POLITECNICO DI TORINO
Repository ISTITUZIONALE

Axp: A hw-sw co-design pipeline for energy-efficient approximated convnets via associative matching

Original
Axp: A hw-sw co-design pipeline for energy-efficient approximated convnets via associative matching / Mocerino, L., Calimera, A.. - In: APPLIED SCIENCES. - ISSN 2076-3417. - 11:23 (11164)(2021). [10.3390/app112311164]

Availability:
This version is available at: 11583/2947616 since: 2021-12-22T17:56:24Z

Publisher:
MDPI

Published
DOI:10.3390/app112311164

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

09 November 2026

applied sciences

MDPI

Article

AxP: A HW-SW Co-Design Pipeline for Energy-Efficient Approximated ConvNets via Associative Matching

Luca Mocerino * and Andrea Calimera *

Department of Control and Computer Engineering, Politecnico di Torino, 10129 Torino, Italy
* Correspondence: luca.mocerino@polito.it (L.M.); andrea.calimera@polito.it (A.C.)

Abstract: The reduction in energy consumption is key for deep neural networks (DNNs) to ensure usability and reliability, whether they are deployed on low-power end-nodes with limited resources or high-performance platforms that serve large pools of users. Leveraging the over-parametrization shown by many DNN models, convolutional neural networks (ConvNets) in particular, energy efficiency can be improved substantially preserving the model accuracy. The solution proposed in this work exploits the intrinsic redundancy of ConvNets to maximize the reuse of partial arithmetic results during the inference stages. Specifically, the weight-set of a given ConvNet is discretized through a clustering procedure such that the largest possible number of inner multiplications fall into predefined bins; this allows an off-line computation of the most frequent results, which in turn can be stored locally and retrieved when needed during the forward pass. Such a reuse mechanism leads to remarkable energy savings with the aid of a custom processing element (PE) that integrates an associative memory with a standard floating-point unit (FPU). Moreover, the adoption of an approximate associative rule based on a partial bit-match increases the hit rate over the pre-computed results, maximizing the energy reduction even further. Results collected on a set of ConvNets trained for computer vision and speech processing tasks reveal that the proposed associative-based hw-sw co-design achieves up to 77% in energy savings with less than 1% in accuracy loss.

Keywords: deep learning; convolutional neural networks; energy efficiency; data reuse; clustering; hw design

check for updates

Citation: Mocerino, L.; Calimera, A. AxP: A HW-SW Co-Design Pipeline for Energy-Efficient Approximated ConvNets via Associative Matching. *Appl. Sci.* **2021**, *11*, 11164. https://doi.org/10.3390/app112311164

Academic Editor: Fabio La Foresta

Received: 29 September 2021
Accepted: 23 November 2021
Published: 24 November 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In the last decade, convolutional neural networks (ConvNets) have outclassed traditional machine learning algorithms in several tasks, from image classification [1,2] to audio [3,4] and natural language processing [5,6]. Many smart applications today make use of ConvNets to infer meaningful information from raw data. Those ConvNets, first trained off-line on a representative set of data and then deployed for on-line inference, can be processed either remotely, on high-performance servers, or locally, close to the source of data, on lightweight end-nodes. In both cases, achieving high energy efficiency is paramount: on the cloud side, it is to reduce the power consumption, the costs of the cooling systems, and hence it is a way to improve maintenance and reliability [7]; on the edge side, is to cope with lower energy budgets and to maximize the battery lifetime still ensuring reasonable processing time [8]. Unfortunately, ConvNets are created heavy and cumbersome, and their algorithmic structure calls for high resource allocation, that is, large memories for storing weights and partial results and highly parallel data-paths for handling massive arithmetic workloads.

The need for practical optimization methods and training flows capable of lowering the hardware requirements attracted large research interest in the last years, leading to many possible alternatives. Most of them are based on different forms of approximate computing strategies enabled by the intrinsic error resilience of ConvNets. Approximations can be applied at different levels by means of different knobs: (i) the data format, with

mini-floats [9,10] or fixed-point quantization [11–13]; (ii) the arithmetic precision, replacing exact multiplications with an approximate version [14,15]; (iii) the algorithmic structure, for instance simplifying standard convolutions with an alternative formulation, such as Winograd [16] or frequency domain convolution [3]. These techniques share a common characteristic: the arithmetic algebra adopted for carrying on matrix convolutions is still built upon the basic multiply and accumulate (MAC) operation.

A broader look at how ConvNets process data may suggest a radical and perhaps more efficient alternative. The convolutional layers are characterized by stencil loops that update array elements according to fixed patterns, thereby producing repetitive workloads with a high degree of temporal and spatial locality. This offers the opportunity to implement reuse mechanisms that alleviate the computational workload. To this end, associative memories that store recurrent results have been proposed as a viable option to replace the arithmetic MAC, and thus to skip redundant computations improving the energy efficiency [17,18]. To further enhance the associative mechanism, additional approximations not strictly related to the arithmetic MAC approximation can offer an orthogonal dimension for optimization. Bit obfuscation and operand precision lowering were used to relax the matching rules indeed, further increasing the repetitiveness of certain patterns and the probability the required data get available in the associative memory [17–19]. To be noted that the error introduced by approximate matching rules might call for auxiliary error-recovering policies through online hardware calibration and/or custom training procedures. This represents a substantial overhead. Furthermore, most of the proposed solutions rely on custom memory architectures thought for Resistive-RAM technologies, which are more difficult to integrate within standard CMOS designs and processes [20].

This work does focus on these aspects introducing a practical alternative to enhance data reuse via approximate associative matching and standard CMOS circuits. Specifically, we propose a hardware–software co-design pipeline consisting of a suite of automatic tools that implement a multi-stage approximation and generate ConvNet models able to maximize the reuse mechanism implemented through a memory-enhanced processing element (PE) borrowed from our preliminary work [21]. Similar input patterns are merged via clustering at first, then a bit-wise approximate matching is implemented in such a way that the associative memory utilization is maximized. Along the pipeline, the error tolerance is regulated through a user-defined threshold, enabling an energy-accuracy tradeoff to satisfy different application-level specifications and/or custom hardware constraints. High prediction accuracy is achieved neither resorting to additional training epochs for weights, nor any custom training procedure. This is an important feature as it ensures the use of pre-trained models maximizing the integration with existing model architectures. The processing element consists of tiny CMOS content addressable memories (CAMs) coupled with a standard floating-point unit (FPU), which is easy to be integrated with modern digital designs, e.g., existing GP-GPUs [17,22,23] or custom spatial accelerators [24–26].

The experiments conducted on computer vision tasks (image classification and object recognition) and keyword spotting reveal that our approach achieves up to 77% of energy savings with a negligible accuracy loss (<1%). Moreover, with the possibility of controlling the accuracy-latency tradeoff at design-time, ConvNets obtained with the proposed technique cover a larger set of operating points (up to 23% energy savings when the accuracy target gets relaxed from 0.5% to 3%). To prove the scalability of our solution, we also repeated experiments considering half-precision floating point data-paths. The results reveal it is possible to achieve a comparable energy efficiency (72%) with negligible accuracy loss (<1%). Finally, the comparison with the state-of-the-art shows remarkable energy saving (up to 29.47%) against prior solutions.

2. Background and Related Works

2.1. Convolutional Neural Networks (ConvNets)

ConvNets belong to a sub-class of deep learning algorithms particularly suited for inference on multidimensional inputs (e.g., images, video, spectrograms). Figure 1 depicts

the internal structure of a classical ConvNet architecture; it consists of two main computing blocks, the first one in topological order (green) dedicated to *features extraction*, the second one (blue) dedicated to *features separation*. The *convolutional* layers (CONV) extract hierarchical features performing multidimensional convolution between the feature maps received as input and their inner filters learned at training time. The CONV layers are usually interleaved with *batch normalization* layers or *pooling* layers (POOL) that normalize or re-scale the feature maps. The high-level and low-dimensional features produced by the extraction block are then fed to the separation block, where *fully-connected* layers (FC) are trained to implement the geometrical, non-linear separation of the features. A final *softmax* layer calculates the output probability score across the available classes.

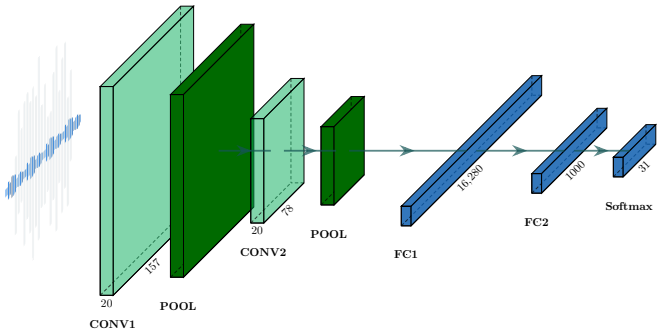


Figure 1. ConvNet architecture adopted on Google Speech Command dataset (GSC) [27].

2.2. ConvNets Approximation via Arithmetic Approximation and Data-Reuse

In this section, we briefly review some of the available energy-driven optimization options for ConvNets, those most related to our proposal in particular. Specifically, we discuss methods based on approximation strategies that do not alter the inner architecture of the model. Interested readers can refer to existing surveys for a comprehensive literature review on the topic [28,29].

Arithmetic precision scaling is by far the most adopted strategy [11,12,30]. The parameters, trained and optimized using a floating-point (FP) representation, are rescaled to integers with a lower bit-width, e.g., 8-bit as the most common option. Even though integer quantization has proven rather quite efficient, especially for resource-constrained devices with limited instruction sets, it is no free lunch. Firstly, handling fixed-point operators introduces additive computational overhead due to range alignment. This calls for optimized kernel implementations depending on the underlying hardware architecture. Secondly, additional *re-training* epochs are often needed in order to recover from the accuracy drop coused by quantization. To notice that re-training is time-consuming and the original training data are not always available for privacy or security reasons [31,32]. Recent trends suggest the use of alternative floating-point formats rather than integers, which require no extra fine-tuning steps, such as Bfloats [10] or reduced precision floating-point [33,34]. Alternatively, custom training loops that implement quantization-aware weights learning [35–37] represent a faster and more reliable option than fine-tuning. At the hardware level, several custom designs make use of lightweight *inexact* MAC units integrated into the data-path [15,38–40] to accelerate the arithmetic workload.

A side-product of the aforementioned algorithmic-level arithmetic approximation via bit-width lowering is the increase of data repetitiveness during the inference stage, and hence the opportunities offered by data-reuse policies. For instance, some prior works investigated the use of memory-based associative mechanisms to replace classical hardware arithmetic units. Razlighi et al. in [18] proposed a look-up search into a special content-addressable memory (CAM) mapped onto a resistive technology as a substitute for multiply-and-accumulate (MAC) units. This approach targeted simple multilayer perceptrons (MLPs), which account for fully connected layers only, while it is known that convolutions layers dominate the energy consumption in ConvNets [41,42]. Also, it

requires additional *re-training* epochs to alleviate the dramatic accuracy drop. Moreover, it was thought for emerging resistive memories. Other proposals elaborated on the same idea embedding associative paths into the standard floating point units: a resistive ternary content addressable Memory (TCAM) was proposed in [43], and a FeFET-based TCAM was used in [44]. Further variants of the same concept foresee the replacement of the associative memory with a classical software-based probabilistic data filter, such as a bloom filter (BF) as proposed in [45], eventually implemented in hardware with resistive technologies. The false positives introduced by BFs when retrieving a pattern contribute to substantial accuracy drops. Moreover, a heterogeneous design with both CMOS and resistive technologies might limit applicability and savings in general. Moreover, even recent general approximate matrix multiplication methods [46], based on product quantization algorithm, target fully connected layers only and require an ad hoc training procedure for learning the hashing functions parameters, which are the basis of the efficient lookup search.

Finally, in [21] we proposed a hardware–software co-design flow to implement a fully-CMOS processing element that integrates an SRAM-based CAM into a standard FPU. The design is enabled by a clustering procedure aimed at boosting the intrinsic reuse opportunity. Experimental results have shown that achieving high accuracy may call for large memory configurations, because of the high number of centroids needed and/or the high number of input activations to be stored, reducing the energy savings achievable. Borrowing the original idea proposed in [21], this work makes a step further introducing the use of approximate matching for associative-based ConvNet processing.

3. Co-Design Pipeline: Concept and Methodology

As anticipated in the previous sections, the basic principle of our proposal is to accelerate the inference stage of a ConvNet by pre-computing and storing the most recurrent multiplications offline and then reusing them whenever the same input pattern occurs. We first introduce the processing element and its design parameters, then we provide a detailed description of the co-design pipeline.

3.1. Hardware Design

An overview of the PE architecture is depicted in Figure 2a. There is an FPU composed of a multi-stage floating-point multiplier (FPMul) and a single-stage accumulator, coupled with an associative memory storing the most representative input patterns and weights values. The PE functionality is simple, yet highly efficient. The CAMs process the input operands in parallel. If they (partially) match those stored, the multiplication result is read from the associative memory and the FPMul clock gated, otherwise, the multiplication is normally computed using the FPMul. Due to the optimization stage done at design time, the FPMul is turned off for most of the cycles with a marginal contribution to the total power consumption.

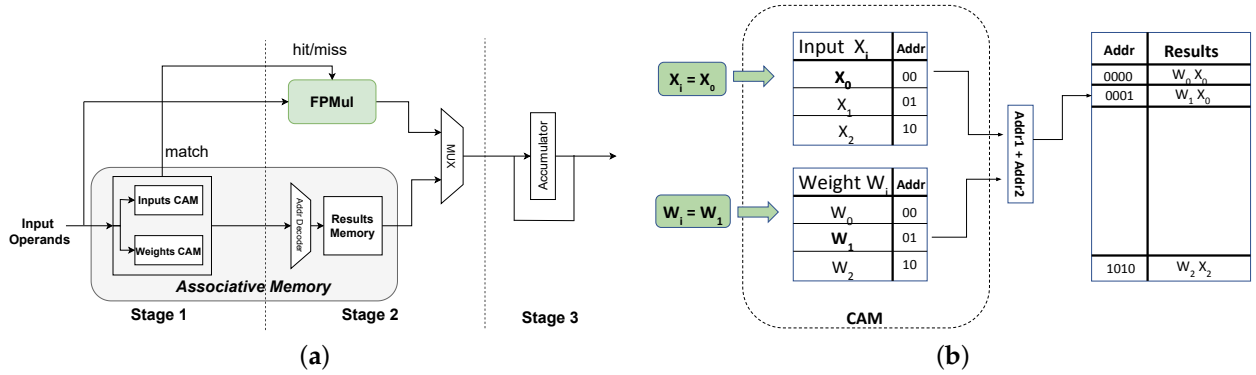


Figure 2. A functional overview of the processing element (**a**) and associative memory (**b**).

We implemented both single and half-precision floating-point units. The architecture of the associative memory is depicted in Figure 2b. It is composed of two CAMs, one dedicated to the most recurrent weights, the other dedicated to activations values. The SRAM stores the results of the pre-computed approximate multiplications. The proposed architecture is inspired to [18], whose design is more efficient in terms of area and energy compared to a single CAM solution with a double word bit-width (weight and input concatenation). If compared to a standard pipelined floating-point MAC, the proposed hybrid processing element (PE) has an additional pipeline stage (Stage 1 in Figure 2a). The latency of such an additional stage is given as the search time consumed by two parallel CAMs and is function of the CAM configuration/size, of course. According to our experiments and simulations, the latency of the largest CAM configuration (512 rows) fits the design clock. To notice that larger CAM configurations can be designed (at no clock-period penalty) leveraging a modular architecture composed of multi-stage sub-CAMs, with a base module that replicates itself in cascading stages [18]. The design and assessment of circuit and/or architecture optimization is out of the scope of this work. It is worth emphasizing that the second and last stage of our hybrid processing element PE (Stage 2 in Figure 2a) has a shorter latency compared to the FPMul. This would enable further speed-up and optimization options, as most of the MAC operations could be faster. Nonetheless, we kept the same clock period of the FPMul thus ensuring synchronization of the workload.

The associative memory size and hence its performances are characterized by a set of parameters defined as follows:

- N_w denotes the number of rows in CAM dedicated to relevant weights value;
- N_{in} indicates the number of rows in CAM dedicated to the most frequent input and activations;
- $Abit$ refers to the CAM and SRAM memory word size.

The parameter set (N_w, N_{in}, $Abit$) describes the size of the two CAMs and hence the SRAM memory defined as $N_m = (N_w \cdot N_{in}) \cdot Abit$. Playing with different configurations of those parameters leads to distinct memory designs and different energy consumption. To this aim, the co-design tool explores those memory configurations to minimize the energy consumption for a given user-defined accuracy constraint. This process might return custom configurations, e.g., uneven word size ($Abit$) or non 2's power N_w and N_{in}. However, as depicted in Figure 3, we opted for a more regular memory structure with 8-bit word bit-width when considering a PE with a half-precision (FPMul 16) and 16-bit word bit-width when considering a 32-bit data format (FPMul 32).

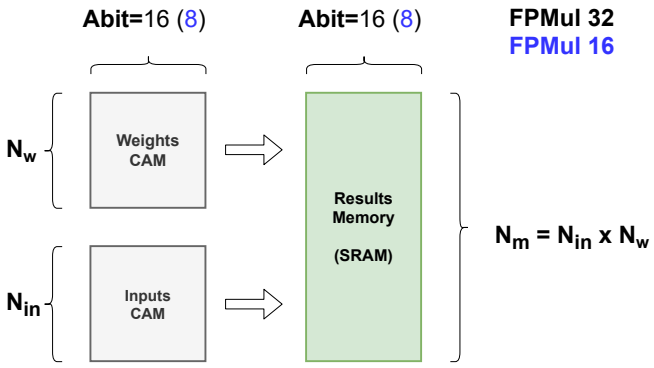


Figure 3. Energy model overview.

3.2. Energy Model

Given N_w, N_{in} and $Abit$, the overall energy consumption for a look-up in the associative memory is calculated through Equation (1), where E_w and E_{in} are the energy contributions due to the CAMs with size N_w and N_{in} respectively, and E_m is the contribu-

tion given by the patterns memory of size N_m. The energy values come from a hardware characterization done offline.

$$E_{lookup} = E_w(N_w, Abit) + E_{in}(N_{in}, Abit) + E_m(N_m) \quad (1)$$

The simulation framework integrates the energy model shown in Equation (2), where hr represents the hit rate, E_{mul} is the energy consumption of the multiplier, E_w and E_{in} refer to the energy consumption of the two CAMs given in Equation (1), E_{hit} is the energy consumed once the inputs do match the CAM content (the pre-computed result is retrieved from the small-size SRAM memory), and E_{miss} is the energy due to a missing search of the pattern (in this case the multiplication is effectively computed).

$$\begin{aligned} E_{hit} &= hr \cdot E_{lookup} \\ E_{miss} &= (1 - hr) \cdot (E_{mul} + E_w + E_{in}) \\ E_{tot} &= E_{miss} + E_{hit} \end{aligned} \quad (2)$$

One can compute the energy-saving by comparing the energy consumed by the stand-alone (single or half precision) FPMul to that retrieved by the energy model above.

3.3. Software Design

To enable such associative-based implementation, we introduce a co-design framework depicted in Figure 4. It consists of two main stages: an *Optimization Stage* and a *Simulation Stage*. The former searches for possible candidates of the memory and model configurations that satisfy the user-defined constraints. The latter does a test over the relevant configurations elaborated in the previous stage emulating the inference stage on the designed PE, providing the weight-set of the clustered ConvNet and the corresponding energy-efficient associative memory settings as main outcome. The *Optimization Stage* is the core of the framework. It has two principal components: (i) the *Approximation Pipeline (AxP)* that performs an iterative dual-step approximation based on a clustering & approximate flow applied on the ConvNet model; (ii) the *Activation Pattern Profiling (APP)* stage that works in parallel extracting relevant statistics on the activation maps produced by the inner processing layers of the ConvNet model. The following sections provide a more detailed description of these components.

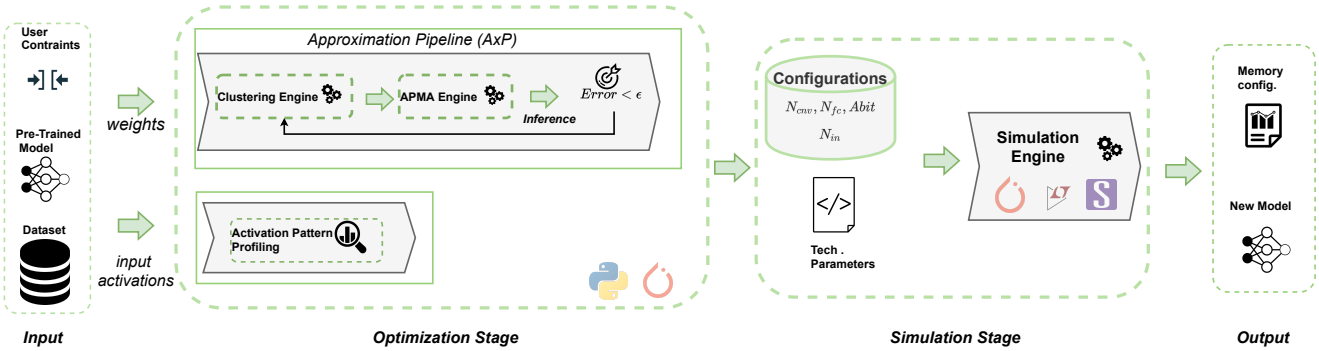


Figure 4. Co-Design tool overview.

3.4. Approximation Pipeline (AxP)

The approximation pipeline (AxP) is built to increase data repetitiveness across the ConvNet model. The rationale is that reducing the number of different parameters and activations minimizes the arithmetic unit utilization since most of the operations can be pre-computed and reused during the inference by exploiting the associative-based processing element. To achieve this, an iterative clustering and approximation procedure is implemented as described by Algorithm 1.

Algorithm 1: Approximation Pipeline (AxP) Algorithm

```
1  Input: model, valid_set, acc, ϵ, max_iter
2  Output: configs
3  configs = list()
4  for N_cnv, N_fc in S_w and max_iter >0 do
5      N_cnv, N_fc, new_model = clustering (model)
6      for Abit in S_ab do
7          Abit, new_model = apma (new_model, Abit)
8          new_acc = forward (new_model, valid_set)
9          Error = acc - new_acc
10         if Error ≤ ϵ then
11             configs.append (N_cnv, N_fc, Abit);
12         end
13     end
14     max_iter- -
15 end
```

The main inputs are (i) the pre-trained *model*, (ii) the validation set (*valid_set*), (iii) a user-defined accuracy drop target (ϵ), and (iv) a stopping variable (*max_iteration*) to ensure convergence of the iterative loop. The algorithm delivers as main outcome the set of configurations (N_{cnv}, N_{fc}, *Abit*) which are fed as input to the latter stage of the framework, namely, the simulation engine. Going deeper into details, the AxP is composed of two sub-stages run sequentially: `clustering` and *Approximate Pattern Matching Analysis* (`apma`). The `clustering` step explores different solutions in a discrete space $S_w \in \{2^1, \ldots, 2^n\}$, n defined empirically, seeking the optimal number of weight clusters for both the convolutional and the fully-connected layers (the parameters N_{cnv} and N_{fc} respectively). The optimal number of bits used for the approximation pattern matching (the parameter *Abit*) is evaluated during the `apma` sub-stage, which explores the viable configurations in the interval $S_{ab} = [\frac{bw}{2}, bw]$, bw as the parallelism of the current hardware data-path (32- or 16-bit in our study). During the iterative loop, shown in Algorithm 1, all the tuples (N_{cnv}, N_{fc}, *Abit*) that satisfy the accuracy constraint are kept as possible optimal solutions. Here the tool employs the solution found in the `clustering` stage, characterized by N_{cnv} and N_{fc} parameters, and iteratively performs the forwarding pass on the validation set. In each iteration, a certain partial matching configuration (while N_{cnv} and N_{fc} are fixed) is evaluated on the validation set and all the solutions that meet the user constraint ϵ are collected. This additional approximation phase introduces a new degree of freedom in the design phase and a finer grain control in the accuracy vs. energy efficiency space.

From here follows the detailed description of `clustering` and `apma` sub-stages and the knob involved.

3.4.1. Clustering Engine

During the clustering stage, the ConvNet's weights are merged into the lowest possible number of clusters using a similarity distance metric. Once the engine returns the cluster centroids, all the weights in a certain range are mapped on the corresponding centroid values. This procedure affects the ConvNet model complexity, that is, the lower the number of clusters, the lower the cardinality of the weight-set. Indeed, only the values of the cluster centroids are the weight patterns to be stored.

The clustering procedure was built upon the jenks natural breaks (JNB) algorithm [47]. It is an iterative method whose objective is to maximize the inter classes variance while minimizing the intra-class variance. The weight tensors are unrolled and treated as a 1D vector, then sorted by magnitude during a pre-processing step to achieve speed-up. We

opted for a layer-wise clustering granularity with a differentiated strategy for convolutional and fully connected layers. Each filter in a CONV layer is clustered independently, while the entire weight matrix is considered for an FC layer. The resulting number of clusters are N_{cnv} and N_{fc} for CONV and FC layers respectively. The differentiated clustering strategy adopted for the two types of layers is motivated by empirical evidence as it decreases both the number of centroids and the accuracy drop.

As a practical example, the histograms reported in Figure 5 show the effect of the clustering step on the weights-set of an FC layer using $N_{fc} = 16$ clusters. The original peak frequency for near-zero values (≈5k) is doubled after the `clustering` step (≈10k), which suggests that the occurrence of zero patterns gets doubled at inference-time, hence opening to a higher matching probability exploitable by an associative-memory based processing element.

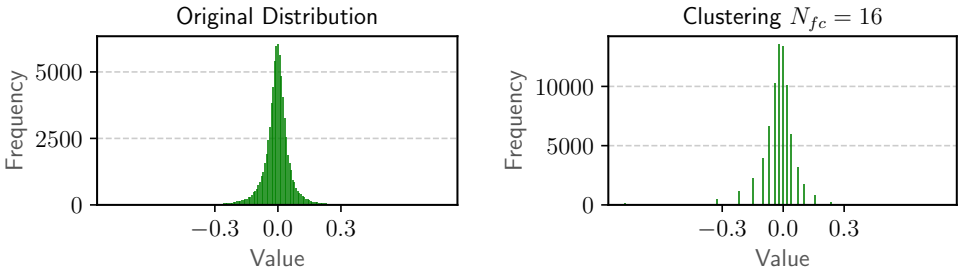


Figure 5. Clustering. Original weights distribution (**left**), weight distribution after clustering (**right**).

3.4.2. APMA Engine

Once the weight-set of the ConvNet model has been discretized across a lower number of clusters, the second stage encompasses the approximate pattern matching analysis (APMA) (see Figure 4). APMA takes as input the model processed by the clustering engine and returns the approximation bits (*Abit*) parameter, which is the number of bits used to implement the approximate matching function for the retrieving of the pre-computed results stored in the associative memory of the processing element. The main objective is to minimize the *Abit* while ensuring an accuracy drop lower or equal to the user-defined constraint.

The target data formats are the single and half-precision floating-point defined within the IEEE 754 standard. Figure 6 reports the single-precision floating-point format, composed of a sign bit, 8 bits for the *exponent* and 23 fraction bits. During the approximate matching, the least significant bits (LSB) are obfuscated, using the most significant bits (MSB) as the search key. For instance, a possible approximate matching schema makes use of 13 MSBs, pruning 19 fraction bits. This mechanism replaces the exact matching between current input patterns (weight and activation) and the pre-computed patterns in the associative memory, increasing the hit rate at the cost of a certain accuracy drop. It is worth noticing this mechanism can be regulated with the *Abit* knob, which affects the memory size, the energy efficiency, and the accuracy drop.

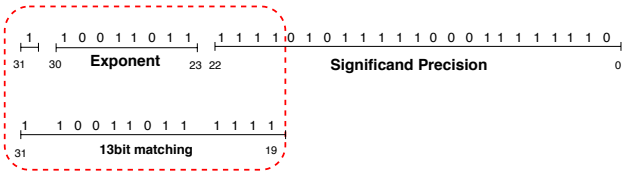


Figure 6. IEEE 754 single-precision floating point format (**top**) and a possible approximate matching (**bottom**).

3.4.3. Activation Pattern Profiling (APP)

We exploit the information stored in the training data to retrieve the best representative values for the possible inputs and activation. The tool performs the feed-forward pass

on the training data to collect and profile the most frequent inputs and the intermediate activation values. Those values fed the dedicated CAM memory (*Inputs CAM* in Figure 2a) and based on the empirical evaluation, we adopted a layer-wise granularity, where (N_{in}) indicates the number of input activations to store for each layer.

3.5. Understanding Co-Design Knobs

The optimization stage returns a set of optimal configurations (N_{cnv}, N_{fc}, N_{in}, $Abit$) and the corresponding accuracy degradation. Each configuration reflects a different hardware setting, which in turn affects the energy efficiency and the accuracy drop as follows:

- N_w defined as max(N_{cnv}, N_{fc}) and $Abit$ affect the weight-CAM size and the overall accuracy;
- N_{in} and $Abit$ impact the activation-CAM size and the overall hit rate in the associative memory;
- N_w, N_{in} and $Abit$ contribute to the static random access memory (SRAM) size for collecting the multiplication results: $N_m = (N_w \cdot N_{in}) \cdot Abit$.

Fixing $Abit$, there exist solutions with the same accuracy degradation (same N_w), but different associative memory sizes (varying N_{in}), hence higher or lower energy savings.

3.6. Simulation Engine

Once all viable configurations have been computed and assessed based on the actual accuracy target, a further iterative step emulates the inference stage on the customized hardware. This step encompasses the assessment of the CAM-enhanced FPU. The associative memory is initialized with the patterns provided by the optimization pipeline. The technological characterization of the PE (*Tech. Parameters* in Figure 4) and the hit rate in the associative memory are used for estimating the energy consumption. The best associative memory configuration that satisfies the user's constraints with minimum energy consumption is finally returned as the main outcome of the framework. It is worth emphasizing that multiple accuracy constraints can be used by the end-user to trade energy efficiency with performance according to specific requirements (hardware equipment or application).

4. Experimental Results

4.1. Benchmarks and Datasets

Table 1 reports the benchmarks adopted. Specifically, the dataset and ConvNet employed for each task, the input data size, and the detailed model topology. It also collects the baseline accuracy for the full-precision model (FP32) and the half-precision one (FP16). A detailed description of the tasks follows.

Table 1. Benchmark overview. Convolutional layers with shape ($ch_o \times kh \times kw$), fully-connected layers with shape ($ch_i \times ch_o$) and pooling layers with shape (kh, kw, s); kh and kw are kernel height and width, s is the stride, ch_i and ch_o refer to the number of input and output channels, respectively.

Task		**IC**		**OR**		**KS**	
Dataset		MNIST [48]		GTSRBD [49]		GSC [27]	
Model		LeNet-Like [45]		LeNet-5 [50]		GscNet [27]	
Input		$1 \times 32 \times 32$		$3 \times 32 \times 32$		$1 \times 44 \times 44$	
Topology		Conv	$6 \times 5 \times 5$	Conv	$6 \times 5 \times 5$	Conv	$20 \times 5 \times 5$
		MaxPool	$(2, 2, 2)$	MaxPool	$(2, 2, 2)$	MaxPool	$(2, 2, 1)$
		Conv	$16 \times 5 \times 5$	Conv	$16 \times 5 \times 5$	Conv	$20 \times 5 \times 5$
		MaxPool	$(2, 2, 2)$	MaxPool	$(2, 2, 2)$	MaxPool	$(2, 2, 1)$
		Conv	$120 \times 5 \times 5$	Conv	$120 \times 5 \times 5$	FC	$16{,}280 \times 1000$
		FC	120×10	Conv	120×84	FC	1000×31
				FC	84×43		
Acc. Top-1 (%)	FP32	98.82		87.13		69.41	
	FP16	98.67		87.10		69.22	

Image Classification (IC): the goal is to classify 10 different handwritten digits. The dataset employed is the popular MNIST [48] consisting of 60 k 32 × 32 gray-scale images, 50k samples used for training, 10k for testing. The model employed is inspired by the LeNet [50] architecture widely adopted in previous works.

Object Recognition (OR): the objective is to recognize 43 different traffic signals, a popular sub-task for autonomous driving. The dataset adopted is the German Traffic Sign Recognition Dataset (GTSRD) [49]. It collects 50 K samples of 32 × 32 RGB samples of traffic signals spread in the German streets, 40 k used for training, 10k testing. For this task, the ConvNet architecture is the LeNet-5 [50] model.

Keyword Spotting (KS): the objective is to recognize 30 simple vocal commands. The adopted dataset comes from Google research, Google Speech Commands (GSC) [27], with 65k one-second-long audio samples of 30 different keywords plus noise, labeled as "unknown". The inputs are the 2D spectrograms of the recorded samples, 56,196 samples are for training, 7518 for testing. The model adopted is the GSCNet taken from an open-source repository [51] ,which is inspired by the original work [27].

4.2. Hardware and Software Setup

We designed a PE with a single and half-precision floating-point multiplier using the open-source FloPoCo library [52]. The energy performance is retrieved from Synopsys Design Compiler and PrimeTime leveraging a commercial CMOS 28nm FDSOI technology library from STMicroelectronics. The CAMs are designed and mapped with a standard 6T cell that comes from the 28 nm technology library. The energy characterization was done in HSPICE by Synopsys. Experimental results show that the adopted design performance is aligned with state-of-the-art CAM memories [53]. The SRAM bank is characterized using CACTI 7.0 [54]. A parametric characterization of the three memory components is depicted in Figure 7. Here, the energy consumption increases with memory size (the number of rows) while fixing the word size as described in the previous section.

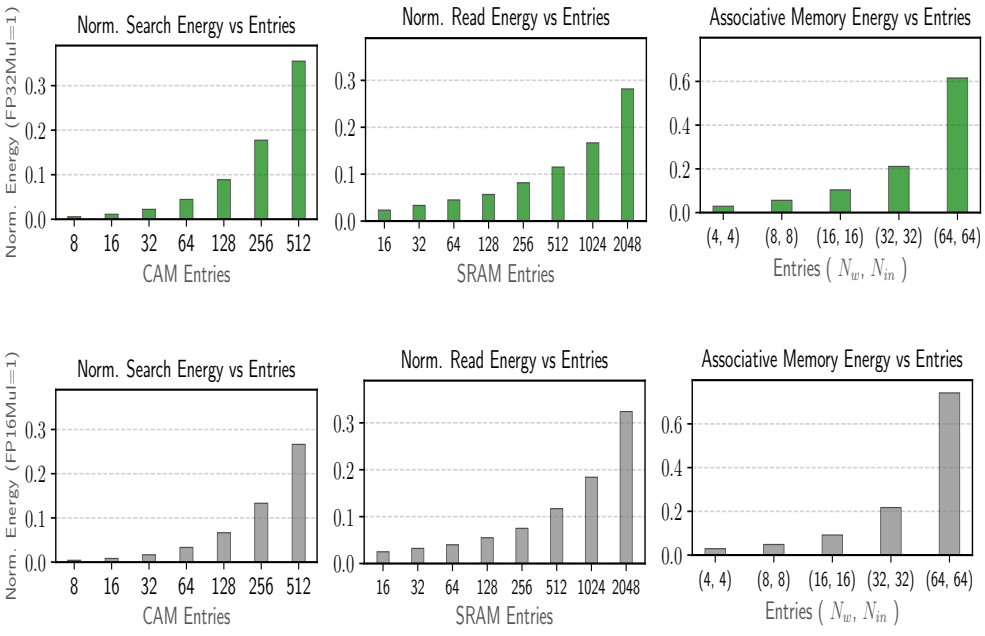


Figure 7. Normalized energy vs. entries: from left to right CAM, SRAM and associative memory. The energy is normalized with respect to single precision (green bars) and half precision multiplier (grey bars).

The entire framework was built upon the deep-learning framework PyTorch (v1.4). The baseline models were trained with hyper-parameters from original papers [27,50]. Both training and inference stages were run on a server powered with 40-core Intel Xeon CPUs and accelerated with NVIDIA Titan Xp GPU (CUDA 10.0). The models were trained twice

using FP32 first and FP16 then, however, Table 1 revealed that there is no difference for the considered benchmarks.

4.3. Weight Approximation Pipeline

This section quantifies the existing relationship between the prediction accuracy of ConvNets and the design knobs explored in the approximation pipeline. A first analysis shows how clustering (i.e., N_{cnv} and N_{fc}) affects the accuracy along with the benefits of the proposed approximate pattern matching strategy.

The `clustering` effects are depicted in Figure 8 where the accuracy drop is linked to the number of clusters used in that stage. For graphical purposes, we just plotted the solutions where $N_{cnv} = N_{fc}$, however, the tool can explore also uneven working points. Intuitively, the accuracy degradation decreases for larger values of N_{cnv} and N_{fc}. This behavior shows the hidden relationship between the number of different parameters, i.e., the model complexity and the generalization ability of a ConvNet. The break-even (near-to-zero accuracy) configuration differs among benchmarks and it is strictly related to the model topology and the task complexity. In particular, for ConvNets with similar size (e.g., for OR and IC tasks), the number of clusters required are strictly related to the task difficulty (43 vs. 10 classes), when the number of weights grows up (wider architecture) more clusters are required to represent the entire weight space, as shown by the KS benchmark for instance. The results achieved are remarkable. It is possible to shirk the complex high-dimensional information of a ConvNet model with no substantial loss in the prediction ability. For instance, negligible accuracy degradation is obtained with $N_{cnv} = N_{fc} = 64$ for OR, $N_{cnv} = N_{fc} = 16$ for IC and $N_{cnv} = N_{fc} = 512$.

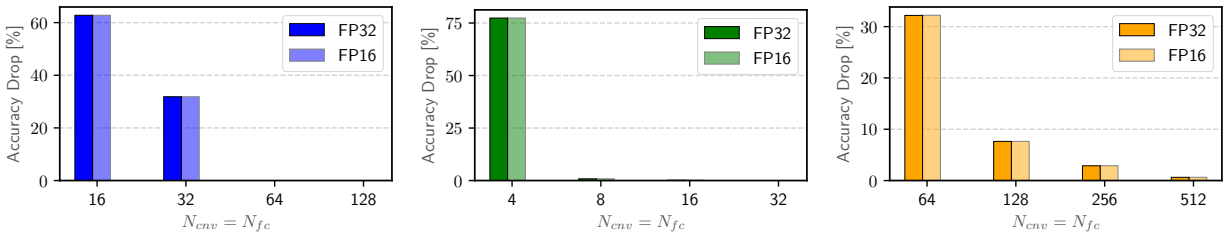
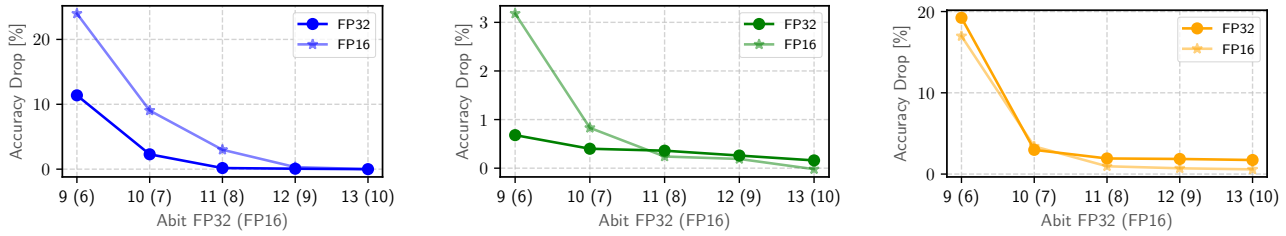


Figure 8. Accuracy drop (%) variation changing the weight clusters configurations (N_{cnv}, N_{fc}). Results for 32 bit (FP32) and 16 bit (FP16) data. (**a**) OR; (**b**) IC; (**c**) KS.

The *Abit* is a key knob in our proposal, as it affects both accuracy and hit rate in the associative memory. Figure 9 shows the accuracy degradation trend for a different number of bits used to implement the approximate matching. In order to assess the impact due to *Abit*, we fixed the number of clusters (N_w) such a way that the accuracy drop is negligible: $N_w = 64$ for OR, $N_w = 16$ for IC and $N_w = 512$ for KS. Those points represent possible candidate solutions indeed. The accuracy drop decreases as *Abit* gets larger. Although the initial error differs among benchmarks, the common ground is that it gets close to zero using 13 MSB over the whole 32-bit; for the FP16 format, 8 MSBs are just enough to achieve negligible accuracy losses (<1%). From Figures 8 and 9, it is also clear that *Abit* is a finer knob when compared to the number of clusters, as it leads to minor accuracy variations.

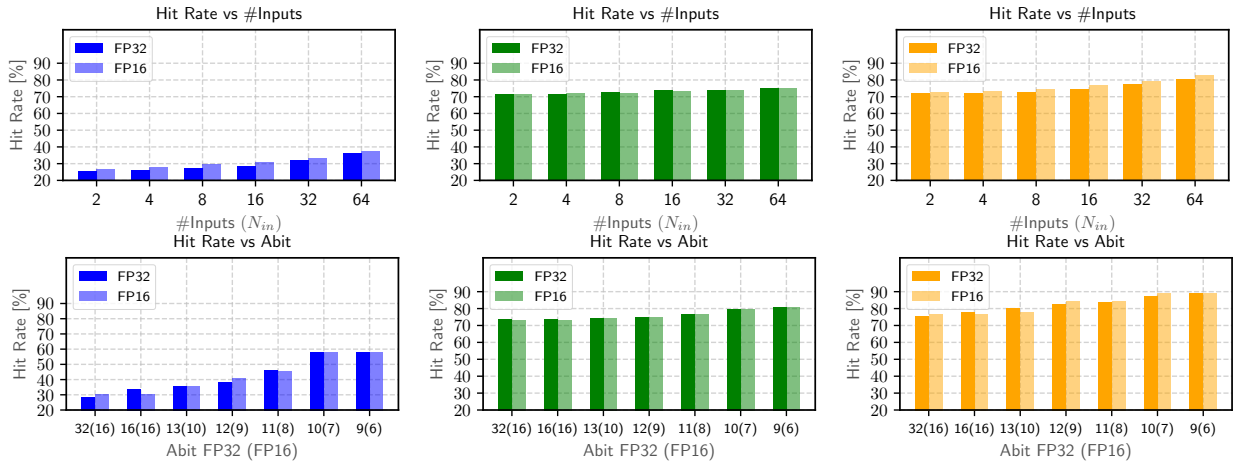


(**a**) (**b**) (**c**)

Figure 9. Accuracy drop (%) when changing the *Abit*, fixing the number of clusters: $N_w = 64$ for OR, $N_w = 16$ for IC and $N_w = 512$ for KS. Results are presented for 32 bit (FP32) and 16 bit data (FP16). (**a**) OR; (**b**) IC; (**c**) KS.

4.4. Input Activations Profiling

Figure 10 (top) shows the hit rate trends as function of the number of input and activation patterns stored N_{in}. As expected, the hit rate increases with N_{in}, yet with different rates depending on the task. IC and KS reach higher hit rates for instance (71–81%). The best case for IC gets close to 75%, whereas KS goes even higher to 81% when $N_{in} = 64$. The direct implication of this observation is that the FPMul workload decreases up to 80%, which means a mere 20–30% of the overall input patterns are handled as classical arithmetic operations. The results get slightly different for the OD task where the hit rate ranges are limited between 25% and 36%. This might be due to the higher variance in input data. Similar behavior is depicted when scaling bit-width (FP16). Here, the hit rate for OR and KS slightly increases (up to ≈3%) due to the natural reduction of the bit-width. The peak performance is recorded for the KS benchmark, where the hit rate ranges from 76% to 89%.



(**a**) (**b**) (**c**)

Figure 10. On top, the hit rate variation changing the number of most frequent inputs and intermediate activation profiled (N_{in}). On bottom, the hit rate (%) variation when adding the approximation pattern matching, controlled by *Abit*. $N_{in} = 16$. (**a**) OR; (**b**) IC; (**c**) KS.

However, an increment in the hit rate due to a higher number of clusters has a side effect on energy consumption. Higher N_{in} means larger associative memory indeed, and hence more energy consumption. This behavior motivates the need to balance those contrasting metrics (hit rate and energy consumption) through extensive co-design exploration.

4.5. Approximate Pattern Matching on Input Activation

Figure 10 (bottom) shows the hit rate when varying *Abit* while fixing the other knobs ($N_{in} = 16$). This configuration guarantees a moderate hit rate (as depicted in Figure 10 (top)) and may be a good representation point to analyse the impact of `apma` stage on hit rate.

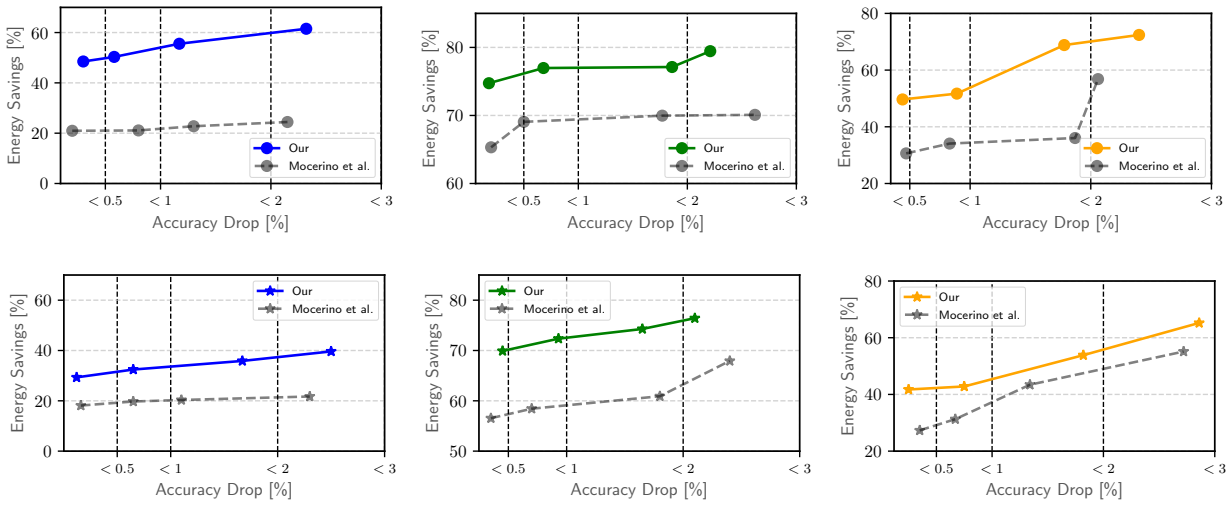
As expected, the hit rate increases when resorting to a lower number of bits for approximate matching. Specifically, the hit rate increases up to 58% for OR, 83% for IC, and 89% for KS. More interesting is the understanding of how the hit rate behavior changes after the `apma` step to isolate the impact on that stage. This is depicted in Figure 10 (bottom), where the leftmost values represent the baseline values (equals to Figure 10 (up), $N_{in} = 16$). When comparing this behavior with the hit rate after the clustering (Figure 10 (bottom), leftmost values) it turns out that the `apma` phase leads to substantial hit rate improvements.

In particular, even on the benchmark with a high hit rate after the clustering step, the `apma` step leads to an additional improvement: up to 14% (13%) for KS and 9% (10%) for IC, referred to 32 bit-data width (16 bit-data). Moreover, the results are impressive on OR benchmark, where this additional approximation step guarantees up to for 30% (FP16 28%) of hit rate improvement. This suggests that the `apma` phase is very effective, as it leads to new energy-efficient solutions, unreachable with the clustering stand-alone.

4.6. Energy-Accuracy Trade-Off and Comparison with Previous Works

The plots collected in Figure 11 show the Pareto analysis obtained running multiple instances of the framework under different user-defined accuracy constraints.

Those constraints on the x-axis (dashed vertical lines) are represented as the accuracy drop w.r.t. the nominal accuracy (in Table 1). On the y-axis, the energy savings w.r.t. a standard convolution-as-GEMM implementation running on fully arithmetic processing element are reported. In this space, the plots depict our solutions (in colored full lines) and our preliminary work (in grey dashed lines). Each color refers to a different benchmark: object recognition with blue, image classification task with green, and keywords spotting with yellow. Moreover, the top row refers to the FP32 data-path (circle marker), whereas the bottom one to FP16 (star marker).



(**a**) (**b**) (**c**)

Figure 11. Pareto point into energy-accuracy space for each benchmark. On the top, results corresponding to single-precision floating-point data-path, on the bottom half-precision. Each column (and color) represents a different benchmark, from left to right: OR, IC and KS. The current approach is compared to our preliminary work [21] (grey dashed lines). (**a**) OR; (**b**) IC; (**c**) KS.

Intuitively, energy savings get lower for stricter accuracy constraints. The amount of energy savings vary across the benchmarks as a consequence of different configurations, hence different associative memory designs. Looking at the FP32 results, with a close-to-zero accuracy drop (<0.5%), energy savings are substantial: 48.50% (OR), 74.75% (IC), 49.65% (KS). Results get even more interesting when relaxing the constraints (<3%): 61.50% (OR), 79.45% (IC), 72.40% (KS).

Efficacy has been proven for FP16 too, yet with narrow margins. Specifically, the energy savings range from 29.32% to 39.5% (OR), 69.91% to 76.42% (IC), and from 41.74% to 65.22% (KS) for the three accuracy constraints respectively. The best scaling ratio is shown by KS with an improvement of 22.75% relaxing the constraints, while IC shows a smoother scaling with 5.45% of accuracy improvement. Reducing data path bit-width leads to similar trends, in particular, KS presents a maximum scaling of 23.38% while the minimum occurs for IC (6.51%).

Concerning prior arts, the first analysis we conducted aims at showing the improvements brought by our new framework when compared to our preliminary work [21] (gray dashed line in Figure 11). The energy efficiency increases up to 37.06% for OR, 9.42% for IC, and 32.79% for KS for FP32 data path, while for FP16 data path it is possible to save up to 13.37% for OR, 17.92% for IC, and 14.42 for KS from a strict to more relaxed accuracy constraint. It is noted that our solution heavily improves the energy efficiency on benchmarks where our preliminary work shows the poorest results (e.g., OR benchmark). This gets empirical evidence that the additional approximation step introduced with this work enhances data-reuse opportunity. Even though savings slightly decrease, the results on the FP16 data path confirm the dominance of the new approximate technique.

As a final remark, Table 2 reports a fair comparison to other prior works based on associative-based computing integrated with a floating-point unit [38,40] or probabilistic data-structure such as bloom filter [45]. Moreover, in Table 2 details on the memory technology adopted, the type of cell, the technological node and the clock period of processing element are reported. The voltage supply for each solution is set to 1V. The analysis involves a common benchmark (IC) under the same accuracy constraint (<1%).

Table 2. Comparison w.r.t. state of the art.

Work	Dataset	Energy Savings [%]	Memory Tech.	Cell Type	Tech. Node (nm)	T_{clk} (ns)
Our	IC	76.97 (+29.47)	CMOS	6T	28	1.5
[45]		47.50	ReRAM	1T1R	45	1.5
[40]		45.92	-	-	45	-
[38]		44.81	FeFET	4T-2FeFET	45	-

The results reveal the proposed framework outperforms prior arts, with up to 29.47% of energy improvement. This achievement is the result of a joint hardware–software optimization process, in which a sophisticated data-reuse strategy fully exploits the compact and energy-efficient associative processing element.

Finally, we found that the search time for a feasible solution grows in the case of large ConvNets (i.e., with a high number of channels). We realized that the bottleneck is the clustering stage, where the clustering iterations in convolutional layers increase with the number of active channels. There are different ways to address this limitation, for instance adopting a recently proposed high parallel implementation [55,56].

5. Conclusions

This work presents a hardware–software co-design tool that affects the arithmetic workload in ConvNet processing: (i) enhancing the recurrent pattern reuse with an *ad hoc* approximation pipeline composed of a clustering step followed by the approximate pattern matching phase; (ii) integrating an FPU with associative memory. We experimentally tested the solution on three different applications reaching up 77% of energy-saving under a negligible accuracy drop (<1%) outperforming all priors-related solutions. Results are almost preserved with precision scaling.

Author Contributions: Conceptualization, L.M. and A.C.; Investigation, L.M. and A.C.; Methodology, L.M. and A.C.; Resources, A.C.; Software, L.M.; Writing—review & editing, L.M. and A.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

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

References

1. Druzhkov, P.; Kustikova, V. A survey of deep learning methods and software tools for image classification and object detection. *Pattern Recognit. Image Anal.* **2016**, *26*, 9–15. [CrossRef]
2. Tuia, D.; Volpi, M.; Copa, L.; Kanevski, M.; Munoz-Mari, J. A survey of active learning algorithms for supervised remote sensing image classification. *IEEE J. Sel. Top. Signal Process.* **2011**, *5*, 606–617. [CrossRef]
3. Abdel-Hamid, O.; Mohamed, A.R.; Jiang, H.; Deng, L.; Penn, G.; Yu, D. Convolutional neural networks for speech recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2014**, *22*, 1533–1545. [CrossRef]
4. Kwon, S. A CNN-assisted enhanced audio signal processing for speech emotion recognition. *Sensors* **2020**, *20*, 183.
5. Lopez, M.M.; Kalita, J. Deep Learning applied to NLP. *arXiv* **2017**, arXiv:1703.03091.
6. Babić, K.; Martinčić-Ipšić, S.; Meštrović, A. Survey of Neural Text Representation Models. *Information* **2020**, *11*, 511. [CrossRef]
7. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; pp. 1–12.
8. Lu, Z.; Rallapalli, S.; Chan, K.; La Porta, T. Modeling the resource requirements of convolutional neural networks on mobile devices. In Proceedings of the 25th ACM International Conference on Multimedia, Mountain View, CA, USA, 23–27 October 2017; pp. 1663–1671.
9. Kang, H.J. Short floating-point representation for convolutional neural network inference. *IEICE Electron. Express* **2018**, *15*, 20180909. [CrossRef]
10. Kalamkar, D.; Mudigere, D.; Mellempudi, N.; Das, D.; Banerjee, K.; Avancha, S.; Vooturi, D.T.; Jammalamadaka, N.; Huang, J.; Yuen, H.; et al. A study of BFLOAT16 for deep learning training. *arXiv* **2019**, arXiv:1905.12322.
11. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June, 2018; pp. 2704–2713.
12. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv* **2016**, arXiv:1609.07061.
13. Mocerino, L.; Calimera, A. Fast and accurate inference on microcontrollers with boosted cooperative convolutional neural networks (bc-net). *IEEE Trans. Circuits Syst. Regul. Pap.* **2020**, *68*, 77–88. [CrossRef]
14. Hashemi, S.; Bahar, R.I.; Reda, S. DRUM: A dynamic range unbiased multiplier for approximate applications. In Proceedings of the 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, USA, 2–6 November 2015; pp. 418–425.
15. Camus, V.; Schlachter, J.; Enz, C.; Gautschi, M.; Gurkaynak, F.K. Approximate 32-bit floating-point unit design with 53% power-area product reduction. In Proceedings of the ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference, Lausanne, Switzerland, 12–15 September 2016; pp. 465–468.
16. Li, S.; Park, J.; Tang, P.T.P. Enabling sparse winograd convolution by native pruning. *arXiv* **2017**, arXiv:1702.08597.
17. Imani, M.; Rahimi, A.; Rosing, T.S. Resistive configurable associative memory for approximate computing. In Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 1327–1332.
18. Razlighi, M.S.; Imani, M.; Koushanfar, F.; Rosing, T. Looknn: Neural network with no multiplication. In Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 1779–1784.
19. Imani, M.; Masich, M.; Peroni, D.; Wang, P.; Rosing, T. CANNA: Neural network acceleration using configurable approximation on GPGPU. In Proceedings of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju Island, Korea, 22–25 January 2018; pp. 682–689.
20. Jalaleddine, S.M. Associative memories and processors: The exact match paradigm. *J. King Saud Univ. Comput. Inf. Sci.* **1999**, *11*, 45–67. [CrossRef]
21. Mocerino, L.; Tenace, V.; Calimera, A. Energy-Efficient Convolutional Neural Networks via Recurrent Data Reuse. In Proceedings of the 2019 Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 848–853. [CrossRef]

22. Peroni, D.; Imani, M.; Nejatollahi, H.; Dutt, N.; Rosing, T. ARGA: Approximate Reuse for GPGPU Acceleration. In Proceedings of the 56th Annual Design Automation Conference 2019 (DAC '19), Las Vegas, NV, USA, 2–6 June 2019; Volume 8, pp. 1–6. [CrossRef]
23. Peroni, D.; Imani, M.; Rosing, T. ALook: Adaptive Lookup for GPGPU Acceleration. In Proceedings of the 24th Asia and South Pacific Design Automation Conference (ASPDAC '19), Tokyo, Japan, 21–24 January 2019; pp. 739–746. [CrossRef]
24. Mittal, S. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Comput. Appl.* **2020**, *32*, 1109–1139. [CrossRef]
25. Locke, K. *Parameterizable Content-Addressable Memory*; Xilinx Application Note XAPP1151; Xilinx: San Jose, CA, USA 2011.
26. Irfan, M.; Ullah, Z.; CC Cheung, R. Zi-CAM: A power and resource efficient binary content-addressable memory on FPGAs. *Electronics* **2019**, *8*, 584. [CrossRef]
27. Sainath, T.N.; Parada, C. Convolutional neural networks for small-footprint keyword spotting. In Proceedings of the Sixteenth Annual Conference of the International Speech Communication Association, Dresden, Germany 6–10 September 2015.
28. Wang, E.; Davis, J.J.; Zhao, R.; Ng, H.C.; Niu, X.; Luk, W.; Cheung, P.Y.; Constantinides, G.A. Deep neural network approximation for custom hardware: Where we've been, where we're going. *ACM Comput. Surv.* **2019**, *52*, 1–39. [CrossRef]
29. Mittal, S. A survey of techniques for approximate computing. *ACM Comput. Surv.* **2016**, *48*, 1–33. [CrossRef]
30. Mocerino, L.; Calimera, A. TentacleNet: A pseudo-ensemble template for accurate binary convolutional neural networks. In Proceedings of the 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Genoa, Italy, 31 August–2 September 2020; pp. 261–265.
31. Choi, Y.; Choi, J.; El-Khamy, M.; Lee, J. Data-free network quantization with adversarial knowledge distillation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Seattle, WA, USA, 14–19 June 2020; pp. 710–711.
32. Bhardwaj, K.; Suda, N.; Marculescu, R. Dream distillation: A data-independent model compression framework. *arXiv* **2019**, arXiv:1905.07072.
33. Sun, X.; Choi, J.; Chen, C.Y.; Wang, N.; Venkataramani, S.; Srinivasan, V.V.; Cui, X.; Zhang, W.; Gopalakrishnan, K. Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 4900–4909.
34. Johnson, J. Rethinking floating point for deep learning. *arXiv* **2018**, arXiv:1811.01721.
35. Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; Cheng, J. Quantized convolutional neural networks for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 25 June–1 July 2016; pp. 4820–4828.
36. Lai, L.; Suda, N.; Chandra, V. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv* **2018**, arXiv:1801.06601.
37. Lin, D.D.; Talathi, S.S. Overcoming challenges in fixed point training of deep convolutional networks. *arXiv* **2016**, arXiv:1607.02241.
38. Ma, D.; Yin, X.; Niemier, M.; Hu, X.S.; Jiao, X. AxR-NN: Approximate Computation Reuse for Energy-Efficient Convolutional Neural Networks. In Proceedings of the 2020 on Great Lakes Symposium on VLSI, Virtual Event China, 7–9 September 2020; pp. 363–368.
39. Jiang, H.; Liu, L.; Lombardi, F.; Han, J. Approximate arithmetic circuits: Design and evaluation. In *Approximate Circuits*; Springer: Cham, Switzerland, 2019; pp. 67–98.
40. Zhang, Q.; Wang, T.; Tian, Y.; Yuan, F.; Xu, Q. ApproxANN: An approximate computing framework for artificial neural network. In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2015; pp. 701–706.
41. Yang, X.; Pu, J.; Rister, B.B.; Bhagdikar, N.; Richardson, S.; Kvatinsky, S.; Ragan-Kelley, J.; Pedram, A.; Horowitz, M. A systematic approach to blocking convolutional neural networks. *arXiv* **2016**, arXiv:1606.04209.
42. Chen, Y.H.; Yang, T.J.; Emer, J.; Sze, V. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2019**, *9*, 292–308. [CrossRef]
43. Imani, M.; Peroni, D.; Kim, Y.; Rahimi, A.; Simunic, T. Efficient neural network acceleration on GPGPU using content addressable memory. In Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 1026–1031.
44. Yin, X.; Niemier, M.; Hu, X.S. Design and benchmarking of ferroelectric FET based TCAM. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 1444–1449.
45. Jiao, X.; Akhlaghi, V.; Jiang, Y.; Gupta, R.K. Energy-efficient neural networks using approximate computation reuse. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1223–1228.
46. Blalock, D.; Guttag, J. Multiplying Matrices Without Multiplying. *arXiv* **2021**, arXiv:2106.10860.
47. Jenks, G.F. The data model concept in statistical mapping. In *International Yearbook of Cartography*; George Phillip and Son: London, UK, 1967; Volume 7, pp. 186–190.
48. LeCun, Y.; Cortes, C.; Burges, C. *MNIST Handwritten Digit Database*; AT&T Labs: Bedminster, NJ, USA, 2010; Volume 2.
49. Stallkamp, J.; Schlipsing, M.; Salmen, J.; Igel, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Netw.* **2012**, *32*, 323–332. [CrossRef]
50. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]

51. Google Speech Commands PyTorch. Available online: https://github.com/adiyoss/GCommandsPytorch (accessed on 28 August 2021).
52. De Dinechin, F.; Pasca, B. Designing custom arithmetic data paths with FloPoCo. *IEEE Des. Test Comput.* **2011**, *28*, 18–27. [CrossRef]
53. Gupta, N.; Makosiej, A.; Vladimirescu, A.; Amara, A.; Anghel, C. 1.56 GHz/0.9 V energy-efficient reconfigurable CAM/SRAM using 6T-CMOS bitcell. In Proceedings of the ESSCIRC 2017—43rd IEEE European Solid State Circuits Conference, Leuven, Belgium, 11–14 September 2017; pp. 316–319.
54. Balasubramonian, R.; Kahng, A.B.; Muralimanohar, N.; Shafiee, A.; Srinivas, V. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. In Proceedings of the ACM Transactions on Architecture and Code Optimization (TACO), New York, NY, USA, 2 July 2017; Volume 14, pp. 1–25.
55. Daoudi, S.; Anouar Zouaoui, C.M.; El-Mezouar, M.C.; Taleb, N. Parallelization of the K-Means++ Clustering Algorithm. *Ingénierie Syst. d'Inform.* **2021**, *26*, 59–66. [CrossRef]
56. Shahrezaei, M.H.; Tavoli, R. Parallelization of Kmeans++ using CUDA. *arXiv* **2019**, arXiv:1908.02136.