Energy-Efficient Convolutional Neural Networks via Recurrent Data Reuse

Luca Mocerino, Valerio Tenace, Andrea Calimera
Dipartimento di Automatica e Informatica
Politecnico di Torino, 10129 Torino, Italy
andrea.calimera@polito.it

Abstract—**Deep learning (DL) algorithms have substantially improved in terms of accuracy and efficiency. Convolutional Neural Networks (CNNs) are now able to outperform traditional algorithms in computer vision tasks such as object classification, detection, recognition, and image segmentation. They represent an attractive solution for many embedded applications which may take advantage from machine-learning at the edge. Needless to say, the key to success lies under the availability of efficient hardware implementations which meet the stringent design constraints.**

Inspired by the way human brains process information, this paper presents a method that improves the processing efficiency of CNNs leveraging their repetitiveness. More specifically, we introduce (i) a clustering methodology that maximizes weights/activation reuse, and (ii) the design of a heterogeneous processing element which integrates a Floating-Point Unit (FPU) with an associative memory that manages recurrent patterns. The experimental analysis reveals that the proposed method achieves substantial energy savings with low accuracy loss, thus providing a practical design option that might find application in the growing segment of edge-computing.

I. Introduction

The recent surge in artificial intelligence, deep learning [1] in particular, is due to the exceptional growth of performance achieved by Convolutional Neural Networks (CNNs). Inspired by the mechanisms that regulate the *primary visual cortex* of the brain [2], such complex computational models are able to match, and exceed, human performance at classifying objects [1]. With those achievements, CNNs are becoming the *de facto* standard not only in terms of visual understanding but also in a wide variety of domains that embrace medical diagnosis tools [3], personal assistants [4], and self-driving vehicles [5], to name a few.

Deep learning, and CNNs in particular, is the *leitmotif* that ignited the Internet-of-Things (IoT). With the objective of empowering human-environment interactions, the backbone of this pervasive-computing paradigm is the *sense-making* process: physical data sampled by ubiquitous sensors are transmitted, stored, and elaborated in the cloud where deep learning algorithms are used to extract relevant features of the environment where those smart objects interact. Although effective, this centralized implementation shows poor efficiency. The valuable information distilled from raw data (507 ZB/year in 2020 as predicted in [6]) is a mere 23% [7]. This means that 77% of IoT resources are wasted to transmit and store useless data. Moreover, since data are processed remotely by data centers, real-time applications may suffer delays. A smarter option to perpetrate a sustainable IoT is to push some computational tasks at the edge, that is embedding CNN mechanisms into the end-points.

This shift hides serious caveats. Firstly, CNN models require huge amounts of memory footprint that are usually not available in embedded devices. Secondly, evaluating a CNN model implies a massive amount of floating-point operations. Previous works demonstrated that quantization via fixed-point scaling [8] can retain accuracy even at lower arithmetic precision. This helps to reduce the memory footprint, but the actual workload is unaffected because: (i) the number of multiply&accumulate (MAC) operations is almost the same, and (ii) working with fixed-point representations might be much more complex than floating-point. This motivates recent works that introduce alternative data representations, e.g. half-precision floating-point [9], or high-level dynamic management schemes [10].

Another interesting alternative comes from the brain: to barter classical MAC arithmetic with a different memory-centric computing strategy. It is known from cognitive sciences that human brain can be regarded as a complex *association machine* [11] where different concepts are recalled by means of combined key information [12]. Inspired by recent works that employ look-up tables to process neural network models, e.g. [13], [14], we embrace the idea of using recurrent data patterns for accelerating the computational workload. More specifically we investigate the benefits of integrating an associative memory within a floating-point processing unit. This allows relevant information, i.e., the most significant intermediate results, to be retrieved by means of a combined key between the two operands, thus skipping the pure arithmetic.

The aim of this work is to devise an efficient embodiment of this computing strategy. We introduce a clustering method that maximizes the reuse property of a neural network, together with the CMOS design of a CAM-enhanced floating-point unit that leverages such property. The methodology has been validated for different applications, i.e., image classification and speech recognition, using three CNNs trained with a dedicated dataset. Experimental results show a 50% energy saving with low accuracy loss ($< 3\%$). When compared to relevant previous works, i.e., [15], the energy saving is substantially improved: 21.56% with $< 1\%$ of accuracy loss.

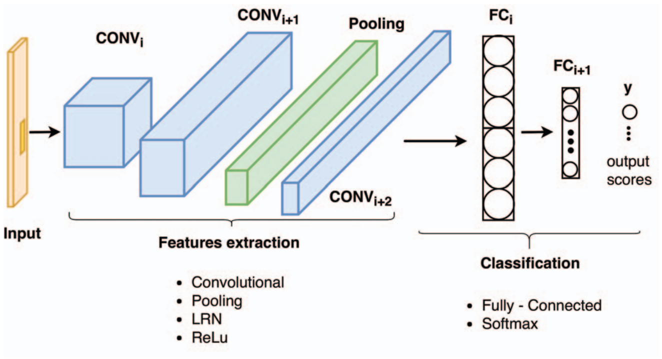

Figure 1: A typical CNN model.

II. Background

CNNs are a special class of deep learning models mostly suited for two-dimensional inputs. Figure 1 depicts the typical internal structure. It consists of several layers, each with a proper function: *input layers* that handle images for computational stages, *output layers* that are in charge of producing the final answer of the classification task, and several *hidden layers* that extract relevant features learned during the training stage. The feature extraction region is made up of layers of different kinds: *convolutional* (CONV) layers that perform multidimensional convolutions, *pooling* layers which reduce the dimension of feature maps, and *dropout* layers that help to mitigate over-fitting. The classification region implements a geometric separation of the extracted features; it typically consists of a *fully-connected* (FC) neural network that computes scores on each feature map.

Among these layers, CONV and FC are the most computational and memory expensive. CONV layers perform several convolutions between multidimensional filters (or kernels) and input feature maps, i.e., the pixel of the image (in the case of the input layer) or the obtained convoluted images (for internal layers). The main parameters of a CONV layer are:

- **NF**: Number of filters and output feature map channels;
- **C**: Number of input feature map and filter channels;
- **FH/FW**: Filters height/width;
- **K**: Input feature map height/width;
- **OH/OW**: Output feature map height/width;
- **S**: Stride;
- **P**: Padding.

The convolution operation can be formally defined as in (1), where $0 \leq u \leq NF$, $0 \leq y \leq OH$ (with $OH = (FH - K + 2P)/S + 1$) and $0 \leq x \leq OW$ (with $OW = (FW - K + 2P)/S + 1$).

$$\mathbf{f\_map}[u][x][y] = \sum_{z=0}^{C-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} I[z][S_{x+i}][S_{y+j}] \cdot W[u][z][i][j] \quad (1)$$

A FC layer is a deep neural network per sé. It integrates a number of layers (L), each of them with an arbitrary number of neurons. Each layer ($l \in L$) has several activations (a_l) multiplied for weights matrix (W_l) with an additional bias (b_l). The output of a generic layer l can be written as in (2).

$$y_l = W_l \cdot a_{l-1} + b_l \quad (2)$$

Both CONV and FC layers are implemented using Multiply&Accumulate units.

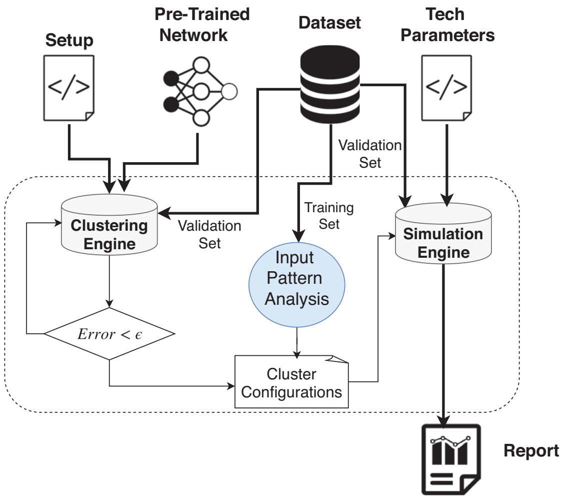

Figure 2: Design flow overview

III. Proposed Methodology

The main idea is to store pre-computed multiplications results and to reuse them whenever the same input pattern occurs, hence avoiding useless computations of frequent operations. To take benefit from this strategy, we propose an optimization framework that aims at improving the computational reuse by means of a non-linear clustering of the weight distribution. A discretized CNN model increases the probability of recurrent paths indeed, thus enabling an efficient use of memory-based computing. To leverage this mechanism we also introduce the design of a hardware processing element (PE) that exploits the joint co-operation of an associative memory and a standard Floating-Point Unit (FPU).

A. Co-design Framework

Energy efficiency can be achieved by increasing the repetitiveness in the workload during the feed-forward pass of the CNN. The rationale is to maximize the frequency of weights and activations. This task is implemented by the optimization tool-chain illustrated in Figure 2. The main inputs are (i) the pre-trained CNN, (ii) the dataset, (iii) a configuration file with the technological parameters of the PE, and (iv) the user-defined accuracy loss threshold (ϵ). The tool produces the most energy-efficient associative memory configuration that obeys to the constraints and the resulting CNN model.

The input CNN model, the dataset, and the setup file are used by the weight clustering engine. Its main goal is to reduce the number of different parameters in the network by merging similar weights according to their magnitude. Once the centroids are found, all the weights that belong to a certain cluster are mapped on centroid values. This procedure allows to reduce the CNN model complexity and to find the most representative weight values of the entire CNNs (represented by cluster centroids). The centroids represent the weight patterns that need to be stored, whereas activations are processed with a different procedure.

The adopted clustering relies on the *Jenks Natural Breaks* (JNB) algorithm [16]. This is an iterative method whose objective is to maximize the inter classes variance while minimizing the intra-class variance. The choice of using a JNB algorithm comes from the experimental evaluation of other two algorithms (*k*-means and mean-shift clustering) on different CNN models (Table I) using the *goodness of variance fit* (GVF) as a metric.

We opted for a different clustering procedure for the CONV layers and the FC layers. Given a CONV layer of dimension NF $\times$ C $\times$ FH $\times$ FW , N_{cnv} represents the number of clusters per filters of size C $\times$ FH $\times$ FW. Given a FC layer of size W $\times$ H, the N_{fc} represents the number of clusters for the whole matrix of dimensions W $\times$ H. On the base of empirical observations, the use of different clustering granularity for each layer allows to reach the same accuracy with a reduced number of centroids.

Figure 3 shows the effect of the adopted clustering on the weights of a sample FC layer. The proposed approach enhances the data reuse opportunity for the following reasons: (i) a non-linear quantization allows to represent the original weight distribution with a limited set of values; (ii) the separation between most occurring values from less occurring ones gets improved. This emerges from Figure 3, where the frequency, i.e., the number of repetitions, of the same value increases after clustering.

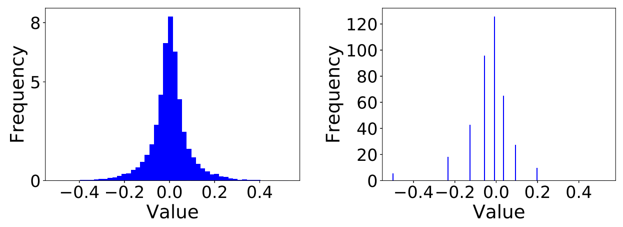

Figure 3: Clustering. Original weights distribution (left), weight distribution after clustering (right).

The tool iteratively selects an appropriate set (N_{cnv}, N_{fc}) such that the resulting accuracy drop, i.e., the *Error*, meets the constraint ϵ.

For what concerns the activations, we use as a reference the information stored in the training set. Partial results collected during the feed-forward pass of the CNN are collected in order to profile the most frequent input activation values (N_{in}) and the intermediate activation values.

Once completed, the clustering stage provides a set of configurations (N_{cnv}, N_{fc}, N_{in}) and the corresponding accuracy degradation. The contribution of each of those parameters on energy saving and accuracy drop is summarized as follows:

- N_w defined as *max*(N_{cnv}, N_{fc}) gives the weight-CAM size and the overall accuracy;
- N_{in} defines the activation-CAM and the overall hit rate in the associative memory;
- Both N_w and N_{in} contribute to the Static Random Access Memory (SRAM) size for collecting the multiplication results: $M_{rows} = N_w \times N_{in}$.

Different configurations explore different memory sizes that adhere to the user-defined constrain. For instance, we can have solutions with the same accuracy degradation (same N_w), but different associative memory sizes (varying N_{in}), hence higher or lower energy savings.

The available configurations are evaluated by emulating the inference stage on the customized hardware. This stage encompasses the simulation of the CAM-enhanced FPU. The associative memory is initialized with the patterns provided by the clustering engine. Using the hit rate and the technological characterization of the PE, the framework provides an estimation of the energy consumption. As a final outcome, the tool returns the best associative memory configuration that satisfies the user's constraints with minimum energy consumption. This allows the user to pick the best configuration according to the requirements (application and hardware platform) by trading accuracy with energy efficiency. For instance, a face recognition system used to unlock the smartphone may put energy efficiency first and tolerate larger accuracy loss.

B. Hardware Design

An overview of the PE structure is provided in Figure 4. It consists of an FPU and an associative memory filled with the centroids returned by the clustering engine. From a functional point of view, the PE works as follows: the input operands are processed in parallel by the CAMs; if they match the ones already stored, the floating point multiplier (FPMul) is clock gated and the multiplication result is retrieved from the associative memory, otherwise it is normally computed. The integration of the FPU and the CAM memory follows a pipelined scheme that improves the power consumption: in most of the cases the FPU is indeed clock-gated.

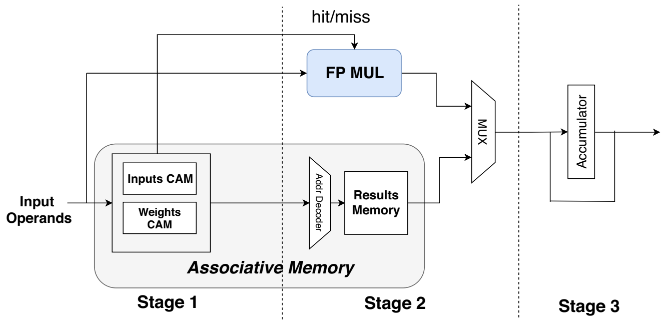

Figure 4: Processing Element (PE)

The FPU is a multi-stage single-precision floating-point multiplier and single-stage accumulator. The architecture of the associative memory is depicted in Figure 5. It consists of two CAMs and an SRAM: the inputs CAM is dedicated to the

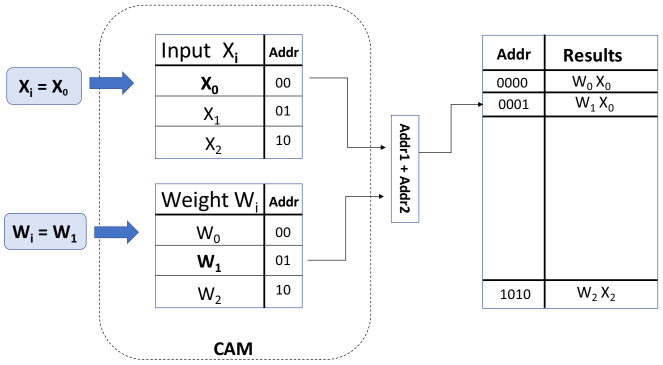

Figure 5: Associative Memory functional overview

most frequent activations, the weight CAM collects the cluster centroids, and the SRAM stores the results.

The proposed memory architecture is similar to the one proposed in [14], but with enhanced performance. Indeed, compared to a single CAM solution with a double word bit-width (weight and input concatenation) and with $\#rows = N_w * N_{in}$, our approach shows less area and better savings.

The associative memory can be designed with a custom solution using the memory configuration obtained from the co-design framework, which is tailored to the specific application. Moreover, a flexible solution can exploit a modular architecture composed of multi-stage CAMs with a base module that replicates itself in cascading stages. By disabling a certain number of base modules, it is possible to achieve the required size. For simplicity, we emulate the associative memory designed with the custom approach.

C. Energy Model

The overall energy consumption for a look-up in the associative memory can be estimated using (3), 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 contribution given by the patterns memory of size M_{rows}.

$$E_{lookup} = E_w(N_w) + E_{in}(N_{in}) + E_m(N_m) \qquad (3)$$

The energy model integrated into the simulation framework is reported in (4), where hr represents the hit rate, E_{mul} is energy of the multiplier, E_w and E_{in} refer to the energy consumption of the two CAMs described by (3), E_{hit} is the energy consumed when inputs do match 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 in that case the multiplication is actually computed).

$$\begin{aligned} E_{hit} &= hr * E_{lookup} \\ E_{miss} &= (1 - hr) * (E_{mul} + E_w + E_{in}) \qquad (4) \\ E_{tot} &= E_{miss} + E_{hit} \end{aligned}$$

The energy saving can be trivially computed comparing the energy calculated as in (4) to that consumed by the FPMul alone.

IV. Experimental results

A. Experimental Setup

The clustering analysis and the simulation tool are built upon the deep-learning libraries provided by PyTorch v0.3 [17]. We evaluated the proposed methodology for three different applications:

- **MNIST** [18] is a database of handwritten digits. It is composed of 60k samples for training and 10k for the test, organized in 10 classes.
- **German Traffic Sign Recognition (GTSRB)** [19] consists of 50k images of street signals used for a multi-class, single-image classification challenge with a total of 43 classes to recognize.
- **Google Speech Commands (GSC)** [20] has 65k one-second long expressions of 30 words made by one thousand different people. The challenge is to recognize 30 different words pronounced.

We trained a dedicated CNN network for each dataset. As proposed in [15], we adopted a LeNet-like CNN for both MNIST and GTSERB, while we deployed a custom network for GSC.

Table I: Baseline Parameters

Network	**Dataset**	**Top-1** (%)
LeNet-like [15]	MNIST	98.8
LeNet-like [21]	GTSRB	87,13
CONV: 20x5x5 - 20x5x5 - FC: 16280x1000 - 1000x30	GSC	69.41

Table I gives an overview of the adopted CNN benchmarks and their top-1 accuracy. Concerning the third CNN (GSC dataset), the table just reports the CONV and FC layers together with their size, although they are interleaved with pooling and dropout layers.

The PE design consists of an associative memory and a single-precision floating-point multiplier from FloPoCo [22]. Energy estimations have been obtained with Synopsys Design Compiler leveraging a commercial CMOS 28nm FDSOI technology library from STMicroelectronics. CAMs are based on a standard 6T cell, and mapped using the same 28nm technology library. The characterization was carried out by means of HSPICE simulations. Results show that the adopted solution shows an energy efficiency comparable to state-of-the-art CAM memories [23]. The characterization of the small-size SRAM-based memory was retrieved from CACTI 6.0 [24]. Figure 6 shows a parametric analysis for the normalized energy consumption of all the three memory components.

B. Clustering Results

Figure 7 quantifies the effect of clustering. As expected, the accuracy drop reduces for lager number of clusters (N_{cnv}, N_{fc}). It appears that the LeNet-like model used on MNIST shows close-to-zero degradation when $N_{cnv} = N_{fc} = 64$. Similar trends apply for GTRSB and GSC, yet for different number of clusters. It is a remarkable result if one considers that very

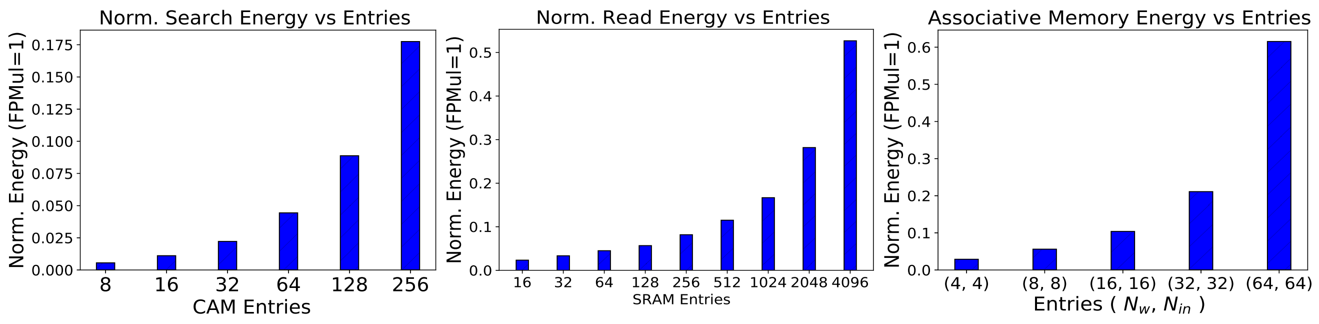

Figure 6: Normalized Energy vs Entries: CAM (a), SRAM (b), Associative Memory (c).

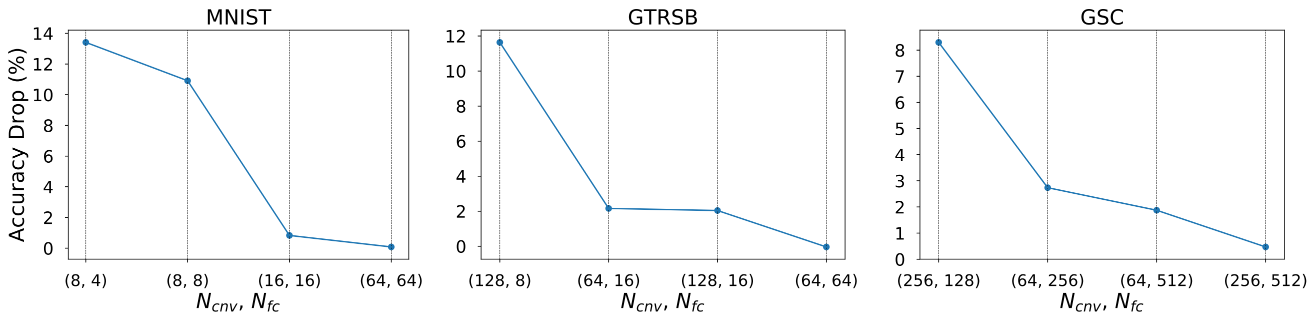

Figure 7: Accuracy drop (%) variation changing the weight clusters configurations (N_{cnv}, N_{fc})

small CAMs can represent the complex set of information stored in a multidimensional CNN model.

What emerges from Figure 8 is that the hit rate increases with N_{in}. For the networks on MNIST and GSC, the hit rate ranges from 70% to 80%: the model used on GSC dataset reaches a hit rate of 80% by using $N_{in} = 64$, whereas that on MNIST gets close to 73%. That implies that just the 20%-30% of the overall workload is actually computed by the FPU. The hit rate of the network on GTSRB swings on a lower and wider range, from 18% to 35%. The GTSRB dataset consists of input activations with a large variance, therefore the number of patterns used for the activations (N_{in}) heavily affects the hit rate. It is worth to notice that a high hit rate does not always mean higher energy savings. Increasing N_{in}, and hence the hit rate, makes the associative memory larger and more power hungry. This motivates the fact that the exploration of different configurations (N_w, N_{in}) is crucial to find the most energy-efficient solution.

While N_{in} affects the hit rate, N_w affects the accuracy; both define the overall energy consumption. This emerges from (3), where both values contribute to the associative memory. The plot in Figure 6-c shows that with $N_w = N_{in} = 64$, the normalized energy for a look-up search is 0.6. To notice that for $N_w = 64$ the accuracy drop is $\cong 0$ for both MNIST and GTSRB. These results suggest that it is possible to optimize the energy of a CNN without paying any accuracy loss.

C. Performance Results

Figure 9 shows the energy savings of the proposed solution under several accuracy constraints for the three benchmarks under analysis.

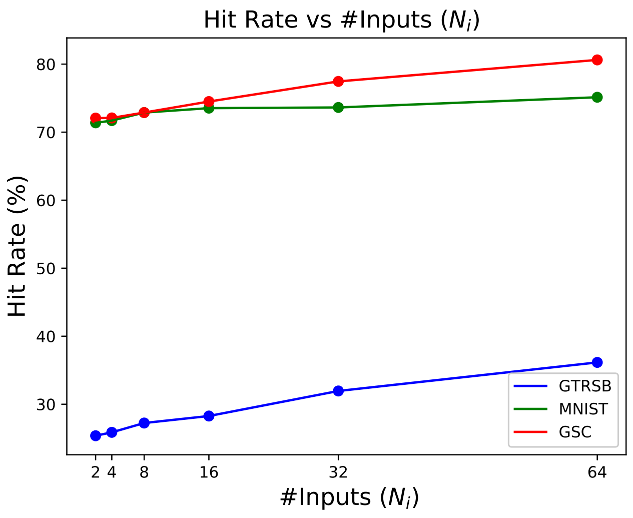

Figure 8: Hit Rate / #inputs (N_i) trends fixing N_{cnv} and N_{fc}

The plot reports the dominant implementations that emerge from the Pareto analysis explored by the co-design tool. As can be seen, the amount of energy savings may change with different CNN models and dataset; those elements indirectly affects the associative memory size, as previously described.

Table II: Comparison with previous work [15].

	Network	Dataset	Energy Savings (%)	Accuracy Drop (%)
Proposed	LeNet-like (Table I)	MNIST	69.06 (**+21.56**)	1
[15]	LeNet-like (Table I)	MNIST	47.5	1

It is also clear from Table II that our approach outperforms [15]: 21.56% energy efficiency improvment.

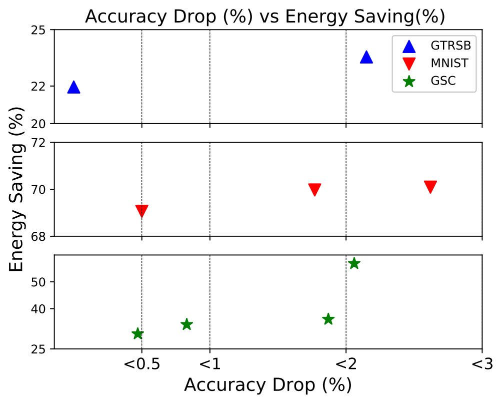

Figure 9: Accuracy Drop vs. Energy Savings tradeoff for the considered dataset. Maximum energy savings for the accuracy drop $< 3\%$: 70% (MNIST), 57% (GSC), 24% (GTSRB).

D. *Related work*

Several approaches exploit recurrent operations due to the spatial locality of the data involved in CONV and FC layer computation. The work in [25] performs voltage over-scaling on a special resistive Ternary Content Addressable Memory (TCAM) integrated within an FPU. That memory performs approximate pattern matching within a specified Hamming distance. Unfortunately, the Hamming distance is not a good metric for floating-point similarity. It does not consider the impact of error position on accuracy drop since the error on the mantissa, or on the exponent, have different contributions. On the other hand, our work exploits typical CMOS memories that have been used for years in standard GPUs or CPUs. Another recent method [15] adopts a Bloom Filter (BF) in resistive technology to store recurrent patterns and to perform approximate pattern matching on the values stored in the Bloom Vector. This approach trades energy savings on multiplications with the accuracy loss. However, this solution presents two main drawbacks: (i) the approximation error and the presence of false positive when retrieving a pattern from a BF heavily impacts the final accuracy; and (ii) BFs built with a resistive technology are hard to integrate into standard architectures and design flows [26].

V. Conclusions

This work proposes a methodology that provides an energy efficient software-hardware solution that: (i) improves the recurrent data reuse opportunities in CNNs via clustering; (ii) integrates an associative memory into standard processing elements in order to exploit pre-computed results. The latter are retrieved from an associative memory when the input patterns match the pre-stored ones, otherwise they are normally computed. Experimental results on different datasets demonstrate that our solution can reach up to 50% of energy saving with a $< 3\%$ accuracy degradation. Our solution outperforms previous works improving the energy saving by 21.56% with a negligible $< 1\%$ performance degradation.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.
[2] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture of monkey striate cortex,” *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.
[3] G. Litjens *et al.*, “A survey on deep learning in medical image analysis,” *Medical image analysis*, vol. 42, pp. 60–88, 2017.
[4] F.-L. Li *et al.*, “Alime assist: An intelligent assistant for creating an innovative e-commerce experience,” in *CIKM’17: ACM Conference on Information and Knowledge Management*, pp. 2495–2498.
[5] M. Bojarski *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.
[6] D. Evans, “The internet of things: How the next evolution of the internet is changing everything,” *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.
[7] T. Tran, “Evolution of machine learning: from expert to exploratory systems,” 2016.
[8] I. Hubara *et al.*, “Quantized neural networks: Training neural networks with low precision weights and activations,” *arXiv preprint arXiv:1609.07061*, 2016.
[9] L. Lai, N. Suda, and V. Chandra, “Deep convolutional neural network inference with floating-point weights and fixed-point activations,” *arXiv preprint arXiv:1703.03073*, 2017.
[10] V. Peluso and A. Calimera, “Scalable-effort convnets for multilevel classification,” in *ICCAD’18: International Conference on Computer-Aided Design*. ACM, 2018, pp. 12:1–12:8.
[11] V. G. Ivancevic and T. T. Ivancevic, *Neuro-fuzzy associative machinery for comprehensive brain and cognition modelling*. Springer, 2007, vol. 45.
[12] W. A. Suzuki, “Associative learning signals in the brain,” *Progress in brain research*, vol. 169, pp. 305–320, 2008.
[13] W. Chen *et al.*, “Compressing neural networks with the hashing trick,” in *ICML’15: International Conference on Machine Learning*, 2015, pp. 2285–2294.
[14] M. S. Razlighi *et al.*, “Looknn: Neural network with no multiplication,” *DATE’17: Design, Automation & Test in Europe Conference*, pp. 1779–1784, 2017.
[15] X. Jiao *et al.*, “Energy-efficient neural networks using approximate computation reuse,” *DATE’18: Design, Automation & Test in Europe Conference*, pp. 1223–1228, 2018.
[16] G. F. Jenks, “The data model concept in statistical mapping,” *International yearbook of cartography*, vol. 7, pp. 186–190, 1967.
[17] A. Paszke *et al.*, “Automatic differentiation in pytorch,” 2017.
[18] Y. LeCun, C. Cortes, and C. Burges, “Mnist handwritten digit database,” *AT&T Labs*, vol. 2, 2010.
[19] J. Stallkamp *et al.*, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural networks*, vol. 32, pp. 323–332, 2012.
[20] T. N. Sainath and C. Parada, “Convolutional neural networks for small-footprint keyword spotting,” in *INTERSPEECH’15: Conference of the International Speech Communication Association*, 2015.
[21] Y. LeCun *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[22] F. De Dinechin and B. Pasca, “Designing custom arithmetic data paths with flopoco,” *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.
[23] N. Gupta *et al.*, “1.56 ghz/0.9 v energy-efficient reconfigurable cam/sram using 6t-cmos bitcell,” in *ESSCIRC’17: IEEE European Solid State Circuits Conference*, 2017, pp. 316–319.
[24] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “Cacti 6.0: A tool to understand large caches,” *University of Utah and Hewlett Packard Laboratories, Tech. Rep*, 2009.
[25] M. Imani *et al.*, “Efficient neural network acceleration on gpgpu using content addressable memory,” *DATE’17: Design, Automation & Test in Europe Conference*, pp. 1026–1031, 2017.
[26] S. M. Jalaleddine, “Associative memories and processors: The exact match paradigm,” *Journal of King Saud University-Computer and Information Sciences*, vol. 11, pp. 45–67, 1999.