Algorithmic enablers for compact neural network topology hardware design: review and trends

William Guicquero, Arnaud Verdant

Algorithmic enablers for compact Neural Network topology Hardware design: review and trends

William Guicquero, Arnaud Verdant
CEA-LETI, F-38000 Grenoble, France
william.guicquero@cea.fr, arnaud.verdant@cea.fr

***Abstract*—This paper reports the main State-Of-The-Art algorithmic enablers for compact Neural Network topology design, while relying on basic numerical experiments. Embedding in-sensor intelligence to perform inference tasks generally requires a proper definition of a Neural Network architecture dedicated to specific purposes under Hardware limitations. Hardware design constraints known as power consumption, silicon surface, latency and maximum clock frequency cap available resources related to the topology, i.e., memory capacity and algorithmic complexity. We propose to categorize into 4 types the algorithmic enablers that force the hardware constraints as low as possible while keeping the accuracy as high as possible. First, Dimensionality Reduction (DR) is used to reduce memory needs thanks to predefined, hardware-coded patterns. Secondly, low-precision Quantization with Normalization (QN) can both simplify hardware components as well as limiting overall data storage. Thirdly, Connectivity Pruning (CP) involves an improvement against over-fitting while limiting needless computations. Finally, during the inference at the feed-forward pass, a Dynamical Selective Execution (DSE) of topology parts can be performed to limit the activation of the entire topology, therefore reducing the overall power consumption.**

***Index Terms*—Neural Network, Compressive Sensing, Random Pruning, Quantized Neural Network, Dynamic Neural Network, Hardware-Algorithm co-design.**

I. Introduction

The recent wide popularity of AI is mainly due to astonishing results of high-end algorithms. For instance, image classification based on Deep Neural Networks (DNN) ran on GPUs can reach close or even beyond human-level efficiency in terms of accuracy and speed [1] [2]. If appropriately designed, it can take advantage of massively parallel computing [3] and/or being optimized thanks to a specific high-level hardware mapping [4] [5]. On the other hand, a current trend from GPU providers is to push forward ML-dedicated hardware platforms [6], promoting the deployment of AI on the edge, for embedded systems. Indeed, low-power computing nodes ($\sim W$) now tend to be relevant for IoT targeted applications.

However, those cutting-edge systems can not be involved for ultra-low power systems ($\ll mW$), in particular for always-on functioning modes. In that context, various types of System-on-Chips have been proposed in the literature, namely for voice detectors [7], smart RF nodes [8] and image sensors providing wake-up triggers [9]. Those prior works involve near-sensor or in-sensor decision making. Note that one major source of leverage to design efficient in-sensor processing is to take carefully into account the raw data format provided by embedded transducer nodes (i.e., mems microphones, RF front-ends, pixels, ...). Up to a certain extent, it can also motivate to revisit how transducers work. For instance, Dynamic Vision Sensors (DVS) [10] are claimed to tackle data compression issues [11] while improving processing performances by involving Spike Neural Networks (SNN) [12], benefiting from intrinsic DVS-type scene feature extraction. Even if not providing relevant proofs of its competitiveness for the moment, it still opens novel research directions.

Apart from those 'unconventional' approaches, linear classifiers working on canonical data are yet efficient for numerous basic applications. As with linear DR, it only involves linear operations thus allowing an easy hardware implementation. However, it exhibits limited performances for advanced inference tasks, especially when data are not linearly describable. Various workarounds have been developed to address this non-linear manifold issue. A very popular one is the Kernel Trick, enabling the introduction of a non-linear function in most of linear frameworks. However, nowadays, a common technique to involve non-linearity –with the least assumptions on data structurality– is to use a Neural Network (NN). Mathematically speaking, a formal definition of NN is a composition of multiple linear and non-linear functions. We can distinguish two classes of linear functions (matrix-to-vector projections, convolutions) as well as two classes of non-linear operator types (activation units, pooling routines). The use of convolution layers (e.g., CNN) is motivated by a reduction of the required memory combined with a better data local structure analysis and a reduction of over-fitting issues. In the case of Neural Networks, the non-linearity is basically introduced by neuron activations which generally correspond to one-dimensional functions. Otherwise pooling functions can also benefit from non-linear operations as in the case of max-pooling that pools data as its local maximum. Depending on the definition of the output loss function, NNs show the advantage of being versatile and being compatible with all three main categories of machine learning tasks (classification, regression and clustering) independently in supervised, non-supervised and semi-supervised learning contexts.

II. Algorithmic enablers for NN-based embedded machine learning inference tasks

While most of Hardware accelerators dedicated to machine learning are for generic tasks, ultra-low power systems with embedded inference are generally designed to address very specific ones. Highly-dedicated NN topology can thus be

designed to manage the issues related to resource-limited hardware, in terms of silicon surface and power consumption.

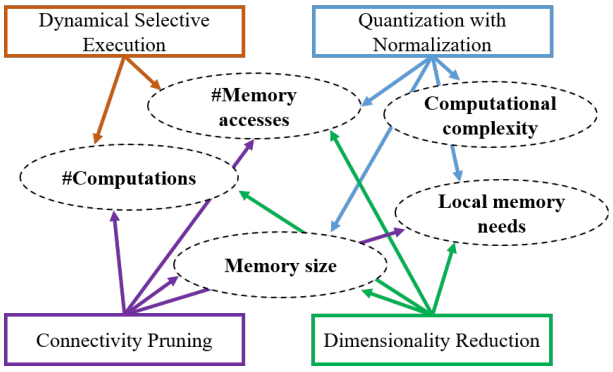

Fig. 1. Enablers for compact Neural Network topology design.

In this dual scope, the main goal when dealing with Neural Network topology design is thus to find the best operating point to reach the trade-off between accuracy of the inference and the overall algorithmic complexity, while limiting the effects of outliers and preventing against over-fitting. In other words, it aims at finding the most compact topology while preserving the best performances in terms of accuracy. To this end, as depicted by Fig. 1, we identified 4 algorithmic enablers to limit processing load of an NN feed-forward pass:

- Dimensionality Reduction *(layer width)*
- Quantization with Normalization *(data dynamic range)*
- Connectivity Pruning *(sparsity of the projections)*
- Dynamical Selective Execution *(topology activations)*

Those levers act on various types of hardware-related limitations as **#Computations** that refers to the overall amount of computational workload of the topology, **#Memory accesses** that is the total number of accesses needed during the forward pass execution, **Memory size** which is related to the overall NN weight storage needs, **Local memory needs** (e.g., cache memory) being induced by the number of nodes in the topology and finally what we call **Computational complexity** which is the complexity related to the node core-computations, induced by both linear and non-linear operations.

A. Dimensionality Reduction (DR)

Dimensionality Reduction corresponds to the process of reducing the size of the data while preserving its information content for a certain targeted use. Typically, DR techniques are massively employed for dataset inspection and easing large data processing by reducing its dimensionality. In this paper, Dimensionality Reduction largely refers to as feature extraction. This operation can be performed via linear or nonlinear, structured or unstructured, data-aware or data-agnostic processing. In the context of hardware-limited Neural Network topology design, this feature extraction stage should not replace a standard projection layer if it is not related to a significant hardware constraint relaxation. Indeed, the use of DR is relevant if performed by optimized hardware structures and thus not being issued from a learning stage, as trained weights. One of the most well-known is ”pooling” that can be implemented without complex hardware. Structured DR could be performed using mathematical transforms too (i.e., Wavelet, Fourier...), combined with subsampling done according to data structure assumptions. However, those approaches still require specific hardware to store or generate all transform coefficients and to perform a high number of MACs.

Introduced in early 2010’, Compressive Sensing (CS) [13] aims at providing techniques to compress signals while enabling its reconstruction via optimization algorithms [14], this, under signal sparsity and sensing scheme versus sparsity representation bases incoherence priors. Note that, recent advances on CS reconstruction allows bypassing iterative algorithms by using learned NNs [15]. Signal processing in the compressed domain already presented in [16] allows to leverage the prohibitive cost of recovery. Indeed, the Restricted Isometry Property guarantees a preservation of Euclidean distances in the CS domain enabling the use of traditional Machine Learning algorithms. In that context, [17] investigates inference performance limits with a universal compressive measurement scheme. At the expense of relatively large Neural Networks, [18], [19] and [20] propose to perform the inference directly on compressed data, yet involving at least a layer with the same size as the original image. To this end, they either use a sensing matrix transpose projection or a learned fully connected layer. In addition to classification applications, CS also seems a good candidate to alleviate algorithm complexity related to regression problems, as reported in [21] that deals with spectrum occupancy sensing. The same applies for clustering as presented in [22] where a variant of Compressive K-Means (CKM) works with a 1-bit quantization. More recently, even the famous Visual Question Answering (VQA) problem has been efficiently addressed in [23], using CS measurements. Finally, [24] depicts an image sensor architecture that directly performs CS and inference in the focal plane benefiting from DR to optimize overall digital hardware resources. As a direct extension of these prior works, integrating a CS layer into a CNN topology is expected to be useful in order to benefit from both relevant first convolutional layers (learned kernel filters) with a reduced size of the fully connected latest layers. Besides, we want to stress that another way to take advantage of CS has been developed in [25] where an already learned topology is being compressed via CS, making assumptions on the nature of the NN learned patterns. Apart from the investigations on compressively sensed data for direct inference, Extreme Learning Machines (ELM) somehow works in an equivalent fashion. Indeed, a basic ELM corresponds to a single-hidden layer NN whose weights of the hidden layer are not tuned [26], and randomly generated. In the direction of CS, [27] proposes to prune useless neurons of the ELM learned layer, therefore intrinsically performing DR. [28] [29] present surveys of direct generalizations of ELMs, namely the concatenation of basic ELMs composed of multiple layers with learned weights. In addition, [30] revisits MLP-ELM topology to replace the greedy deep learning strategy by unsupervised ELM feature learning, being performed in a forward manner.

B. Connectivity Pruning (CP)

As for DR, Connectivity Pruning (CP) can be either learned or not, structured or not. Learned pruning has been deeply investigated, mainly because of its interests in terms of memory and its negligible impact on inference accuracy. To this end, [31] presents a common 3-stage pruning strategy, learn, select-prune and relearn. This "oversized-to-adequate" connectivity learning approach has the advantage of being simple and generic, typically enabling a reduction by around a factor of 10 on the number of weights. On the contrary, [32] presents a "small-to-adequate" topology learning approach they call Continuous Growth and Pruning (CGaP). Indeed, they claim that instead of starting from an over-parametrized NN, it is more efficient to iteratively enlarge it until reaching an asymptotic performance. On the other hand, [33] proposes efficient representations for NN projection matrices thanks to low-entropy statistics while replacing standard matrix sparse representation for improved hardware mapping. This approach, yet based on a specific coefficient coding and feed-forward execution architecture, seems to enable up to $\times 90$ energy savings without major accuracy loss. In the specific CNN case, various pruning strategies also exist. We can mention [34] whose goal is to cap the inter-band connectivity density between consecutive conv. layers using their so-called group-wise pruning to exhibit structured connectivity. Otherwise, still in the scope of CNN, [35] presents a technique to directly reduce the kernel filters redundancy using filter pruning. However, those previous works do not tend to reduce complexity once and for all from the hardware point of view, because of requiring specific learning and being dedicated to complex inference problems aiming at being ran on a generic hardware.

The goal of [36] is yet to propose a hardware-aware pruning in order to efficiently perform connectivity reduction with a relevant compact hardware mapping. It compares sparse patterns resulting from different weights pruning strategies, evaluating how crossbar patterns enable dedicated hardware designs with improved performances. Apart from structured and learned pruning, as for the randomly based approach of CS, [37] shows that random pruning strategy performs at par with principled pruning strategies. Even if this statement cannot be generalized for any inference problem, it still opens a new path in terms of hardware design to optimize such random connectivity as it has been shown relevant in the case of CS. Finally, other works such as [38] do not only optimize structured sparsity of weights but also their quantization by limiting each weights to be a ternary value (i.e., $\{-1, 0, +1\}$).

C. Quantization with Normalization (QN)

[39], [40] and [41] recently show that training quantized NN (quantized weights and activations) can be efficiently performed, without a large degradation of performances, thanks to a dedicated training namely with proper batch normalizations. One of the main trick of quantized learning is that a non-quantized version of the coefficients is still used for the learning stage. They are used for weight updates during back propagation steps while their quantized version are only used for the forward pass. These works present extensive numerical experiments, demonstrating the interest of using 1bit quantization because of enabling ultra-light XOR computations. The state-of-the-art training algorithm presented in [41] further involves specific improvements based on NN regularization functions and scaling parameters which are dedicated to binary weights learning. In order to fully validate this learning strategy, [42] compares a specific training approach involving quantization with the direct use of the same topology whose weights are quantized after a standard training. It demonstrates possible trade-offs in terms of latency-vs-accuracy between floating point and integer-only computations. Typically, [43] studies the effect of the number of bits used to code the weights on the NN performance, showing the asymptotic nature of the NN accuracy for a given number of quantization levels. Furthermore, [44] reviews Binary Neural Networks (BNN), especially pointing out that BNN are often confused with ternary-weighted NN. Both have advantages and drawbacks, ternary weights indeed provide better accuracy by enabling connections zeroing with a high level of compression and enabling simple hardware mapping, but still requiring somehow 2-bit per coefficient instead of a single one.

More generally, non-uniform (i.e., logarithmic) quantization of weights has been investigated in [45]. A part from specific topology training and without requiring full network retraining, [46] formulates the weights&activations quantization tasks as a Minimum Mean Squared Error (MMSE) problem applied to a pretrained NN. In order to further improve competitiveness of BNNs compared to floating-points NNs, [47] proposes an extended version of BNNs using Stochastic Computing. Indeed, since hardware mapping is fixed (i.e., XNOR and pop-count), stochastic computing can take advantage of it without deep hardware design modifications. On the other hand, [48] proposes to use "dithering" by distributing quantization around neighboring layer input nodes.

D. Dynamically Selective Execution (DSE)

[49] gathers various Dynamically Selective Executions (DSE) strategies for optimizing accuracy with respect to efficiency. In particular, they list four topologies: low to high precision, cascaded networks, chained graph and hierarchical decision tree. The underlying idea is to adapt the NN topology according to its application context, lowering the averaged dynamic power consumption. Indeed, in the case of always-on systems, [50] shows that a multi-level wake-up enables to optimize ratios between power and event occurrence.

Apart from the NN-related literature and for a long time, hierarchical inference has been deeply investigated [51]. For instance, [52] presents an iterative decision-making approach which is based on successive projections of CS measurements. [53] presents a framework that enables the integration of both input-dependent and resource-dependent dynamic inference mechanisms applied to high-end CNN backbones. Typically, they analytically demonstrate that FLOPS used for the CIFAR-10 classification problem can be reduced by a factor of more than 3. As for previous sections (DR, CP and QN),

various works start from an already learned topology in order to enable DSE. Namely, [54] aims at partitioning existing topology between the edge device and the cloud at the layer's granularity to relax average power consumption of IoT nodes.

III. Numerical experiments

To illustrate previous statements, we carried out basic numerical experiments on DR, CP and QN. For the sake of simplicity and reproducibility, we considered the MNIST classification problem, with the NN following topology:

layer type	input	output	enabler	figures
sample	-	784	-	-
DR	784	rd	CS	*(a)(d)(e)(f)*
Con. layer	rd	rd	CP, QN	*(b)(c)(d)(e)(f)*
ReLU	rd	rd	QN	*(c)(e)(f)*
Con. layer	rd	32	CP, QN	*(b)(c)(d)(e)(f)*
Softsign	32	32	-	-
Con. layer	32	10	-	-
Softmax	10	10	-	-

We thus have arbitrarily chosen this topology because of its small size. It is learned using a Momentum based Stochastic Gradient Descent combined with the algorithm presented in [40] to deal with QN. Note that activation functions are bypassed if QN involves a quantization <4bit and the same quantization is applied to all layers (which is not optimal). Fig. 2 reports curves of accuracy degradation linked to different setups. DR is simply performed using CS via a Rademacher projection (with a normalized matrix), CP consists in randomly pruned connections (uniformly distributed) and quantization is done with the same resolution on both weights and activations. *(a)*, *(b)* and *(c)* respectively show the effect of DR, CP and QN on the classification accuracy while being used independently. In that specific MNIST case, a DR of 25% compression ratio (i.e., $rd = 196$) with a CP of 50% and a QN of 5bit all represent proper operating points for such a topology (even better than the original one). But, while being combined *(f)*, they can further contribute to limit the overall hardware needs, with only very limited impacts on the output inference results.

IV. Discussion and Conclusion

Recent AI-accelerators and smart sensors already follow this trend. First, Compressive Sensing sensors tend to implement decision making [24] because of its light algorithmic complexity due to intrinsic DR. Secondly, [55] and [56] propose algorithmic training frameworks involving both CP and QN. Moreover, the Binary CNN hardware accelerator of [5] exhibits particularly remarkable operating points in terms of energy-accuracy. [57] (2019) already implements both DSE ternary quantization, whereas [58] (2017) only benefits from zero-weight activation knowledge for its CNN. Indeed, the proposed CMOS image sensor in [57] enables always-on Face Detection for on-demand Face Recognition thanks to a hybrid analog-digital CNN accelerator. In the specific case of binary computing, unconventional hardware structures are thus once again highlighted, for example with differential crosspoint memristor arrays [59]. SRAM in-memory computing also shows promising results [4] [50], typically for 1bit quantized NN. In addition to binary and ternary quantization, logarithmic quantization [60] can also relax hardware structures when low precision quantization too much impacts the overall NN accuracy. Other recent works implement DSE strategies [61] while relying on upstream results as depicted in [62].

Two distinct types of inference-dedicated hardware can thus be identified, first low-power edge-AI accelerator and secondly smart sensors with embedded decision making. The NN topology optimizations for both are of a highly different nature because of not sharing all the same design constraints. In particular, as detailed in this paper, two approaches have been developed, either starting from an already learned NN then reducing its complexity or retraining from scratch a specifically-dedicated NN topology. Moreover, in a near future, always-on sensor based systems will massively benefit from algorithm-hardware co-design. Thanks to our intensive literature study, we can conclude that next edge-AI devices will most probably involve NN topologies taking advantage of all the algorithmic enablers depicted in Fig. 1.

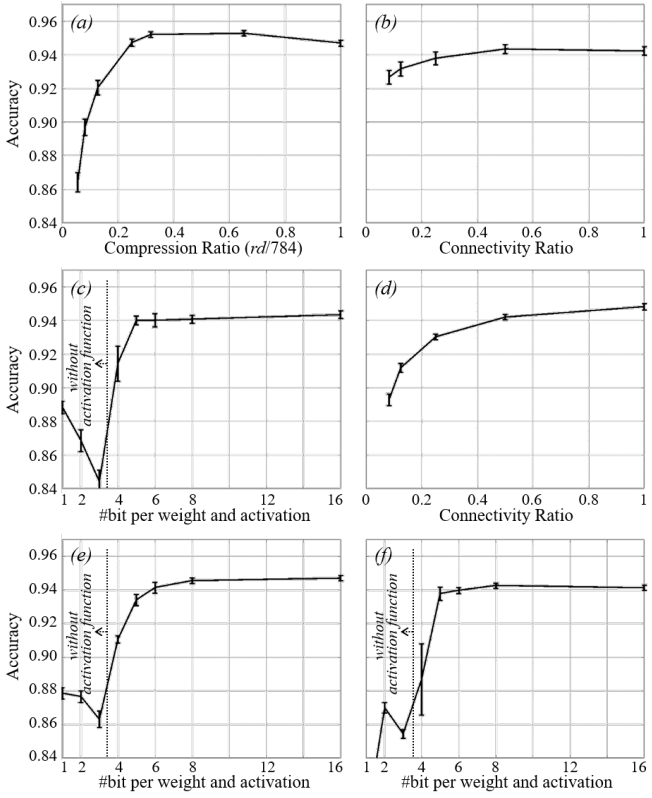

Fig. 2. Average accuracy performance on the MNIST test dataset (30 training& testing draws, error bar widths representing standard deviations), using DR via CS *(a)*, CP via random pruning *(b)*, QN (without activation functions if <4bit) *(c)*, CP with DR ($rd = 196$) *(d)*, QN with DR ($rd = 196$) *(e)*, and QN with DR ($rd = 196$) and CP (connectivity ratio of 50%) *(f)*.

Part of this work has been supported by the EU ECSEL project named OCEAN12 (Grant number 26 056 947).

REFERENCES

[1] K.-H. Tan and B. P. Lim, "The artificial intelligence renaissance: deep learning and the road to human-Level machine intelligence," *APSIPA*, 2018.

[2] J. Hestness *et al.*, "Beyond Human-level Accuracy: Computational Challenges in Deep Learning," *PPoPP*, 2019.

[3] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," *ISCA*, 2017.

[4] K. Bong *et al.*, "A 0.62mw ultra-low-power convolutional-neural-network face-recognition processor and a CIS integrated with always-on haar-like face detector," *ISSCC*, 2017.

[5] D. Bankman *et al.*, "An Always-On 3.8μJ/86% CIFAR-10 Mixed-Signal Binary CNN Processor With All Memory on Chip in 28-nm CMOS," *JSSC*, 2019.

[6] "Bringing the Power of AI to Millions of Devices, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/."

[7] M. Yang *et al.*, "Design of an Always-On Deep Neural Network-Based 1μW Voice Activity Detector Aided With a Customized Software Model for Analog Feature Extraction," *JSSC*, 2019.

[8] B. Chatterjee *et al.*, "RF-PUF: Enhancing IoT Security Through Authentication of Wireless Nodes Using In-Situ Machine Learning," *ITJ*, 2019.

[9] J. Choi *et al.*, "A 3.4μW Object-Adaptive CMOS Image Sensor With Embedded Feature Extraction Algorithm for Motion-Triggered Object-of-Interest Imaging," *JSSC*, 2014.

[10] C. Posch *et al.*, "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output," *Proceedings of the IEEE*, 2014.

[11] S. Dong *et al.*, "Spike Coding for Dynamic Vision Sensor in Intelligent Driving," *ITJ*, 2019.

[12] J. A. Prez-Carrasco *et al.*, "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing Application to Feedforward ConvNets," *TPAMI*, 2013.

[13] E. J. Candes and M. B. Wakin, "An Introduction To Compressive Sampling," *SPM*, 2008.

[14] E. C. Marques *et al.*, "A Review of Sparse Recovery Algorithms," *IEEE Access*, 2019.

[15] S. Lohit *et al.*, "Convolutional Neural Networks for Noniterative Reconstruction of Compressively Sensed Images," *TCI*, 2018.

[16] M. A. Davenport *et al.*, "Signal Processing With Compressive Measurements," *JSTSP*, 2010.

[17] T. Wimalajeewa *et al.*, "Performance Limits of Compressive Sensing-Based Signal Classification," *TSP*, 2012.

[18] A. Adler *et al.*, "Compressed Learning: A Deep Neural Network Approach," *SPARS*, 2016.

[19] S. Lohit *et al.*, "Direct inference on compressive measurements using convolutional neural networks," *ICIP*, 2016.

[20] A. Deerli *et al.*, "Compressively Sensed Image Recognition," *EUVIP*, 2018.

[21] B. Khalfi *et al.*, "When machine learning meets compressive sampling for wideband spectrum sensing," *IWCMC*, 2017.

[22] V. Schellekens and L. Jacques, "Quantized Compressive K-Means," *SPL*, 2018.

[23] L. Huang *et al.*, "CS-VQA: Visual Question Answering with Compressively Sensed Images," *ICIP*, 2018.

[24] W. Benjilali *et al.*, "An Analog-to-Information VGA Image Sensor Architecture for Support Vector Machine on Compressive Measurements," *ISCAS*, 2019.

[25] Y. Wu *et al.*, "Compressing YOLO Network by Compressive Sensing," *ACPR*, 2017.

[26] G.-B. Huang *et al.*, "Extreme learning machine: Theory and applications," *Neurocomputing*, 2006.

[27] Y. Miche *et al.*, "OP-ELM: Optimally Pruned Extreme Learning Machine," *TNN*, 2010.

[28] G.-B. Huang *et al.*, "Extreme learning machines: a survey," *IJMLC*, 2011.

[29] S. Ding *et al.*, "Extreme learning machine: algorithm, theory and applications," *AIR*, 2015.

[30] J. Tang *et al.*, "Extreme Learning Machine for Multilayer Perceptron," *TNNLS*, 2016.

[31] S. Han *et al.*, "Learning both Weights and Connections for Efficient Neural Networks," *NIPS*, 2015.

[32] X. Du *et al.*, "Towards Efficient Neural Networks On-A-Chip: Joint Hardware-Algorithm Approaches," *CSTIC*, 2019.

[33] S. Wiedemann *et al.*, "Compact and Computationally Efficient Representation of Deep Neural Networks," *TNNLS*, 2019.

[34] N. Yu *et al.*, "Accelerating convolutional neural networks by group-wise 2d-filter pruning," *IJCNN*, 2017.

[35] S. Lin *et al.*, "Toward Compact ConvNets via Structure-Sparsity Regularized Filter Pruning," *TNNLS*, 2019.

[36] L. Liang *et al.*, "Crossbar-Aware Neural Network Pruning," *IEEE Access*, 2018.

[37] D. Mittal *et al.*, "Recovering from Random Pruning: On the Plasticity of Deep Convolutional Neural Networks," *WACV*, 2018.

[38] Y. Boo and W. Sung, "Structured sparse ternary weight coding of deep neural networks for efficient hardware implementations," *SiPS*, 2017.

[39] M. Courbariaux *et al.*, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *NIPS*, 2015.

[40] I. Hubara *et al.*, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *arXiv:1609.07061*, 2016.

[41] S. Darabi *et al.*, "BNN+: Improved Binary Network Training," *arXiv:1812.11800*, 2018.

[42] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *arXiv:1712.05877*, 2017.

[43] Y. Boo *et al.*, "Memorization Capacity of Deep Neural Networks under Parameter Quantization," *ICASSP*, 2019.

[44] T. Simons and D.-J. Lee, "A Review of Binarized Neural Networks," *Electronics*, 2019.

[45] J. Cai *et al.*, "A Deep Look into Logarithmic Quantization of Model Parameters in Neural Networks," *ICAIT*, 2018.

[46] Y. Choukroun *et al.*, "Low-bit Quantization of Neural Networks for Efficient Inference," *arXiv:1902.06822*, 2019.

[47] T. Hirtzlin *et al.*, "Stochastic Computing for Hardware Implementation of Binarized Neural Networks," *IEEE Access*, 2019.

[48] K. Ando *et al.*, "Dither NN: An Accurate Neural Network with Dithering for Low Bit-Precision Hardware," *FPT*, 2018.

[49] L. Liu and J. Deng, "Dynamic Deep Neural Networks: Optimizing Accuracy-Efficiency Trade-offs by Selective Execution," *arXiv:1701.00299*, 2017.

[50] K. Bong *et al.*, "A Low-Power Convolutional Neural Network Face Recognition Processor and a CIS Integrated With Always-on Face Detector," *JSSC*, 2018.

[51] M. I. Jordan and R. A. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm," *Neural Computation*, 1994.

[52] W. Benjilali *et al.*, "Exploring Hierarchical Machine Learning for Hardware-Limited Multi-Class Inference on Compressed Measurements," *ISCAS*, 2019.

[53] Y. Wang *et al.*, "Dual Dynamic Inference: Enabling More Efficient, Adaptive and Controllable Deep Inference," *arXiv:1907.04523*, 2019.

[54] C. Hu *et al.*, "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge," *INFOCOM*, 2019.

[55] G. Srivastava *et al.*, "Joint Optimization of Quantization and Structured Sparsity for Compressed Deep Neural Networks," *ICASSP*, 2019.

[56] X. Long *et al.*, "Learning Sparse Convolutional Neural Network via Quantization With Low Rank Regularization," *IEEE Access*, 2019.

[57] J. Kim *et al.*, "An Ultra-Low-Power Analog-Digital Hybrid CNN Face Recognition Processor Integrated with a CIS for Always-on Mobile Devices," *ISCAS*, 2019.

[58] D. Kim *et al.*, "A novel zero weight/activation-aware hardware architecture of convolutional neural network," *DATE*, 2017.

[59] P. Chiu *et al.*, "A Binarized Neural Network Accelerator with Differential Crosspoint Memristor Array for Energy-Efficient MAC Operations," *ISCAS*, 2019.

[60] T. Ueki *et al.*, "Learning Accelerator of Deep Neural Networks with Logarithmic Quantization," *IIAI-AAI*, 2018.

[61] H. Wu *et al.*, "A 0.23mw Heterogeneous Deep-Learning Processor Supporting Dynamic Execution of Conditional Neural Networks," *ESSCIRC*, 2018.

[62] K. Goetschalckx *et al.*, "Optimized Hierarchical Cascaded Processing," *JESTCAS*, 2018.