# MATIC: Adaptation and In-Situ Canaries for Energy-Efficient Neural Network Acceleration

Sung Kim†, Patrick Howe†, Thierry Moreau‡, Armin Alaghi‡, Luis Ceze‡, Visvesh Sathe†
Department of Electrical Engineering†, Paul G. Allen School of Computer Science and Engineering‡
University of Washington
Seattle, WA, USA

***Abstract*—We present MATIC (Memory Adaptive Training with In-situ Canaries), a voltage scaling methodology that addresses the SRAM efficiency bottleneck in DNN accelerators. To overscale DNN weight SRAMs, MATIC combines the characteristics of destructive SRAM reads with the error resilience of neural networks in a memory-adaptive training process. PVT-related voltage margins are eliminated using bit-cells from synaptic weights as in-situ canaries to track runtime environmental variation. Demonstrated on a low-power DNN accelerator fabricated in 65nm CMOS, MATIC enables up to 3.3× total energy reduction, or 18.6× application error reduction.**

## I. Introduction

Deep neural networks (DNNs) have demonstrated state-of-the-art performance on a variety of signal processing tasks, and there is growing interest in DNNs for next-generation IoT and embedded platforms. However, recent work has shown that for accelerators that reduce or eliminate DRAM access, on-chip SRAM dedicated to synaptic weights accounts for greater than 50% of total system power [1]. The on-chip memory problem is particularly acute in DNNs with classifier layers, where data-reuse techniques [2], [3] are ineffective since classifier weights are unique, and account for greater than 80% of total weight parameters [4]. Voltage scaling can enable significant static and dynamic power reduction, however read and write stability constraints have historically prevented more aggressive scaling for SRAM - SRAM is either placed on a separate voltage rail hundreds of millivolts higher than the rest of the design, or the system shares a unified voltage domain. In either case, significant energy savings from voltage scaling remain unrealized due to SRAM operating voltage constraints, translating to shorter operating lifetime.

## II. Background and Contributions

### A. Deep Neural Networks

Deep neural networks (DNNs) are a class of bio-inspired machine learning models that are represented as a directed graph of neurons [5] (Figure 1). DNN operation can be divided into two key mechanisms: **(1)** *training* and **(2)** *inference*:

**(1)** During inference, a neuron $k$ in layer $j$ implements the composite function $z_k^{(j)} = f\left(\sum_{i=1}^{N^{(j-1)}} w_{k,i}^{(j)}\ z_i^{(j-1)}\right)$. $z_i^{(j-1)}$ denotes the output from neuron $i$ in the previous layer, and $w_{k,i}^{(j)}$ denotes the weight in layer $j$ from neuron $i$ in the previous layer to neuron $k$. $f(x)$ is a non-linear function, typically a sigmoidal function or rectified linear unit (ReLU).

In matrix form, the neuron composite function is equivalent to the matrix-vector product in Eq.1 (with $f(x)$ computed element-wise). Hence, DNN execution is especially amenable to dataflow hardware architectures designed for linear algebra.

$$\mathbf{z}^{(j)} = f\begin{bmatrix} w_{1,1}^{(j)} & \dots & w_{1,N}^{(j)} \\ \vdots & \ddots & \vdots \\ w_{M,1}^{(j)} & \dots & w_{M,N}^{(j)} \end{bmatrix}\begin{bmatrix} z_1^{(j-1)} \\ \vdots \\ z_N^{(j-1)} \end{bmatrix} = f\left(\mathbf{W}^{(j)}\mathbf{z}^{(j-1)}\right) \quad (1)$$

**(2)** Training involves iteratively solving for weight parameters using some variation of gradient descent. Given a weight $w_{i,k}^{(j)}$, its value at training iteration $n$ is given by $w_{k,i}^{(j)}[n] = w_{k,i}^{(j)}[n-1] - \alpha\dfrac{\partial J}{\partial w_{k,i}^{(j)}[n-1]}$, where $J$ is a suitable loss function (e.g., mean-squared error or cross-entropy).

The partial derivatives of the loss function w.r.t. the weights are computed by propagating error backwards via the chain rule (backprop). For example, for a network with a single hidden layer, sigmoid activations $f(x) = 1/(1+e^{-x})$ and mean-squared loss, the error gradient w.r.t. a weight $w_{k,i}^{(2)}$ is given by $\dfrac{\partial J}{\partial w_{k,i}^{(2)}} = \dfrac{\partial J}{\partial z^{(J)}}\dfrac{\partial z^{(J)}}{\partial x}\dfrac{\partial x}{\partial w_{k,i}^{(2)}}$. MATIC relies on the observation that backprop makes error at the output, including artificial error created by weight perturbations, observable by *all* weights in the network.

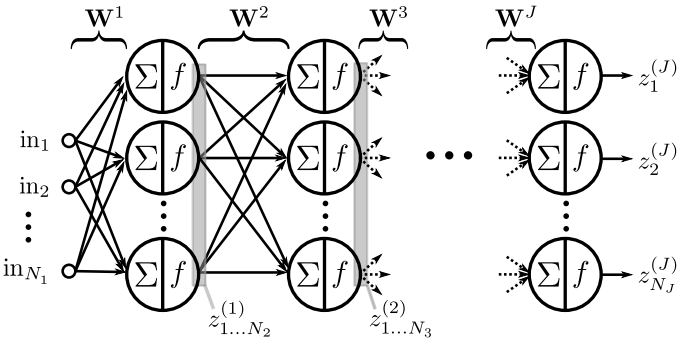
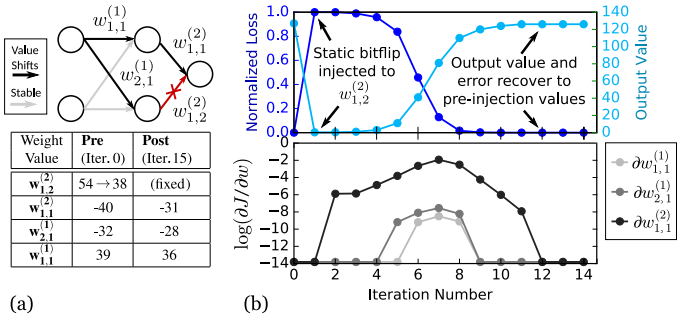



Fig. 1. General architecture of a fully-connected DNN

### B. Weight Adaptation

The simplified example in Figure 2 shows how trainable weight parameters imbue neural networks with intrinsic resilience to error. The network in Figure 2(a) is initialized with 8-bit integer weights such that the network loss is zero for the

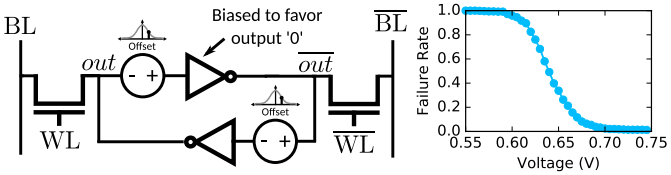
Technical Report UW-CSE-2017-05-01




| Weight Value | Pre (Iter. 0) | Post (Iter. 15) |
|---|---|---|
| $w^{(2)}_{1,2}$ | 54 → 38 | (fixed) |
| $w^{(2)}_{1,1}$ | -40 | -31 |
| $w^{(1)}_{2,1}$ | -32 | -28 |
| $w^{(1)}_{1,1}$ | 39 | 36 |

(a) (b)

Fig. 2. **(a)** Weight values pre and post-adaptation. **(b)** Error, output value, and error gradient characteristics across SGD iterations.




(a) (b)

Fig. 3. **(a)** SRAM 6T bit-cell with mismatch-induced input offset **(b)** Read-failure rate with Monte Carlo simulation of 10K bit-cells.

training set. The network uses a sigmoid activation function, and square loss with stochastic gradient descent (SGD) for training [5]. At iteration 1 we apply a static mask to bit '4' of $w^{(2)}_{1,2}$ to simulate a fault, and observe the adaptation of surrounding weights after several iterations of SGD. Figure 2(b) shows that while masking $w^{(2)}_{1,2}$ initially degrades error performance, the surrounding weights adapt after several iterations of SGD; the large, non-zero gradients illustrate the backprop mechanism compensating for the error injected to $w^{(2)}_{1,2}$.

### C. SRAM Read Failures

Figure 3 shows a simplified 6T SRAM bit-cell model. Variation-induced mismatch between bit-cell devices creates an inherent state-independent offset [6]. This offset results in each bitcell having a "preferred state." For instance, the bit-cell depicted in Figure 3 favours driving $\overline{\text{out}}$ to logic '0'. Due to statistical variation, larger memories are likely to see a number of cells with significant offset error.

As supply voltage scales, the diminished noise margin allows the bit-cell to be flipped to its preferred state during a read. Once flipped, the bit-cell retains state, favouring its (now incorrect) bit value due to the persistence of the built-in offset. Consequently, the occurrence of memory bit-cell read failure at low supply voltages is random in space, but essentially provides *stable* read outputs consistent with its preferred state. The read failures described above are in distinct from bitline access-time failures, which can be corrected with ample timing margin.

### D. Contributions

In this paper we present MATIC (Memory Adaptive Training with In-situ Canaries), the first hardware/algorithm co-design methodology that exploits the inherent error tolerance of DNNs to aggressively scale the voltage of weight SRAMs with little to no accuracy loss. In addition to the development of MATIC, we design and implement SNNAC, a low-power DNN accelerator for energy-constrained mobile devices, and demonstrate state-of-the-art performance.

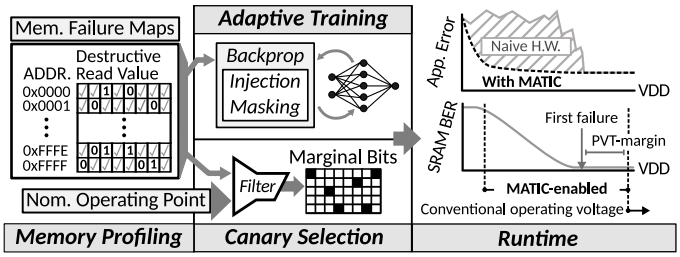



Fig. 4. Overview of the MATIC compilation and deployment flow.

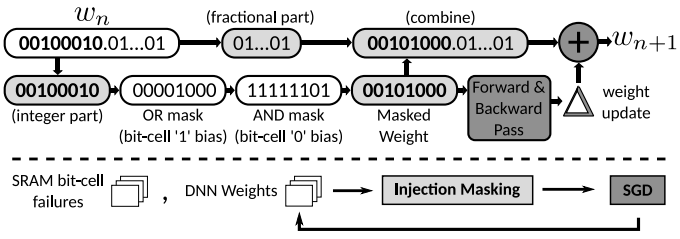



Fig. 5. The modified DNN training algorithm and injection masking process.

## III. Voltage Scaling for DNN Accelerators

MATIC (Figure 4) uses two techniques to operate SRAMs past their point of failure while maintaining *application* accuracy across PVT variation: **(A.)** Memory-adaptive training, and **(B.)** in-situ synaptic canaries.

### A. Memory-adaptive Training

Memory-adaptive training leverages the inherent resilience of neural networks to adapt *around* bit-errors in synaptic weights that result from voltage scaling past $V_{min,read}$. Random mismatch results in bit-cells that are biased towards a given storage state. If a bit-cell stores the complement of its "preferred" state, performing a read at a sufficiently low *VDD* flips the cell and subsequent reads will be incorrect but largely stable [6]. SRAM read failures, as described above, are profiled post-silicon and incorporated into the backpropagation (backprop) algorithm (as described in Section II-B) with an *injection mask* (Figure 5). During every training iteration, the injection mask preserves the fractional portion of a given synaptic weight and applies bit-masks (that correspond to bit failures in SRAM) to the fixed-point weight. The masked weight is used in SGD before being recombined with its fractional part and weight update. Preserving the fractional weight is critical, since it enables gradual value-shifts that occur over multiple (fractional) weight updates.

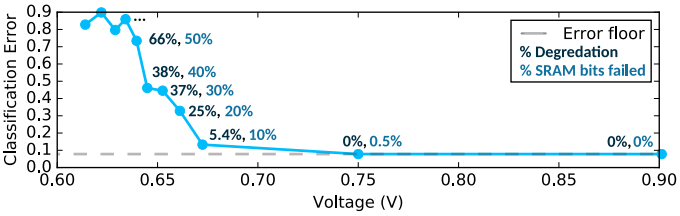



Fig. 6. Simulated performance of memory-adaptive training on MNIST.

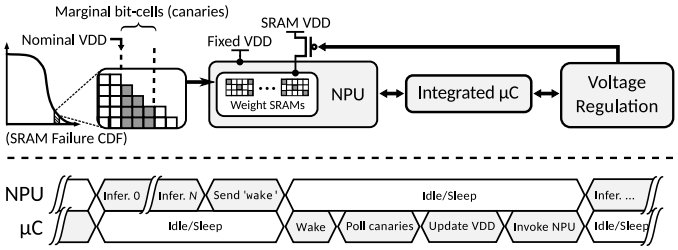



Fig. 7. NPU-µC interaction, and in-situ canary-based SRAM VDD control.

To evaluate the feasibility of memory-adaptive training, we first examine the memory-adaptive training flow with simulated SRAM failure rates. At each voltage, a proportion of randomly selected weight bits are fixed to either '1' or '0', where the proportion of fixed bits is determined from SPICE Monte Carlo simulation. Figure 6 shows that a significant fraction of bit errors can be tolerated before application error degrades.

### B. In-Situ Synaptic Canaries

The in-situ canary circuits are bit-cells directly from synaptic weight SRAMs that facilitate SRAM supply-voltage control (Figure 7). Traditional canary circuits replicate critical circuits to detect imminent failure, but require added margin and are vulnerable to PVT-induced mismatch. Instead, MATIC uses synaptic weights themselves as in-situ canary circuits, leveraging a select number of bit-cells that are on the margin of failure to maintain a target bit-cell read failure rate. The operation of canary relies on two key observations:

1. Since the most marginal, failure prone bit-cells are chosen as canaries, canaries fail before other performance-critical bit-cells.
2. Neural networks are robust to a small fraction of *uncompensated* errors [7]. As a result, the failure states of canary bit-cells are not critical for overall performance, and canaries can be selected directly from synaptic weights.

At runtime, in-situ canary bits are polled periodically to determine whether supply voltage modifications should be applied. While we use an integrated microcontroller in the test chip described below, the runtime controller can be implemented with faster or more efficient circuits, if required. For our tests we use a binary control policy where SRAM supply voltage is adjusted in 1mV steps upon detecting a failing/succeeding canary bit-cell. Since we pick the most marginal, failure prone bit-cells, at runtime only canary bits are re-written. For canary selection, we conservatively select eight distributed, marginal canary bit-cells from each weight-storage SRAM.

## IV. DNN Accelerator Architecture

To demonstrate the effectiveness of MATIC, we implement a light-weight SoC called SNNAC (Systolic Neural Network AsiC) in 65nm CMOS. The SNNAC architecture (Figure 8) is based on the systolic dataflow design from SNNAP [8], heavily modified for SoC integration. The SNNAC core consists of a fully-programmable central Neural Processing Unit (NPU) that contains eight multiply-accumulate (MAC)-based Processing Elements (PEs) arranged in a systolic ring. Energy-efficient PE compute is acheived with fixed-point arithmetic with 8-22 bit precision, and each PE uses a dedicated voltage-scalable

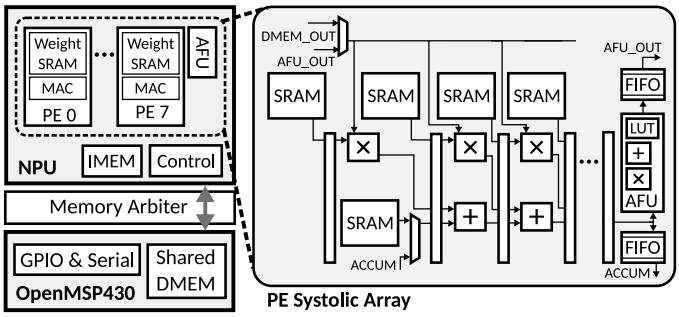



Fig. 8. Architecture of the SNNAC DNN Accelerator.

SRAM bank to enable local storage of synaptic weights. The systolic ring is attached to the activation function unit (AFU), which minimizes energy footprint with piecewise-linear approximations of activation functions (e.g., sigmoid, ReLU). Programmability is acheived with statically compiled instruction schedules, which time-multiplex the computation of each DNN layer onto the systolic array. SNNAC also includes a sleep-enabled OpenMSP430-based microcontroller (µC), which handles runtime control and off-chip communication.

## V. Experimental Results

At 25C and 0.9V, SNNAC nominally operates at 250 MHz and dissipates 16.8 mW, achieving a 90.6% classification rate on MNIST [9]. The other application benchmarks include face detection (MIT CBCL face database [10]), and 2 benchmarks from the approximate computing suite AxBench [11]. We find that the compiled SRAMs (rated at 0.9V) exhibit read failures starting from 0.53V, with all reads failing at ~0.4V (Figure 9(a)). Figure 11 shows how MATIC recovers application error, resulting in an 18.6× reduction in average error-increase (AEI) versus naive hardware (Table I). To avoid biasing the application error analysis, all benchmarks use compact DNN topologies that minimize intrinsic over-parameterization (Figure 9(b)).

For energy-efficiency we consider the operation of SNNAC in 3 feasible operating scenarios, *HighPerf* (high performance), *EnOpt_A* (energy optimal, separate logic/SRAM voltages), and *EnOpt_B* (energy optimal, single voltage domain). At the minimum-energy point (MEP) across the 3 cases, MATIC

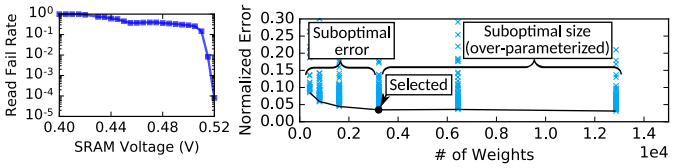



Fig. 9. **(a)** Measured SRAM read-failure rate at 25C. **(b)** Topology selection to avoid overparameterization - each point is a unique DNN topology.

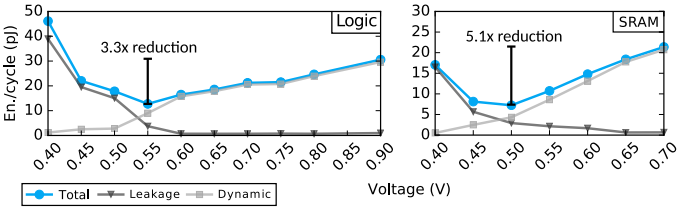



Fig. 10. Energy-per-cycle measurements for the SNNAC test chip.

TABLE I. DNN BENCHMARKS, AND APPLICATION ERROR MEASUREMENTS.

| Benchmark | Description | Error Metric | DNN Topology | Error@0.9V (nominal) | Error@0.50V (naive) | Error@0.50V (adaptive) | Error@0.46V (naive) | Error@0.46V (adaptive) | AEI (naive) | AEI (adapt.) | Reduction (0.46V-0.52V) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *mnist* [9] | Digit recognition | Classif. rate | 100-32-10 | 9.4% | 70.7% | 13.0% | 84.0% | 15.6% | 0.65 | 0.050 | **12.5** |
| *facedet* [10] | Face detection | Classif. rate | 400-8-1 | 12.5% | 33.6% | 15.6% | 47.7% | 15.8% | 0.38 | 0.056 | **6.7** |
| *inversek2j* [11] | Inverse kinematics | Mean sq. error | 2-16-2 | 0.032 | 0.169 | 0.040 | 0.245 | 0.050 | 0.50 | 0.019 | **26.7** |
| *bscholes* [11] | Option pricing | Mean sq. error | 6-16-1 | 0.021 | 0.094 | 0.023 | 0.094 | 0.026 | 0.67 | 0.023 | **28.4** |
| Average | - | - | - | - | - | - | - | - | - | - | **18.6** |

TABLE II. ENERGY EVALUATION WITH MATIC-ENABLED SCALING.

| Param/Config. | ***HighPerf*** | Base | ***EnOpt_A*** | Base | ***EnOpt_B*** | Base |
|---|---|---|---|---|---|---|
| Logic Voltage (V) | 0.9 | 0.9 | 0.55 | 0.55 | 0.55 | 0.9 |
| SRAM Voltage (V) | 0.65 | 0.9 | 0.5 | 0.9 | 0.55 | 0.9 |
| Frequency (MHz) | 250 | - | 17.8 | - | 17.8 | - |
| **Total (pJ/cycle)** | **48.96** | **67.08** | **19.98** | **49.23** | **20.60** | **67.08** |
| Logic | 30.58 | 30.58 | 12.73 | 12.73 | 12.73 | 30.58 |
| SRAM | 18.37 | 36.50 | 7.24 | 36.50 | 7.86 | 36.50 |
| **Energy Reduction** | **1.4**× | - | **2.5**× | - | **3.3**× | - |

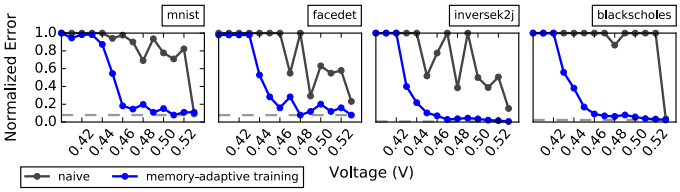
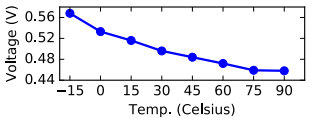


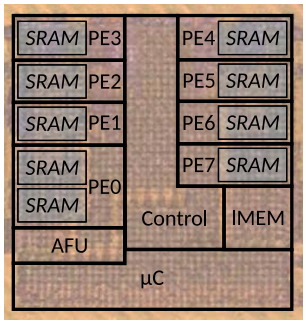

Fig. 11. Error performance of SNNAC, with and without MATIC deployed.




(a)

| Technology | TSMC GP 65nm |
|---|---|
| Core Area | 1.15×1.2mm |
| Voltage | 0.44-0.9V |
| Frequency | 1.8-250 MHz |
| Power | 0.1-16.8 mW |
| Energy | 19.9-67.1 pJ/cycle |

(b)




(c)

Fig. 12. **(a)** SRAM VDD set by the in-situ canary system, initialized at 0.5V, 25C. **(b)** Chip performance summary. **(c)** Die microphoto.

TABLE III. PERFORMANCE COMPARISON

| | Low-power embedded, FC | | | High-performance, Conv. | |
|---|---|---|---|---|---|
| | **This Work** | ISSCC'17 [12] | ISCA'16 [1] | VLSI'16 [3] | ISSCC'16 [2] |
| Process | **65nm** | 40nm | 45nm | 40nm | 65nm |
| Area | **1.4 mm sq.** | 7.1 mm sq. | 0.64 mm sq. | 2.4 mm sq. | 12.2 mm sq. |
| DNN Type | **FC** | FC | FC | Conv. | Conv. |
| Power | **0.37 mW** | 0.29 mW | 9.2 mW | 33 mW | 278 mW |
| Frequency | **17.8 MHz** | 3.9 MHz | 800 MHz | 204 MHz | 200 MHz |
| Voltage | **0.44-0.9V** | 0.63-0.9V | 1V | 0.55-1.1V | 0.82-1.17V |
| Efficiency | **400.5 GOPS/W** | 374 GOPS/W | 174 GOPS/W | 1.6 TOPS/W | 243 GOPS/W |

enables up to 3.3× total energy reduction, and 5.1× energy reduction in SRAM (Table II, Figure 10). We note that SRAM energy is minimized at 0.5V with a 38% SRAM bit-cell failure rate, which translates to an 87% classification rate on MNIST (versus 29.3% for naive hardware). To demonstrate system stability over temperature, we execute the application benchmarks in a chamber with ambient temperature control, and sweep temperature from -15C to 90C for a given nominal voltage. Figure 12(a) shows the SRAM voltage settings dictated by the in-situ canary system for an initial setting at 0.5V, with 0% mismatch with the expected error. Table III shows that the MATIC-SNNAC combination achieves state-of-the-art energy-efficiency and wider operating-voltage range compared to other FC DNN accelerators. The performance of SNNAC is either better than or comparable to state-of-the-art convolutional-centric (Conv.) accelerators, despite the lack of data and filter reuse techniques in unique FC-layers.

## VI. CONCLUSION

In this paper we presented MATIC, the first hardware/algorithm co-design methodology that addresses the energy-efficiency bottleneck imposed by synaptic weight SRAMs. Key developments and contributions include

1. Memory-adaptive training - a technique that leverages the adaptability of neural networks to train around errors resulting from SRAM voltage scaling.
2. In-situ synaptic canaries - the use of bit-cells directly from weight SRAMs for voltage control and variation-tolerance.

In addition, we designed and implemented SNNAC, a low-power DNN accelerator fabricated in 65nm CMOS (Figure 12(b-c)). Demonstrated on SNNAC, the application of MATIC results in 3.3× total energy reduction and 5.1× energy reduction in SRAM, or 18.6× reduction in application error, thus enabling robust and energy-efficient operation for a general class of DNN accelerators.

## ACKNOWLEDGMENT

The authors would like to thank Fahim Rahman, Rajesh Pamula, and John Euhlin for design support.

## REFERENCES

[1] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *ISCA*, 2016, pp. 243–254.

[2] Y.-H. Chen *et al.*, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *ISSCC*, 2016.

[3] B. Moons *et al.*, "A 0.3-2.6 TOPS/W precision-scalable processor for real-time large-scale convnets," in *VLSIC*, 2016, pp. 1–2.

[4] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006, 2006.

[6] A. J. Bhavnagarwala *et al.*, "The impact of intrinsic device fluctuations on CMOS SRAM cell stability," *JSSC*, vol. 36, pp. 658–665, Apr. 2001.

[7] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *ISCA*, June 2012, pp. 356–367.

[8] T. Moreau *et al.*, "SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration," in *HPCA*, 2015.

[9] Y. LeCun *et al.*, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[10] M. Alvira *et al.*, "An Empirical Comparison of SNoW and SVMs for Face Detection," MIT, Cambridge, MA, Tech. Rep., 2001.

[11] A. Yazdanbakhsh *et al.*, "AxBench: A Multi-Platform Benchmark Suite for Approximate Computing," *IEEE Des. Test*, 2016.

[12] S. Bang *et al.*, "14.7 A 288 uW programmable deep-learning processor with 270KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *ISSCC*, Feb 2017, pp. 250–251.