Offline Training for Memristor-based Neural Networks

Guillem Boquet, Edwar Macias, Antoni Morell, Javier Serrano, Enrique Miranda and Jose Lopez Vicario
Universitat Autònoma de Barcelona (UAB)
Email: {guillem.boquet, edwar.macias, antoni.morell, javier.serrano, enrique.miranda, jose.vicario}@uab.cat

***Abstract*—Neuromorphic systems based on Hardware Neural Networks (HNN) are expected to be an energy-efficient computing architecture for solving complex tasks. Due to the variability common to all nano-electronic devices, HNN success depends on the development of reliable weight storage or mitigation techniques against weight variation. In this manuscript, we propose a neural network training technique to mitigate the impact of device-to-device variation due to conductance imperfections at weight import in offline-learning. To that aim, we propose to add said variation to the weights during training in order to force the network to learn robust computations against that variation. Then, we experiment using a neural network architecture with quantized weights adapted to the design constrains imposed by memristive devices. Finally, we validate our proposal against real-world road traffic data and the MNIST image data set, showing improvements on the classification metrics.**

***Index Terms*—Neuromorphic, Deep learning, RRAM, Memristor, Traffic forecasting**

I. Introduction

Neuromorphic computing, which imitates the principle behind biological synapses with a high degree of parallelism, has recently emerged as a promising candidate for novel and sustainable computing technologies [1]. In the transportation field and many road networks, accurate real-time road traffic prediction is required to prevent the risks and inconveniences associated to traffic congestion, which demands a larger amount of data processing with high energy efficiency. Associated to that and with the addition of Intelligent Transportation Systems (ITS), it is expected an increase in the number of edge devices such as Road Side Units (RSU) to collect data from the physical world.

In that context, neuromorphic systems based on Hardware Neural Networks (HNN) implemented with memristive devices are a feasible solution and an energy-efficient computing architecture for solving such tasks in the edge [2]. One of the major challenges of hardware networks is the development of reliable weight storage due to the variability common to all nano-electronic devices [3]. Weights are efficiently implemented using memristive devices or memristors in hardware networks, which are devices that behaves as a resistor with memory. Thus, memristors are very promising candidates owing to their analogue memory functionality [1], [4] and their simple structure that allows for crossbar circuit integration and aggressive size scaling, which are necessary for the practical implementation of deep neural networks (DNN). Memristor-based networks can be trained by offline (or *ex situ*) and online (or *in situ*)-learning methods. In the first case, which is the focus of this manuscript, weights are calculated on a precursor software-based network and then imported sequentially into the crossbar circuit. In the second case, training is implemented *in situ* in hardware and only for small neural networks [5], so the weights are adjusted in parallel which is significantly more demanding [4]. In both cases, a high precision weight import is required to implement complex networks and achieve the expected performance when the network is realized. However, various properties of memristors are known to affect negatively the performance of neuromorphic systems [1]. Specifically, the conductance response of any real nonvolatile memory (NVM) device exhibits imperfections that derive in unreliable performance of the network. Those imperfections include nonlinearity, stochasticity, varying maxima, asymmetry between increasing and decreasing responses, and non-responsive devices at low or high conductance [6]. For example, most memristor show a nonlinear weight update, where the conductance change gradually saturates [1].

Recently, [2] stated that many issues still need to be resolved both at the material, device and system levels to simultaneously achieve high accuracy, low variability, high speed, energy efficiency, small area, low cost, and good reliability. Thus, in order to memristor-based networks match the state-of-the-art networks that are currently used in software literature, more advanced software and hardware mitigation techniques are needed. First, in order to achieve high reliability at network realization and, secondly, to reduce the cost of production since networks that are more robust against variation can be implemented with low-cost devices. In that sense, [5] presented a mask technique to capture the sneak path problem stating that any kind of training incorporating the knowledge of the crossbar array behavior will likely improve the accuracy of memristor-based networks significantly. Similar to our proposal, their technique is integrated into the weights during neural network training. In this work, we propose an offline neural network training technique to mitigate the impact of device-to-device variation due to conductance imperfections at weight import in neuromorphic systems. First, we propose a DNN architecture with quantized weights adapted to the design constrains imposed by memristive devices currently used in neuromorphic literature. In addition, we propose to add to the weights during training a variable that models the conductance variation to force the network to learn robust calculations against weight variation. Finally, we experiment against real-world road traffic data and the well-known MNIST data set and provide extensive results to validate our proposal.

This research is supported by the Catalan Government under Project 2017 SGR 1670 and the Spanish Government under Project TEC2017-84321-C4-4-R co-funded with European Union ERDF funds.

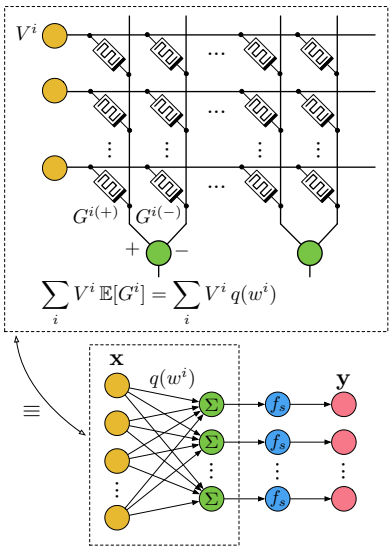

Fig. 1: Schematic (bottom) and hardware RRAM crossbar array equivalent (top) of a hidden layer of the quantized neural network described in Sec. II. Two RRAM cells per weight are used to create negative weights. The addition of HRS and LRS conductances leads to the equivalent negative weight.

II. Methodology

Quantized neural networks (QNN) have been proposed because of their benefits in terms of hardware such as area and power saving, e.g., Binary Neural Networks (BNN) [7]. QNN use quantized values of inputs, weights and/or activations to become smaller and faster, achieving similar accuracy to floating point DNN. These networks are realized with binary memristor crossbars, where memristors are programmed by high-resistance state (HRS) and low-resistance state (LRS) accordingly to the training algorithm. Typical memristive neural networks store weights in Resistive Random Access Memory (RRAM) crossbar arrays (RCA), Fig. 1. For example, a binary RRAM Convolutional Neural Network (CNN) was recently designed and optimized in [8] while in [9] a full hardware implementation of an RRAM-based convolutional block was presented.

Next, we propose a QNN architecture adapted to the design constrains imposed by RCA-based networks currently used in neuromorphic literature. Our proposed network is based on Ternary Weights Networks (TWN) [10], which have stronger expressive abilities than their binary counterpart. Additionally, we propose to add a random variable to weights during training to mitigate the device-to-device variation due to conductance imperfections and the small window between LRS and HRS at network realization.

A. Quantized neural network

We start by defining the activation $\mathbf{y}$ (i.e., output) of the l-th layer of our DNN as

$$\mathbf{y}^l = f_s^l\big([g \oplus q(\mathbf{W}^l)]\,\mathbf{x}^l + \mathbf{b}^l\big), \tag{1}$$

where $\mathbf{x}^l$ is the input, $\mathbf{b}^l$ the bias term, $q(\mathbf{W}^l)$ the quantized weights, f_s^l an scaled activation function and g a random variable $g \sim \mathcal{N}(0, \sigma_G^2)$ described by the combination of HRS and LRS conductances' variability. Please, notice that we used here $\oplus$ to denote the element-wise addition of a scalar to a matrix.

B. Input range

Typically, the dynamic range of RCA is limited to values where the conductance exhibits linear behavior. Hereafter, we assume an input voltage range where the conductance is constant and the voltage-intensity ratio is linear. For example, Fig. 3 in [11] shows the linear behavior from 0 to 0.2V. Thus, we limit the values of $\mathbf{x}$ to a range of low values, $\mathbf{x} \in [0, V_{max}]$ [4]. Accordingly, we scale the activation function f to match the input range of the next layer as $f_s = V_{max}\, f$, which is crucial to allow the network to easily train, achieve much better accuracies and ensure that the input to the next layer stays inside the dynamic operation range.

C. Ternarization

Weights of DNN need to be quantized in order to be implemented in hardware because memristor conductances are limited to few values. We define the deterministic quantization function q as

$$q(w^{i,l}) = \begin{cases} +\mu_G & w^{i,l} > \Delta \\ 0 & |w^{i,l}| \le \Delta \\ -\mu_G & w^{i,l} < \Delta \end{cases}, \tag{2}$$

where Δ is a positive defined threshold [10], μ_G the quantization value and $w^{i,l}$ are the elements of $\mathbf{W}^l$, that is, the real-valued weights of the l-th layer of the software-based network. As q limits to three possible values, 2-bit storage requirement is needed for a unit of weight, which achieve up to $16\times$ or $32\times$ model compression rate compared with the float (32-bit) or double (64-bit) precision counterparts.

D. Conductance model

The conductance response of memristors exhibits imperfections that can decidedly affect performance. Those imperfections derive in an uncertainty of the values of weights at the time of device realization, that is, there exists a device-to-device variation governed by the memristor's characteristics. We are interested in modeling the variation of the final conductance value at weight import because we focus on offline-learning. Example measures of real device-to-device variation of the conductance were shown in [12] and [13], where Pham *et al.* [13] reported the memristance's statistical distribution. Based on that, we modeled the realized memristor conductance $G \sim \mathcal{N}(\mu_G, \sigma_G^2)$ as a random variable that follows a Normal distribution with fixed moments μ_G and σ_G^2. As we aim to mitigate the impact of said uncertainty during offline-learning, we incorporated that knowledge using μ_G in (2) and $g \sim \mathcal{N}(0, \sigma_G^2)$ as an addition term in (1). In other words, we proposed to add random noise to weights after its quantization during training in order to force the network to learn robust computations against weight variation.

TABLE I: Parameter values used for simulation.

Parameter	Value	Parameter	Value
V_{max}	0.2 V	$\hat{\mu_L}$	1
Δ	0.05	$\hat{\mu_H}$	$\{0.1, 0.3, 0.5\}$
$f_s(a)$	$0.2 \times \frac{1}{1+e^{-a}}$	$\sigma_H/\hat{\mu_H}$	21%
t_c	$\hat{\mu_L} - \hat{\mu_H}$	$\sigma_L/\hat{\mu_L}$	$\{3, 10, 20, 30, 40\}$%
σ_G	$\{0.1, 0.2, 0.3\}$		

E. Value of weights

Here, we consider using two RRAM cells per weight as the weight realization technique in order to create negative weights [5]. Consider $\mu_L^+ = \mu_L^- > \mu_H^+ = \mu_H^-$ as the respective average conductances of LRS and HRS with their associated standard deviations $\sigma_L^+ = \sigma_L^-$ and $\sigma_H^+ = \sigma_H^-$, where the superscripts $^+$ and $^-$ denote the first and second RRAM cell respectively (Fig. 1). Direct linear combinations of the four average conductances result in the three possible weight values $\{\mu_H^+ - \mu_L^-, 0, \mu_L^+ - \mu_H^-\}$. Thus, we define the quantization value of our quantized network as $\mu_G = \mu_L^+ - \mu_H^- = |\mu_H^+ - \mu_L^-|$. To calculate its variation σ_G, we assume variables are independent and that $\sigma_L \leq \sigma_H \ll \mu_G$ [11]. Then, the standard deviation of the sum is computed taking the square root of the combined variance for two different cases,

$$\sigma_G = \begin{cases} \sqrt{\sigma_L^2 + \sigma_H^2} & w^{i,l} \neq 0 \\ \sqrt{\sigma_H^2 + \sigma_H^2} & w^{i,l} = 0 \end{cases},$$

where in the case $w^{i,l} = 0$ we considered the highest variance. Then, $q(w^{i,l})$ is the expected weight value at realization of the hardware network and σ_G^2 its variance or the second moment of G. Finally, we normalized μ_H and μ_L using the maximum value μ_L as the base value, so that $\hat{\mu_L} = 1$.

F. Weight update

QNN learning is done via *backpropagation* and Stochastic Gradient Descent (SGD). Ternary-valued weights are used during the forward and backward propagations but not during the parameters update, because keeping good precision weights during the updates is necessary for SGD to work [14]. Additionally, the quantization function (2) is not differentiable, thus its gradient is estimated using the Straight-Through Estimator (STE):

$$\frac{\delta q(w^{i,l})}{\delta w^{i,l}} = \begin{cases} 1 & |w^{i,l}| \leq t_c \\ 0 & |w^{i,l}| > t_c \end{cases},$$

where t_c is a threshold for clipping the gradients. The quantization is replaced by a clipped identity on the backward pass, which preserves the gradient's information and cancels large gradients without any impact on the ternary weights.

III. Experimentation

In this section, we experimented with the proposed QNN. Table I summarizes the values used for simulation. We used common values of conductance and dynamic range [4]. For comparison, we trained two different networks: with and without the proposed g factor in (1), which we refer as *Proposed* and *Original* respectively. For each experiment, we used the same random seeds to initialize the pseudo random number generators. Regarding the software-network simulation, we trained 10 times the *Proposed* and *Original* networks starting from the same initialization and kept the best performing networks for further evaluation. The proposed network was trained with σ_G varying from 0.1 to 0.3. Regarding the hardware-network simulation, we simulated 1000 realizations of the networks using different values of conductance's variation from 3 to 21%. A greater distance between conductance of HRS and LRS is preferred to allow the network to differentiate between weight values despite their variation. However, not all types of memristors achieve greater distances, thus we fixed $\hat{\mu_L} = 1$ and varied $\hat{\mu_H}$ from 0.1 to 0.5. Finally, we fixed $\sigma_H/\hat{\mu_H} = 21\%$ and varied $\sigma_L/\hat{\mu_L}$ from 3 to 40% [5], [11], because memristors provide different values of LRS and the variation of HRS is always greater than LRS.

During experimentation, all code was written in Python with the help of Tensorflow library for DNN coding. All testing was done using an Ubuntu server mounting an Intel Xeon W-2123 and 3 $\times$ NVIDIA GeForce RTX 2080 Ti.

A. Classification of MNIST

First, we experimented with the well-known MNIST handwritten digit classification problem. The QNN architecture used was 728–1000–1000–10 with 1.79 M trainable parameters, where the three hidden layers followed (1) and the last layer used a softmax activation. The model size was 0.43 MB, being 6.84 MB the equivalent size in float-32. Pixel data was scaled between 0 and 0.2. The categorical cross entropy was minimized until *earlystopping*, where a random 10% of the training data was used for validation.

B. Classification of real-world traffic

To evaluate our proposal against real-world data, we collected from the Performance Measurement System (PeMS) two years of speed and flow measures of sensor 1118421 (hereafter referred as RSU) installed on a south-bound section of Interstate 5 in California. 2015 data was used for training and 2016 for testing, with a total of 210501 5-minute samples. Similar to [15], we considered two Levels Of Service (LOS) based on flow and speed: free-flow and congested. We fitted a two-regime speed-flow model [16] to data (Fig. 2) and used it to set labels accordingly to the defined LOS. By using the model and visually inspecting the labels, most of the samples marked as congested matched the peak hours of the working days. This resulted in a binary classification problem with an imbalance ratio of 0.0572, where very few samples were labeled as congested. Thus, we weighted the loss function according to the class ratio to penalize errors on the minority class during training. Besides, we used a batch size of 2048 to ensure the batch had a chance of containing at least few congested samples. The classification problem to solve by the RSU consisted of using the last 3 hours of speed and flow measures to predict its own LOS in a 15-minute time horizon. The QNN architecture used was 36–50–50–1 with 4.35 k

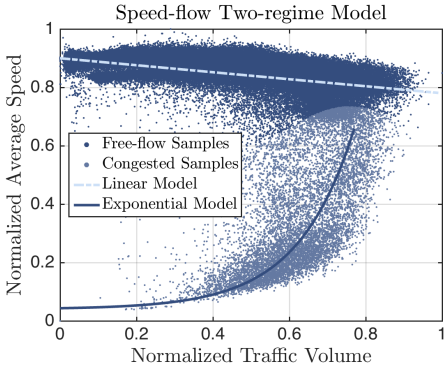

Fig. 2: Normalized to maximum speed and flow samples labeled by fitting a two-regime speed-flow model [16] as two traffic Levels Of Service (LOS): free-flow and congested.

trainable parameters, where the hidden layers followed (1) and the last layer used a sigmoid activation. The model size was 1.06 kB, being 0.02 MB the equivalent size in float-32. Speed data was scaled between 0 and 0.2 km/h. The binary cross entropy was minimized until *earlystopping*, where a random 10% of the training data was used for validation.

C. Results

To keep all measured information and for a clear statistical comparison of methods, we reported the Complementary Cumulative Distribution Function (CCDF), which we defined as the probability at device realization of the classification metric to be greater than n, i.e., $P(metric > n)$. For MNIST, we reported the classification accuracy in Fig. 3, 4 and 5. However, for PeMS, we reported the CCDF of the Area Under the Receiver Operating Curve (AUROC) in Fig. 4 and 5, because of class imbalance. Additionally, we defined the Improvement as the % difference on the obtained metric n when $P(metric > n) \rightarrow 1$ between our best performing proposal and the original network.

First, Fig. 3 shows the impact of device-to-device variation on classification performance for different values of conductance's variation and normalized distances between HRS and LRS. Distance of 0.7 was omitted as the results can be extrapolated from the other distances. The original accuracies on MNIST of the software-based networks (no convolution, no unsupervised pre-training and no data augmentation) were 97.88% and 98.10% for 0.5 and 0.9 distances, respectively. Fig. 3 clearly shows that performance becomes less reliable as the uncertainty of conductance increases. This is more accentuated when the distance between HRS and LRS is smaller. For example, there is a 60% chance that a realized network classifies erroneously more than 5% of the test images, when the variation is 40% and distance is 0.5. Besides, it shows that our network with ternary weights is quite robust against conductance variations of less than 20%, meaning that there is small margin of improvement below that. On PeMS data, the CCDF exhibits similar behavior but with less acute impact on the classification metric AUROC because it is a simpler binary classification problem with less learnable parameters. This motivates the need of weight variation mitigation techniques as the complexity of memristor-based networks will keep increasing to match the sate-of-the-art software architectures.

Improvements of our proposal are summarized in Fig. 5 and reference values used to calculate them are depicted using green crosses in Fig. 4. Improvements under 0.5% were omitted. As expected, the network using the proposed training scheme achieved greater improvements when there is a small distance between conductance of HRS and LRS and high variation, almost 10% when the variation is 40% and distance is 0.5 on MNIST data. Contrary, improvements on PeMS data are not that significant because there was less margin to improve, there was less impact of the weight variation on the classification metric as shown in Fig. 3. A more detailed view of the results is shown in Fig. 4 using the CCDF, which describes the expected behavior at realization. Fig. 4 shows that the CCDF of the proposal had a steeper slope compared to the original network, which means that there is less variability on the final performance of the realized network. Hence, we can conclude that our proposal is a valid scheme to mitigate the impact of memristors with small windows between LRS and HRS and high variability, as it improved quite significantly the classification performance. Besides, results suggested that training with $\sigma_G = 0.3$ rather than lower values of it is beneficial to achieve the best reliability. Nonetheless, using higher values than that derived in unstable training where networks cannot converge to an optimal solution, thus σ_G should be optimized as a hyper-parameter or much further evaluation is needed to derive the optimal one.

IV. Conclusion

In real memristor-based networks, there is a loss in performance due to device variability when the networks are realized. In that context, we designed a network architecture with quantized weights adapted to the design constrains of devices currently used in neuromorphic systems. At the same time, we proposed an offline training technique that incorporates the knowledge of the device variability due to conductance imperfections. We showed a mitigation of the impact of device-to-device variation and obtained improvements of up to 9.71% and 3.3% in MNIST and a road traffic classification problem, respectively. Hence, our proposal helped to achieve high reliability at network realization, which can be used to increase the expected performance of memristor-based networks or reduce the cost of production since networks that are more robust against variation can be implemented with low-cost devices. Future work will aim to extend the experiments to more complex architectures such as CNN and classification problems such as CIFAR-10 and regression.

References

[1] S. Choi, S. Ham, and G. Wang, "Memristor synapses for neuromorphic computing," in *Memristors-Circuits and Applications of Memristor Devices*. IntechOpen, 2019.

[2] E. Miranda and J. Suñé, "Memristors for neuromorphic circuits and artificial intelligence applications," 2020.

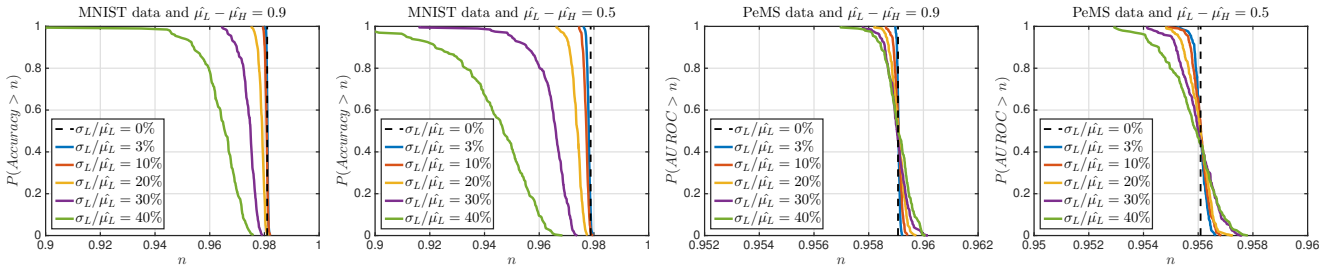

Fig. 3: Impact of conductance variation on the accuracy and AUROC of MNIST and PeMS binary classification, respectively. Two different plots of the empirical CCDF for 0.9 and 0.5 normalized distances between HRS and LRS.

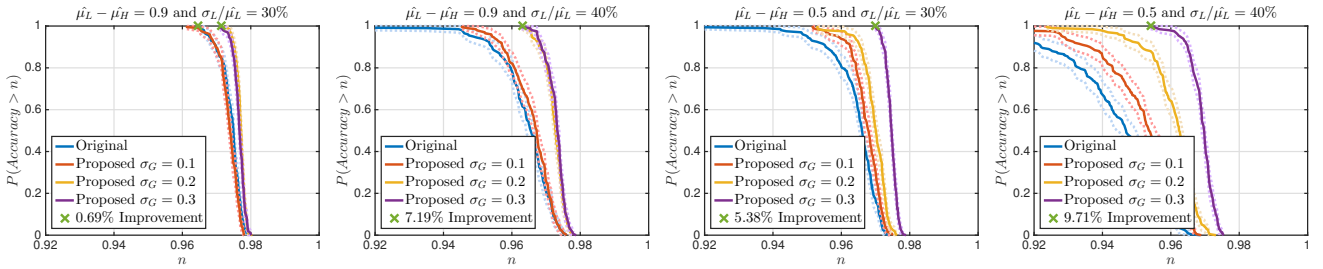

Fig. 4: Plots of the empirical CCDF and confidence bounds for MNIST classification. The proposed mitigation technique increases the accuracy of the original network at realization.

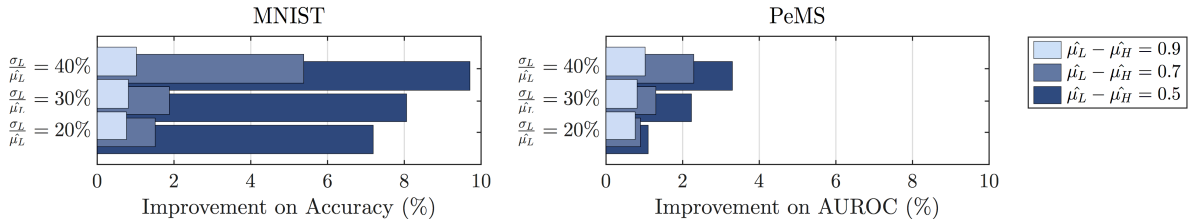

Fig. 5: Improvements of the proposed method on the accuracy and AUROC for MNIST and PeMS data, respectively. Improvements were computed when the probability of getting a certain minimum performance at device realization tends to 100%. $\hat{\mu_L} - \hat{\mu_H}$ is the normalized distance between LRS and HRS conductances. $\sigma_L/\hat{\mu_L}$ is the standard deviation of the LRS conductance, $\sigma_H/\hat{\mu_H}$ was fixed to 21%.

[3] S. Kim, H.-D. Kim, and S.-J. Choi, "Impact of Synaptic Device Variations on Classification Accuracy in a Binarized Neural Network," *Scientific reports*, vol. 9, no. 1, pp. 1–7, 2019.

[4] F. Alibart, E. Zamanidoost, and D. B. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," *Nature communications*, vol. 4, no. 1, pp. 1–7, 2013.

[5] M. E. Fouda, S. Lee, J. Lee, A. Eltawil, and F. Kurdahi, "Mask technique for fast and efficient training of binary resistive crossbar arrays," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 704–716, 2019.

[6] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti *et al.*, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

[7] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, p. 661, 2019.

[8] L. Ni, Z. Liu, H. Yu, and R. V. Joshi, "An energy-efficient digital reram-crossbar-based cnn with bitwise parallelism," *IEEE Journal on Exploratory solid-state computational devices and circuits*, vol. 3, pp. 37–46, 2017.

[9] E. Giacomin, T. Greenberg-Toledo, S. Kvatinsky, and P.-E. Gaillardon, "A robust digital rram-based convolutional block for low-power image processing and learning applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 2, pp. 643–654, 2018.

[10] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[11] A. Prakash, J. Park, J. Song, S. Lim, J. Park, J. Woo, E. Cha, and H. Hwang, "Multi-state resistance switching and variability analysis of hfo x based rram for ultra-high density memory applications," in *2015 International Symposium on Next-Generation Electronics (ISNE)*. IEEE, 2015, pp. 1–2.

[12] S. Kim, M. Lim, Y. Kim, H.-D. Kim, and S.-J. Choi, "Impact of synaptic device variations on pattern recognition accuracy in a hardware neural network," *Scientific reports*, vol. 8, no. 1, pp. 1–7, 2018.

[13] K. V. Pham, S. B. Tran, T. V. Nguyen, and K.-S. Min, "Asymmetrical training scheme of binary-memristor-crossbar-based neural networks for energy-efficient edge-computing nanoscale systems," *Micromachines*, vol. 10, no. 2, p. 141, 2019.

[14] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[15] T. Pamula, "Road traffic conditions classification based on multilevel filtering of image content using convolutional neural networks," *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 3, pp. 11–21, 2018.

[16] A. D. May, *Traffic flow fundamentals*, 1990.