Enabling Extreme Energy Efficiency Via Timing Speculation for Deep Neural Network Accelerators

Zhang, Jeff
jeffjunzhang@nyu.edu

Ghodsi, Zahra
ghodsi@nyu.edu

Rangineni, Kartheek
karthek@iitk.ac.in

Garg, Siddharth
sg175@nyu.edu

Abstract

Due to the success of deep neural networks (DNN) in achieving and surpassing state-of-the-art results for a range of machine learning applications, there is growing interest in the design of high-performance hardware accelerators for DNN execution. Further, as DNN hardware accelerators are increasingly being deployed in datacenters, accelerator power and energy efficiency have become key design metrics. In this paper, we seek to enhance the energy efficiency of high-performance systolic array based DNN accelerators, like the recently released Google TPU, using voltage underscaling based timing speculation, a powerful energy reduction technique that enables digital logic to execute below its nominal supply voltage. The design of MAC-Drop is motivated by the observation that conventional voltage underscaling techniques proposed in literature are not well-suited to large, high-performance DNN accelerators. MAC-Drop encompasses three new architectural innovations that enable aggressive voltage underscaling for DNN accelerators without compromising performance or classification accuracy. Our empirical results indicate that MAC-Drop enables between $32\% - 56\%$ *energy savings when evaluated over four state-of-the-art benchmark DNNs.*

1. Introduction

Deep neural networks (DNN) have achieved or surpassed state-of-the-art results in a wide range of machine learning applications, from image, video and text classification [1, 2, 3], to speech recognition [4] and language translation [5]. DNNs contain multiple layers of computation, where each layer multiplies inputs from the previous layer with a matrix of weights followed by a non-linear transformation such as a sigmoid or linear rectification. State-of-the-art DNNs have *millions* of parameters requiring *large* matrix multiplications (or convolutions) and are thus computationally challenging workloads.

Owing to their wide applicability and computational cost, there is growing interest in the design of special-purpose hardware accelerators for neural network training and inference [6, 7, 8, 9, 10, 11].

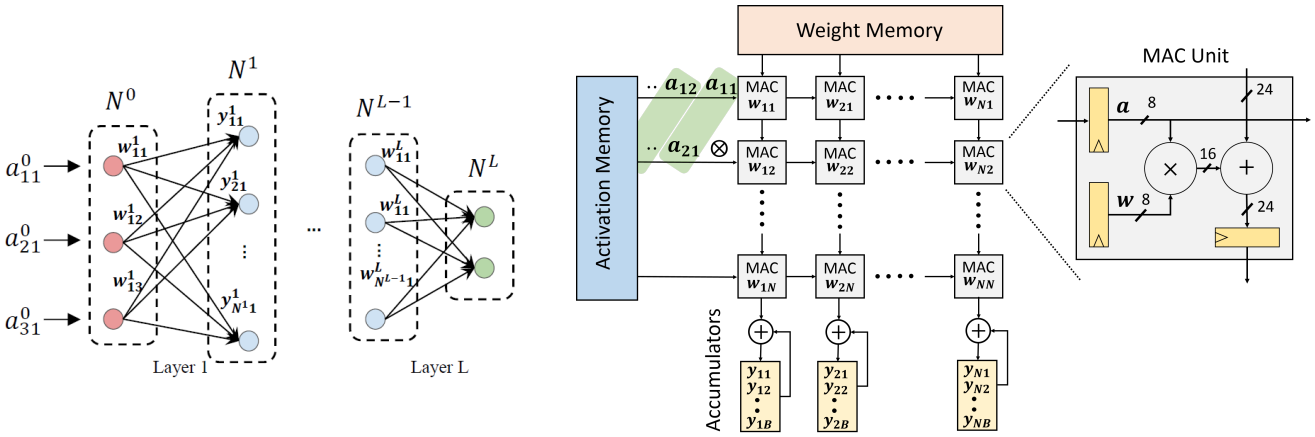
This paper focuses specifically on the design of hardware accelerators for DNN *inference*. While several different accelerator architectures have been proposed, *systolic array* based implementations are among the most promising [10, 12, 13]. Systolic arrays are 2-D grids of homogeneous processing elements (PE) that perform only nearest neighbour communication. Each PE receives data from its upstream neighbors, computes on the data and forwards the result to its downstream neighbors. Systolic arrays with multiply and accumulate (MAC) units as PEs can efficiently compute matrix multiplications and convolutions [14, 15, 16], and are thus well-suited for DNN acceleration.

As such, systolic arrays obviate the need for input buffering (since data arrive "just in time") and complex routing, thus enabling a hardware friendly implementation. Further, systolic arrays increase energy efficiency by amortizing frequent (and power-hungry) reads of weights and activation over multiple MAC operations. Harnessing these advantages, the recently released Google Tensor Processing Unit (TPU) uses a systolic array with a 256×256 grid of MAC units at its core and provides $30\times$-$80\times$ greater performance/Watt than CPU and GPU based servers at a raw throughput of 90 TOPS/second. As high performance DNN accelerators are increasingly deployed in data-centers, their energy efficiency (performance/Watt or performance/Joule) is a key metric that impacts data-center operational cost.

In this paper we seek to further enhance the energy efficiency of high-performance systolic array based DNN accelerators (like the Google TPU) with minimal impact on classification accuracy. We do so by leveraging voltage underscaling based timing speculation, a powerful energy saving technique that has previously been used to reduce general-purpose processor energy consumption by as much as 40% [17]. The premise underlying this technique is that worst-case timing critical paths in digital logic are *rarely* exercised, making it possible to run digital logic at supply voltages lower than the nominal voltage if timing errors can be dealt with.

Our work is motivated by the observation that state-of-the-art voltage underscaling techniques that have been successfully applied to general-purpose processors (and some other application domains) do *not* provide the same energy savings for systolic array based DNN accelerators. We briefly explain why this is the case below, and a provide a more detailed explanation along with empirical data in Section 2.1.

State-of-the-art mechanisms to deal with timing errors can be broadly grouped into two categories: (i) *timing error detection and recovery* (TED) [17, 18, 19, 20]; and (ii) *timing error propagation* (TEP) [21, 22]. TED detects timing errors by replacing conventional flip-flops in the design with those that can detect timing errors, for instance, Razor flip-flops [17] and recovers from errors by safely re-executing the offending input. TEP, on the other hand, allows errors to propagate to subsequent logic stages in the expectation that the algorithm



(a) Generic structure of a DNN consisting of multiple fully-connected layers.

(b) Baseline DNN accelerator architecture

Figure 1: (a) high-level overview of the structure of a DNN; (b) and it's execution on a systolic array based DNN accelerator.

itself is error resilient.

The fundamental challenge in applying TED to large systolic arrays is that re-executing even a single MAC operation that has a timing error disrupts the precisely orchestrated movement of data through the array and requires, in effect, all MACs to re-execute their inputs. For a TPU-like array with roughly 65K MACs, the probability that at least one out of the 65K MAC units has a timing error is high; even if each MAC unit experiences only 10^{-5} errors/clock, there is a roughly 50% chance that at least one MAC has a timing error in each clock cycle. The extra cycles required to recover from frequent timing errors incur a *large performance penalty.*

Compared to TED, TEP does not have a performance penalty, but causes the outputs of MAC units to be erroneous in the presence of timing errors. Unfortunately, timing errors for arithmetic operations typically occur in higher order bits, resulting in potentially large changes in the value of the partial sum computed by the MAC unit. We show, empirically, that because of the errors in the partial sum computation, the DNN classification accuracy drops significantly with TEP even for small timing error rates.

In this paper, we propose **MAC-Drop**, a new technique to recover from timing errors in large DNN accelerators Like TED, MAC-Drop detects timing errors using Razor flip-flops, but instead of re-executing the operation, steals a cycle from the succeeding MAC by *dropping* the succeeding MACs update to the partial sum. While MAC-Drop does introduce errors in the computation, we show empirically that the mean error in the network output is small, enabling execution at at timing error rates as high as 30% without compromising accuracy.

We have empirically evaluated MAC-Drop based on detailed post-synthesis timing simulations of a systolic array with 256×256 MAC units using four benchmark DNN architectures, including state-of-the-art speech and image recognition networks. Our experimental results show that MAC-Drop enables between 32%-56% energy savings with no loss in performance and within 1% reduction in classification accuracy relative to a baseline that executes at nominal voltage.

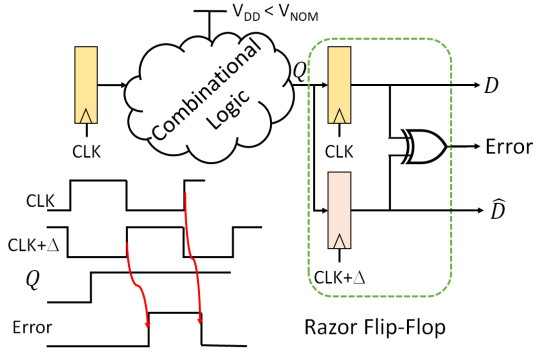


Figure 2: Timing error detection using the Razor flip-flop [17].

2. Voltage Underscaling Based Timing Speculation

Figure 2 illustrates the basic idea behind timing speculation. Conventional digital logic is run at a supply voltage such that the worst-case delay between any pair of flip-flops is smaller than the clock period. The combinational logic in Figure 2 runs at a voltage *lower* than its nominal voltage, reducing its power and energy consumption but increasing its delay. Therefore, the delay of some paths in the logic might *exceed* the clock period at the reduced supply voltage, causing timing errors. The timing error rate depends on the workload, since combinational logic delay is highly data dependent, and on factors such process variability and temperature [23].

Timing Error Detection and Recovery (TED) Ernst et al were the first to leverage the Razor architecture [17]. The

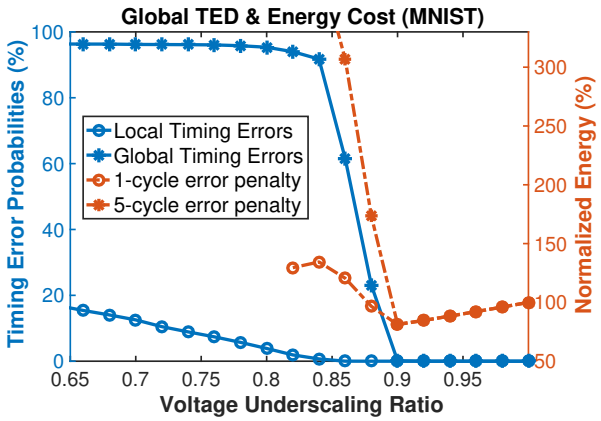


Figure 3: Timing error probabilities as a function of voltage underscaling ratio for the MNIST benchmark, and the corresponding energy cost for global TED for different timing error recovery penalties.

idea is to replace regular flip-flops in the design with Razor flip-flops, as shown in Figure 2 . A razor flip-flop samples its inputs twice, using the regular clock *and* a delayed clock and flags a timing error if the two outputs are different.

Razor proposes two recovery mechanisms when a timing error occurs. Global error recovery combines errors from each flip-flop into a single global error signal and gates the next clock edge if the global error signal is asserted. Global error recovery is conceptually simple and has only a single cycle error recovery penalty, but infeasible for large designs because it requires the global error signal and clock gating signals to be computed and routed in one clock cycle.

To address this issue, Razor also proposes a distributed error recovery mechanism called counterflow pipelining. In this technique, when a pipeline stage incurs an error, it re-executes its computation in the next clock cycle, inserts a bubble in downstream stages, and propagates a pipeline flush signal to upstream stages. Counterflow pipelining does not need a global error or stall signal, but has a higher recovery penalty of up to five clock cycles for a 5-stage MIPS processor. In general, the total execution time, N_{total} (in clock cycles) depends on the timing error rate, p_{err}, and the recovery penalty $N_{recovery}$ as follows $N_{total} = N_{exec} + p_{err} N_{recovery}$, where N_{exec} is the error-free execution time [23].

Timing Error Propagation (TEP) An alternative approach is to exploit algorithmic noise tolerance and simply allow timing errors to propagate to subsequent stages of computation, instead of re-executing inputs that cause errors [24, 21]. While these techniques have been shown to work for DSP applications like FIR filters, we show in this paper that naively applying TEP for DNN accelerators results in a significant drop in classification accuracy at relatively low timing error rates, motivating the design of MAC-Drop.

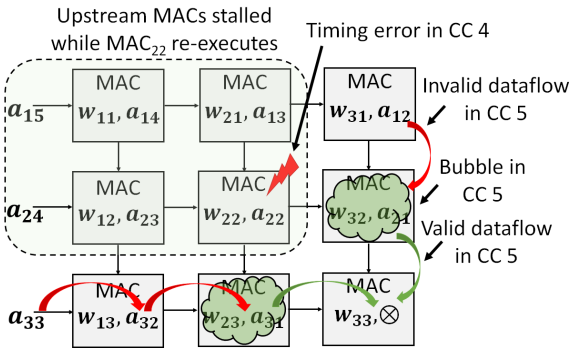


Figure 4: An illustration of the upstream and downstream dependencies in a distributed TED scheme.

2.1. Challenges with Using Existing Voltage Underscaling Techniques for DNN Acceleration

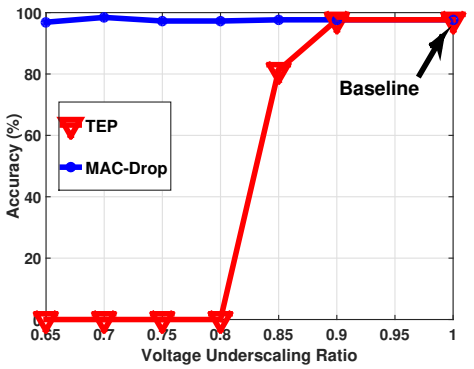
We motivate MAC-Drop by evaluating the state-of-the-art voltage underscaling based timing speculation techniques described in the prior section on systolic array based DNN accelerators. As we show, these techniques only work up to relatively small timing error rates, thus limiting the opportunities for aggressive voltage underscaling. We define the global timing error rate as the probability of any MAC unit in the systolic array being erroneous in a clock cycle, and the local timing error rate as the error rate per MAC unit. The *voltage underscaling ratio* (r), is defined as the ratio of the underscale voltage at which the circuit is running, over its nominal voltage.

Global TED Figure 3 plots the local timing error rate and the global timing error rate for an execution of the MNIST digit recognition network on a systolic array with 256×256 MACs as the function of the voltage underscaling ratio (see Section 4 for further details on experimental settings). Although the local timing error rate increases only gradually with voltage underscaling, the global error rate shoots up below a voltage underscaling ratio of $r = 0.9$.

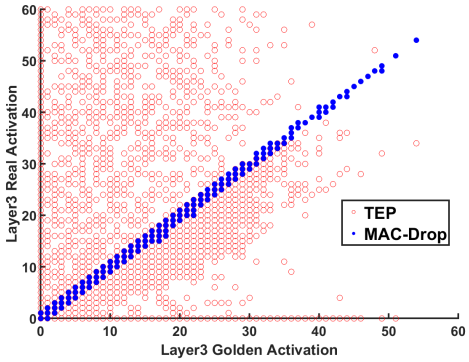
Figure 3 also plots the energy consumed by the accelerator assuming an error recovery penalty of $N_{recovery} = 1$ and $N_{recovery} = 5$ cycles. We assume that even a recovery penalty of five clock cycles is optimistic — computing the global error signal for a 256×256 array requires a 65K input OR gate and a globally routed clock gating signal.

Distributed TED Using counterflow pipelining as an exemplar [17], we show that even with distributed TED a timing error in any MAC causes all inputs to the systolic array, both upstream *and* downstream from the offending MAC, to stall. In other words, the performance of distributed TED is still dominated by the *global* and not per MAC error rate.

Figure 4 illustrates why this is the case using a 3×3 systolic array in which MAC_{22} incurs a timing error in clock cycle 4 (CC 4). In the next clock cycle, MAC_{22} re-executes its computation, pushes bubbles (or nops) to its downstream consumers



(a)



(b)

Figure 5: Classification accuracy of TEP as a function of voltage underscaling; and a scatter plot of the outputs of Layer 3 of the MNIST network with TEP versus the error-free outputs. Also shown as motivation are the corresponding data for our proposed scheme, MAC-Drop.

(MAC_{23} and MAC_{32}), and we optimistically assume that all upstream MACs are stalled.

Now, in CC 5, downstream MAC_{33} can indeed continue operating, but MAC_{13} and MAC_{31} must also stall. Consequently, the downstream activation input a_{33} must stall as well. Most generally, when a single timing error occurs in any MAC in a systolic array, all MACs in the same and prior rows and columns, and importantly, all inputs to the array must (eventually) stall to maintain synchrony. Consequently, the performance of a distributed TED technique like counterflow pipelining can be no better than an idealized global TED.

TEP Figure 5a shows the classification accuracy for the MNIST benchmark as a function of the voltage underscaling ratio when errors are allowed to propagate through the systolic array. We note that as soon as timing errors start appearing, i.e., below a voltage underscaling ratio of $r = 0.9$, the classification accuracy for TEP drops sharply. Figure 5b plots the error-free (golden) activations for Layer 3 of the MNIST network versus the activations obtained from TEP for a voltage underscaling ratio of $r = 0.85$. Observe that the differences between the golden and TEP activations are *large*, with a mean absolute error of 564 times than our proposed scheme.

Figure 5a and Figure 5b also show the corresponding data for MAC-Drop, our proposed scheme for dealing with timing errors. Note that the classification accuracy is the same as the baseline even with aggressive voltage underscaling. We will describe MAC-Drop in more detail in Section 3.

3. MAC-Drop

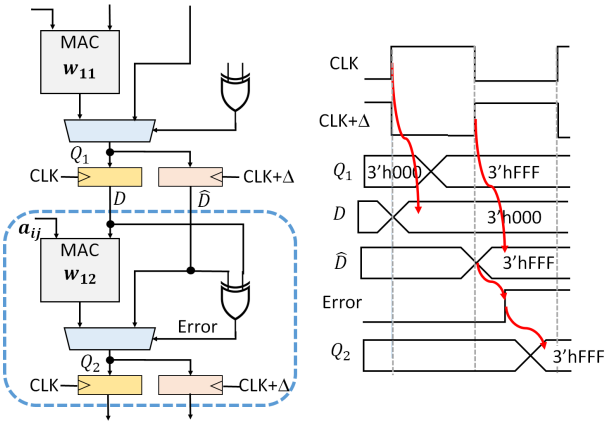


Figure 6: A block-level diagram illustrating MAC-Drop.

At the heart of the proposed architecture is MAC-Drop, a new technique to deal with timing errors in MAC units. Like TED, MAC-Drop instruments MAC-Drop with Razor flip-flops to detect timing errors, but recovers from timing errors *without* re-executing erroneous MAC operations. MAC-Drop builds on the observation that the distribution of weights in DNNs (and CNNs) is biased towards small values [25] (we also validate this observation in our own experiments); hence, the contribution of each individual MAC operation to the output of a neuron is small.

When a MAC incurs a timing error, MAC-Drop steals the next clock cycle from its successor MAC to correctly finish its own update to the partial sum, and *bypass* (or drops) the successor MACs update.

As shown in Figure 6, MAC-Drop requires minimal hardware changes. In addition to Razor flip-flops, MAC-Drop adds a multiplexor (MUX) controlled by the error signal from the prior MAC unit. If the previous MAC unit incurs an error, the MUX forwards the previous MAC's correctly computed partial sum (obtained from Razor's shadow flip-flop) to the next MAC unit; if not, the current MAC unit updates the partial sum as it normally would and forwards to the next MAC.

Figure 6 illustrates MAC-Drop's operation with a timing diagram. As in prior work, we assume that the shadow clock is delayed by 50% of the clock period, after which the error signal and correct partial sum from the prior MAC become available. Note that, although not shown in Figure 6, the error signal is obtained by OR-ing the error outputs of each

of individual Razor flip-flop at the output of the MAC. This requires at worst a 24-input OR gate, although, as we note in Section 4, only a fraction of the MAC units outputs need to be protected using Razor flip-flops. For correct operation, the error signal computation and propagation through the MUX must complete within the remaining half clock cycle. This constraint is easily met; in our implementation, error signal computation and MUX propagation consume only 16.7% and 2% of the clock period.

MAC-Drop can be viewed as randomly dropping a fraction of connection between input and output neurons, where the randomness arises from timing errors. Interestingly, randomly dropping neurons and connections during training are commonly used techniques, referred to as dropout [26] and dropconnect [27], respectively, that prevent overfitting and help reduce generalization error.

We note that MAC-Drop is unrelated to zero-skipping, a technique proposed in prior literature that skips over MAC units whose activation inputs are zero to save energy [7, 28]. In fact, MAC-Drop can be applied on top of zero-skipping to achieve further energy savings. We defer an exploration of the combination of these ideas to future work.

4. Experimental Setup

We now describe the experimental setup used to evaluate MAC-Drop. This includes the DNN benchmarks used, the details of the baseline hardware platform that we model and our simulation methodology.

Benchmarks. We evaluate MAC-Drop on four benchmarks, MNIST digit classification, Reuters text categorization [29], TIMIT speech recognition [30] and image recognition using the ImageNet dataset [31]. For the first three benchmarks, we trained multi-layer perceptrons (MLPs), i.e., DNNs with only fully connected (FC) layers, from scratch. For ImageNet classification, we used the pre-trained AlexNet CNN from PyTorch torchvision package [32] based on [33] with five convolutional layers. We note that fully connected MLPs form more than 66% of Google's datacenter workload [10].

In order to prepare the DNNs for execution on MAC-Drop, we quantized the weights and activations to 8-bit fixed point values. The number of bits dedicated to the integer and fractional parts are determined for each layer separately based on the minimum and maximum weight and activation values for that layer.

MAC-Drop Hardware Parameters Our baseline DNN accelerator is modeled around the Google TPU, pictured in Figure 1b. We model a systolic array with a grid of 256×256 MACs, with 8-bit weights and activations, and an adder with a 24-bit output. We implemented a prototype of the systolic array in synthesizable Verilog, and synthesized it with OSU FreePDK 45 *nm* standard cell library using the Cadence Genus synthesis tool. The systolic array synthesizes at 658 MHz at a nominal supply voltage of 1.1 V and consumes 19.7 W of dynamic power. We note that the actual TPU was implemented in a 28 *nm* process and will consequently have lower power consumption.

We note that although MAC-Drop only performs voltage underscaling on the systolic array, the array consumes a *significant* fraction of our baseline chip's power. We estimate that the systolic array consumes between $50\% - 80\%$ of the chip's dynamic power consumption (ignoring peripherals).

Simulation Methodology We scheduled our benchmark DNNs on the systolic array using the mapping described in [34, 35, 36]. Output neurons and output channels map to columns in the array, and input neurons and input channels to rows. The three MLPs are run with a batch size of 256 inputs, while for AlexNet each batch has a single image. However, a single input image corresponds to an *effective* batch size of 3025 for the input convolutional layer and 169 for the output convolutional layer. AlexNet's FC layers have an effective batch size of 1 and have only 0.3% utilization of the systolic array. Thus, we assume that MAC-Drop is only used to accelerate AlexNet's convolutional layers, which make up more than 90% of its computational cost.

To obtain dynamic delay data, we use the synthesized systolic array netlist along with the standard delay file (SDF) generated by the synthesis tool to run detailed gate level timing simulations using ModelSim, and record the delay of *each* MAC unit for each clock cycle in which it is utilized. We post-process the simulated delay values using a script that emulates the execution of MAC-Drop, injects timing errors and outputs the resulting classification accuracy.

For the three MLPs, we executed 7 batches of 256 inputs each. One batch was used as a validation data (to determine voltage underscaling levels), and the remaining are used as test data to get the final classification accuracy. For AlexNet, we simulated 256 images of which 8 are used as validation and the remaining are used as test. We note that the primary constraint in using more validation/test data is computational time: running detailed timing simulations for a *single* batch of 256 inputs for TIMIT takes 8 hours using 24 parallel threads on a 40-core Intel Xeon server. Simulating a single image for AlexNet takes roughly the same amount of time.

5. Experimental Results

We evaluate two variants of MAC-Drop: **static voltage underscaling**, where each layer executes at the same voltage underscaling ratio, and **dynamic voltage underscaling**, where each layer operates at its own optimal voltage underscaling ratio.

Figure 7 shows the classification accuracy versus the total energy consumption (relative to baseline) for TIMIT obtained for the static and dynamic variants of MAC-Drop on the validation data. Note that for both techniques, we can obtain large reductions in energy consumption *without* compromising classification accuracy, even in the presence of high timing

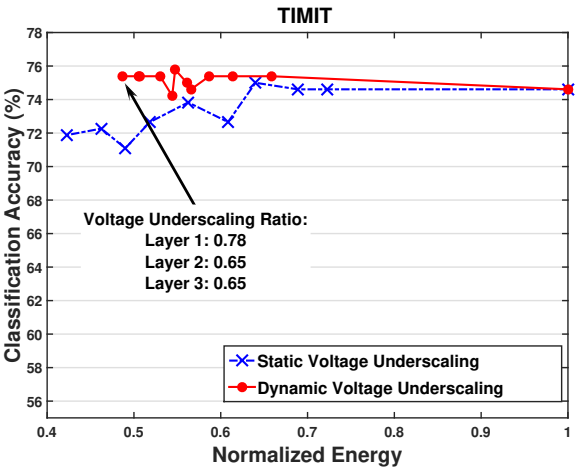


Figure 7: Classification accuracy vs. total energy consumption for TIMIT on the validation dataset using both static and dynamic voltage underscaling.

error rates. Because the dynamic technique is able to evenly distribute timing errors across both layers, it enables more aggressive voltage underscaling than the static technique.

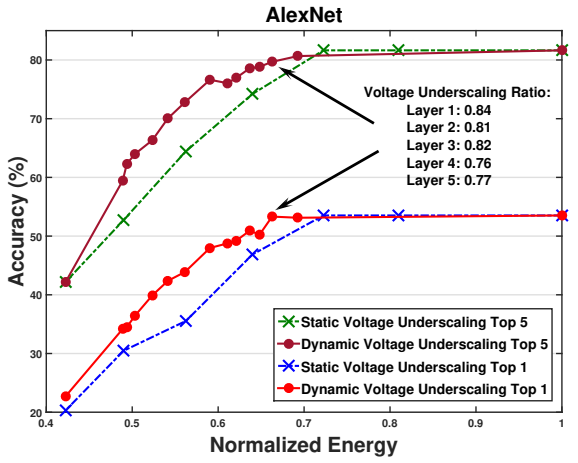


Figure 8: Classification accuracy vs. total energy consumption for AlexNet on the validation dataset using both static and dynamic voltage underscaling.

Figure 8 plots the same data as above for AlexNet. As is conventional, we report both top-1 and top-5 accuracies for AlexNet. Note, as before, that dynamic voltage underscaling technique provides higher classification accuracy for the same energy savings, or conversely greater energy savings for the same classification accuracy compared to static voltage underscaling.

However, we note that compared to TIMIT, AlexNet's classification accuracy drops more markedly with either static or dynamic voltage underscaling. This is in part because AlexNet has 1000 output classes while TIMIT has only 138. Hence, for AlexNet, the distance between the prediction probabilities for the top two classes will be lower, making it more susceptible to noise.

6. Conclusion and Future Work

In this paper, we have presented MAC-Drop, architectural support for enabling aggressive voltage underscaling of high-performance DNN accelerators like the Google TPU without compromising performance or classification accuracy. MAC-Drop provides between $32\% - 56\%$ energy savings with at most 1% drop in classification accuracy and no loss in performance.

As future work, we would like to explore the synergy between MAC-Drop and techniques such as time borrowing and approximate or reduced precision MAC units. Further, given the appealing analogy between MAC-Drop and neural network regularization techniques like dropconnect, we would like to explore the use of MAC-Drop in accelerating DNN training.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[3] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering*, 18(10):1338–1351, 2006.

[4] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[5] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[6] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 247–257. ACM, 2010.

[7] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[8] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[9] Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto, and Hans Peter Graf. A massively parallel coprocessor for convolutional neural networks. In *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, pages 53–60. IEEE, 2009.

[10] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. *arXiv preprint arXiv:1704.04760*, 2017.

[11] Seongwook Park, Kyeongryeol Bong, Dongjoo Shin, Jinmook Lee, Sungpill Choi, and Hoi-Jun Yoo. 4.6 a1. 93tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[12] Hsiang-Tsung Kung. Why systolic architectures? *IEEE computer*, 15(1):37–46, 1982.

[13] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 29:1–29:6, New York, NY, USA, 2017. ACM.

[14] HT Kung and Charles E Leiserson. Systolic arrays (for vlsi). In *Sparse Matrix Proceedings 1978*, volume 1, pages 256–282. SIAM, 1979.

[15] Guo-Jie Li et al. The design of optimal systolic arrays. *IEEE Transactions on Computers*, 100(1):66–77, 1985.

[16] HT Kung and SW Song. A systolic 2-d convolution chip. Technical report, Carnegie-Mellon University, 1981.

[17] Dan Ernst, Shidhartha Das, Seokwoo Lee, David Blaauw, Todd Austin, Trevor Mudge, Nam Sung Kim, and Krisztián Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24(6):10–20, 2004.

[18] Shidhartha Das, Carlos Tokunaga, Sanjay Pant, Wei-Hsiang Ma, Sudherssen Kalaiselvan, Kevin Lai, David M Bull, and David T Blaauw. Razorii: In situ error detection and correction for pvt and ser tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):32–48, 2009.

[19] B. Greskamp, L. Wan, U. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen, and C. Zilles. Blueshift: Designing processors for timing speculation from the ground up. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pages 213–224, Feb 2009.

[20] Matthew Fojtik, David Fick, Yejoong Kim, Nathaniel Pinckney, David Harris, David Blaauw, and Dennis Sylvester. Bubble razor: An architecture-independent approach to timing-error detection and correction. In *International Solid-State Circuits Conference*, pages 488–490. IEEE, 2012.

[21] Paul N Whatmough, Shidhartha Das, David M Bull, and Izzat Darwazeh. Circuit-level timing error tolerance for low-power dsp filters and transforms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(6):989–999, 2013.

[22] Paul N Whatmough, Sae Kyu Lee, Hyunkwang Lee, Saketh Rama, David Brooks, and Gu-Yeon Wei. 14.3 a 28nm soc with a 1.2 ghz 568nj/prediction sparse deep-neural-network engine with> 0.1 timing error rate tolerance for iot applications. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 242–243. IEEE, 2017.

[23] Marc De Kruijf, Shuou Nomura, and Karthikeyan Sankaralingam. A unified model for timing speculation: Evaluating the impact of technology scaling, cmos design style, and fault recovery mechanism. In *International Conference on Dependable Systems and Networks*, pages 487–496. IEEE, 2010.

[24] Rajamohana Hegde and Naresh R Shanbhag. Soft digital signal processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):813–823, 2001.

[25] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[26] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[27] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1058–1066, 2013.

[28] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 267–278. IEEE Press, 2016.

[29] Ana Margarida de Jesus Cardoso Cachopo. Improving methods for single-label text categorization. *Instituto Superior Técnico, Portugal*, 2007.

[30] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[31] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[32] PyTorch torchvision models. http://pytorch.org/docs/0.2.0/torchvision/models.html.

[33] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[34] J. Ross, N.P. Jouppi, A.E. Phelps, R.C. Young, T. Norrie, G.M. Thorson, and D. Luu. Neural network processor, November 24 2016. US Patent App. 14/844,524.

[35] CA US) Ross, Jonathan (Mountain View. Prefetching weights for use in a neural network processor, April 2017.

[36] CA US) Phelps Andrew Everett (Middleton WI US) Ross, Jonathan (Mountain View. Computing convolutions using a neural network processor, April 2017.