ACOUSTIC: Accelerating Convolutional Neural Networks through Or-Unipolar Skipped Stochastic Computing

Wojciech Romaszkan, Tianmu Li, Tristan Melton, Sudhakar Pamarti, Puneet Gupta
Department of Electrical and Computer Engineering
University of California, Los Angeles
Los Angeles, USA
{wromaszkan, litianmu1995, tristanmelton, spamarti, puneetg}@ucla.edu

***Abstract*—As privacy and latency requirements force a move towards edge Machine Learning inference, resource constrained devices are struggling to cope with large and computationally complex models. For Convolutional Neural Networks, those limitations can be overcome by taking advantage of enormous data reuse opportunities and amenability to reduced precision. To do that however, a level of compute density unattainable for conventional binary arithmetic is required. Stochastic Computing can deliver such density, but it has not lived up to its full potential because of multiple underlying precision issues. We present ACOUSTIC: Accelerating Convolutions through Or-Unipolar Skipped sTochastIc Computing, an accelerator framework that enables fully stochastic, high-density CNN inference. Leveraging split-unipolar representation, OR-based accumulation and novel computation-skipping approach, ACOUSTIC delivers server-class parallelism within a mobile area and power budget - a 12mm^2 accelerator can be as much as 38.7x more energy efficient and 72.5x faster than conventional fixed-point accelerators. It can also be up to 79.6x more energy efficient than state-of-the-art stochastic accelerators. At the lower-end ACOUSTIC achieves 8x-120X inference throughput improvement with similar energy and area when compared to recent mixed-signal/neuromorphic accelerators.**

***Index Terms*—Neural nets, hardware architecture**

I. Introduction

Energy, area and latency optimization of deep learning accelerators is especially important for resource-constrained edge devices, which are taking on the increasing share of inference workloads due to privacy, energy and latency concerns. Low precision [1–5] and data reuse [6, 7] are crucial for reducing the number of memory accesses, on- and off-chip, which are the major contributors to overall system energy consumption [8]. Various hardware architectures to leverage low-precision (e.g., bit serial [9, 10]) have also been proposed.

Due to unparalleled levels of compute density it provides, stochastic computing (SC) can be an excellent candidate for convolutional neural network (CNN) acceleration where low bit precision (roughly equivalent to 8bit fixed point) is enough [11–15]. However, accuracy degradation in addition operation, forces early conversion or even abandoning stochastic accumulation altogether, reducing SC domain to just multiplication [11–13]. Further, most prior SC works develop network-specific ASICs (e.g., [12]) rather than programmable accelerators.

In this paper we present ACOUSTIC - Accelerating Convolutional neural networks through Or-Unipolar Skipped sTochastIc Computing, a scalable, programmable and fully-stochastic CNN accelerator framework. ACOUSTIC integrates multiple algorithm and architecture optimizations that allow us to harness full benefits of SC:

- ***Optimization of SC primitives for deep neural networks.*** We develop a temporally processed *split-unipolar stochastic representation* that enables 2X+ shorter streams. Second, we enable practical stochastic OR-based accumulation which has unique scale-free saturating addition properties. This reduces MAC size by 4X-20X while retaining comparable accuracy.
- ***Computation skipping based stochastic average pooling*** that can deliver latency and energy reduction proportional to the size of pooling window (4X-9X) on the convolutional layer itself.
- A ***programmable stochastic CNN accelerator*** architecture: ACOUSTIC, built to harness SC compute density to maximize activation and weight reuse, while minimizing or completely removing partial sums to dramatically reduce conversion overheads and number of memory accesses required.
- We develop ***training optimization*** to model the peculiarities of ACOUSTIC and speed up stochastic stream-based CNN training for ACOUSTIC by almost 10X.

The result is that ACOUSTIC can deliver real time inference performance at ultra-low energies (4ms/0.4mJ per image using AlexNet on Imagenet with batch size of 1) and small area (12mm^2) making it very suitable for learning at the edge.

II. ACOUSTIC Optimizations for CNNs

SC due to its number representation (proportion of 1's in a random bit stream) allows multiplication and addition operations using single gates [16, 17] but suffers from several issues that we address in ACOUSTIC.

A. Split Unipolar Representation

Stochastic computing offers two alternative number representation formats: unipolar and bipolar [1]. For the former, each bit in the stream has possibility v of being one, and for the latter the possibility is $(v+1)/2$, where v is the value being represented. In neural networks, maintaining high accuracy mandates using weights with both positive and negative values, which makes bipolar representation the most common choice when implementing SC-based accelerators [11, 12, 15]. However, umipolar requires at least *2X shorter streams* than bipolar for same representational error. RMS error of unipolar and bipolar SC representations can be calculated as $\frac{\sqrt{nv(1-v)}}{n} = \sqrt{\frac{v(1-v)}{n}}$ and $\sqrt{\frac{1-v^2}{n_b}}$ respectively, where n is the length of the bit stream.

[1]Alternate representations have been proposed but have not been popular due to larger error [18] or larger area [19].

We develop a *split-unipolar representation* which uses two streams to represent each weight, one for the positive and one for the negative component. For a positive weight value, its corresponding negative stream is 0, and vice-versa. Because activations (inputs) of a neural network layer are typically non-negative due to ReLU activation function in the previous layer, they can be represented using only a single positive stream. The activation streams are multiplied and accumulated separately with positive and negative weight components using up counters (in ACOUSTIC architecture, as described later, layer activations are converted to binary and stored in memory), whose values are then subtracted from each other to obtain final result. Since the counter output is in fixed-point binary domain, ReLU activation is easily implemented as a bitwise AND of the inverted sign with every other bit [2].

Split-unipolar computation can be realized in hardware using temporal unrolling with just a single MAC array, where computation is done in two phases. In the first phase, all negative weights, and by extension their respective multipliers, are gated using their sign. This means only results corresponding to positive weights are being accumulated and the output counters are counting up. In the second phase, the mask is inverted, only negative weights contribute to the outputs, and counters count down. A simple example of a 2-wide MAC with one positive and one negative weight and stream length of 8 is shown on Figure 1. Spatial unrolling (similar idea has been recently independently proposed by [20]) simply doubles the compute arrays (i.e., 50% utilization) but we do not further explore as our target is resource-constrained devices. Split unipolar SC results in overall

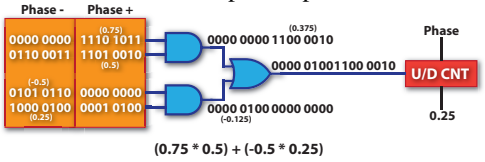

Fig. 1. Circuit level support for split-unipolar representation.

smaller energy/latency due to $> 2x$ shorter bitstreams [3] and allows for more accurate OR-based accumulation described next.

B. OR-based Scaling-Free Accumulation

One of the main selling points of SC is that computation can be performed using bit-wise operations between two input bit streams. An AND gate performs multiplication: $AND(v_1,v_2) = v_1 \times v_2$, where v_1, v_2 are the input possibilities for two unipolar streams. Similarly, a 2:1 multiplexer (MUX) can be used to compute $MUX(v_1,v_2,s) = s \times v_1 + (1-s) \times v_2$, i.e., scaled addition between two input streams, where s is the select input. Thus, MUX can act as a stochastic adder by using a 50% random stream at the select input. However, MUX-based addition degrades accuracy of computation (primarily due to the scaling factor), especially when wide accumulation is performed. Since neural networks generally perform very large matrix multiplications, prior work in SC-based neural network acceleration was often forced to perform accumulation in binary domain, by either using costly parallel counters [12] or converting the results back to binary domain after every multiplication [21].

We use OR-based stochastic accumulation [22]. It is scaling-free (important for very wide accumulations in deep CNNs), has reasonable accuracy for unipolar streams (a monte-carlo analysis of 3 × 3 × 256 = 2304 wide accumulation reveals OR having 8x *less* absolute error than MUX-based accumulation) and is also much more compact (4.2x than [12] and 23.8X than [21] for a 128 wide accumulate) than alternative accumulation methods. However, a two-input OR, the result is equal to $v1 + v2 - v1v2$ instead of $v1 + v2$. We show later how we address this imperfect accumulation in training of the networks.

C. Computation Skipping for Stochastic Average Pooling

Although max-pooling is commonly used in CNN for dimensionality reduction, it has to be implemented as a FSM in SC [12, 23]. As a result of it can be 2X more expensive in area/power than average pooling [24]. Average pooling simply averages the activations in the pooling window (usually 2x2 or 3x3) to generate the output activations of a layer. In SC, this can be simply accomplished by a MUX (i.e., scaled addition in SC). Accuracy difference between the two styles of pooling is minimal (<0.3% for a small CNN for CIFAR10 as well as AlexNet on ImageNet dataset) Here we make an interesting observation that to get the required output value, the select signal does not need to be random as long as the inputs are random and independent from each other. Since we know the bits the multiplexer "chooses" a priori, we can skip computation for all other inputs' bits. Instead of passing multiple streams through the pooling multiplexer we concatenate shorter streams, either in stochastic or fixed-point binary domain. This allows us to *reduce the computation required by the convolutional layer preceding a pooling operator by 4x to 9x*, depending on the pooling window size. The area overhead of supporting computation skipping is minimal - it increases the size of an individual activation counter by 2.7% to 8.7%, depending on the pooling window size, which is $< 1\%$ of the overall accelerator area. The issue with the computation skipping scheme is that the results are correlated, thus necessitating randomization of outputs. However, ACOUSTIC architecture converts the streams to binary after each layer (and regenerates random sequences for the next layer), completely removing the correlation problem.

D. Network Training Enhancements for ACOUSTIC

As mentioned in Section II-B, our use of split unipolar streams enables OR-based accurate accumulation. Though it has a systematic error (it computes $v1 + v2 - v1v2$), it is well-defined and therefore can be taken into account by replacing all additions with OR-addition during training of a neural network. This in turn, requires multiplications in the neural network to be performed explicitly while training, and also slows down addition during both forward and backward pass (∼15X longer training runtime). To speed up training, we approximate the effect of OR addition using the following function:

$$OR(a_1,a_2,...,a_n) \approx 1 - \prod_{i=1}^{n}(1-\frac{s}{n}) \approx 1 - e^{-s} \qquad (1)$$

where s is the sum of inputs. This approximates OR-addition (approximation error $< 5\%$ as extracted from actual training runs) by adding an activation function after normal network layer and therefore offers 10X+ speedup in training using SC streams.

[2]Other activation functions require FSM implementations [12, 15] and we do not explore them here.

[3]For rest of the paper, our stream length refers to temporal unrolled split unipolar stream length. I.e., 256 long stream implies 128×2.

III. ACOUSTIC Architecture

In this section, we motivate and develop the ACOUSTIC accelerator architecture.

A. Understanding SC Benefits

Despite single gate operations, SC does not offer any energy advantage even with random number generator (RNG) sharing across multiple stochastic number generators (SNGs), as is common practice, due to conversion overheads and long stream lengths required to achieve equivalent fixed-point precision. The conversion overheads can be amortized by input (across multiple convolution filters in a CNN layer) and output (by reducing partial sums) reuse but our experiments using TSMC 28nm library and LFSR-based SNGs indicate that energy benefits are modest. Where SC excels is in compute density. With input and output reuse, SC MACs can be 47X smaller than 8-bit fixed-point MACs, enabling much higher levels of parallelism (which can be effectively leveraged in CNNs). While high compute density directly improves system cost through reduced area footprint, it is much more valuable as a means of reducing overall energy consumption, and inference latency by reducing DRAM and on-chip memory accesses (which account for bulk of energy in neural network compute [25, 26]). To leverage this density however, communication-related overheads, like wiring and network-on-chip (NoC) have to be minimized, which is non-trivial for single-gate compute units.

This conclusion leads us to two important dataflow considerations for SC-based accelerator design: (1) maximizing reuse through parallelism and avoiding partial sums; and (2) reducing wiring complexity to maximize density. Unlike conventional fixed-point binary accelerators which rely on tightly coupling compute units with small, local scratchpads for cheap intermediate value buffering [6, 25, 26], ACOUSTIC leverages much larger available compute density to eliminate partial sums and amortize SC to binary conversion costs (otherwise partial sums will either need to be stored as expensive unrolled streams or be converted to binary). Similarly, to not impact the compute density and to not route long bistreams, sophisticated network on chip approaches (as used by [6, 26, 27]) are not used (recall that ACOUSTIC will have 50X more processing elements than conventional architectures).

B. Accelerator Architecture

ACOUSTIC (see Figure 2) uses external control interface to write the program representing a given network model into instruction memory and enable the Dispatcher control module to execute that program (details in section III-C). The direct memory access (DMA) controller is responsible for loading initial activations (inputs) and weights for each layer to on-chip memories and storing final outputs back in DRAM. There are two types of on-chip memories: weight buffers and activation scratchpads. To reduce the reliance on external DRAM bandwidth, ACOUSTIC overlaps computation with weight loading phases, i.e. fetching weights for the next layer while computing the current one.

There are three activation scratchpads, corresponding to three MAC columns, to provide optimal support for the commonly used 3x3 convolutional kernels. Each of the MAC columns, described in detail below, has a corresponding set of activation and weight SNGs and buffers, which are loaded with fixed-point binary values and generate stochastic sequences of a given

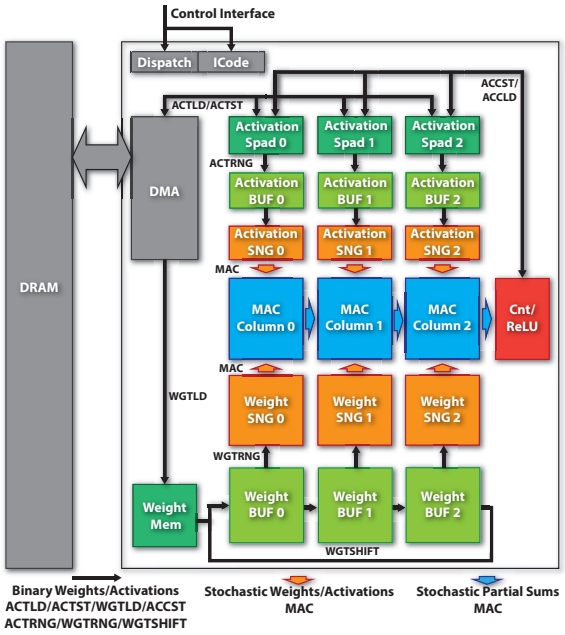

Fig. 2. ACOUSTIC accelerator overview.

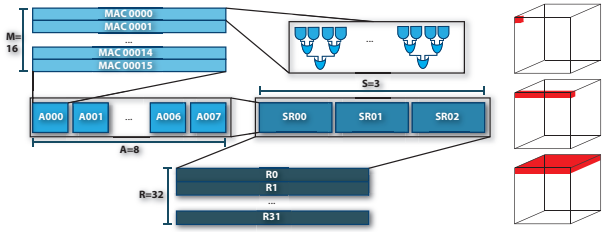

Fig. 3. Hierarchical organization of the SC MAC array, with the outputs generated at corresponding levels. Rows span across all three columns.

length. Corresponding partial sums generated by MAC columns are then accumulated together and sent to the activation counter modules which convert final results back to fixed-point binary representation and perform ReLU activation. Results in the activation counter blocks are written back to one of the activation scratchpads as inputs for the next layer.

ACOUSTIC compute engine has a highly hierarchical organization, shown on Figure 3, to balance parallelism with flexibility. Fixed, 96-wide multiply-accumulate units perform reduction before configurable fabric decides which partial sums map onto which outputs. Those 96:1 MAC units are the basic building blocks of our compute engine. A group of M=16 MACs with partially shared inputs and shared weights forms a MAC array. A=8 arrays form a sub-row, which shares a single activation scratchpad, and S=3 sub-rows together form a row, which corresponds to a single kernel. There are R=32 rows, which means 32 kernels can be computed in parallel, using the same activations.

Though we omit detailed explanations of how computation is mapped onto the ACOUSTIC engine for brevity reasons, ACOUSTIC can support kernel sizes from 1x1 to 3x3 and larger kernels using multiple computational passes with activation reloading in between. Padding (as is common at the edge of inputs in CNN) is supported with the help of a low-overhead shifting fabric which is shared across all rows. Though ACOUSTIC can perform strided convolutions by underutilizing the compute fabric, the preferred method of dimensionality reduction is pooling.

For pooling across output height, ACOUSTIC exploits compu-

tation skipping by using proportionally shorter streams and not resetting output counters between successive computation phases. As a result, outputs that fall under the same pooling window are averaged together using the scaled addition property of stochastic stream concatenation. This means each individual compute pass is shortened proportionally to the pooling size. Pooling across output width is done by output counters. Output counter with pooling support have small (2x-3x) parallel counters before them, which allows them to accumulate adjacent outputs that fall under the same pooling window. This again allows us to shorten the streams proportionally to the pooling window width. As explained in Section II-C the area overhead of this solution is minimal.

ACOUSTIC supports fully-connected (FC) layers in the most straightforward manner possible. Since FC layers cannot re-use weights without employing batching, ACOUSTIC cannot capitalize on weight re-use within an array. This means that only a single MAC in a given array can be used. However, if the fully-connected kernel is extended across 6 successive rows, their collective arrays can cover the whole 512 inputs with individual weights. The corresponding outputs of those rows need to be then accumulated together, which is supported by the ACOUSTIC fabric. While this is highly unoptimized, and leads to a 87.5% underutilization we argue that there is not much point in further optimizing the FC performance as newer CNN architectures like ResNet or Inception rely on a single, relatively small FC layer, which has very negligible impact on overall performance [28–31].

Given the sheer number of multiply accumulate units present in our architecture and its reliance on an inflexible connectivity fabric, non-ideal resource utilization is unavoidable. We do not consider this a major issue of our architecture, for two reasons. First, even with 50% or lower utilization, the effective number of multiply accumulate units is still on the order of hundreds of thousands. Second, unused MACs and SNGs do not contribute to dynamic energy consumption - because their input values are zeroes, AND-based multipliers perform operand gating, removing any switching propagation.

C. Control in ACOUSTIC

We opted for a distributed control scheme to keep individual control modules simple, while enabling overlapping different phases together to reduce overall latency. We have developed a restricted instruction set shown in Table I. The main control unit is the Dispatcher which reads the instructions, distributes them to other control units, maintains execution loops and enforces synchronization through barriers.

TABLE I
ACOUSTIC CONTROL MODULES AND THEIR RESPECTIVE INSTRUCTIONS.

Module	Instruction	Description
DMA	ACTLD/ST WGTLD	Load/store activations from/to DRAM Load weights from DRAM
MAC	MAC	Compute
ACTRNG	ACTRNG	Load activations into SNGs
WGTRNG	WGTRNG WGTSHIFT	Load weights into SNGs Shift weight SNG buffers
CNT	CNTLD/ST	Load/store activations from/to counter/ ReLU units
DISPATCH	FOR*/END* K/B/R/P BARR	Kernel/batch/row/pooling loop Barrier

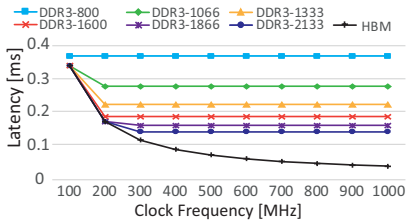

Fig. 4. Latency of processing a convolutional layer with 16x16x512 inputs and 512 3x3x512 kernels and pre-loading 512 3x3x512 kernels for the subsequent layers using different clock frequency and external memory interfaces, using temporarily-unrolled 256-long split-unipolar streams.

The dispatcher will modify and send out instructions to other control units, which are implemented as simple counter-based FSMs. Each one of them maintains a small FIFO to buffer multiple instructions and output an IDLE signal to the dispatcher once all instructions are processed. Instructions will continue to be dispatched until a barrier is encountered. Once that happens, the barrier mask will be compared with combined IDLE signals to determine if execution can continue. This allows ACOUSTIC to run multiple operations concurrently e.g. loading weights for the next layer while processing the current one.

Overall, ACOUSTIC supports an extensive number of operations which allows it to implement majority of image recognition models. Convolutions with different kernel, padding and pooling sizes, fully connected layers, ReLU activations, residual connections are all supported.

D. Evaluated ACOUSTIC Architectures

In this section, we parametrize the ACOUSTIC architecture to two reasonable choices that we evaluate in the next section. Besides the compute engine size, there are three main factors affecting ACOUSTIC system's performance: clock frequency, off-chip memory bandwidth and on-chip memory size. Increasing the clock frequency speeds up the computation, but may require higher memory bandwidth not to be memory bound. Figure 4 shows that for the bandwidth achievable using different DDR3 standards, latency becomes memory limited at around 300 MHz or below. For smaller layers, that "boundary" frequency will be much higher (e.g., above 1 Ghz for 128 3x3x128 kernels). Further, for ultra-low energy accelerators, support for large model sizes may be unnecessary and therefore all the support for DRAM can be omitted. Finally, activation memory can be sized up to support larger batch sizes if desired.

We evaluate two versions of ACOUSTIC architecture - low power (LP) and ultra-low power (ULP). Performance estimation for both configurations was done using TSMC 28nm library. LP variant (details in Table III) is intended to be integrated in mobile SoC, with limited area and power budgets. It has enough on-chip weight memory (147.5KB) to fully store weights for commonly encountered convolutional layers. For large fully-connected layers, a new batch of weights is fetched while the current one is being processed. It has enough on-chip activation memory (600KB) to process most commonly used CNNs without ever having to offload activations off-chip [28–31]. In cases where that is not possible, outputs are offloaded to external memory and fetched back when necessary for the next layer, which is supported by ACOUSTIC ISA. The ULP variant (Table IV) targets low-complexity inference (e.g. MNIST digit recognition using LeNet-5), on extremely resource

constrained devices. It is meant to compete with analog and neuromorphic approaches in terms of energy efficiency [32]. It has 2KB of activation memory and 3KB of weight memory.

IV. Evaluation & Results

A. Evaluation Methodology

SC is extremely slow to accurately simulate in software, mainly because of randomization [14]. To aid in computationally tractable design space exploration, we opted to decouple functional and performance simulations. For any trained neural network model, accuracy is evaluated using our custom SC functional simulator, which models just the computation part using bitstreams. It is given the network model, test dataset, trained weights and SC configuration i.e. stream lengths, RNG scheme etc. which it uses to compute test accuracy and compare against training results. The same configuration and neural network model (described in ACOUSTIC ISA), is then fed to the custom performance simulator, whose goal is to accurately model execution time and data movement without simulating the actual computation. The performance simulator is also fed power, area and latency numbers for individual system components, which it uses to generate accurate processing energy and latency numbers. We used TSMC 28nm library with Synopsys Design Compiler synthesis tool to obtain area, latency and power numbers for the MAC array, buffers, SNGs and counter/ReLU units. Memory, both SRAM and DRAM, were modelled using CACTI 6.5 [33].

We use Eyeriss as a baseline for the LP variant, which is a template for most spatial accelerators [6, 25, 26]. To model latency and energy consumption we use the simulator presented in [34]. We compare our numbers to original Eyeriss configuration with 168 processing elements (PEs), as well as a scaled-up version with 1024 PEs, both scaled to 28nm technology and 8-bit precision. Where possible, we also compare to SCOPE, a state-of-the-art SC neural network accelerator [14]. SCOPE is a flexible DRAM-based in-memory compute accelerator, with only multiplication performed in the stochastic domain. SCOPE numbers are reproduced from [14, 35] and scaled to 28nm. For the ULP variant we compare with MDL-CNN [32], time-based convolution engine with a similar area footprint and Conv-RAM [36], analog, in-memory convolutional engine, both scaled to 28nm. All ACOUSTIC configurations use split-unipolar representation with 2x128-bit streams.

B. ACOUSTIC Accuracy

Accuracy results are shown in Table II. AlexNet on ImageNet is too large for our current SC simulator, so SC validation accuracy is not available (SCOPE [14] only reports results on MNIST.). We do stochastic stream based training with an approximate OR as described in Section II-D to show achievable accuracy of ACOUSTIC. As Table II shows that ACOUSTIC accuracy is same as SCOPE and it can achieve accuracy similar to 8-bit fixed point hardware with stream lengths of 512 (i.e., 256x2 for split-unipolar). We believe part of the remaining gap is due to use of approximate OR during training and better but computationally tractable approximations are part of our ongoing work.

C. Area & Power Breakdown

Area breakdowns for ACOUSTIC LP and ULP configurations are shown on Figures 5 a) and b) respectively, while power breakdowns are shown on Figures 5 c) and d). As can be seen, MAC arrays are the major contributors to both area and power on the ACOUSTIC LP variant. Weight buffers, while being major contributors to area, have much lower relative power consumption due to very infrequent switching. A more area efficient implementation of weight buffers should therefore be explored. The area and energy of the ULP variant is dominated by activation and weight memories.

TABLE II
Accuracy comparisons.

Validation Accuracy [%] Network	Dataset	Stream Length	8-bit Fixed Pt	SCOPE	ACOUSTIC
LeNet-5	MNIST	128	99.2	99.32	99.3
CNN	SVHN	256	90.29	N/A	86.75
		512		N/A	89.02
	CIFAR-10	256	79.9	N/A	74.9
		512		N/A	78.04

D. Performance Comparisons

Table III compares area, power, clock frequency for all accelerators, as well as inference throughput (frames/s) and energy efficiency (frames/J) for different models. ACOUSTIC clearly outperforms conventional fixed-point accelerators. LP variant can be as much as 38.7x more energy efficient than Eyeriss with 1k PEs, depending on the network model. It is also more energy efficient than SCOPE - up to 79.6x higher frames per Joule. SCOPE require hundreds of mm^2 of area, which makes it unsuitable for edge inference. SCOPE multiplies stochastic streams in parallel, instead of using streaming like ACOUSTIC. This mandates using multiple, large DRAM arrays to achieve low latency.

The latency of AlexNet and VGG is largely dominated by fully-connected layers. Both VGG and AlexNet have multiple, large FC layers, with tens of MB of weights, and ACOUSTIC is not optimized towards those. On the Resnet-18 model, which has only a single, small FC layer, ACOUSTIC delivers lower latency than for AlexNet, despite Resnet-18 being ≈2x more computationally intensive.

Table IV shows that the ACOUSTIC ULP variant, with a comparable area footprint, can deliver up to 123x speedup over MDL-CNN, and is 1.33x more energy efficient on convolutional layers (CONV-RAM and MDL-CNN do not report performance on FC layers). It has 8.2X higher throughput than Conv-RAM with similar energy efficiency. Furthermore, ACOUSTIC uses 8-bit precision for both weights and activations, when both MDL-CNN and Conv-RAM use binarized weights, resulting in 1%-3% drop in accuracy for MNIST. For fair comparison, we're using 128-long bitstreams for ACOUSTIC ULP and non-accelerated MDL, such that neither architecture sacrifices any accuracy.

V. Conclusion

In this paper we presented ACOUSTIC accelerator for convolutional neural networks. ACOUSTIC incorporates multiple algorithm optimizations: split-unipolar representation, stochastic average pooling with computation skipping and training-hardware co-optimization. ACOUSTIC accelerator architecture is build around the idea of density-enabled data reuse, which allows it to significantly reduce the number of on- and off-chip memory accesses. ACOUSTIC architecture delivers server-class parallelism within a mobile area and power budget - a 12mm^2 accelerator can be as much as 38.7x more energy efficient and 72.5x faster than

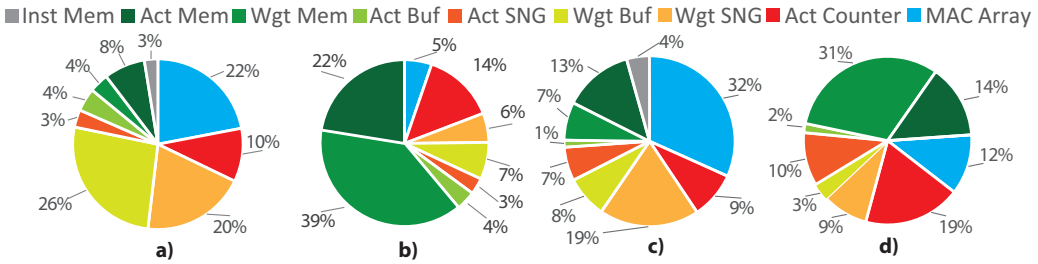

Fig. 5. Area breakdown for ACOUSTIC LP (a) and ULP (b) and power breakdown for ACOUSTIC LP (c) and ULP (d).

TABLE III
PERFORMANCE COMPARISON BETWEEN ACOUSTIC LP AND OTHER FIXED-POINT AND STOCHASTIC ACCELERATORS.

	Eyeriss 8-bit Base	Eyeriss 8-bit 1k PEs	SCOPE	ACOUSTIC LP
Area[mm2]	3.7	15.2	273.0	12.0
Power[W]	0.12	0.45	N/A	0.35
Clock[MHz]	200	200	125	200
AlexNet Fr/J	306.9	381.2	136.2	2590.6
Fr/s	41.1	210.7	5771.7	238.5
VGG-16 Fr/J	14.4	18.7	9.1	723.8
Fr/s	1.8	8.4	755.9	93.2
Resnet-18 Fr/J	295.6	380.3	N/A	2471.6
Fr/s	34.0	182.5	N/A	542.6
CIFAR-10 CNN Fr/J	N/A	N/A	N/A	131k
Fr/s	N/A	N/A	N/A	46,168

TABLE IV
PERFORMANCE COMPARISON BETWEEN ACOUSTIC ULP, MDL CNN [32] AND CONV-RAM [36] ON CONV LAYERS OF LENET5 AND CIFAR10 CNN.

	Conv-RAM	MDL CNN	ACOUSTIC ULP
Domain	Analog	Time	SC
Precision [A/W]	6b/1b	8b/1b	8b/8b SC
Area [mm2]	0.02	0.124	0.18
Power [mW]	0.016	0.03	3
Clock [MHz]	364	24	200
LeNet-5 Fr/J	40M	33.6M	41.7M
Fr/s	15,200	1009	125,000
CIFAR-10 CNN Fr/J	N/A	N/A	697K
Fr/s	N/A	N/A	2100

conventional fixed-point accelerators. It can also be up to 79.6x more energy efficient than state-of-the-art stochastic accelerators and can be implemented in an order of magnitude less area than recent stochastic computing based accelerators, delivering real-time performance in a mobile/IoT energy/area envelope. Our ongoing primarily addresses developing fast training algorithms for ACOUSTIC to improve accuracy for large networks.

ACKNOWLEDGMENT

This work was supported by the DARPA FRANC program.

REFERENCES

[1] D. D. Lin et al. “Fixed Point Quantization of Deep Convolutional Networks”. In: *CoRR* (2015). arXiv: arXiv:1511.06393v3.
[2] M. Courbariaux et al. “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1”. In: *CoRR* (2016). arXiv: 1602.02830.
[3] V. Vanhoucke et al. “Improving the speed of neural networks on CPUs”. In: *Proc. Deep Learning and ...* (2011).
[4] P. Judd et al. “Stripes: Bit-Serial Deep Neural Network Computing”. In: *IEEE Computer Architecture Letters* (2017). arXiv: arXiv:1011.1669v3.
[5] S. Anwar et al. “Fixed point optimization of deep convolutional neural networks for object recognition”. In: *2015 ICASSP* (2015).
[6] Y. H. Chen et al. “Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks”. In: *ISCA 2016* (2016).
[7] H. Kwon and T. Krishna. “A Communication-Centric Approach for Designing Flexible DNN Accelerators”. In: *IEEE Micro* (2018).
[8] M. Horowitz. “Computing’s energy problem (and what we can do about it)”. In: *ISSCC* (2014).
[9] Y. Wang et al. “FPAP : A Folded Architecture for Efficient Computing of Convolutional Neural Networks”. In: *ISVLSI* (2018).
[10] C. Eckert et al. “Neural Cache : Bit-Serial In-Cache Acceleration of Deep Neural Networks”. In: *ISCA*. 2018.
[11] H. Sim et al. “Scalable Stochastic-Computing Accelerator for Convolutional Neural Networks”. In: *ASP-DAC 2017*. 2017.
[12] A. Ren et al. “SC-DCNN : Highly-Scalable Deep Convolutional Neural Network using Stochastic Computing”. In: *ASPLOS*. 2017.
[13] V. T. Lee et al. “Energy-Efficient Hybrid Stochastic-Binary Neural Networks for Near-Sensor Computing”. In: *DATE*. 2017.
[14] S. Li et al. “SCOPE: A Stochastic Computing Engine for DRAM-based In-situ Accelerator”. In: *IEEE Micro*. 2018.
[15] Z. Li et al. “HEIF: Highly Efficient Stochastic Computing based Inference Framework for Deep Neural Networks”. In: *TCAD* (2018).
[16] B. R. Gaines. “Stochastic Computing”. In: *SJCC 1967*. 1967.
[17] A. Alaghi et al. “Optimizing stochastic circuits for accuracy-energy tradeoffs”. In: *2015 ICCAD*. 2015.
[18] Y. Liu et al. “A stochastic computational multi-layer perceptron with backward propagation”. In: *IEEE TC* (2018).
[19] A. Ardakani et al. “VLSI implementation of deep neural networks using integral stochastic computing”. In: *2016 ISTC*. 2016.
[20] S. R. Faraji et al. “Energy-Efficient Convolutional Neural Networks with Deterministic Bit-Stream Processing”. In: *DATE*. 2019.
[21] H. Sim and J. Lee. “A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks”. In: (2017).
[22] J. A. Dickson et al. “Stochastic arithmetic implementations of neural networks with in situ learning”. In: *IEEE ICNN 1993*. 1993.
[23] J. Yu et al. “Accurate and Efficient Stochastic Computing Hardware for Convolutional Neural Networks”. In: *2017 ICCD* (2017).
[24] Z. Li et al. “Structural design optimization for deep convolutional neural networks using stochastic computing”. In: *DATE* (2017).
[25] Y. H. Chen et al. “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks”. In: *ISSCC* (2016). arXiv: 1512.04295.
[26] H. Kwon et al. “A Data-Centric Approach for Modeling and Estimating Efficiency of Dataflows for Accelerator Design”. In: *CoRR* (2018). arXiv: arXiv:1805.02566v3.
[27] Y.-h. Chen et al. “Eyeriss v2 : A Flexible and High-Performance Accelerator for Emerging Deep Neural Networks”. In: *CoRR* (2018). arXiv: arXiv:1807.07928v1.
[28] A. Krizhevsky et al. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *NIPS 2012* (2012). arXiv: 1102.0183.
[29] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv preprint arXiv:1409.1556* (2014). arXiv: 1409.1556.
[30] C. Szegedy et al. “Going Deeper with Convolutions”. In: *CVPR*. 2015.
[31] K. He et al. “Deep Residual Learning for Image Recognition”. In: *CVPR*. 2015. arXiv: 1512.03385.
[32] A. Sayal et al. “All-Digital Time-Domain CNN Engine Using Bidirectional Memory Delay Lines for Energy-Efficient Edge Computing”. In: *ISSCC’2019*. IEEE. 2019.
[33] H. Labs. *CACTI-6.5 (Cache Access Cycle Time Indicator)*.
[34] M. Gao and M. Horowitz. “TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory”. In: *ASPLOS*. 2017.
[35] S. Li et al. “DRISA : A DRAM-based Reconfigurable In-Situ Accelerator”. In: *IEEE MICRO* (2017).
[36] A. Biswas and A. P. Chandrakasan. “Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications”. In: *ISSCC’2018*. 2018.