Chipmunk: A systolically scalable 0.9 mm2, 3.08Gop/s/mW @ 1.2 mW accelerator for near-sensor recurrent neural network inference

CHIPMUNK: A Systolically Scalable 0.9 mm^2, 3.08 Gop/s/mW @ 1.2 mW Accelerator for Near-Sensor Recurrent Neural Network Inference

Francesco Conti*†, Lukas Cavigelli*, Gianna Paulin*, Igor Susmelj*, Luca Benini*†

*Integrated Systems Laboratory, ETH Zürich, Switzerland

†Energy-Efficient Embedded Systems Laboratory, University of Bologna, Italy

***Abstract*—Recurrent neural networks (RNNs) are state-of-the-art in voice awareness/understanding and speech recognition. On-device computation of RNNs on low-power mobile and wearable devices would be key to applications such as zero-latency voice-based human-machine interfaces. Here we present CHIPMUNK, a small (<1 mm^2) hardware accelerator for Long-Short Term Memory RNNs in UMC 65 nm technology capable to operate at a measured peak efficiency up to 3.08 Gop/s/mW at 1.24 mW peak power. To implement big RNN models without incurring in huge memory transfer overhead, multiple CHIPMUNK engines can cooperate to form a single systolic array. In this way, the CHIPMUNK architecture in a 75 tiles configuration can achieve real-time phoneme extraction on a demanding RNN topology proposed in [1], consuming less than 13 mW of average power.**

I. INTRODUCTION

In the last few years, we have witnessed an "artificial intelligence" revolution that has been fueled by the concurrent availability of huge amounts of training data, computing power to learn upon it, and evolution of "smart" algorithms, in particular those based on deep learning. Within this field, *recurrent neural networks* (RNNs), particularly Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), are receiving increasing attention: They have shown state-of-the-art accuracy in tasks such as speech recognition [1], [2] and language translation [3], making them the forefront of the "intelligent" user interfaces of products such as Amazon Alexa, Google Assistant, Apple Siri, Microsoft Cortana and others.

One of the key limitations of the current generation of commercial products based on RNNs is that these embedded, edge devices depend on remote servers taking care of the computational workload necessary for the deployment of these algorithms. Moreover, when RNNs are used as a component of human-machine interfaces, the intrinsic latency of network communication can also be problematic, as people expect the "smart" devices to reply not only accurately, but also timely.

For these reasons, it is very attractive to integrate RNN capabilities locally in embedded mobile and wearable platforms, making them capable of state-of-the-art voice and speech recognition autonomously and independent from external servers. Nonetheless, while much attention has recently been dedicated to the deployment of embedded low-power inference accelerators for forward-only deep networks deployment [4]–[8], making RNNs energy-efficient is a fundamentally harder problem: the necessity to keep and update an internal state and the widespread usage of densely connected layers translate to very large memory footprint and high bandwidth requirements.

In this work, we present a twofold contribution towards the deployment of RNN-based algorithms in devices such as smartphones, smartwatches and wearables. First, we designed CHIPMUNK, a small and low-energy hardware accelerator engine targeted at real-time speech recognition and capable to operate autonomously on moderate size LSTM networks. We present silicon results from a prototype chip containing a CHIPMUNK engine, which has been fabricated in UMC 65 nm technology; the chip can achieve up to 3.8 Gop/s at maximum efficiency operating point (@0.75 V), consuming only 1.24 mW.

Second, we conceived a scalable computing architecture, apt to operate on bigger LSTM models as well. As the main limitation to the deployment of big RNNs in embedded scenarios stems from their memory boundedness, we designed the CHIPMUNK engines so that they can be replicated in a systolic array, cooperating on a single bigger LSTM network. This methodology allows the acceleration of large-scale RNNs, which can be made fast enough to operate in real-time under realistically tight time, memory and battery constraints without requiring complex, power hungry and expensive high-bandwidth main memory interfaces.

II. RELATED WORK

A recent thorough survey of efforts on hardware acceleration and design of efficient shows that few efforts have been focused on RNN inference [9]. We thus focus on this application, surveying state-of-the-art implementations from data-center to ultra-low power accelerators in the remainder of this section.

Data center workloads for RNNs are often offloaded to GPUs or specialized semi-independent co-processors such as Google's Tensor Processing Unit (TPU) [10] consuming in the order of 50-300 W. The TPU is a unified architecture to target DNNs with convolutional and densely connected layers as well as LSTMs. However, TPUs suffer from low utilization when running RNNs. Yet 29% of the workload running on Google's TPUs is devoted to RNN inference [10], showing their relevance in commercial applications.

In a lower power range (tens of Watts), several FPGA implementations can be found. The Efficient Speech-recognition Engine (ESE) [11] targets the deployment of RNNs on a Xilinx UltraScale FPGA. To maximize efficiency and address the memory boundedness of RNNs, it heavily focuses on network quantization and pruning of the recurrent topologies and thus this accelerator engine is mainly targeted at sparse matrix-vector operations. Rybalkin *et al.* [12] also target bidirectional LSTMs in their FPGA accelerator. Bidirectional LSTMs have been shown to obtain better accuracy in some cases [1] but are less attractive for an online, real-time scenario as they inherently increase the network latency. Finally, DeepStream [13] is a small hardware accelerator deployed on a Xilinx Zynq 7020 targeted at text recognition with RNNs. It requires to

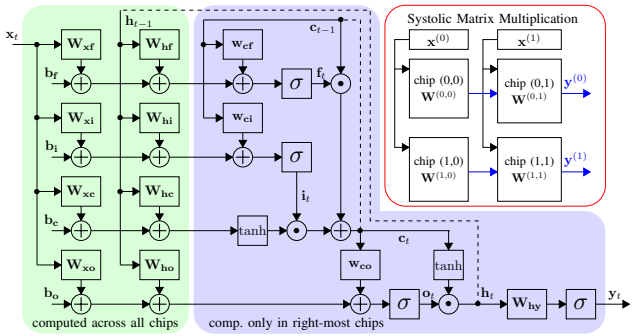


Fig. 1. Data dependency graph of a LSTM. The majority of computations are the vector-matrix mult. (green) and can be distributed across multiple chips. Top-right: distribution of a vector matrix mult. to a systolic array of chips.

continuously stream in weights, which makes it impractical for big RNN topologies with millions of weights.

The only published ultra-low power (few mW) implementation, the DNPU [14], uses two separate special-purpose engines for convolutional layers (called CP), on one side, and fully-connected and recurrent ones on the other (called FRP). The FRP does not include any particular facilities to address the stateful nature of RNNs, and it includes only a small amount of memory (10 kB) making external memory accesses necessary for even small RNNs, thus limiting peak performance by introducing a serious bandwidth bottleneck.

III. Architecture

A. Operating principle

Long Short-Term Memory (LSTM) network layers [15] are often described with the following set of canonical equations:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{w}_{ci} \odot \mathbf{c}_{t-1} + \mathbf{b}_i) \quad (1)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{w}_{cf} \odot \mathbf{c}_{t-1} + \mathbf{b}_f) \quad (2)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c) \quad (3)$$

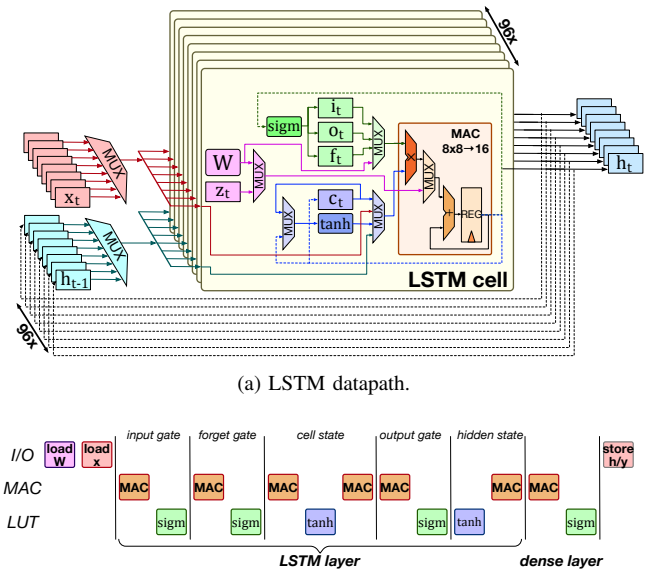
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{w}_{co} \odot \mathbf{c}_t + \mathbf{b}_o) \quad (4)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (5)$$

where $\mathbf{x}$ is the *input state* vector; $\mathbf{i}$, $\mathbf{f}$, $\mathbf{o}$ are called *input*, *forget* and *output gates* respectively; $\mathbf{c}$ and $\mathbf{h}$ are the *cell* and *hidden states*. The subscript indicates either the current state t or the previous $t-1$, and $\odot$ denotes element-wise multiplication[1]. The characteristic dimensions of all vectors and matrices depend on the size of the input state (N_x) and on that of the hidden state (N_h). Multiple LSTM layers can be connected by using the hidden state of one layer as input of the next. Finally, LSTM networks often include a final densely connected layer without recurrence: $\mathbf{y}_t = \sigma(\mathbf{W}_{hy}\mathbf{h}_t)$.

In Chipmunk, we exploit two distinct observations regarding LSTMs. First, all compute steps are based on the same set of basic operations: *i)* matrix-vector products, *ii)* element-wise vector products, and *iii)* element-wise non-linear activations. The internal datapath of Chipmunk can be configured to execute these three basic operations (Section III-B) and the LSTM state parameters are stored on-chip. Second, the vast amount of data required to compute one time step of a RNN

[1] In most literature Eqs. (1), (2) and (4) use matrix notation for $\mathbf{W}_{ci}$, $\mathbf{W}_{cf}$ and $\mathbf{W}_{co}$; however as these matrices are diagonal by construction, we use the element-wise product notation here for consistence with what is actually implemented in the Chipmunk hardware.



(a) LSTM datapath.

(b) Sequence of datapath basic operation loops.

Fig. 2. LSTM datapath used in Chipmunk and typical sequence of operations. The datapath can be used to implement the operations in Eqs. (1) to (5) by appropriately controlling the muxes and clearing the register states.

are the weights. Storing them on-chip is thus essential to achieve high energy efficiency. To this end, we a large share of the overall chip area is dedicated to SRAM to keep the weights local. For larger LSTMs not fitting on a single chip, we allow operation in a systolic mode where the weights are split across multiple connected chips and only the much smaller intermediate results are exchanged as further discussed in Section III-C.

B. Tile architecture

A product between a matrix of size $A \times B$ and a vector of size B is composed of two nested loops, i.e. in pseudo-code:

```
for a in range(0, A): # row loop
  for b in range(0, B): # column loop
    z += W[a,b] * x[b]
```

In Chipmunk, the highlighted row loop is executed on multiple parallel units, while the inner loop is executed sequentially. Fig. 2a shows a high-level diagram of the Chipmunk LSTM datapath that implements this functionality. N_{lstm} parallel LSTM units are used to execute all the iterations of the row loop at the same time. Each LSTM unit is composed of an embedded memory bank to store weights (W), registers for storing the $\mathbf{o}_t$, $\mathbf{f}_t$, $\mathbf{i}_t$ and $\mathbf{c}_t$ values locally, a multiply-accumulate unit and two lookup tables to implement the non-linear activation functions. $\mathbf{x}_t$ and $\mathbf{h}_t$ are kept outside of the LSTM units, in a bank of N_{lstm} registers. At each cycle of a column loop, one element of the input state and one of the hidden state are selected depending on the iteration index and broadcast to all LSTM units. Fig. 2b shows the basic operation loops composing a LSTM network deployed on Chipmunk.

All state variables use 8 bit fixed point precision, while 16 bits are used within the multiply-accumulate block to minimize overflows. I/O is performed via an input stream port and an output stream port, each consisting of 8 bits of data and 2 bits to enable a simple ready/valid handshake. Weights are loaded

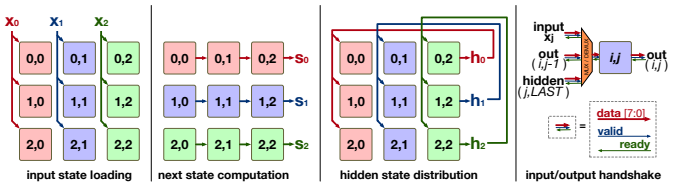


Fig. 3. CHIPMUNK tile I/O and operation of a 3×3 systolic array during the load of the input state $\mathbf{x}_t$, computation of the new $\mathbf{i}_t$, $\mathbf{o}_t$, $\mathbf{f}_t$, $\mathbf{c}_t$, $\mathbf{h}_t$ state values, and redistribution of the updated hidden state $\mathbf{h}_t$.

at the beginning of the computation of a LSTM layer, and inputs are streamed in sequentially. The internal state of the LSTM cell in terms of cell state and hidden state is retained between consecutive LSTM input "frames" to implement the recurrent nature of the network. A CHIPMUNK engine can be used to implement a full LSTM network with $N_x, N_h \leq N_{lstm}$ storing the weights on chip. Larger networks require to stream them in from an external source.

C. Systolic scaling

As the main target of the CHIPMUNK accelerator is to enable ultra-low latency applications such as on-device real-time speech recognition, the computing power of a single engine might not be sufficient. A single engine cannot be arbitrarily scaled up: LSTM units are all coupled to the same set of registers via simple multiplexers, making it impractical to increase N_{lstm} above a few hundred units. Instead, to provide a more scalable and elegant solution, we designed CHIPMUNK so that multiple engines can be connected as tiles and share the burden of the RNN computation in a spatial fashion.

Fig. 3 shows how the computation is split between multiple tiles in the case of a 3×3 array. The input state is split into vectors of size N_{lstm} and each vector is broadcast vertically along a column. The new value for the internal gates/states is computed by accumulating the results computed by each row. Finally, the last column can compute the output hidden state, which is broadcasted vertically to the columns for the next iteration (cf. Fig. 3c). For a given network size/systolic configuration, these connections can be hard-wired such that no external multiplexing is required.

IV. RESULTS & DISCUSSION

A. Silicon prototype & Comparison with State-of-the-Art

We designed and built a silicon prototype based on a single CHIPMUNK tile as described in Section III-B. The prototype chip was fabricated in UMC 65 nm technology, using high voltage threshold cells to minimize leakage. It features $N_{lstm} = 96$ LSTM units, which hold their weight and bias parameters in 12 separate SRAM banks (81.7 kB in total). The full chip, shown in Fig. 4, occupies $1.57\,\mathrm{mm}^2$ including the pads. The chip exposes the interface described in Section III-C for tile-to-tile communication, so that it would be possible to prototype a systolic array using many discrete chips.

Fig. 4 shows the experimental results obtained by testing the CHIPMUNK prototype at room temperature (25 °C). The prototype is fully functional in an operating range between 0.75 V (limited by SRAM failure) and 1.24 V, corresponding to a range of 20 to 168 MHz of maximum clock frequency and from 1.24 to 29 mW of power consumption. The peak performance in terms of operations per second[2] of one CHIPMUNK chip is 32.2 Gop/s (at 1.24 V) and the peak energy efficiency (3.08 Gop/s/mW) is reached at 0.75 V.

Table I compares architectural parameters and synthetic results between CHIPMUNK and the existing VLSI and FPGA-based implementations for which performance and energy numbers have been published. Our work reaches comparable performance with the DNPU proposed by Shin *et al.* [14]. Performance is obviously below that claimed by Google TPU [10], but this is mostly due to the different size. In fact, despite the TPU uses 28 nm integration, CHIPMUNK has 2.8× better area efficiency - and a performance-wise "TPU-equivalent" array with ∼115 CHIPMUNK engines would consume only 3.33 W, an order of magnitude less than the TPU. CHIPMUNK advances the state-of-the-art energy efficiency with respect to the DNPU, showing a 39% improvement. Moreover, the DNPU does not include any provision to address the fundamental memory boundedness of RNNs, which CHIPMUNK addresses via systolic scaling. All FPGA implementations [11]–[13] are at least two orders of magnitude less energy-efficient.

In terms of arithmetic precision we have chosen to use 8 bit fixed-point representations for storage and perform the MAC operations with 16 bit precision. This is in line with Google's TPU and higher than the 4-7 bit of the DNPU.

B. Real-world speech recognition

To evaluate CHIPMUNK on a real-world problem, we targeted *CTC-3L-421H-UNI*, a 3-layer, 421-hidden units per layer LSTM topology introduced by Graves *et al.* [1], which takes as input a stream of 123 Mel-Frequency Cepstral Coefficients (MFCCs) extracted from an audio stream and identifies phonemes with an error rate of 19.6%, evaluated on the TIMIT database. The MFCC input "frames" are produced with a 10 ms rate, which means that any embedded low-latency real-time RNN implementation should be able to elaborate the full network in less than this time. We evaluate three different CHIPMUNK configurations: a systolic array of 75 units, divided in 3 sub-arrays of 5×5 engines; a single array of 5×5 engines; and a single CHIPMUNK engine. The largest configuration can host the full topology in a spatial fashion; each of the sub-arrays hosts one layer of the RNN. After the initial programming phase, it does not need any reprogramming. The smaller arrays need to be reconfigured at each new layer (in the 5×5 array case) or multiple times per layer (in the single unit case).

Table II reports execution time and power for these three configurations. Execution times include both computation and reconfiguration, excluding only the initial configuration which doesn't need to be repeated for each new frame/layer. Bold time/power values indicate configurations that can meet the 10 ms deadline. As the *CTC-3L-421H-UNI* topology has $\sim 3.8 \times 10^6$ weights, a $3 \times 5 \times 5$ systolic configuration is best used (all weights stored locally). Smaller configurations imply a $> 80\%$ overhead for reloading weights.

Average power, also shown in Table II, is computed under the assumption that the array is perfectly duty cycled when not in use over the 10 ms window. Even in the assumption that the CHIPMUNK array is always-on, the 12.55 mW required to process this network would only add ∼4% to idle power on

[2] As customary for neural network accelerators, we count 1 multiply-accumulate as 2 operations.

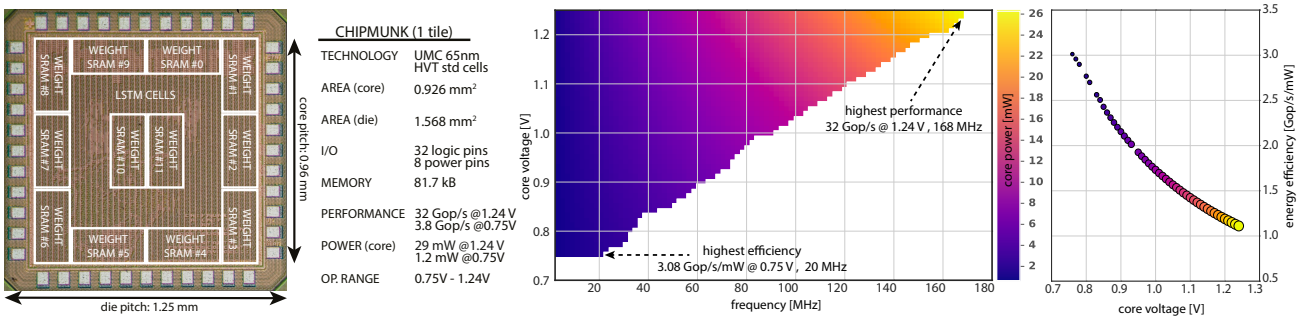


Fig. 4. Left: Microphotograph of a CHIPMUNK die. Right: Frequency, power and performance of the CHIPMUNK prototype versus operating voltage at room temperature (25 °C). The left shmoo plot shows core voltage versus operating frequency; the color shade corresponds to the core power consumption (darker=less power). The right plot shows energy efficiency versus core voltage; the color shade of the scattered dots corresponds to the core power, while their size is proportional to the maximum frequency.

TABLE I. COMPARISON TO EXISTING VLSI AND FPGA IMPLEMENTATIONS

	THIS WORK	DNPU*[14]	Google TPU†[10]	Han *et al.*‡[11]	Rybalkin *et al.* [12]	Chang *et al.* [13]
Technology	UMC 65 nm CMOS	65 nm CMOS	28 nm CMOS	Xilinx XCKU060	Xilinx Z7045	Xilinx Z7020
Area	core: 0.93 mm²	core*: ∼2.0 mm²	–	294k LUT	33k LUT	7.6k LUT
	die: 1.57 mm²	die: 16.0 mm²	die: ∼300 mm²	453k FF	15k FF, 33 DSP	13k FF, 50 DSP
On-chip memory	82 kB	10 kB	28 MB	4.2 MB	332 kB	–
Arithmetic	8-16 bit	4-7 bit	8-16 bit	12 bit, pruned	5-16 bit	16 bit
Number of MACs	96	64	66k	–	–	4
Core voltage	1.24 V / 0.75 V	1.1 V / 0.77 V	–	–	–	–
Frequency	168 / 20 MHz	200 / 50 MHz	700 MHz	200 MHz	166 MHz	142 MHz
Power	29.03 / **1.24 mW**	21 / 2.6 mW	40-28 W	41 W	∼10 W	2.3 W
Peak performance	32.3 / 3.8 Gop/s	25 / 6.25 Gop/s	**3.7-2.8 Top/s**	2.5 Top/s (equiv.)‡	152 Gop/s	0.389 Gop/s
Energy efficiency	1.11 / **3.08 Gop/s/mW**	1.10 / 2.22 Gop/s/mW	<0.13 Gop/s/mW	0.061 Gop/s/mW‡	0.0152 Gop/s/mW	0.000146 Gop/s/mW
Area efficiency	**34.4 Gop/s/mm²**	12.5 Gop/s/mm²	12.3-9.3 Gop/s/mm²	–	–	–

* The DNPU is a mixed CNN-RNN processor. We report here only the figures related to the RNN subunit.
† We present here the values from [10] based on the two LSTMs for which they measured the performance. For both, the TPU is severely memory bandwidth limited.
‡ They assume a well-structured sparsity of 11.2% in the weight matrices. Reported numbers are dense-equivalent throughput. Underlying compute throughput: 282 Gop/s.

TABLE II. CTC-3L-421H-UNI SPEECH RECOGNITION LSTM EXECUTED ON CHIPMUNK WITH A 10 MS CONSTRAINT

	Configuration	PERF @1.24 V	EFF @0.75 V
Execution time	systolic 3×5×5	**0.09 ms**	**0.76 ms**
	systolic 5×5	**1.59 ms**	13.31 ms
	single	38.23 ms	321.14 ms
Peak power	systolic 3×5×5	**1833.75 mW**	**165.75 mW**
	systolic 5×5	**611.25 mW**	55.25 mW
	single	24.45 mW	2.21 mW
Average power	systolic 3×5×5	**16.53 mW**	**12.55 mW**
	systolic 5×5	**96.89 mW**	-

a typical smartphone (300 to 400 mW [16]). Adding a filter to drop clearly uninteresting input (e.g. silence) would likely decrease this overhead by an order of magnitude.

V. CONCLUSION

We have presented an architecture and silicon measurement results for a small (0.9 mm²) RNN hardware accelerator providing 3.8 Gop/s at 1.2 mW in 65 nm digital CMOS technology, resulting in new state-of-the-art energy and area efficiencies of 3.08 Gop/s/mW and 34.4 Gop/s/mm². The systolic design is scalable to accommodate also large RNNs efficiently by connecting multiple identical chips on the circuit board.

ACKNOWLEDGEMENTS

This work was supported in part by the EU project ExaNoDe under grant H2020-671578, and in part by the Swiss National Science Foundation project Micropower Deep Learning.

REFERENCES

[1] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech Recognition With Deep Recurrent Neural Networks," in *Proc. IEEE ICASSP*, 2013.

[2] W. Xiong, J. Droppo *et al.*, "The Microsoft 2016 Conversational Speech Recognition System," in *Proc. IEEE ICASSP*, 2017, pp. 5255–5259.

[3] K. Cho, B. van Merrienboer *et al.*, "Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation," in *Proc. ACL EMNLP*, 2014, pp. 1724–1734.

[4] L. Cavigelli and L. Benini, "A 803 GOp/s/W Convolutional Network Accelerator," *IEEE TCSVT*, 2016.

[5] F. Conti, R. Schilling *et al.*, "An IoT Endpoint System-on-Chip for Secure and Energy-Efficient Near-Sensor Analytics," *IEEE TCAS*, vol. 64, no. 9, pp. 2481–2494, 9 2017.

[6] R. Andri, L. Cavigelli *et al.*, "YodaNN: An Architecture for Ultra-Low Power Binary-Weight CNN Acceleration," *IEEE TCAD*, 2017.

[7] Y.-H. Chen, T. Krishna *et al.*, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *Proc. IEEE ISSCC*, 2016, pp. 262–263.

[8] Z. Du, R. Fasthuber *et al.*, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *Proc. ACM/IEEE ISCA*, 2015, pp. 92–104.

[9] V. Sze, Y.-H. Chen *et al.*, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *arXiv:703.09039*, 2017.

[10] N. P. Jouppi, A. Borchers *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proc. ACM ISCA*, 2017.

[11] S. Han, J. Kang *et al.*, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *Proc. ACM/SIGDA FPGA*, 2016.

[12] V. Rybalkin, N. Wehn *et al.*, "Hardware Architecture of Bidirectional Long Short-Term Memory Neural Network for Optical Character Recognition," in *Proc. IEEE DATE*, 2017, pp. 1390–1395.

[13] A. X. M. Chang and E. Culurciello, "Hardware Accelerators for Recurrent Neural Networks on FPGA," in *Proc. IEEE ISCAS*, 2017.

[14] D. Shin, J. Lee *et al.*, "DNPU: An 8.1TOPS/W Reconfigurable CNN-RNN Processor for General-Purpose Deep Neural Networks," in *Proc. IEEE ISSCC*, vol. 60, 2017, pp. 240–241.

[15] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," in *USENIX ATC*, 2010.