Towards Ultra-High Performance and Energy Efficiency of Deep Learning Systems: An Algorithm-Hardware Co-Optimization Framework

Yanzhi Wang[1], Caiwen Ding[1], Zhe Li[1], Geng Yuan[1], Siyu Liao[2], Xiaolong Ma[1], Bo Yuan[2], Xuehai Qian[3], Jian Tang[1], Qinru Qiu[1], Xue Lin[4]

[1]Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244
[2]Department of Electrical Engineering, City University of New York, New York, NY 10031
[3]Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089
[4]Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115

Abstract

Hardware accelerations of deep learning systems have been extensively investigated in industry and academia. The aim of this paper is to achieve ultra-high energy efficiency and performance for hardware implementations of deep neural networks (DNNs). An algorithm-hardware co-optimization framework is developed, which is applicable to different DNN types, sizes, and application scenarios. The algorithm part adopts the general block-circulant matrices to achieve a fine-grained tradeoff of accuracy and compression ratio. It applies to both fully-connected and convolutional layers and contains a mathematically rigorous proof of the effectiveness of the method. The proposed algorithm reduces computational complexity per layer from O(n^2) to O($n \log n$) and storage complexity from O(n^2) to O(n), both for training and inference. The hardware part consists of highly efficient *Field Programmable Gate Array* (FPGA)-based implementations using effective reconfiguration, batch processing, deep pipelining, resource re-using, and hierarchical control. Experimental results demonstrate that the proposed framework achieves at least 152X speedup and 71X energy efficiency gain compared with IBM TrueNorth processor under the same test accuracy. It achieves at least 31X energy efficiency gain compared with the reference FPGA-based work.

Introduction

The recent *deep neural networks* (DNNs), especially deep *convolutional neural networks* (CNNs), have been able to deliver remarkable success in visual and recognition tasks (Deng et al.,Taigman et al.) and real-world applications (Huval et al., Collobert and Weston, Burbidge et al.), by leveraging large-scale neural network sizes and learning from a huge volume of data. Despite the advantage of improved overall accuracy, the deep layered structure and large model sizes increase the computational complexity and memory requirements. It is projected that the majority of inference tasks will be performed on embedded, IoT and mobile systems which are with limited power and computational resources. In order to achieve higher scalability, performance, and energy efficiency, two orthogonal research and development trends have both attracted enormous interests.

The first is hardware accelerations of deep learning systems/applications, which have been extensively investigated in industry and academia (Farabet et al.,Suda et al.,Qiu et al., Zhang et al.,Zhang et al.,Han et al.,Zhao et al.,Zhang and Li, Umuroglu et al.,com,com,Chen et al.,Han et al.,Chen et al.). As a representative technique, FPGA-based accelerators can offer the advantages of programmability, high degree of parallelism and short development cycle. Important progresses have been reported on FPGA accelerations of original DNNs (Farabet et al., Suda et al., Zhang et al., Zhang et al.), binary neural networks (Zhao et al., Umuroglu et al.), and more recently, on DNNs and recurrent neural networks (RNNs) with model compression techniques (Qiu et al., Han et al.). These prior work mainly focus on the inference phase of DNNs, and suffer from frequent access to off-chip memory systems because the limited on-chip memory can hardly accommodate the large model sizes. Accessing off-chip memory is highly energy inefficient. As pointed out in (Han et al., Han, Mao, and Dally), the per-bit access energy of off-chip memory is 200X compared with on-chip memory storage, and dominates the whole system power consumptions. Besides, it is also desirable to achieve algorithmic-level accelerations to accommodate the further scaling of DNNs, instead of simply adding more and more hardware devices.

The second important trend is the model size compression and algorithmic-level acceleration of DNNs (with very minor accuracy loss), including weight quantization (Lin, Talathi, and Annapureddy, Lin et al.), sparsity regularization (Feng and Darrell, Wen et al., Li, Park, and Tang), connection pruning (Han et al., Han, Mao, and Dally), and low rank approximation (Denil et al., Denton et al.). These approaches can offer a reasonable amount of parameter reduction (e.g., by $9\times$ to $13\times$ in (Han et al., Han, Mao, and Dally)) and/or a reasonable speedup (e.g., around 50% to $2\times$ in (Wen et al.)). However, they suffer from the following limitations: (i) the sparsity regularization and pruning methods will likely result in an irregular and sparse network structure, thereby undermining the compression ratio and increasing computation time (especially inefficient on GPUs and dedicated hardware which has high parallelism capability); (ii) the training complexity will be increased by incorporating additional pruning process (Han et al., Han, Mao, and Dally), additional low rank approximation step (Denil et al., Denton et al.), or extra trade-off parameters (Wen et al.); (iii) the compression or acceleration factors are heuristic numbers that cannot be precisely controlled, not to mention a mathematically rigor-

ous proof of the effectiveness of these methods.

To combine these two directions, the aim of this paper is to address the limitations of existing model size compression and acceleration work and to achieve ultra-high energy efficiency and performance for FPGA-based hardware implementations of DNNs, by (i) deriving a highly suitable algorithm for efficient computation and storage reduction without significant accuracy loss, and (ii) deriving the corresponding optimized hardware implementations. We develop an algorithm-hardware co-optimization framework, which is applicable to different DNN types, sizes, and application scenarios. The proposed framework comprises algorithm and hardware parts. The **algorithm part** extends reference (Cheng et al.), which applies circulant matrices to the whole fully-connected (FC) layer for model compression, to (i) the adoption of the general *block-circulant matrices* to achieve fine-grained tradeoff of accuracy and compression ratio, (ii) the generalization to the convolutional (CONV) layers for significant acceleration as CONV layers dominate the computation of DNNs (Krizhevsky, Sutskever, and Hinton, He et al.), (iii) providing a mathematically rigorous proof that the proposed algorithm will asymptotically converge to the same "effectiveness" as DNNs without compression, and (iv) decoupling the fast Fourier transform (FFT) and inverse FFT computations in the framework for accelerating computation and facilitating hardware implementations. The proposed algorithm reduces computational complexity per layer from O(n^2) to O($n \log n$) and storage complexity from O(n^2) to O(n), both for training and inference, with negligible degradation in DNN accuracy. The **hardware part** consists of highly efficient FPGA-based implementations using effective reconfiguration, batch processing, deep pipelining technique, effective resource re-using, and a hierarchical control framework. The proposed FPGA-based implementation can accommodate the whole DNN model using on-chip block memory, thereby significantly improving the overall energy efficiency. Finally, a comprehensive **algorithm-hardware co-optimization** is proposed which comprises (i) model selection and optimization, (ii) hardware optimization, and (iii) variational inference-based Bayesian learning for enhancing accuracy and robustness. In summary, the major contributions of this work include both algorithm and hardware parts. The algorithm part adopts block-circulant matrices for weight representation, which could achieve a significant model compression ratio with minor accuracy degradation. It applies to the whole network, both fully-connected and convolutional layers. The hardware part consists of highly efficient FPGA-based implementations with multiple innovative parts of reconfiguration, batch processing, deep pipelining, resource re-using, etc.

Please note that the proposed framework **is distinct from** the prior work (Mathieu, Henaff, and LeCun), which applies FFTs to accelerate the computations in the CONV layers. The prior work applies only to a single filter in the CONV layer and achieves no storage reduction (in fact it results in storage increase), whereas the proposed method applies both to CONV and FC layers and achieves simultaneous acceleration and storage reduction.

Because we focus on highly energy-efficient FPGA-based implementations for low-power embedded applications, we focus on the inference phase of small to medium-scale DNNs (e.g., for MNIST, SVHN, CIFAR datasets) on high energy-efficiency FPGAs. Compared with the IBM TrueNorth neurosynapstic processor (Merolla et al.), our FPGA-based implementation achieves at least 152X speedup in throughput and 71X energy efficiency gain under the same test accuracy. Similarly, our actual FPGA implementations outperform (in performance) the state-of-the-art analog-based and emerging device-based implementations. Our framework achieves at least 31X gain in equivalent energy efficiency compared with the reference FPGA-based work that achieves the best efficiency.

Related Works

FPGA Accelerations of DNNs. FPGA-based accelerations of DNNs have been extensively investigated recently due to the advantages of programmability, high degree of parallelism and short development cycle. Based on the early work of direct acceleration of FPGAs (Farabet et al.), recently researchers have investigated energy-efficient implementations using the batch processing technique (Zhang et al., Zhang et al.) or on compressed models using singular value decomposition (SVD) (Qiu et al.). In this year the research on this topic has exploded, including accelerations of DNNs with weight pruning (Han et al.), binary neural networks (Zhao et al., Umuroglu et al.), and high-level synthesis for fast generation of FPGA implementations (Zhao et al.,Zhang and Li). These work typically suffer from frequent access to off-chip memory systems because their model sizes cannot be effectively reduced for on-chip memory storage, thereby resulting in high energy consumptions. The typical (equivalent) energy efficiency range is from 7 GOPS/W to less than 1 TOPS/W, depending on the testing FPGA platform, implementation details, and compression techniques.

Connection Pruning and Weight Sparsifying. Han *et al.* (Han et al., Han, Mao, and Dally) reduced the number of parameters by 9X - 13X using connection pruning. Since most reduction is achieved on FC layers, no significant speedups of CONV layers can be observed (Wen et al.). As CONV layers have become the computational bottleneck, compression and acceleration on CONV layers become essential. Liu *et al.* achieved layer-wise 4.59X speedup on the CONV layers of *AlexNet* with 2% accuracy loss. Recently, (Wen et al.) adopts a *structured sparsity learning* method and derives an effective tradeoff between acceleration on CPU/GPU and test accuracy for the CONV layers. More specifically, for *ResNet-20* on CIFAR-10 and *AlexNet* on ImageNet benchmarks, more than 50% acceleration can be achieved without any accuracy loss, while around 3X acceleration is achieved with an acceptable accuracy loss of 2%.

FFTs for CONV Layer Accelerations. LeCun *et al.* have proposed using FFTs to accelerate the computations in the CONV layers, which applies only to a single filter in the CONV layer (Mathieu, Henaff, and LeCun). It uses FFT to calculate the traditional inner products of filters and input feature maps, and can achieve speedup for large filter sizes. The underlying neural network structure remains

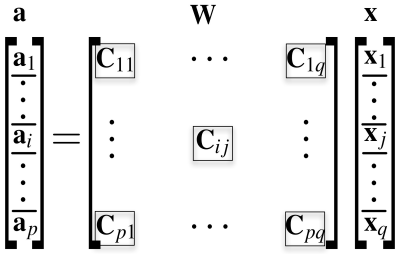

Figure 1: **Block-circulant matrix-vector multiplication in the proposed framework.**

unchanged. The speedup is due to filter reuse and it cannot achieve either asymptotic speedup in Big-O notation or weight compression.

Structured Matrices in FC Layers for Model Compression. The most relevant work to this paper is (Cheng et al.), which directly applies circulant matrices to the FC layers for model compression. As an example, an FC layer of DNN can be represented as $\mathbf{y} = \psi(\mathbf{W}\mathbf{x}+\theta)$, where vectors $\mathbf{x}$ and $\mathbf{y}$ represent the outputs of all neurons in the previous layer and the current layer, respectively; $\mathbf{W}$ is an n-by-n weight matrix; and $\psi(\cdot)$ is the activation function. When $\mathbf{W}$ is a circulant matrix, the fast Fourier transform (FFT)-based fast multiplication method can be utilized, and the computational complexity and weight storage complexity will be reduced from O(n^2) to O($n \log n$) and from O(n^2) to O(n), respectively. Despite the significant reduction in computation and weight storage, this approach has the limitations of (i) resulting in a huge number of padding 0's when the numbers of inputs and outputs are not equal, (ii) resulting in certain accuracy degradation for large-scale FC layers because of the aggressive weight reduction, and (iii) only applicable to the FC layer, whereas the CONV layers are the most computationally intensive in DNNs.

Algorithm Development of Block-Circulant Matrix-Based DNNs

In this section, we develop the algorithmic framework of block-circulant matrix-based DNNs for simultaneous acceleration and model compression, for both inference and training phases. The proposed framework is able to accommodate arbitrary size and aspect ratio of weight matrices, and achieves a fine-grained tradeoff between test accuracy and compression/acceleration ratio (Ding et al.). Unlike (Cheng et al.), we develop algorithms for both FC and CONV layers as shown in the following. We provide a mathematically rigorous proof of the proposed algorithm that it satisfies the *universal approximation property* as uncompressed DNNs. Finally, we develop decoupling technique for FFT/IFFT pairs for further acceleration and facilitating hardware (FPGA) implementations.

Inference and Training Algorithms for FC Layers

The key idea of block-circulant matrix-based FC layers is to partition the original arbitrary-size unstructured weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ into 2D blocks of square sub-matrices. Such partitioning strategy has two advantages: 1) It is suitable for arbitrary-size weight matrices without any requirement on the aspect ratio of $\mathbf{W}$; and 2) it is an adjustable approach that can conveniently control the compression ratio and potential accuracy loss by only changing the size of sub-matrices.

For formal discussions on the proposed inference and training procedures, let k denote the *block size* (size of each sub-matrix) and there are $p \times q$ blocks after partitioning $\mathbf{W}$, where $p = m \div k$ and $q = n \div k$. Zero padding is required if k does not directly divide m or n, but the amount of zero padding will be significantly reduced compared with (Cheng et al.). Then $\mathbf{W} = [\mathbf{C}_{ij}]$, $i \in \{1 \dots p\}$, $j \in \{1 \dots q\}$. Correspondingly, the input $\mathbf{x}$ is also partitioned as $\mathbf{x} = [\mathbf{x}_1^T, \mathbf{x}_2^T, \dots, \mathbf{x}_q^T]^T$. Then the forward propagation process in the inference phase is given by:

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \begin{bmatrix} \sum_{j=1}^{q} \mathbf{C}_{1j}\mathbf{x}_j \\ \sum_{j=1}^{q} \mathbf{C}_{2j}\mathbf{x}_j \\ \cdots \\ \sum_{j=1}^{q} \mathbf{C}_{pj}\mathbf{x}_j \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \cdots \\ \mathbf{a}_p \end{bmatrix}, \tag{1}$$

where $\mathbf{a}_i \in \mathbb{R}^k$ is a column vector. Assume each circulant matrix $\mathbf{C}_{ij}$ is defined by a vector $\mathbf{w}_{ij}$, i.e., $\mathbf{w}_{ij}$ is the first row vector of $\mathbf{C}_{ij}$. Then according to the *circulant convolution theorem* (Pan, Bini, Pan, and Eberly), the calculation of $\mathbf{C}_{ij}\mathbf{x}_j$ can be performed as $\mathrm{IFFT}\big(\mathrm{FFT}(\mathbf{w}_{ij}) \circ \mathrm{FFT}(\mathbf{x}_j)\big)$, where $\circ$ denotes element-wise multiplications. The operation procedure is shown in Fig. 1. For the inference phase, the computational complexity of this FC layer will be $O(pqk \log k)$, which is equivalent to $O(n \log n)$ for small p, q values. Similarly, the storage complexity will be $O(pqk)$ because we only need to store $\mathbf{w}_{ij}$ or $\mathrm{FFT}(\mathbf{w}_{ij})$ for each submatrix, which is equivalent to $O(n)$ for small p, q values. Simultaneous acceleration and model compression compared with the original DNN can be achieved.

Now consider the backward propagation process in the training phase. Let a_{il} be the l-th output element in $\mathbf{a}_i$. Then by using the chain rule we can derive the backward propagation process as follows:

$$\frac{\partial L}{\partial \mathbf{w}_{ij}} = \sum_{l=1}^{k} \frac{\partial L}{\partial a_{il}} \frac{\partial a_{il}}{\partial \mathbf{w}_{ij}} = \frac{\partial L}{\partial \mathbf{a}_i} \frac{\partial \mathbf{a}_i}{\partial \mathbf{w}_{ij}}, \tag{2}$$

$$\frac{\partial L}{\partial \mathbf{x}_j} = \sum_{i=1}^{p} \sum_{l=1}^{k} \frac{\partial L}{\partial a_{il}} \frac{\partial a_{il}}{\partial \mathbf{x}_j} = \sum_{i=1}^{p} \frac{\partial L}{\partial \mathbf{a}_i} \frac{\partial \mathbf{a}_i}{\partial \mathbf{x}_j}. \tag{3}$$

We have proved that $\frac{\partial \mathbf{a}_i}{\partial \mathbf{w}_{ij}}$ and $\frac{\partial \mathbf{a}_i}{\partial \mathbf{x}_j}$ are block-circulant matrices. Therefore, $\frac{\partial L}{\partial \mathbf{w}_{ij}}$ and $\frac{\partial L}{\partial \mathbf{a}_i}\frac{\partial \mathbf{a}_i}{\partial \mathbf{x}_j}$ can be calculated as the "FFT→element-wise multiplication→IFFT" procedure and is equivalent to $O(n \log n)$ computational complexity per layer. Due to space limitation, the algorithmic descriptions of forward and backward propagations are omitted.

Please note that there is no special need to translate into or approximate each sub-matrix of $\mathbf{W}$. Instead, as shown in

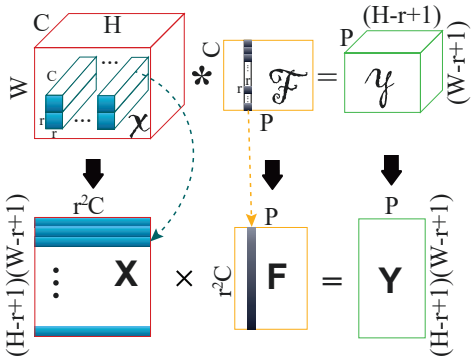

Figure 2: **Reformulation of Eqn. (4) to matrix multiplication.**

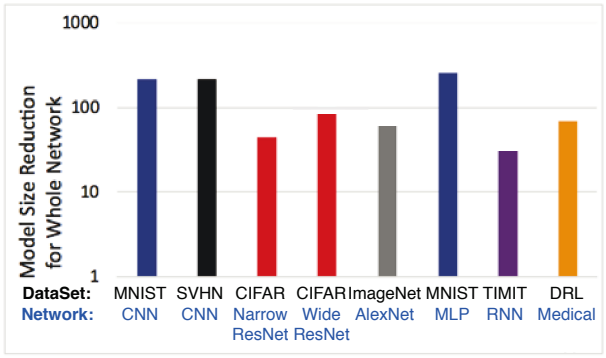

Figure 3: **Weight storage reduction results.**

Eqns. (2) and (3), we directly learn the vector $\mathbf{w}_{ij}$ (the first-row vector) of each sub-matrix of $\mathbf{W}$ in the training process. The assumption is that the other rows of the sub-matrix follow the circulant formulation. In other words, when following the learning process Eqns. (2) and (3), the learnt weight matrices naturally follow the block-circulant format. In fact, this is a key advantage of this proposed method in that there is no need for additional "translation" or "approximation" steps.

Inference and Training for CONV Layers

We generalize the inference and training algorithms to CONV layers, which have become the computation bottleneck of the whole DNN. The CONV layers are often associated with multiple input and multiple output feature maps:

$$\mathcal{Y}(x,y,p) = \sum_{i=1}^{r}\sum_{j=1}^{r}\sum_{c=1}^{C}\mathcal{F}(i,j,c,p)\mathcal{X}(x+i-1,y+j-1,c), \tag{4}$$

where $\mathcal{X} \in \mathbb{R}^{W\times H\times C}$, $\mathcal{Y} \in \mathbb{R}^{(W-r+1)\times(H-r+1)\times P}$, $\mathcal{F} \in \mathbb{R}^{r\times r\times C\times P}$ represent the input, output, and weight tensors of the CONV layer, respectively. Here W and H are the spatial dimensions of the input feature maps, C is the number of input feature maps, r is the size of the convolutional kernel, and P is the number of output feature maps.

Efficient software tools such as Caffe provide an efficient methodology of transforming tensor-based operations in the CONV layer to matrix-based operations (Jia et al., Vedaldi and Lenc), in order to enhance the implementation efficiency (GPUs are optimized for matrix operations.) Fig. 2 illustrates the application of the method to reformulate Eqn. (4) to the matrix multiplication $\mathbf{Y} = \mathbf{XF}$, where $\mathbf{X} \in \mathbb{R}^{(W-r+1)(H-r+1)\times Cr^2}$, $\mathbf{Y} \in \mathbb{R}^{(W-r+1)(H-r+1)\times P}$, and $\mathbf{F} \in \mathbb{R}^{Cr^2\times P}$.

We generalize the concept of "block-circulant structure" to the rank-4 tensor ($\mathcal{F}$) in the CONV layer, i.e., *all the slices of the form $\mathcal{F}(\cdot,\cdot,c,p)$ are block-circulant matrices.* Then we can prove that $\mathbf{F}$ is actually a block-circulant matrix. Hence the fast multiplication approach for block-circulant matrices, as the "FFT→component-wise multiplication →IFFT" procedure, can now be applied to accelerate $\mathbf{Y} = \mathbf{XF}$, thereby resulting in the acceleration of (4). The training phase can be derived similarly. The overall degrees of reduction in computational and storage complexities are similar to those in FC layers.

Theoretical Foundation and Software Results

With the substantial reduction of weight storage and computations, we also attempt to prove that the proposed block-circulant matrix-based framework will consistently yield the similar overall accuracy compared with DNNs without compression. The theoretical proof will make the proposed method theoretically rigorous and distinct from prior work.

In the theory of neural networks, the *universal approximation property* states that a neural network should be able to approximate any continuous or measurable function with arbitrary accuracy provided that an enough large number of parameters are available. This property provides the theoretical guarantee of using neural networks to solve machine learning problems, since machine learning tasks can be formulated as finding a proper approximation of an unknown, high-dimensional function. We have proved the universal approximation property of block circulant matrix-based neural networks, and more generally, for arbitrary structured matrices satisfying the low displacement rank γ. As a result, we can guarantee the universal "effectiveness" of the proposed framework on different DNN types and sizes, application domains, and hardware/software platforms. Detailed proof procedure is provided in the supplementary file (pro).

Fig. 3 shows the model compression results on MNIST, SHVN, CIFAR-10, ImageNet, TIMIT (speech recognition) benchmarks, etc., using various DNN models. The accuracy degradations are constrained to be 1% to 2% between the original models and block-circulant matrix-based models. The overall model compression is contributed by both weight parameter reduction and bit quantization. It can be observed that a significant model size compression, and therefore acceleration, can be achieved using the proposed framework.

Accelerating Computation and Facilitating Hardware Implementations

We propose the decoupling technique of FFTs and IFFTs, which applies to both inference and training phases. We take the inference phase of FC layer as an illustrative example. First, we make the observation that the FFT results of $\mathbf{x}_j$, i.e., $\text{FFT}(\mathbf{x}_j)$, need to be utilized to calculate all $\mathbf{a}_i$ vectors. Similar observation also holds for $\mathbf{w}_{ij}$. Hence, we could perform pre-calculation of $\text{FFT}(\mathbf{x}_j)$ and $\text{FFT}(\mathbf{w}_{ij})$ and store them in memory for effective re-use. The $\text{FFT}(\mathbf{w}_{ij})$ values can even be pre-calculated and stored in memory before the inference phase because they are fixed after training. By performing such pre-calculation of $\text{FFT}(\mathbf{x}_j)$, the total number of FFTs needed to calculate $\mathbf{Wx}$ reduces from $p \cdot q$ to q (assuming $\text{FFT}(\mathbf{w}_{ij})$'s are calculated and stored in prior), achieving a significant reduction in total computations.

Similarly, each vector $\mathbf{a}_i$ to be calculated in Eqn. (1) is given by $\sum_{j=1}^{q} \text{IFFT}\big(\text{FFT}(\mathbf{w}_{ij}) \circ \text{FFT}(\mathbf{x}_j)\big)$, which requires q IFFT calculations. Because FFTs and IFFTs are linear operations (Oppenheim), we can calculate IFFT in the last step, i.e., calculate $\mathbf{a}_i$ as $\text{IFFT}\big(\sum_{j=1}^{q} \text{FFT}(\mathbf{w}_{ij}) \circ \text{FFT}(\mathbf{x}_j)\big)$. In this way the total number of IFFT calculations can be reduced by q times.

High Energy Efficiency and Performance Implementation in FPGAs

Based on the algorithmic framework, we describe the developed high-efficiency FPGA-based implementation of DNNs. Since the target is low-power embedded applications, we focus on the inference phase of small to medium-scale DNNs, e.g., for MNIST, SVHN, CIFAR datasets. We leave the large-scale DNNs, e.g., for ImageNet dataset, for future investigation because they do not target at embedded applications. We first describe the proposed FPGA implementations using a set of reconfiguration and performance/efficiency enhancement techniques, then present the algorithm-hardware co-optimization framework.

FPGA Implementations: Reconfigurability, In-Place Computation, Batch Processing, Deep Pipelining, and Resource Re-Use

Reconfigurability, In-Place Computation, and Batch Processing. In order to accommodate different DNN models, sizes, and application scenarios, the proposed FPGA implementation possesses *reconfigurability* for different layer sizes and layer types (FC or CONV layers). The reconfigurability is achieved because (i) both FC and CONV layers are formulated as the "FFT→component-wise multiplication →IFFT" procedure; (ii) IFFT can be implemented using the FFT structure with simple pre-processing step (Salehi, Amirfattahi, and Parhi); and (iii) the FFT structure possesses inherent *recursive property* in that small-scale FFTs can be implemented in parallel in larger-scale FFT structures (Oppenheim). More specifically, the first and second properties enable the implementation of a single FFT structure in a time-multiplexed manner for both FFTs and IFFTs and both FC and CONV layers. For instance, a 128-input FFT structure can be implemented in FPGA if a block size of 128 is utilized. The third property enables that a single FFT structure can be utilized even if we use different block sizes for FC and CONV layers. Finally, *in-place computation* is utilized such that the same memory space can be utilized to store the outputs of every layer in the DNN, i.e., the outputs of each neuron layer i will replace the inputs (outputs of layer $i-1$). In this way, the execution of an overall DNN will use the single FFT structure in a sequential, time-multiplexed manner without extra memory requirements.

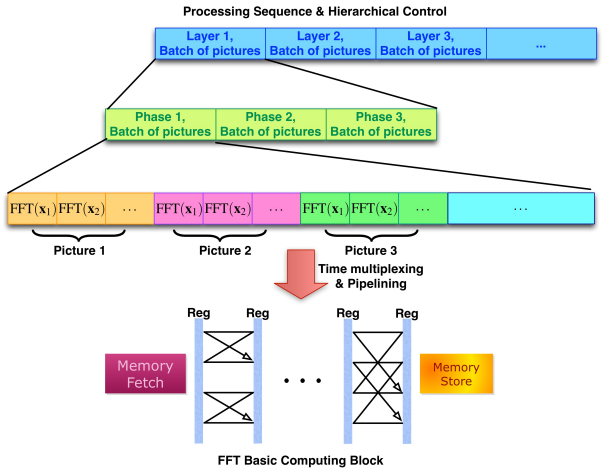

Figure 4: **The execution of the whole DNN inference phase in FPGA.**

The execution of the inference phase of the whole DNNs is shown in Fig. 4. The *batch processing* technique is utilized, in that a batch of input pictures are processed in an interleaved manner in the FPGA. As shown in Fig. 4, we first compute the first layer of all input pictures in this batch, then the second layer, and so on. Different layers of a neural network will be time-multiplexed on the basic block. The computations are all based on the implemented FFT structure discussed previously in a time-multiplexed manner. All operations will be pipelined on the basic computing block. The reason of batch processing is the *deep pipelining* (to be discussed later) utilized in the hardware implementation. Otherwise, pipeline bubbles have to be injected when computing all layers for one input picture consecutively, which results in timing overheads. A typical batch consists of around 50-100 pictures, because (i) state-of-the-art FPGAs have more than 2MB on-chip memory storage (e.g., Intel (Altera) CyClone V 5CEA9, Xilinx Kintex-7 XC7K325T) and (ii) the intermediate results of small to medium-scale DNNs (e.g., DNNs for CIFAR-10) typically take several KBs per picture.

Three-Phase Operations, Deep Pipelining, and Resource Re-Use. As described before, the calculation of $\mathbf{Wx}$ consists of three phases: calculation of $\text{FFT}(\mathbf{x}_j)$ vectors for each j, calculation of element-wise multiplications $\text{FFT}(\mathbf{w}_{ij}) \circ \text{FFT}(\mathbf{x}_j)$ for each i, j (and corresponding additions), and IFFTs for each i. For example, if $\mathbf{W}$ is 1024-by-1024 and the block size is 128, a total of 8 FFTs, 8 IFFTs, and 64 groups of element-wise multiplications will be performed. As shown in Fig. 4, the three-phase operations are

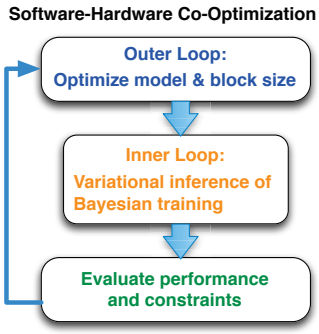

Figure 5: **The software-hardware co-optimization of model and block size, and Bayesian training.**

integrated with batch processing. More specifically, an outer loop iterates on all layers of the DNN. Within the outer loop is the three calculation phases. Within each phase is the calculations for every i, j in each picture and for all pictures. In this way the timing overheads can be minimized to close to zero.

The *deep pipelining* technique is utilized for FFTs and IFFTs in order to improve throughput and energy efficiency, as illustrated in Fig. 4. For example, if a 128-point FFT is implemented as the basic computing block in FPGA, it needs 7 pipeline stages plus 4 additional stages corresponding to memory reading and writing. When IFFT is implemented on such basic computing block, 2 additional stages are needed corresponding to the preprocessing, and biasing and ReLU activation. The element-wise multiplications and additions in the second phase are also pipelined.

One clear advantage of the FPGA-based hardware implementation is the ability of *resource re-use*. Besides the effective time multiplexing of FFTs and IFFTs on the same hardware, the hardware multipliers utilized in the second phase can also re-use those in the FFT computing block. This effective resource re-use can be automatically determined in the FPGA synthesis process (qua), which could improve the area and energy efficiency of FPGA implementations.

Algorithm-Hardware Co-Optimizations

Finally, an algorithm-hardware co-optimization framework is developed, which comprises (i) model selection and optimization, (ii) hardware optimization, and (iii) variational inference-based Bayesian learning. The overall objective is to maximize the performance (throughput) and energy efficiency of FPGA hardware implementation subject to certain accuracy requirements. More specifically, the first aspect determines the proper block size and weight matrix size, in order to facilitate FPGA-based FFT implementations while satisfying the overall accuracy requirement. For state-of-the-art FPGAs, a proper block size ranges from 64 to 256 (should better be a power of 2) for FC layers and may be smaller for CONV layers. The second aspect includes the exploitation of FFTs with real-valued inputs, i.e., the FFT results of a real-valued vector is symmetric except for the base (first) component (Oppenheim). Because both $\mathbf{x}_j$ and $\mathbf{w}_{ij}$ are real-valued vectors, we only need to store the first half of vectors FFT($\mathbf{x}_j$) and FFT($\mathbf{w}_{ij}$), which significantly reduce the storage requirement and computations required in element-wise multiplications. The last aspect uses the variational inference process of Bayesian learning (Blei, Jordan, and others), which is compatible with the proposed framework and can result in accuracy and robustness enhancements. Bayesian training using variational inference (Blei, Jordan, and others) is an effective training method to enhance accuracy and robustness of machine learning systems, including neural networks. During training phase, it assumes that each weight is a variable that satisfies certain prior distribution at the beginning. For each training sample, it generates a collection of random weights based on the distribution, and learns both the average and variance of each weight variable. The inference phase (implemented in hardware) will be the same, using the average estimate of each weight. Based on our results, Bayesian training is the most effective for small data training and small-to-medium neural networks. The algorithm-hardware co-optimization framework is shown in Fig. 5. Overall, the proposed FPGA-based implementation can accommodate the whole DNN model using on-chip block memory, thereby significantly improving the overall energy efficiency.

Experimental Results

In this section, we provide the experimental results on FPGA implementations of the proposed framework on small to medium-scale DNNs, using MNIST, SVHN, and CIFAR-10 benchmarks. Our FPGAs for implementation include the low-power FPGA Intel (Altera) CyClone V 5CEA9, and the one with higher performance Xilinx Kintex-7 XC7K325T. The former one is the default FPGA used in experiments. We compare the performance (throughput), energy efficiency, and accuracy with the best state-of-the-arts including IBM TrueNorth neurosynaptic processor, emerging device (e.g., memristor crossbar) based neuromorphic systems, analog-based neuromorphic systems, and reference FPGA implementations. IBM TrueNorth (Esser et al., Esser et al.) is a neuromorphic CMOS chip fabricated in 28nm technology, with 4096 cores each simulating 256 programmable silicon neurons in a time-multiplexed manner. It implements the *spiking neural network*, which is a bio-inspired type of neural networks and benefits from the ability of globally asynchronous implementations. It can accommodate MNIST, SVHN, and CIFAR-10 benchmarks[1] in the experiments.

First, we provide the comparison results on accuracy, performance (throughput, in kilo-frames per second (kFPS)), and energy efficiency (in kFPS/W) on the three benchmarks, as shown in Table 1. The baselines include IBM TrueNorth processor and reference FPGA implementations of these benchmarks. We provide results of the proposed framework on three DNNs of MNIST data set with different target accuracies, one for SVHN, and two for CIFAR-10 data set. The first two DNNs of the MNIST data set are multi-layer perceptron (MLP) models that achieve 92.9% and 95.6% accuracies, respectively. Prior pooling is applied to reduce the

[1]Please note that ImageNet is not currently supported by IBM TrueNorth due to the high-degree neural connections.

input size to 256 and 128, respectively. The third DNN of the MNIST data set is a CNN similar to the LeNet-5 structure (LeCun et al.). The baseline IBM TrueNorth processor also has different implementations with different accuracy levels for the MNIST data set. For the CIFAR-10 data set, the first DNN is a simple CNN structure, whereas the second is a wide ResNet model (He et al.) that can achieve 94.75% accuracy, only 0.75% lower than the best state-of-the-art software implementation. We can observe that under the similar accuracy level, the speedup and energy efficiency gain compared with IBM TrueNorth are at least 152X and 71X, respectively. Under the similar accuracy level, the energy efficiency gain is at least 31X compared with the reference FPGA-based implementation that achieves the highest energy efficiency (Umuroglu et al.) (using binary neural networks). Besides the reduction in computational complexity, the high suitability of the proposed framework for hardware implementation, and the highly efficient deep pipelined hardware structure, the reasons for such significant gains also include the requirement of increasing neuron numbers for spiking or binary neural networks to achieve the same accuracy as MLP or CNN, and the inherent long latency in spiking neural networks.

Next, we provide sample comparison results with emerging device and analog-based implementations. Because the neural networks and applications may be different, we use the equivalent performance in giga-operations per second (GOPS) and energy efficiency in GOPS/W for fair comparisons. The term "equivalent" is utilized because we normalize the number of (multiplication and addition) operations to the original matrix-vector multiplication format. The proposed framework achieves around 5.14 Tera OPS/W (TOPS/W) energy efficiency, which outperforms representative latest results using analog computing and emerging devices. For example, (Shafiee et al., Song et al., Lu et al.) achieve 380.7 GOPS/W, 142.9 GOPS/W, and 1.04 TOPS/W, respectively. The reference work can be either manufactured or device modeling based. Performance wise, as analyzed in (Bayat et al., Liu et al., Li et al.), a matrix-vector multiplication will take around 100ns and it takes around $1\mu s$ to perform one inference sample on the MNIST data set (with 90% - 94% accuracy). Our achieved highest performance (throughput) for the MNIST data set, i.e., 11.6ns per image recognition in CyClone V FPGA or around 4ns per image in Kintex-7 FPGA, is difficult to achieve even using emerging devices and technology.

Finally, we provide the comparison results with other FPGA implementations in terms of the equivalent performance (in GOPS) and energy efficiency (in GOPS/W), as shown in Fig. 6. These metrics are relatively fair comparisons although the DNNs for implementations may be different. The baseline FPGA implementations include high-level synthesis-based implementations, implementations of compressed models, etc. A minimum of more than 84X energy efficiency gain can be achieved compared with the reference FPGA implementations. Besides the reduced computational complexity and the high-efficiency hardware implementation, another key reason for such significant energy efficiency gain is because the proposed FPGA-based implementation can accommodate the whole DNN model using on-chip block memory, thereby significantly improving the overall energy efficiency.

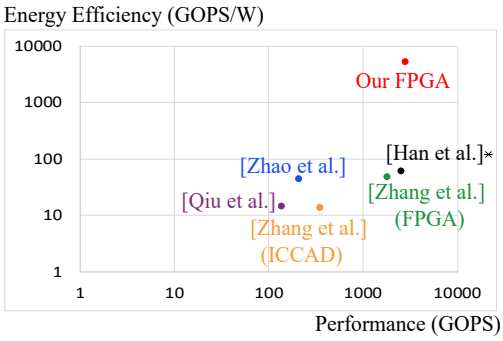

Figure 6: **Comparisons of different FPGA implementations on performance (throughput) and energy efficiency.**

Conclusion

This paper presents an algorithm-hardware co-optimization framework to facilitate ultra high-performance and high energy efficiency hardware implementations of DNNs on FPGAs. The algorithm part adopts the general block-circulant matrices to achieve a fine-grained tradeoff of accuracy and compression ratio. It applies to both FC and CONV layers and contains a mathematically rigorous proof. The proposed algorithm reduces computational complexity per layer from O(n^2) to O($n \log n$) and storage complexity from O(n^2) to O(n), both for training and inference phases. The hardware part consists of highly efficient FPGA-based implementations using effective reconfiguration, batch processing, deep pipelining, resource re-using, and a hierarchical control framework. Experimental results demonstrate that the proposed framework achieves at least 152X speedup in throughput and 71X energy efficiency gain compared with IBM TrueNorth processor under the same test accuracy. It achieves at least 31X energy efficiency gain compared with the reference FPGA-based work.

Acknowledgement

This work is funded by the National Science Foundation Awards CNS-1650469, CCF-1733701, CNS-1704662, CCF-1657333, CNS-1739748, and CCF-1733834.

References

Alemdar, H.; Caldwell, N.; Leroy, V.; Prost-Boucle, A.; and Pétrot, F. 2016. Ternary neural networks for resource-efficient ai applications. *arXiv preprint arXiv:1609.00222*.

Bayat, F. M.; Guo, X.; Klachko, M.; Prezioso, M.; Likharev, K.; and Strukov, D. 2016. Sub-1-us, sub-20-nj pattern classification in a mixed-signal circuit based on embedded 180-nm floating-gate memory cell arrays. *arXiv preprint arXiv:1610.02091*.

Table 1: Comparison results on accuracy, performance, and energy efficiency of the proposed FPGA designs and baselines.

Name	Dataset	Platform	Precision	Accuracy	Performance (kFPS)	Energy eff. (kFPS/W)
Proposed MNIST 1	MNIST	CyClone V	12	92.9%	8.6×10^4	1.57×10^5
Proposed MNIST 2	MNIST	CyClone V	12	95.6%	2.9×10^4	5.2×10^4
Proposed MNIST 3	MNIST	CyClone V	12	99.0%	363	659.5
Proposed SVHN	SVHN	CyClone V	12	96.2%	384.9	699.7
Proposed CIFAR-10 1	CIFAR-10	CyClone V	12	80.3%	1383	2514
Proposed CIFAR-10 2	CIFAR-10	CyClone V	12	94.75%	13.95	25.4
TrueNorth (Esser et al.)	MNIST	TrueNorth	2	99%+	1.0	9.26
TrueNorth (Esser et al.)	MNIST	TrueNorth	2	95%	1.0	250
TrueNorth (Esser et al.)	SVHN	TrueNorth	2	96.7%	2.53	9.85
TrueNorth (Esser et al.)	CIFAR-10	TrueNorth	2	83.4%	1.25	6.11
Umuroglu et al. (Umuroglu et al.)	MNIST	ZC706	1	95.8%	1.23×10^4	1693
Umuroglu et al. (Umuroglu et al.)	SVHN	ZC706	1	94.9%	21.9	6.08
Umuroglu et al. (Umuroglu et al.)	CIFAR-10	ZC706	1	80.1%	21.9	6.08
Alemdar et al. (Alemdar et al.)	MNIST	Kintex-7	2	98.3%	255.1	92.59

Bini, D.; Pan, V.; and Eberly, W. 1996. Polynomial and matrix computations volume 1: Fundamental algorithms. *SIAM Review*.

Blei, D. M.; Jordan, M. I.; et al. 2006. Variational inference for dirichlet process mixtures. *Bayesian analysis* 1(1):121–143.

Burbidge, R.; Trotter, M.; Buxton, B.; and Holden, S. 2001. Drug design by machine learning: support vector machines for pharmaceutical data analysis. *Computers & chemistry* 26(1):5–14.

Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; and Temam, O. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*.

Chen, Y.-H.; Krishna, T.; Emer, J. S.; and Sze, V. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52(1).

Cheng, Y.; Yu, F. X.; Feris, R. S.; Kumar, S.; Choudhary, A.; and Chang, S.-F. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *ICCV*, 2857–2865.

Collobert, R., and Weston, J. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, 160–167. ACM.

http://www.techradar.com/news/computing-components/processors/google-s-tensor-processing-unit-explained-this-is-what-the-future-of-computing-looks-like-1326915.

https://www.sdxcentral.com/articles/news/intels-deep-learning-chips-will-arrive-2017/2016/11/.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*, 248–255. IEEE.

Denil, M.; Shakibi, B.; Dinh, L.; de Freitas, N.; et al. 2013. Predicting parameters in deep learning. In *NIPS*, 2148–2156.

Denton, E. L.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 1269–1277.

Ding, C.; Liao, S.; Wang, Y.; Li, Z.; Liu, N.; Zhuo, Y.; Wang, C.; Qian, X.; Bai, Y.; Yuan, G.; et al. 2017. C ir cnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 395–408. ACM.

Esser, S. K.; Appuswamy, R.; Merolla, P.; Arthur, J. V.; and Modha, D. S. 2015. Backpropagation for energy-efficient neuromorphic computing. In *NIPS*, 1117–1125.

Esser, S. K.; Merolla, P. A.; Arthur, J. V.; Cassidy, A. S.; Appuswamy, R.; Andreopoulos, A.; Berg, D. J.; McKinstry, J. L.; Melano, T.; Barch, D. R.; et al. 2016. Convolutional networks for fast, energy-efficient neuromorphic computing. *NAS* 201604850.

Farabet, C.; Poulet, C.; Han, J. Y.; and LeCun, Y. 2009. Cnp: An fpga-based processor for convolutional networks. In *FPL*, 32–37.

Feng, J., and Darrell, T. 2015. Learning the structure of deep convolutional networks. In *ICCV*, 2749–2757.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *NIPS*, 1135–1143.

Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M. A.; and Dally, W. J. 2016. Eie: efficient inference engine on compressed deep neural network. In *ISCA*, 243–254. IEEE Press.

Han, S.; Kang, J.; Mao, H.; Hu, Y.; Li, X.; Li, Y.; Xie, D.; Luo, H.; Yao, S.; Wang, Y.; et al. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *FPGA*, 75–84. ACM.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Huval, B.; Wang, T.; Tandon, S.; Kiske, J.; Song, W.; Pazhayampallil, J.; Andriluka, M.; Rajpurkar, P.; Migimatsu, T.; Cheng-Yue, R.; et al. 2015. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*.

Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; and Darrell, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *MM*, 675–678. ACM.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

LeCun, Y.; Jackel, L.; Bottou, L.; Brunot, A.; Cortes, C.; Denker, J.; Drucker, H.; Guyon, I.; Muller, U.; Sackinger, E.; et al. 1995. Comparison of learning algorithms for handwritten digit recognition. In *ICANN*, volume 60, 53–60. Perth, Australia.

Li, S.; Liu, X.; Mao, M.; Li, H. H.; Chen, Y.; Li, B.; and Wang, Y. 2016. Heterogeneous systems with reconfigurable neuromorphic computing accelerators. In *ISCAS*, 125–128. IEEE.

Li, S.; Park, J.; and Tang, P. T. P. 2017. Enabling sparse winograd convolution by native pruning. *arXiv preprint arXiv:1702.08597*.

Lin, Z.; Courbariaux, M.; Memisevic, R.; and Bengio, Y. 2015. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*.

Lin, D.; Talathi, S.; and Annapureddy, S. 2016. Fixed point quantization of deep convolutional networks. In *ICML*, 2849–2858.

Liu, X.; Mao, M.; Liu, B.; Li, B.; Wang, Y.; Jiang, H.; Barnell, M.; Wu, Q.; Yang, J.; Li, H.; et al. 2016. Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators. *IEEE Transactions on Circuits and Systems I: Regular Papers* 63(5):617–628.

Lu, J.; Young, S.; Arel, I.; and Holleman, J. 2015. A 1 tops/w analog deep machine-learning engine with floating-gate storage in 0.13 μm cmos. *IEEE Journal of Solid-State Circuits* 50(1).

Mathieu, M.; Henaff, M.; and LeCun, Y. 2013. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*.

Merolla, P. A.; Arthur, J. V.; Alvarez-Icaza, R.; Cassidy, A. S.; Sawada, J.; Akopyan, F.; Jackson, B. L.; Imam, N.; Guo, C.; Nakamura, Y.; et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345(6197):668–673.

Oppenheim, A. V. 1999. *Discrete-time signal processing*. Pearson Education India.

Pan, V. 2012. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media.

`https://drive.google.com/open?id=0B19Xkz1gXlwAYjVjWC1Kc2xSRm8`.

Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S.; et al. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *FPGA*, 26–35.

`https://dl.altera.com`.

Salehi, S. A.; Amirfattahi, R.; and Parhi, K. K. 2013. Pipelined architectures for real-valued fft and hermitian-symmetric ifft with real datapaths. *IEEE Transactions on Circuits and Systems II: Express Briefs* 60(8):507–511.

Shafiee, A.; Nag, A.; Muralimanohar, N.; Balasubramonian, R.; Strachan, J. P.; Hu, M.; Williams, R. S.; and Srikumar, V. 2016. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ISCA*, 14–26. IEEE Press.

Song, L.; Qian, X.; Li, H.; and Chen, Y. 2017. Pipelayer: A pipelined reram-based accelerator for deep learning. In *HPCA*.

Suda, N.; Chandra, V.; Dasika, G.; Mohanty, A.; Ma, Y.; Vrudhula, S.; Seo, J.-s.; and Cao, Y. 2016. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In *FPGA*, 16–25. ACM.

Taigman, Y.; Yang, M.; Ranzato, M.; and Wolf, L. 2014. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 1701–1708.

Umuroglu, Y.; Fraser, N. J.; Gambardella, G.; Blott, M.; Leong, P.; Jahre, M.; and Vissers, K. 2016. Finn: A framework for fast, scalable binarized neural network inference. *arXiv preprint arXiv:1612.07119*.

Vedaldi, A., and Lenc, K. 2015. Matconvnet: Convolutional neural networks for matlab. In *MM*, 689–692. ACM.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *NIPS*, 2074–2082.

Zhang, J., and Li, J. 2017. Improving the performance of opencl-based fpga accelerator for convolutional neural network. In *FPGA*.

Zhang, C.; Fang, Z.; Zhou, P.; Pan, P.; and Cong, J. 2016a. Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks. In *ICCAD*, 12. ACM.

Zhang, C.; Wu, D.; Sun, J.; Sun, G.; Luo, G.; and Cong, J. 2016b. Energy-efficient cnn implementation on a deeply pipelined fpga cluster. In *ISLPED*, 326–331. ACM.

Zhao, R.; Song, W.; Zhang, W.; Xing, T.; Lin, J.-H.; Srivastava, M.; Gupta, R.; and Zhang, Z. 2017. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *FPGA*, 15–24. ACM.