DANoC: An Efficient Algorithm and Hardware Codesign of Deep Neural Networks on Chip

Xichuan Zhou, *Member, IEEE,* Shengli Li, Fang Tang, *Member, IEEE,*
Shengdong Hu, Zhi Lin, and Lei Zhang, *Member, IEEE*

***Abstract*— Deep neural networks (NNs) are the state-of-the-art models for understanding the content of images and videos. However, implementing deep NNs in embedded systems is a challenging task, e.g., a typical deep belief network could exhaust gigabytes of memory and result in bandwidth and computational bottlenecks. To address this challenge, this paper presents an algorithm and hardware codesign for efficient deep neural computation. A hardware-oriented deep learning algorithm, named the deep adaptive network, is proposed to explore the sparsity of neural connections. By adaptively removing the majority of neural connections and robustly representing the reserved connections using binary integers, the proposed algorithm could save up to 99.9% memory utility and computational resources without undermining classification accuracy. An efficient sparse-mapping-memory-based hardware architecture is proposed to fully take advantage of the algorithmic optimization. Different from traditional Von Neumann architecture, the deep-adaptive network on chip (DANoC) brings communication and computation in close proximity to avoid power-hungry parameter transfers between on-board memory and on-chip computational units. Experiments over different image classification benchmarks show that the DANoC system achieves competitively high accuracy and efficiency comparing with the state-of-the-art approaches.**

***Index Terms*— Binary weights, deep belief network (DBN), deep learning, embedded system, field-programmable gate array (FPGA), sparse connections.**

I. Introduction

THE deep neural networks (NNs) have demonstrated their remarkable performance for feature extraction and pattern recognition in the last a few years [1]–[4]. However, due to the contradiction between limited hardware resources and the requirement of high computational performance, it is still a challenge to implement large-scale deep NNs for embedded real-time applications [5]. For example, research indicates that the unsupervised deep belief network (DBN) shows the state-of-the-art accuracy for classifying hyperspectral remote sensing images [3]. However, it is technically impractical to deploy the DBN to process remote sensing images in real time, because the embedded systems carried by satellites or unmanned aerial vehicles are generally limited in memory space, computational resources, and power budget.

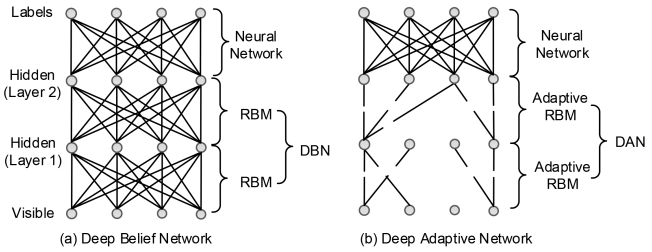


Fig. 1. Typical deep NNs implemented for pattern recognition applications. (a) DBN. (b) DAN. The neurons of the DBN are fully connected between adjacent layers, whereas in a DAN, the majority of neural connections (with zero weights) can be removed and the reserved connections can be represented using single-bit integers, which is much more efficient for hardware implementation.

The fundamental computations of a feedforward DBN involve a large number of high-precision multiplications between the connection weights and the input data, which can be implemented using clusters of central processing units (CPUs) or general-purpose graphics processing units (GPUs) in powerful computers [6]. However, when memory and computational resources are limited in hardware, a more efficient algorithm would be ideal, one that is designed for efficient hardware computing, and only requires simple fix-point computation and much fewer parameters, allowing larger networks to be implemented using system on chips (SoCs) for real-time pattern recognition.

This paper presents an algorithm-and-hardware codesign of the widely applied DBN for embedded real-time image classification. For efficient hardware implementation, one important algorithmic consideration is the number of neural connections. The classic DBN has fully connected neurons between adjacent layers (Fig. 1), resulting in high memory and computational complexity [7]. The second algorithmic consideration is data representation, which is an essential tradeoff between accuracy and cost. Studies indicate that the DBNs trained with parameters of limited precision suffer from significant loss of accuracy [8]–[10]. To address these challenges, an efficient training algorithm, named the DAN,

Manuscript received June 15, 2016; revised December 19, 2016 and April 11, 2017; accepted June 14, 2017. This work was supported in part by the National Natural Science Foundation of China under Contract 61471073, Contract 61401048, Contract 61404016, and Contract 61471071, and in part by the Fundamental Research Funds for the Central Universities under Project 106112017CDJQJ168818 and Project 106112016CDJZR168803. *(Corresponding author: Xichuan Zhou.)*

X. Zhou is with the Key Laboratory of Dependable Service Computing in Cyber Physical Society of Ministry of Education, College of Communication Engineering, Chongqing University, Chongqing 400044, China, and also with the Chongqing Engineering Laboratory of High Performance Integrated Circuits, College of Communication Engineering, Chongqing University, Chongqing 400044, China (e-mail: zxc@cqu.edu.cn).

S. Li, F. Tang, S. Hu, Z. Lin, and L. Zhang are with the Chongqing Engineering Laboratory of High Performance Integrated Circuits, College of Communication Engineering, Chongqing University, Chongqing 400044, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2017.2717442

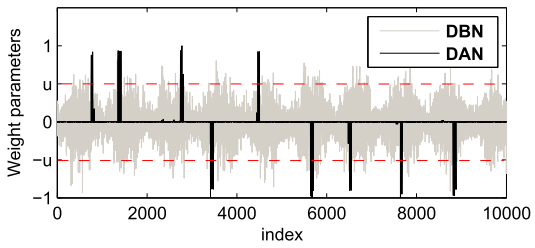


Fig. 2. Connection weights of the DBN and the DAN trained with the MNIST data set. The DAN weights are sparse and separated as three groups, i.e., the zero weights, the positive weights, and the negative weights, leading to robust thresholding results and single-bit representation (± 1).

is proposed to explore the sparsity of neural connections. The DAN adaptively reduces the values of connection weights associated with negligible neurons to zeros (Fig. 2) and robustly quantizes the small proportion of reserved connections using single-bit integers. A novel sparse-mapping-memory (SMM)-based architecture is designed to integrate the DAN on chip (DANoC). The characteristics of the DANoC coprocessor are summarized as follows.

1) *Memory Efficient:* The majority of the neural connections are removed by the DAN algorithm. Experiments show that over 80% of neural connections can be removed without degrading the classification accuracy. The reserved connection weights can be robustly quantized and represented using single-bit integers. Compared with the single-precision DBN, the proposed method could reduce the memory and computational resources by up to 99.9%.
2) *Power Efficient:* High memory efficiency enables the DANoC to reserve all the parameters on chip and reduces the power-hungry transfer operations between on-board memory and the DANoC coprocessor. Furthermore, the DANoC adopts an event-driven architecture, where the computation core is active only if the incoming visible unit is one.
3) *Computationally Efficient:* The single-bit representation allows the DANoC to replace the complicated high-precision multipliers with fast area-efficient accumulators, which further relieves the computational bottleneck.
4) *Scalable and Pipelined:* The DANoC hardware is flexible and scalable in three ways. First, multiple layers of sparse neural cores (SNCs) can be concatenated as a pipeline to form a multilayer deep NN. Second, multiple pipelines are integrated in a chip for parallel acceleration. Third, the hardware implementation of the SNC is optimized and designed as a four-stage subpipeline for high-throughput applications.

With algorithm and hardware optimization, the DANoC could achieve the state-of-the-art performance of over 2000 effective giga-operations per second with less than 2 W of power consumption. The rest of this paper gives more detailed information of the proposed approach and is organized as follows. Section II introduces the related work. Section III presents the proposed algorithm. Section IV describes the hardware design of the DANoC prototype. Section V presents the experiments. We conclude this paper in Section VI.

II. Related Work

Our method explores the sparsity of neural connections via a mixed norm-based regularization approach. It is a standard approach to achieve sparsity via L_1 norm-based regularization. One famous example is the group lasso approach proposed by Yuan and Lin [11]. Our method can be seen as an extension of the group lasso for deep NNs. Different from the group lasso, this paper compresses the weight parameters using a mixed norm regularization, which allows us to control the tradeoff between rowwise and columnwise sparsity in order to achieve the highest compression rate.

Other attempts have been made to introduce sparsity into deep NNs. Ranzato *et al.* [13] proposed a deep encoder–decoder architecture to learn sparse representations. Lee *et al.* [14] developed a variant of the DBN to learn the sparse representations of the input images and found that the selected sparse features had some properties similar to visual area V2. Ji *et al.* [15] proposed a sparse-response DBN based on the rate-distortion theory, which attempted to encode the original data using as few bits as possible. Generally speaking, these studies focused on the sparsity of output activations; however, motivated by learning efficient architectures for hardware implementation, this paper focused on exploring the sparsity of neural connections.

Previous studies have been attempted to incorporate ternary neural connections in traditional NNs [16], [17]. Very recently, there has been a growth of interest to compress the deep NNs at algorithmic level for embedded applications. For example, Han *et al.* [18] proposed a two-step training procedure to remove the small connection weights in the NNs. Chen *et al.* [19] used a hash function to group neural connections into different hash buckets according to different weight values. Han *et al.* [20] proposed a three-step method to prune, quantize and code the connection weights during the training process of the deep NNs. Courbariaux *et al.* [21] proposed a training process to learn the deep NNs with weights and activations constrained to $+1$ or -1. Rastegari *et al.* [22] proposed a binary deep NN, which quantized the weight parameters in each iteration of the training process. Generally speaking, these algorithms attempted to reduce the number of bits needed to represent the parameters, which yielded 1.2–49 fold of improvement in memory efficiency at the cost of degrading classification accuracy.

As a more efficient strategy, the DAN algorithm also allows the connections and the activations to be represented using single-bit integers. Moreover, the DAN learns the optimal architecture of a sparse NN where the majority of neural connections are removed. Similar ideas have been recently demonstrated by Alvarez and Salzmann [23] and Pan *et al.* [24], who proposed different regularization approaches to reduce the number of neuron connections in a deep network by up to 80% during training. Wen *et al.* [25] proposed a structured sparsity learning method, which learned a compact structure from a bigger deep NN and reduced computational cost by 5.1-fold. As the result of our two-step (regularize-and-quantize) strategy, the DAN hardware could achieve 160-to-640 fold compression rate while still achieving competitively high accuracy comparing with the state-of-the-art approaches.

Thanks to the fast development of the deep learning approaches, the research of implementing dedicated hardware to accelerate NN computation is booming. Himavathi and Ahn presented respective digital implementations using reconfigurable field-programmable gate arrays (FPGAs) [7], [26]. Sanni *et al.* [27] presented an FPGA-based DBN using stochastic computation. Zhang *et al.* [28] quantitatively analyzed the convolutional NN (CNN) and implemented the quantized CNN using the FPGA device. Recently, Gokhale *et al.* [29] implemented an FPGA-based coprocessor to accelerate the deep NNs for mobile applications. Performed independently of the algorithmic researches, these hardware designs generally adopted different quantization approaches for embedded implementation, which saved memory footprint but resulted in notable loss of classification accuracy. On the other hand, the proposed DANoC system was optimized from algorithm to hardware, which led to over three orders of magnitude of improvement in hardware efficiency without degrading classification accuracy.

III. Methods

This section presents the proposed hardware-oriented unsupervised deep learning algorithm. As the background knowledge, we first introduce the training algorithm of the famous DBN.

A. Deep Belief Network and Restricted Boltzmann Machine

A DBN is constructed by stacking multiple layers of restricted Boltzmann machines (RBMs) and using the output of the previous-layer RBM as the input of the next-layer RBM (Fig. 1). It is found that the higher layer RBM tends to encode informative abstraction for classification. A standard RBM consists of two layers of units: first, a matrix $\mathbf{W} \in \mathbb{R}^{n\times d}$ is defined as the connection weights, where w_{ij} represents the connection between the visible unit v_i and the hidden unit h_j. Second, the parameters b_j and c_i are the biases for the hidden and visible units, respectively. Given the vector forms of the hidden units $\mathbf{h}$, the visible units $\mathbf{v}$, and the biases $\mathbf{b}$ and $\mathbf{c}$, the energy of a configuration ($\mathbf{v}and\mathbf{h}$) can be written as

$$E(\mathbf{v},\mathbf{h}) = -\mathbf{b^T h} - \mathbf{c^T v} - \mathbf{v^T W h}. \tag{1}$$

As in general Boltzmann machines, the probability distributions over the hidden and visible vectors are defined as

$$p(\mathbf{v},\mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}, \quad Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}. \tag{2}$$

Given (2), the marginal probability of the visible vector is

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}. \tag{3}$$

Since there are no direct connections between two hidden units at the same layer, the hidden units conditioned on $\mathbf{v}$ are independent of each other. Similarly, the visible units conditioned on $\mathbf{h}$ are also independent of each other. The units of a binary hidden layer, conditioned on the visible layer, are independent Bernoulli random variables. The binary state h_j of the jth hidden unit is set to 1 with probability

$$p(h_j = 1|\mathbf{v}) = \delta\left(\sum_i w_{ij} v_i + b_j\right) \tag{4}$$

where $\delta(x) = 1/(1+\exp(-x))$ is the sigmoid activation function. Similarly, if the visible units are binary, the visible units, conditioned on the hidden layer, are also independent Bernoulli random variables. In this case, the binary state v_i of the ith visible unit is set to 1 with probability

$$p(v_i = 1|\mathbf{h}) = \delta\left(\sum_j w_{ij} h_j + c_i\right). \tag{5}$$

On the other hand, if the visible units have real values, then the visible units, conditioned on the hidden layer, are independent Gaussian random variables defined as

$$p(v_i|\mathbf{h}) = \mathcal{G}\left(\sum_j w_{ij} h_j + c_i, 1\right) \tag{6}$$

where $\mathcal{G}(\cdot)$ represents the Gaussian distribution. Suppose $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$ is the parameter set of the RBM. Since the RBM is a generative model, the parameters can be calculated by performing stochastic gradient descent on the log likelihood of the training samples [30]. The probability that the network assigns to a sample $\mathbf{v}^{(k)}(k = 1, \ldots, K)$ is given by summing over all possible hidden vectors as

$$\arg\min_{\boldsymbol{\theta}} -\sum_k \log\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v}^{(k)},\ \mathbf{h}^{(k)})}\right). \tag{7}$$

By solving (7), one could calculate the parameters offline and use them to configure the RBM. After training the RBM, the DBN can be built by stacking multiple layers of RBMs trained in a layer-by-layer manner.

B. Adaptive Restricted Boltzmann Machine

For efficient hardware implementation, we propose a sparsely weighted variant of the RBM, named the Adaptive RBM (AdaRBM), which adds an extra regularization term in (7) to shrink the weights adaptively. The regularization term is based on a mixed matrix norm defined as

$$\|\mathbf{W}\|_M = \sum_i \left(\sum_j |w_{ij}|^2\right)^{1/2} \tag{8}$$

where the two indices i and j are treated differently. It is easy to prove that the mixed norm is a legitimate matrix norm, and it is different from the standard L_1 and L_2 matrix norms, i.e., $\|\mathbf{W}\|_{L_1} = \sum_i \sum_j |w_{ij}|$ and $\|\mathbf{W}\|_{L_2} = (\sum_i \sum_j w_{ij}^2)^{1/2}$.

The mixed matrix norm defined in (8) adds the vector norms of all rows in a matrix; therefore, minimizing the mixed norm reduces the lengths of the matrix's rows. It is worth noting that the shrinking process does not apply evenly to all rows. Shorter rows shrink faster than the rows with larger weights in the stochastic gradient descent process. As a result, the weights in short rows tend to shrink to zero after finite iterations

Similarly, minimizing the mixed norm of a transposed matrix $\mathbf{W}^T$ could reduce the weights in shorter columns to zero. To achieve the maximum compression rate, the AdaRBM attempts to shrink the weight parameters in shorter rows and columns simultaneously by minimizing

$$\mathcal{R}_s(\mathbf{W}) = \lambda(\gamma \|\mathbf{W}\|_M + (1-\gamma)\|\mathbf{W}^T\|_M) \tag{9}$$

where λ controls the sparsity of the weight parameters, and γ controls the balance between row sparsity and column sparsity. Formally, the AdaRBM training algorithm attempts to shrink the regularization term by incorporating it in the standard RBM of (7) as

$$\arg\min_{\boldsymbol{\theta}} -\sum_k \log\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v}^{(k)},\ \mathbf{h}^{(k)})}\right) + \lambda(\gamma \|\mathbf{W}\|_M + (1-\gamma)\|\mathbf{W}^T\|_M) \tag{10}$$

C. Training Algorithm

The objective function of (10) is the sum of a log-likelihood term and a regularization term. The derivatives of the log probability and the regularization term with respect to the parameters can be expressed as

$$\frac{\partial \log\ p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j\rangle_{\text{data}} - \langle v_i h_j\rangle_{\text{model}} + \frac{\partial \mathcal{R}_s(\mathbf{W})}{\partial w_{ij}} \tag{11}$$

$$\frac{\partial \log\ p(\mathbf{v})}{\partial b_j} = \langle h_j\rangle_{\text{data}} - \langle h_j\rangle_{\text{model}} \tag{12}$$

$$\frac{\partial \log\ p(\mathbf{v})}{\partial c_i} = \langle v_i\rangle_{\text{data}} - \langle v_i\rangle_{\text{model}} \tag{13}$$

$$\frac{\partial \mathcal{R}_s(\mathbf{W})}{\partial w_{ij}} = \lambda\left(\gamma \frac{w_{ij}}{\sqrt{\sum_i w_{ij}^2}} + (1-\gamma)\frac{w_{ij}}{\sqrt{\sum_j w_{ij}^2}}\right) \tag{14}$$

where $\langle\cdot\rangle_p$ indicates the expectation of the distribution p. Unfortunately, similar to the RBM training process, (11)–(14) are not tractable, because the computation of the expectations is very difficult; however, one could use the contrastive divergence (CD) with Gibbs sampling to approximate the optimal parameters in an iterative way [30]. On each iteration, we apply the CD update rule, followed by one step of gradient descent of the regularization term as in Algorithm 1. According to 14, the shrinking process of the weight parameters is uneven. After a few hundred times of iterations, the weights in short rows and columns can be reduced to near zero, leading to sparse weight parameters. It is worth noting that, the mix-norm regularization is not differentiable at point zero. In practice, a small constant, e.g., $\alpha = 0.01$, can be added in the denominator to improve the robustness of the training algorithm.

Similar to the DBN, multiple layers of AdaRBMs can be stacked to compose a DAN, and the DAN can also be trained in a layer-by-layer style. Specifically, one could train the bottom AdaRBM with CD on the training data. With the parameters frozen and the hidden unit values inferred, these inferred values can be used as the input data to train the next layer of the network.

The DAN, built with stacked AdaRBMs, is designed for efficient hardware implementation. Technically, the parameters of an AdaRBM are calculated offline with a four-step procedure and used to configure the DANoC hardware prototype: 1) the single-precision sparse parameters are calculated according to Algorithm 1; 2) the weight parameters are thresholded using a small positive value u, and the weights with small absolute values are removed; 3) the reserved positive and negative weights are represented as $+1$ and -1 respectively; and 4) the activations are binarized using zero and one.

Algorithm 1 Training Algorithm of the Adaptive RBM

1: Given $\langle v_i h_j\rangle_{\text{model}}$ represents the distribution defined by running a Gibbs chain, the parameters can be updated using the contrastive divergence rule as

$$w_{ij} \Leftarrow w_{ij} + \epsilon(\langle v_i h_j\rangle_{\text{data}} - \langle v_i h_j\rangle_{\text{model}})$$
$$b_j \Leftarrow b_j + \epsilon(\langle h_j\rangle_{\text{data}} - \langle h_j\rangle_{\text{model}})$$
$$c_i \Leftarrow c_i + \epsilon(\langle v_i\rangle_{\text{data}} - \langle v_i\rangle_{\text{model}})$$

where ϵ is a learning rate, and $\langle\cdot\rangle_{\text{model}}$ is the expectation over the reconstruction data;

2: Update w_{ij} using the gradient of $\mathcal{R}_s(\mathbf{W})$ as

$$w_{ij} \Leftarrow w_{ij} - \lambda(\gamma \frac{w_{ij}}{\alpha + \sqrt{\sum_i w_{ij}^2}} + (1-\gamma)\frac{w_{ij}}{\alpha + \sqrt{\sum_j w_{ij}^2}})$$

where α is a small constant to avoid zero denominator.

3: Check the constraint and repeat the update process until it achieves convergence.

D. Properties of the Deep Adaptive Network

The DAN learns an efficient hardware-oriented network architecture featuring two properties.

1) The neural connections of the DAN are sparse. The DAN adaptively reduces the connection weights associated with negligible visible units to zero, which can be removed for efficient hardware implementation.
2) The activations of the DAN hidden units are sparse. Specifically, during the training phase, the AdaRBM uses $p(h_j = 1|\mathbf{v})$ as the jth output activation to the next layer. Since the short columns of the weight matrix are reduced to zero vectors, the output activations associated with zero columns become close to a constant $\delta(b_j)$ independent of the visible units. This property potentially makes the connection weights of the next-layer AdaRBM to be sparser.

Fig. 3 shows these two properties. Two deep NNs of the same configuration (784-800-800) are built to select the features from the MNIST data set. It seems that most DAN weights are close to zero, which are much sparser than the DBN weights. Meanwhile, by subtracting the constant vector $\delta(\mathbf{b})$, the output activations of the AdaRBMs become notably sparser than the standard RBM [Fig. 3(c) and (d)]. It is worth noting that, as shown in Fig. 3(b), the weight parameters become sparser in the second layer than the first layer. It seems that the sparse activations of the first-layer make the second-layer connection weights sparser.

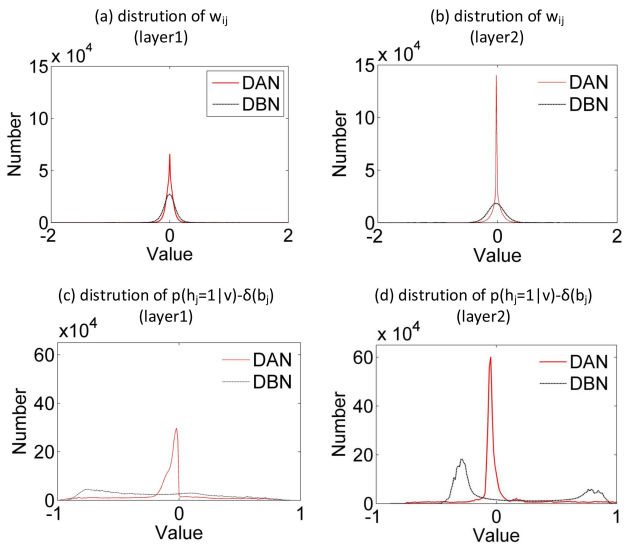


Fig. 3. Empirical distributions of the connection weights and the activations of a two-layer DAN. (a) Distribution of w_{ij} (layer 1). (b) Distribution of w_{ij} (layer 2). (c) Distribution of $p(h_j = 1|\mathbf{v}) - \delta b_j$ (layer 1). (d) Distribution of $p(h_j = 1|\mathbf{v}) - \delta b_j$ (layer 2). It is worth noting that, for the DAN, the second-layer weights and activations seem sparser than the first layer.

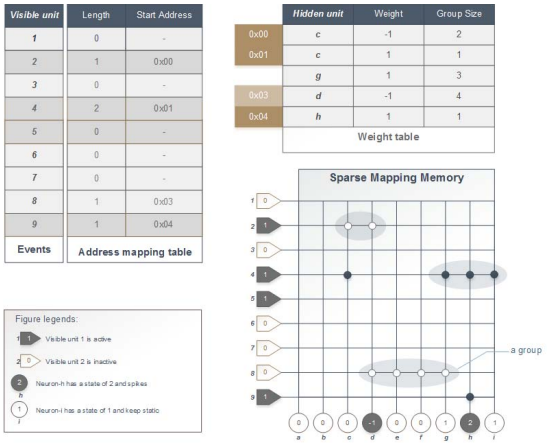


Fig. 4. SMM in the SNC. Each SMM consists of two tables to maintain the group addresses and the nonzero weight groups.

IV. Hardware Prototype

The DANoC system is an efficient, high-throughput and scalable hardware prototype of the DAN built using off-the-shelf FPGA devices. The basic building block of the hardware is an SNC featuring an efficient SMM-based architecture. Fig. 4 shows an example of the cross-bar SMM designed to implement a sparsely connected AdaRBM with nine visible units and nine hidden units. Each cross point in the SMM represents a connection weight w_{ij} between the ith visible unit (left) and the jth hidden unit (down). In a functional point of view, the input data of each core are processed as coded address events. The address event i is active when the ith visible unit is one, which triggers a lookup operation; otherwise, the address event i is inactive and the core will check the next visible unit for an active address event. No lookup operation will be fired until a nonzero visible unit is found. The zeros in the input data stream are omitted and only the spikes of ones could activate the core.

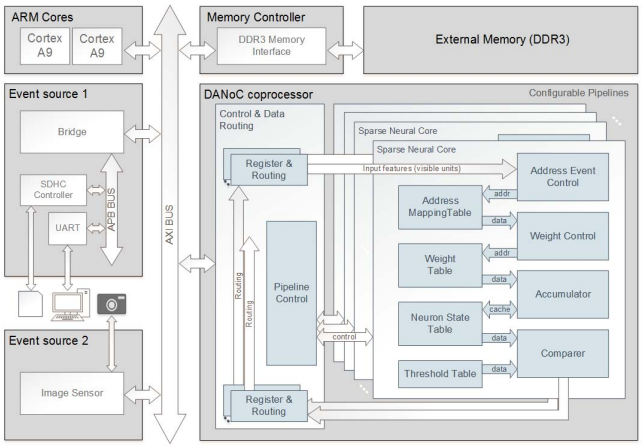


Fig. 5. Block diagram of the DANoC which consists of two host processors, a coprocessor, and an external memory controller. The coprocessor consists of an array of pipelined deep NNs, and each pipeline is composed of multiple SNCs. Each core can be configured as an AdaRBM or an NN classifier.

The lookup operation of the ith address event has three steps. First, the incoming event activates the address mapping table, which reads out the starting address i and the length of the weight group. If the ith visible unit is connected to zero hidden units, the core goes on to check the next visible unit. Second, the binary weights of all hidden units connected with the visible unit i are read out from the weight table. Then, each hidden unit updates the state value s_j in the neural state table by w_{ij}. When the neural state exceeds its threshold b_j, the neuron produces a spike and its neural state is reset to 0; this spike is then encoded and sent off as an address event to the next SNC. The binary representation allows the DANoC to use power-efficient threshold operations to implement the sigmoid activation function.

The weight-decay optimization of the AdaRBM leads to rowwise and columnwise sparse weight matrix. The hardware design of the SNC takes advantage of the 3-D sparsity to improve hardware efficiency. Since the SNC only activates lookup operations when connections associated with a given visible unit exist, the rowwise sparsity allows the DANoC to save time and power by overlooking the majority of the visible units whose associated connections are all removed. On the other hand, the columnwise sparsity makes the nonzero weights in a row to group together; Therefore, the weight table of the SMM could reduce memory consumption and computational time by preserving and reading weights in groups.

A block diagram of the DANoC coprocessor is shown in Fig. 5. The SoC has three main components: two host ARM Cortex A9 processors, a coprocessor, and an external memory controller. The coprocessor comprises an array of pipelined SNCs and a control module. Multiple pipelines of deep NNs can be implemented in parallel in the DANoC. Each pipeline contains multiple cascaded SNCs, and each core can be configured as an AdaRBM layer or a classifier layer. The DANoC uses respective on-chip block RAMs to

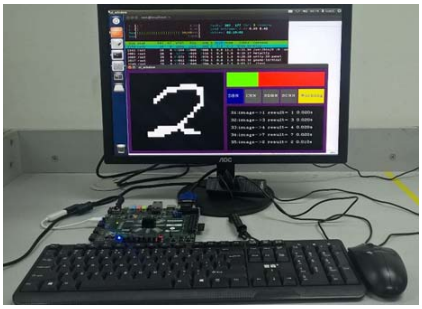


Fig. 6. DANoC hardware prototype applied for classifying images of handwritten digits. The classification of each image contains 1.28 million operations, and the DANoC can process one image in less than 0.24 ms, and the low-price Zynq7Z020-based DANoC achieves effective 319 GOPS at 495 mW.

implement the address mapping table, the weight table, the neural state table, and the threshold table. The accesses of these block RAMs are coordinated as a four-stage subpipeline to maximize the throughput.

The memory complexity of the DANoC chip is mainly determined by the size of the weight table, which is proportional to $n * d * \sigma * \log(s)/s$, where $n * d$ is the size of the weight matrix, σ is the ratio of reserved connections, and s is the average size a weight group. In practice, the DANoC could save 99.3%–99.9% weight memory compared with the standard single-precision DBN. The binary representation achieved by the DAN algorithm enables the DANoC to substitute the complicated floating-point multipliers for power-efficient accumulators. Moreover, since the latency of a binary accumulator is significantly lower than a floating-point multiplier, the DANoC can process the input remote sensing and video images in real time without jamming the pipeline.

The DANoC prototype is implemented using the Xilinx Zynq FPGA device, which is a programmable SoC (Fig. 6). The ARM host processors work at 800 MHz and the coprocessor works at 100 MHz in the SoC. The Zynq7Z020 FPGA has 4.9-Mb on-chip block RAM. The hardware prototype contains 1-GB 533-MHz DDR3 on-board memory and 3.8-GB/s full-duplex memory bandwidth. The peak power consumption of the entire board is 8 W, and the power consumption is less than 2 W for the FPGA device. The Zynq platform is chosen, because its performance increases linearly as the number of pipelines increases. To evaluate the scalability of the DANoC, a high-end version of the DANoC coprocessor is implemented using the Xilinx Zynq7100 FPGA, which contains 26.5-Mb on-chip block RAM, and allows us to fit up to 30 pipelines in a single chip.

The host ARM processors are responsible for parsing a deep network and controlling the transfer of input and configuration data to the coprocessor. The coprocessor is implemented on programmable logic and interfaces with the host processors via the AXI bus. Input data are encoded as address events and streamed into the coprocessor, one data word per clock cycle. Data words are organized as an array, with data words streamed in one row at a time. These data words can be pixels in case of images or videos. The DDR memory controller interfaces the pipelines with the external memory. Its purpose is to route independent data streams and feed data to the DANoC pipelines. The router is implemented as a crossbar switch, allowing the coprocessor to access multiple memory buffers at once with full-duplex data transactions.

V. Experiments

In this section, we evaluate the DAN algorithm and the DANoC hardware prototype using three different applications, i.e., recognizing images of handwritten digits, classifying hyperspectral remote sensing images and an application of video-based self-driving toy robot car.

A. Experiment Setting and Measurements

We implement the sparsely connected DAN with binary connections and activations in four steps. In each step, the feedforward NN becomes more efficient. To distinguish these networks, we list the DANs with different precisions as follows.

1) *DAN:* Single-precision DAN.
2) *DAN$_t$:* Single-precision DAN whose majority of connections associated with zero weights are removed.
3) *DAN$_b$:* Fix-point DAN with binary connection weights.
4) *DAN$_B$:* Fix-point DAN with binary connection weights and binary activations.

To evaluate the sparsity of neural connections, we use the ratio of reserved weights σ as the sparsity measurement, which is controlled by the threshold value u as

$$\sigma = 1 - \frac{\mathcal{N}(u)}{\text{total number of weights}} \times 100\% \qquad (15)$$

where $\mathcal{N}(u)$ indicates the number of weights whose absolute values are smaller than u. Specifically, the sparsity measurement $\sigma \in [0, 1]$ reaches 0 if $u = \max_{ij}|w_{ij}|$. The σ measurement is an important tradeoff between efficiency and classification accuracy, and a typical range of σ is between 5% and 25% for the examined data sets.

B. MNIST Handwritten Images

MNIST is a benchmark image classification data set [31]. It consists of a training set of 60 000 and a test set of 10 000 28×28 grayscale images representing digits ranging from 0 to 9 (Fig. 7). Two networks, i.e., a DAN (784-800-800-10) and a DBN (784-800-800-10), are built to extract features from the images in the MNIST data set. We use the training set of 60 000 images to fit the DAN and DBN simultaneously. The parameters of the DAN are then thresholded and binarized. Respective NN classifiers are connected with the DAN and the DBN. Experiment shows that DAN$_t$ with only 25% connections reserved has almost the same classification accuracy (98.83%) as the original DBN (98.84%) with full connections.

Fig. 7 shows DAN$_b$ with binary connection weights. The weight parameters are shown in Fig. 7(b). The inferred values are shown in Fig. 7(a). As a result of the mixed-norm weight-decay and threshold process, the weight matrices of the first-layer and the second-layer AdaRBMs become very sparse, and the nonzero weights tend to cluster into different groups.

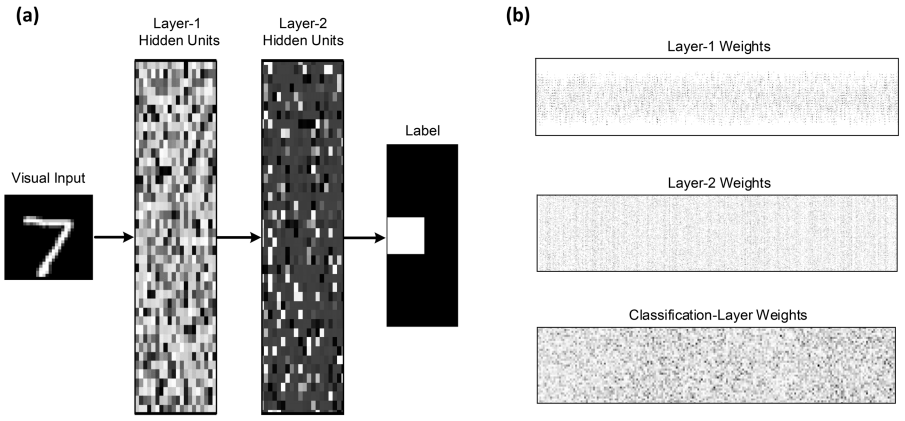


Fig. 7. Sparsely connected binary-weighted DAN (DAN_b) learned from the MNIST data set. An NN classifier is connected to the DAN for recognizing the label (0–9) of a given image. (a) The input image and the associated sparse features extracted using the DAN. (b) The sparse weight parameters learned by the DAN.

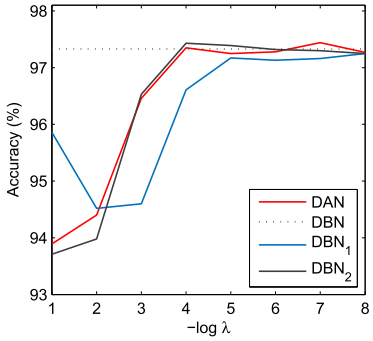


Fig. 8. Classification accuracies of the DBNs and the DAN change as λ changes from 10^{-1} to 10^{-8} ($\gamma = 0.5$). DBN_1 and DBN_2 are single-precision DBNs with L_1 and L_2 norm weight decay, respectively.

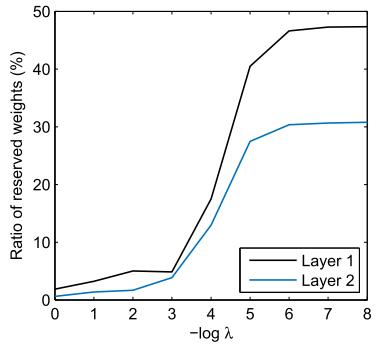


Fig. 9. Ratio of reserved weights ($\sigma\%$) of each layer in the DAN is controlled by the parameter $\lambda (u = 0.1, \gamma = 0.5)$.

We change the parameter λ from 10^{-1} to 10^{-8} to evaluate its relation to classification accuracy. Our experiment randomly selects 10 000 images for training and tests the DAN, the DBN, the DBN with L_1-norm weight-decay (DBN_1), and the DBN with L_2-norm weight-decay (DBN_2) over the rest images. The experiment is carried out ten times and the average accuracies are shown in Fig. 8. It seems that the DAN, the DBN, and DBN_2 have a similar classification accuracy when λ is smaller than 10^{-4}; however, DBN_1 has noticeably lower accuracy.

Since the parameter λ controls the sparsity of the weight parameters in a DAN, it could affect the efficiency of hardware implementation. Fig. 9 shows the relation between the ratio of reserved weights (σ) and the value of λ. A typical threshold value $u = 0.1$ and $\gamma = 0.5$ is set. Results show that the ratio of reserved weights drops significantly from about 50% to less than 5% when λ changes from 10^{-8} to 10^{-1}. Experiment also shows that the second layer of the DAN is over 15% sparser than the first layer.

An experiment is performed to illustrate how the parameter γ controls the tradeoff between rowwise sparsity and columnwise sparsity of the weight matrix. The weights of DAN_t with different values of γ are examined. We compare the rowwise matrix norm $\|\mathbf{W}\|_M$ and the columnwise matrix norm $\|\mathbf{W}^T\|_M$, and have two observations. First, $\|\mathbf{W}^T\|_M$ increases as γ increases, whereas $\|\mathbf{W}\|_M$ drops as γ increases. This observation is coherent with the optimization formulation of (10). Second, the weight matrix of the second-layer AdaRBM has about 20% smaller mixed norm than the first-layer AdaRBM. This observation indicates that the second-layer weights may be sparser than the first-layer weights, which is coherent with the results of Fig. 9.

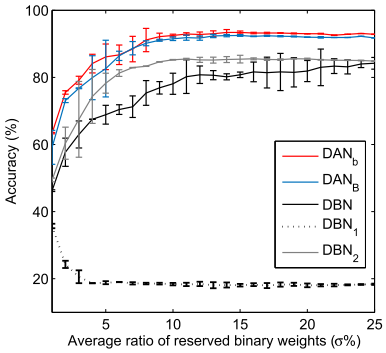


Fig. 10. Classification accuracy of the DANs and the DBNs with different weight decay approaches. The connection weights are thresholded with the $\sigma\%$ significant weights reserved and represented using single-bit integers.

Fig. 10 compares the accuracy of the classic NN classifiers proceeded by the DBN and the DANs of different precisions.

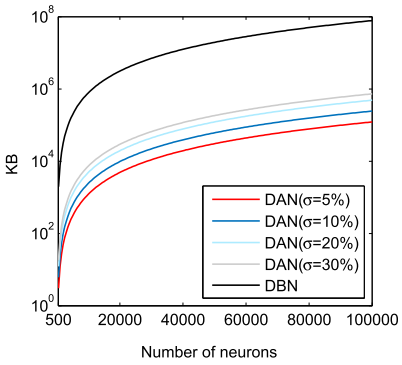


Fig. 11. Memory required for weight parameters of the DBN and the DAN.

To compare the influence of quantization on the examined approaches, the weights of the DBNs are thresholded and quantized with the same setting as the DAN. Experiment shows that, generally speaking, the DAN is sparse and robust with the binary quantization operation. When the ratio of reserved weights σ changes from 0% to 5%, the classification accuracy of DAN_b quickly climbs to over 90%. However, the classification accuracies of the DBN (54.2%), DBN_1 (19.2%), and DBN_2 (67.4%) show much worse results with 5% connections reserved. It seems that DAN_b shows almost no drop in classification accuracy when more than 10% connections are reserved. Moreover, the deviation results indicate that DBN_1, DBN_2, and the DAN are all relatively stabler than the original DBN when the ratio of reserved weights decreases. The deviation of the DAN quickly shrinks when more than 15% weights are reserved.

To achieve higher hardware efficiency, the DAN could use binary activations as well as binary weights. Fig. 10 also shows the influence of adopting binary activation on classification accuracy. The activations of DAN_B are thresholded and binarized using a constant 0.5. As shown in Fig. 10, the accuracy of the NN classifier proceeded by DAN_B is significantly higher than the DBN. By adopting binary weights and activations, the DANoC coprocessor could replace the complex floating-point multipliers with simple fix-point accumulators and implements the sigmoid activation function using simple threshold operations.

By adopting the sparse binary connections, the DAN becomes 99.3% ($\sigma = 20\%$) to 99.9% ($\sigma = 5\%$) more memory efficient than the single-precision DBN. Fig. 11 shows the theoretical results of the memory used by the DAN and the original DBN. The memory complexity of the deep NNs increases as the number of neurons increases. The present DBNs are usually implemented using 32-b representation; however, with sparse and binary weights, DAN_b could improve the memory efficiency by two to three orders of magnitude.

C. Hyperspectral Remote Sensing Images

The second experiment applies the DAN algorithm and the DANoC system to the hyperspectral remote sensing application. Different from visible-light images, the hyperspectral images contain information from across the electromagnetic spectrum. Fig. 12 shows the Pavia Center data set, which contains 1.2 million samples. Each pixel in the image is recorded with 102 spectral bands covering the wavelengths from 401 to 889 nm. The goal of classification is to determine nine labels associated with each pixel.

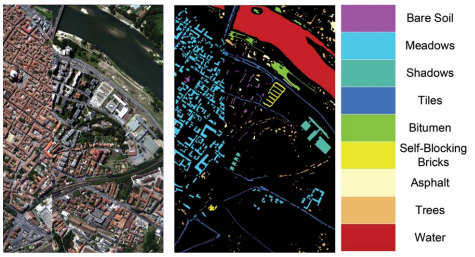


Fig. 12. Pavia Center image and the ground-truth labels for each pixel. The Pavia Center set has 1.2 million samples. The DBN-based classification of each sample contains 10.9 million operations, and the Zynq7100-based DANoC achieves 2036 effective GOPS at 1.9-W power consumption.

TABLE I
EXAMINED HYPERSPECTRAL DATA SETS

Datasets	Bands	Samples (million)	Features	Classes
Indian Pines	224	0.02	2016	16
Salinas	224	0.11	2016	16
Pavia University	103	0.37	927	9
Pavia Centre	102	1.20	918	9

Our experiments examine four well-known data sets of hyperspectral images (Table I). All the examined data sets are downloaded from the hyperspectral website [32]. The scene of the India Pines is gathered by the AVIRIS sensor and consists of 21 025 samples with 224 spectral reflectance bands in the wavelength ranging from 0.4×10^{-6} to 2.5×10^{-6} m. The label of each pixel falls into 16 classes. The Salinas data set is collected by the 224-band AVIRIS sensor over the Salinas Valley, CA, USA, and is characterized by high spatial resolution (3.7-m pixels). The area covered comprises 11 110 thousand samples of 16 classes. Similar to the Pavia Center data set, the Pavia University set has nine classes and 0.37 million samples recorded with 103 spectral bands.

Different from traditional technologies, the hyperspectral imaging can get the spatial and spectral data simultaneously, resulting in high-dimensional samples. For practical remote sensing applications, the number of labeled training samples for each class is usually less than a few hundreds. The scarcity of labeled training samples could cause a significant drop in classification accuracy for supervised approaches, which is known as the Hughes phenomenon. Therefore, an *unsupervised* feature extraction process is usually applied before hyperspectral classification. Recently, there has been a lot interest in the remote sensing area to apply the deep learning algorithms for hyperspectral classification. And the DBN has been reported as the state-of-the-art for extracting features from hyperspectral images [3]. However, it is usually a challenge to deploy the DBN to process remote sensing images in real time, because the embedded systems carried by satellites or unmanned aerial vehicles are generally limited in hardware resources. In this experiment, we show that the efficient DAN algorithm and the DANoC system could provide competitive results with the state-of-the-art approaches for classifying hyperspectral images (Table II).

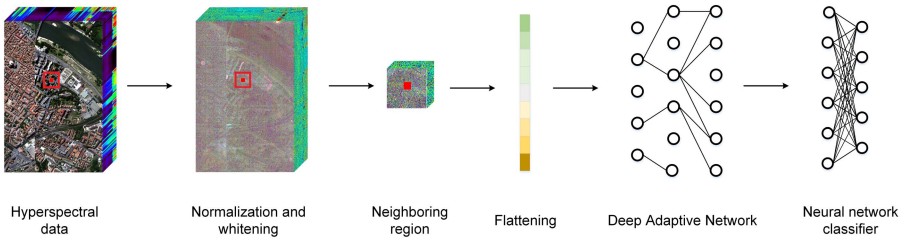


Fig. 13. Spatial–spectral classification of the hyperspectral data using the proposed DAN. Each pixel in the hyperspectral data is recorded with 102 spectral bands covering the wavelengths from 401 to 889 nm.

TABLE II
LATEST STUDIES OF HYPERSPECTRAL IMAGE CLASSIFICATION

Author	Approaches	Salinas	Indian Pines	Pavia University
Zhou	DANoC hardware[1]	94.19	89.03	95.33
Yue [33]	CNN regression	-	-	95.18
Yuan [34]	CART kNN	89.00	81.00	89.00
Wang [35]	sparse coding	-	93.11	90.41
Kuo [36]	kernel SVM	-	88.70	94.00
Ramzi [37]	adaboost SVM	93.12	91.59	94.28
Li [38]	Markov random filed	-	-	94.96
Plaza [39]	multinomial regression	-	76.71	76.03
Li [40]	Tikhonov method	-	89.00	92.00
Chen [41]	Auto-encoder	-	-	98.52
Liu [42]	Auto-encoder	95.50	-	96.40
Zhao [43]	BLDE+CNN	-	-	96.98

[1] The DANoC hardware implements the DAN with binary connections and activations. The compared approaches are generally implemented using 32-bit floating-point representation.

TABLE III
STANDARD HYPERSPECTRAL CLASSIFICATION APPROACHES

Methods	Indian Pines	Salinas	Pavia Center	Pavia University
DAN + neural network	**90.90** ±0.06	*95.52* ±0.03	**99.59** ±0.23	**96.72** ±0.12
DBN + neural network	*90.31* ±0.17	**96.02** ±0.13	*99.50* ±0.13	*95.89* ±0.08
PCA + linear regression	66.76 ±0.16	93.14 ±0.07	99.13 ±0.03	93.48 ±0.07
PCA + naive Bayes	62.11 ±0.21	90.35 ±0.10	96.06 ±0.09	83.2 ±0.13
PCA + decision tree	68.18 ±0.16	89.32 ±0.08	97.57 ±0.05	88.78 ±0.02
PCA + SVM	68.77 ±0.23	93.35 ±0.22	99.16 ±0.05	93.3 ±0.15
ICA + linear regression	68.26 ±0.16	93.51 ±0.07	99.32 ±0.03	93.69 ±0.07
ICA + naive Bayes	66.41 ±0.21	93.44 ±0.10	99.38 ±0.07	85.94 ±0.09
ICA + decision tree	59.71 ±0.18	90.47 ±0.09	96.02 ±0.05	87.87 ±0.13
ICA + SVM	68.26 ±0.22	93.51 ±0.22	99.32 ±0.28	93.3 ±0.21

Fig. 13 shows our DAN-based classification process of the spatial–spectral samples, which consists of four steps: 1) normalization and whitening are applied to reduce the correlation between features; 2) the neighbor region of the target pixel is selected (red square), and the spatial–spectral sample is flattened and arranged as a vector; 3) a DAN is trained and built to extract the features from the high dimensional vector; and 4) the extracted features are classified using an NN classifier.

Given a target pixel, a neighbor region of nine pixels is constructed to form a high-dimensional spatial–spectral sample. Different deep NNs are trained over the Indian Pines data set (1800-2000-2000), the Salinas data set (1836-2000-2000), the Pavia Center data set (918-1000-1000), and the Pavia University data set (927-1000-1000) for feature selection. Our experiment compares the proposed method with the DBN, the principal component analysis (PCA), and the independent component analysis (ICA), all of which have been proven to be effective for processing the hyperspectral data. Different classifiers, including the NN classifier, the logistic regression classifier, the support vector machine, the naive Bayes, and the decision tree, are examined. All the compared classifiers are implemented using the WEKA software [45]. To calculate the classification accuracy, we randomly select 50% of unlabeled hyperspectral samples for training the unsupervised DAN, the PCA, and the ICA, and the rest samples are used for testing. The classifiers are trained with 25% labels randomly selected from the training set. The experiment is repeated ten times, and the average classification accuracy and derivation are calculated. Experiment results in Table III show that the DAN algorithm achieves the highest classification accuracy among all compared approaches over three of the examined data sets (90.90%–99.59%).

Table II compares the binary DAN implemented in the DANoC system with the optimal results reported by the latest hyperspectral studies [33]–[40]. All these studies use the same spatial–spectral samples and similar experiment settings. It seems that the DANoC system could achieve competitive results with the-state-of-the-art approaches by taking advantage of the large-volume unlabeled samples. It is worth noting that the DAN even outperforms some deep learning-based approaches, including the CNN. Since the labeled training samples are limited and expensive for remote sensing applications, the lack of labeled training data degrades the performance of supervised approaches.

With algorithm-level and hardware-level optimization, the DANoC coprocessor achieves competitive performance with the state-of-the-art hardware implementations [27]–[29], [46]. In our hardware experiments, the performance of the DANoC system is estimated using the number of multiplications required in a standard DBN. Two versions of DANoC systems

TABLE IV
PERFORMANCE OF THE DANOC HARDWARE PROTOTYPE

Data sets	Zynq7Z020 @ 410mw - 495 mw				Zynq7100 @ 1760 mw - 1930 mw			
	Number of pipelines	Throughput (thousand samples per second)	Effective performance (GOPS/watt)	Single chip performance (GOPS)	Number of pipelines	Throughput (thousand samples per second)	Effective performance (GOPS/watt)	Single chip performance (GOPS)
MNIST	5	60.1	774	381	25	300.5	992	1903
Indian Pines	1	31.2	615	237	5	156.0	714	1185
Salinas	1	31.2	615	238	5	156.0	714	1189
Pavia Centre	4	53.8	833	407	20	215.2	1073	2036
Pavia University	4	53.9	841	415	20	215.6	1079	2077
Robot Car (sign/road)	3/3	52.5/75.2	677	319	15/15	262.5/376.0	879	1593

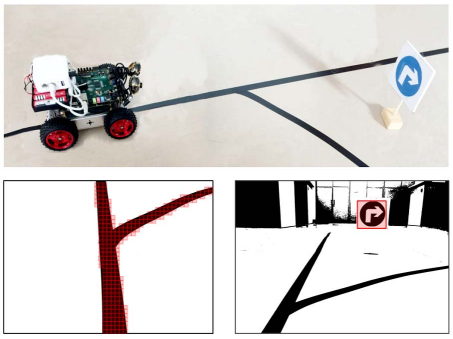


Fig. 14. Self-driving robot car equipped with a DANoC hardware prototype. The robot car has two cameras, one is used to track the road (left) and the other is used to recognize the traffic signs (right). Two types of deep NNs are integrated in the DANoC, which achieve 98.2% correct rate for recognizing 12 traffic signs, and 98.9% correct rate for recognizing the road. The Zynq7100-based DANoC coprocessor achieves the state-of-the-art peak performance of 1593 GOPS and is able to process 67 frames of 640–480 video in real time.

are implemented using the Zynq7Z020 and the Zynq7100 FPGA devices, respectively. As shown in Table IV, the low-cost Zynq7Z020 DANoC achieves high performance of 615–841 giga-operations per second per watt, which allows the DANoC chip to process 31.2–53.9 thousand hyperspectral samples per second. The peak performance achieved by the Zynq7100 DANoC chip is 2077 effective giga-operations per second, which is the state-of-the-art among the latest FPGA-based hardware implementations (Table V). The high performance is achieved at less than 2 W of chip-level power consumption, which is orders of magnitude more efficient than CPU- and GPU-based solutions.

D. Real-Time Analysis of Video Images

Besides the remote sensing applications, which are our primary motivation, we also evaluate the DANoC hardware for real-time video processing. In this experiment, a DANoC hardware prototype is mounted on a toy robot car for automatic driving (Fig. 14). The robot car has two 640–480 video cameras connected with the hardware board via respective USB ports. One camera is used to track the road and the other is used to recognize 12 traffic signs. The video streams are captured and thresholded as binary frames using software running on the ARM host processors. Then, the image frames are split as 16–16 (road) and 32–32 (traffic sign) windows with

TABLE V
PEAK PERFORMANCE OF THE LATEST HARDWARE DEEP NN ACCELERATORS

Author	Type	Platform	Speed (GOPS)	Power (mw)	Precision
This work	FPGA	Zynq7Z020	415	495	fix 1
	FPGA	Zynq7100	2077	1930	fix 1
Sanmi [27]	FPGA	Kintex7350	-	-	fix12
Zhang [28]	FPGA	Virtex7	612	18610	fix32
Gokhale [29]	FPGA	Zynq7Z045	227	4000	fix16
Li [46]	FPGA	StratixV	410	1114	fix 16
Du [47]	layout	65nm	194	320	fix16
Pham [48]	layout	45nm SOI	320	600	fix1
Cavigelli [49]	silicon	65nm	196	510	fix12
Andri [50]	silicon	65nm	423	153	fix1

8-pixel step size, and the images are arranged as binary streams and routed to the DANoC coprocessor for feature extraction and classification. The classification results are then sent back to the host processor to control the toy robot car automatically.

Two types of DAN pipelines are implemented in the DANoC coprocessor for recognizing the traffic signs and the road, respectively. Each pipeline uses two SNCs to implement a stacked DAN and one core to implement the NN classifier. Eight video clips recoded with different light conditions and camera angles are used to train and test the DAN algorithm offline. Experiment shows the DANoC coprocessor achieves 98.2% correct rate for recognizing 12 traffic signs, and 98.9% correct rate for recognizing the road. The performance of the DANoC system is estimated using the number of multiplications required in a standard DBN of the same configuration. The Zynq7Z020 DANoC with six pipelines of deep NNs achieves 319 giga-operations per second with 478-mW power consumption. The high-performance Zynq 7100-based DANoC could integrate up to 30 pipelines in single FPGA chip, which enables it to process 67 frames in real time with the peak performance of 1593 effective giga-operations per second.

VI. CONCLUSION AND DISCUSSION

This paper proposes an algorithm and hardware codesign of the famous DBN for embedded applications. The proposed DAN algorithm is optimized to learn deep NNs with sparse connections, which can be robustly represented using single-bit integers. The proposed efficient learning algorithm could reduce up to 99.9% memory consumption and replace

the complex floating-point multipliers with efficient fix-point accumulators, which enables the DANoC hardware to preserve all the parameters on chip and achieve the state-of-the-art performance for embedded remote sensing and computer vision applications.

Motivated by accelerating unsupervised deep NNs for embedded pattern recognition applications, this paper mainly focuses on the algorithm and hardware optimization of the famous DBN. However, the sparse weight decay approach can also be applied to other deep learning methods. For example, we find that the connection weights in the CNN can also be reduced by the mixed norm regularization approach, and the CNN could be robustly thresholded and represented using binary integers. In the future, we plan to implement and optimize the hardware design for CNN with sparse binary connections and activations.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive comments on this paper. They would also like to thank L. Yu and K. Li for their contribution in developing and evaluating the hardware prototype.

REFERENCES

[1] G. E. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Oct. 2012.

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2015.

[3] Y. Chen, X. Zhao, and X. Jia, "Spectral-spatial classification of hyperspectral data based on deep belief network," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 1–12, Jan. 2015.

[4] O. Vinyals and S. V. Ravuri, "Comparing multilayer perceptron to deep belief network tandem features for robust ASR," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Prague, Czech Republic, May 2011, pp. 4596–4599.

[5] L. P. Maguire *et al.*, "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomputing*, vol. 71, nos. 1–3, pp. 13–29, Dec. 2007.

[6] N. Lopes and B. Ribeiro, "An evaluation of multiple feed-forward networks on GPUs," *Int. J. Neural Syst.*, vol. 21, no. 1, pp. 31–47, Feb. 2011.

[7] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization," *IEEE Trans. Neural Netw.*, vol. 18, no. 3, pp. 880–888, May 2007.

[8] S. Draghici, "On the capabilities of neural networks using limited precision weights," *Neural Netw. J. Int. Neural Netw. Soc.*, vol. 15, no. 3, pp. 395–414, Apr. 2002.

[9] J. L. Holi and J.-N. Hwang, "Finite precision error analysis of neural network hardware implementations," *IEEE Trans. Comput.*, vol. 42, no. 3, pp. 281–290, Mar. 1993.

[10] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Mach. Learn.*, pp. 1737–1746, 2015.

[11] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *J. Roy. Statist. Soc., B (Statist. Methodol.)*, vol. 68, no. 1, pp. 49–67, 2006.

[12] J. Friedman, T. Hastie, and R. Tibshirani, "A note on the group lasso and a sparse group lasso," *Statistics*, Jan. 2010.

[13] M. Ranzato, Y.-L. Boureau, and Y. LeCun, "Sparse feature learning for deep belief networks," in *Advances in Neural Information Processing Systems*, vol. 20. Red Hook, NY, USA: Curran Associates, 2007, pp. 1185–1192.

[14] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area V2," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 20. 2008, pp. 873–880.

[15] N.-N. Ji, J.-S. Zhang, and C.-X. Zhang, "A sparse-response deep belief network based on rate distortion theory," *Pattern Recognit.*, vol. 47, no. 9, pp. 3179–3191, 2014.

[16] S. Abramson, D. Saad, and E. Marom, "Training a network with ternary weights using the CHIR algorithm," *IEEE Trans. Neural Netw.*, vol. 4, no. 6, pp. 997–1000, Nov. 1993.

[17] F. Aviolat and E. Mayoraz, "A constructive training algorithm for feedforward neural networks with ternary weights," in *Proc. Eur. Symp. Artif. Neural Netw.*, Brussels, Belgium, Apr. 1994, pp. 20–22.

[18] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[19] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberge, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *Fiber*, vol. 56, no. 4, pp. 3–7, 2016.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Neural Inf. Process. Systems*, pp. 4107–4115, 2016.

[22] M. Rastegari *et al.*, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur Conf Comput. Vis.*, Sep. 2016, pp. 525–542.

[23] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2262–2270.

[24] W. Pan, H. Dong, Y. Guo. (Jun. 2016). "DropNeuron: Simplifying the structure of deep neural networks." [Online]. Available: https://arxiv.org/abs/1606.07326

[25] W. Wen *et al.*, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[26] B. Ahn, "Computation of deep belief networks using special-purpose hardware architecture," in *Proc. IEEE Int. Conf. Neural Netw.*, Beijing, China, Jul. 2014, pp. 141–148.

[27] K. Sanni *et al.*, "FPGA implementation of a deep belief network architecture for character recognition using stochastic computation," in *Proc. 49th Conf. Inf. Sci. Syst.*, Mar. 2015, pp. 1–5.

[28] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. Acm/sigda Int. Symp.*, Monterey, CA, USA, Feb. 2015, pp. 161–170.

[29] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Columbus, OH, USA, Jun. 2014, pp. 696–701.

[30] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," *Momentum*, vol. 9, no. 1, pp. 599–619, 2010.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[32] *ECU Hyperspectral Website, Hyperspectral Remote Sensing Datasets*, accessed on Jun. 13, 2016. [Online]. Available: http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes#Pavia_Centre_scene

[33] J. Yue, W. Zhao, S. Mao, and H. Liu, "Spectral–spatial classification of hyperspectral images using deep convolutional neural networks," *Remote Sens. Lett.*, vol. 6, no. 6, pp. 468–477, May 2015.

[34] Y. Yuan, G. Zhu, and Q. Wang, "Hyperspectral band selection by multitask sparsity pursuit," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 2, pp. 631–644, Feb. 2015.

[35] Z. Wang, N. M. Nasrabadi, and T. S. Huang, "Semisupervised hyperspectral classification using task-driven dictionary learning with Laplacian regularization," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 3, pp. 1161–1173, Mar. 2015.

[36] B. C. Kuo, H. H. Ho, C. H. Li, C. C. Hung, and J. S. Taur, "A kernel-based feature selection method for SVM with RBF kernel for hyperspectral image classification," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 1, pp. 317–326, Jan. 2014.

[37] P. Ramzi, F. Samadzadegan, and P. Reinartz, "Classification of hyperspectral data using an AdaBoostSVM technique applied on band clusters," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2066–2079, Jun. 2014.

[38] W. Li, S. Prasad, and J. E. Fowler, "Hyperspectral image classification using Gaussian mixture models and Markov random fields," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 1, pp. 153–157, Jan. 2014.

[39] M. Khodadadzadeh, J. Li, A. Plaza, and J. M. Bioucas-Dias, "A subspace-based multinomial logistic regression for hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 12, pp. 2105–2109, Dec. 2014.
[40] W. Li, S. Prasad, J. E. Fowler, and L. M. Bruce, "Locality-preserving dimensionality reduction and classification for hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 4, pp. 1185–1198, Apr. 2012.
[41] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu, "Deep learning-based classification of hyperspectral data," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2094–2107, Jun. 2014.
[42] Y. Liu, G. Cao, Q. Shen, and M. Siegel, "Hyperspectral classification via deep networks and superpixel segmentation," *Int. J. Remote Sens.*, vol. 36, no. 13, pp. 3459–3482, Jul. 2015.
[43] W. Zhao, and S. Du, "Spectral–spatial feature extraction for hyperspectral image classification: A dimension reduction and deep learning approach," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 8, pp. 4544–4554, Aug. 2016.
[44] K. Huang, S. Li, X. Kang, and L. Fang, "Spectral–spatial hyperspectral image classification based on KNN," *Sens. Imag.*, vol. 17, no. 1, pp. 1–13, Dec. 2016.
[45] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explor. Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.
[46] N. Li, S. Takaki, Y. Tomiokay, and H. Kitazawa, "A multistage dataflow implementation of a deep convolutional neural network based on FPGA for high-speed object recognition," in *Proc. IEEE Southwest Symp. Image Anal. Interpretation*, Mar. 2016, pp. 165–168.
[47] Z. Du *et al.*, "Shidiannao: Shifting vision processing closer to the sensor," *Acm Sigarch Comput. Archit. News*, vol. 43, no. 3, pp. 92–104, 2015.
[48] P. H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. Lecun, and E. Culurciello, "NeuFlow: Dataflow vision processing system-on-a-chip," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Sep. 2012, pp. 1044–1047.
[49] L. Cavigelli and L. Benini, "A 803 GOp/s/W convolutional network accelerator," *IEEE Trans. Circuits Syst. Video Technol.*, doi: 10.1109/TCSVT.2016.2592330.
[50] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," in *Proc. ISVLSI*, 2016, pp. 236–241.

Xichuan Zhou (S'06–M'13) received the B.S. and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 2005 and 2010, respectively.

He is currently an Associate Professor and the Assistant Dean of the College of Communication Engineering with Chongqing University, Chongqing, China. He has authored or co-authored over 20 papers on peer-reviewed journals, such as the IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, the IEEE TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS, the IEEE TRANSACTIONS ON ELECTRON DEVICES, and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I. His current research interests include parallel computing, circuits and systems, and machine learning.

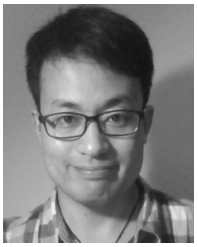
Dr. Zhou was a recipient of the Outstanding Young-and-Middle-Age Faculty Award of Chongqing in 2016.

Shengli Li received the B.S. degree from Chongqing University (CQU), Chongqing, China, in 2013.

He is currently a Graduate Student with the College of Communication Engineering, CQU. His current research interests include circuit and system design for image and video processing.

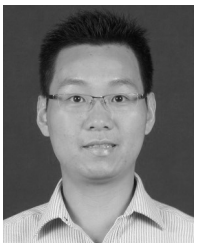
Fang Tang (S'07–M'14) received the B.S. degree from Beijing Jiaotong University, Beijing, China, in 2006, and the M.Phil. and Ph.D. degrees from the Hong Kong University of Science and Technology, Hong Kong, in 2009 and 2013, respectively.

He was a Research Associate with the Hong Kong University of Science and Technology. Since 2013, he has been a Distinguished Research Fellow and Tenure-Track Assistant Professor with the College of Communication Engineering, Chongqing University, Chongqing, China. His current research interests include circuit and system design for biomedical applications.



Shengdong Hu received the M.S. degree in material science and engineering from Sichuan University, Sichuan, China, in 2005, and the Ph.D. degree in microelectronics from the University of Electronic Science and Technology of China, Chengdu, China, in 2010.

Since 2010, he was with Chongqing University, Chongqing, China, where he was involved in research on semiconductor devices and integrated circuits. In 2011, he joined the No.24 Research Institute of China Electronics Technology Group Corporation Chongqing, as a Post-Doctoral Researcher.



Zhi Lin received the B.S. and Ph.D. degrees from the University of Electronic Science and Technology of China, Chengdu, China, in 2009 and 2015, respectively.

Since 2016, he has been with Chongqing University, Chongqing, China, where he was involved in research on semiconductor devices and integrated circuits.

Lei Zhang (M'14) received the Ph.D. degree in circuits and systems from the College of Communication Engineering, Chongqing University, Chongqing, China.

He has authored over 50 scientific papers in top journals. His current research interests include machine learning, pattern recognition, computer vision, electronic olfaction, and intelligent systems.

Dr. Zhang was a recipient of the Hong Kong Scholar Award in 2014 and the New Academic Researcher Award for Doctoral Candidates from the Ministry of Education, China, in 2012.