Parallel Computing 73 (2018) 40–51

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Proteus: Exploiting precision variability in deep neural networks

Patrick Judd[a,*], Jorge Albericio[a,1], Tayler Hetherington[b], Tor Aamodt[b], Natalie Enright Jerger[a], Raquel Urtasun[c], Andreas Moshovos[a]

[a] Department of Electrical and Computer Engineering, University of Toronto, Canada
[b] Department of Electrical and Computer Engineering, University of British Columbia, Canada
[c] Department of Computer Science, University of Toronto, Canada

ARTICLE INFO

Article history:
Received 24 November 2016
Revised 11 April 2017
Accepted 20 May 2017
Available online 24 May 2017

Keywords:
Machine learning
Neural networks
Deep learning
Accelerators
Approximate computing
Reduced precision

ABSTRACT

This work investigates how using reduced precision data in Deep Neural Networks (DNNs) affects network accuracy during classification. We observe that the tolerance of DNNs to reduced precision data not only varies *across* networks, but also *within* networks. We study how error tolerance across layers varies and propose a method for finding a low precision configuration for a network while maintaining high accuracy.

To exploit these low precision configurations, this work proposes PROTEUS, which reduces the data traffic and storage footprint needed by DNNs, resulting in reduced energy for DNN implementations. PROTEUS uses a different representation per layer for both the data (neuron activations) and the weights (synapses) processed by DNNs. PROTEUS is a layered extension over existing DNN implementations that maintains the native precision of the compute engine by converting to and from a fixed-point reduced precision format used in memory. PROTEUS uses a novel memory layout, enabling a simple, low-cost and low-energy conversion unit.

We evaluate PROTEUS as an extension to a state-of-the-art accelerator [1] which uses a uniform 16-bit fixed-point representation. On five popular Convolutional Neural Networks (CNNs) and during inference PROTEUS reduces data traffic by 43% on average while maintaining accuracy within 1% compared to the full precision baseline. As a result, PROTEUS improves energy by 15% with no performance loss.

1. Introduction

Deep Neural Networks (DNNs) have emerged as the state of the art when solving supervised learning tasks, particularly in the context of object recognition, e.g., detection [2,3], segmentation [4], image classification [5].

Current approaches utilize deep network architectures and require massive amounts of memory making it difficult to train networks. Meeting these memory requirements is also problematic during inference under real-time processing in small devices such as embedded systems or mobile devices.

* Corresponding author.
E-mail addresses: juddpatr@ece.utoronto.ca, patrick.judd@mail.utoronto.ca (P. Judd), jalbericiola@nvidia.com (J. Albericio), taylerh@ece.ubc.ca (T. Hetherington), aamodt@ece.ubc.ca (T. Aamodt), enright@ece.utoronto.ca (N. Enright Jerger), urtasun@cs.utoronto.edu (R. Urtasun), moshovos@ece.utoronto.ca (A. Moshovos).

[1] This work was completed while Jorge Albericio was a Postdoctoral Fellow at the University of Toronto.

http://dx.doi.org/10.1016/j.parco.2017.05.003
0167-8191/

A popular approach for reducing computation and memory demands during DNN processing has been to *reduce the precision of the data*, that is the number of bits used. While this introduces approximation error in the internal calculations, DNNs have proven to be very tolerant to such errors [6]. Software implementations of DNNs commonly use single-precision floating point values [7,8], while other works, including custom hardware, use 16 bit fixed-point values [1,9,10], or even lower precisions [11–14]. Using a reduced precision representation has many benefits: 1) saves energy in the memory and communication channels (e.g., memory and on-chip links), 2) improves performance in memory-bound systems through better memory bandwidth utilization and effective cache capacity, and 3) supports larger networks on systems with a fixed memory budget.

Previous work has primarily used a one-size-fits-all approach to choosing precision, resulting in a precision length that is short enough to work well for *all* networks. This is a *worst case* analysis approach where all networks are forced to use the longest representation needed by *any* network.

In contrast, this work analyzes the tolerance of DNNs to reduced precision error at a *per-layer* granularity. It first corroborates that there is significant variance in the precision needed for the weights *across* different networks and layers. Building upon this observation, it describes a method for selecting per layer precisions that maintain network accuracy within a desired range.

We study the effects of per-layer precision selection in the context of five well known CNN architectures for image classification. We show that to maintain accuracy within 1% of the full precision obtained using a network with 32-bit single-precision floating-point values, layers can instead use a fixed-point representation of only 14 bits in the worst case and of just 2 bits in the best case.

PROTEUS (PR) is presented which exploits the aforementioned DNN property to reduce the data communication of DNNs improving energy efficiency and optionally implementation cost. PR uses a different, per-layer representation when storing both the DNN's intermediate results (neurons) and the weights (synapses). As a result, PR reduces on- and off-chip data traffic improving energy, while enabling larger DNNs to be stored and processed given a fixed set of memory resources. Alternatively, PR can reduce the area needed and thus reduce implementation cost. We evaluate PR as a layered extension over existing DNN compute engines. PR stores data in memory in a fixed-point representation with a width chosen in advance on a per-layer basis. As data is brought in from memory, PR expands it to the internal representation used by the compute units such as a wider fixed-point representation. Conversely, prior to writing data to memory, PR converts it to the appropriate form for the per-layer fixed-point representation. As a layered extension over existing DNN engines, PR can be incorporated into general purpose cores, GPUs or accelerators. In this work, we demonstrate PR over a state-of-the-art DNN accelerator.

Using a different representation per layer may seem simple at first; however, converting back and forth from the memory representation to the representation used by the wide datapaths of the accelerator and aligning the data accordingly requires a shuffling network whose complexity, energy and area overheads may negate any potential benefits. To overcome this challenge, PR employs a novel memory layout enabling the use of a low-cost and low-energy conversion network. Simulation results demonstrate that PR can reduce data traffic by 43% and energy by 15% on average compared to using 16-bit fixed-point uniformly in a state-of-the-art accelerator [1].

The rest of this paper is organized as follows. Section 2 reports an analysis of the per layer error tolerances of five networks, followed by a method for finding the best mixed representation for a given network. Section 3 presents PROTEUS and describes the accelerator architecture. Section 4 evaluates PROTEUS in terms of energy savings. Section 5 discusses related work and Section 6 summarizes our observations and results.

2. Accuracy vs. representation length

This section studies the data representation length requirements for our targeted DNNs. Section 2.1 defines our methodology and terminology. Section 2.2 reports the accuracy when precision is reduced uniformly for each network. Section 2.3 studies per layer precision requirements for each layer in isolation. Section 2.4 describes a method to explore the precision-accuracy trade-off space and reports the best per-layer precisions for given accuracy constraints.

2.1. Measurement methodology

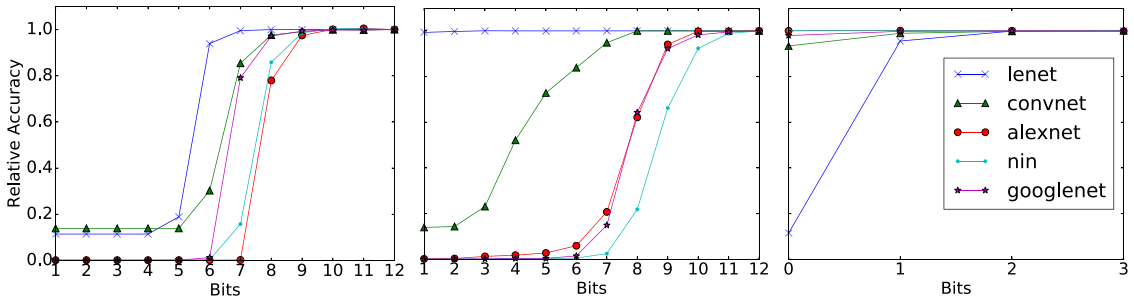
DNN library: We use the popular Caffe framework [7] to measure the effects of reduced precision on the overall network classification accuracy. The different DNNs are implemented in Caffe using a 32-bit single-precision floating-point representation for all numerical data.

How was precision varied per layer: To study the effect of numerical representation on accuracy, we convert the values to the desired representation and then back to single-precision floating-point prior to processing them in each layer. This is appropriate for any potential memory and communication optimizations that would not change the way computations are performed but rather how data is represented when communicated across layers either on-chip or through memory.

To perform this analysis, we modified the Caffe framework to capture data read and write calls and convert the default single-precision floating-point values into the specified lower precision fixed-point representation. Prior to starting the computation for each layer, we convert the fixed-point numbers into single-precision floating point.

Table 1
Networks studied: Accuracy reported is for the baseline configuration. Conv = convolution, FC = fully connected, IM = inception module.

Task	Data set	Network	Layer	Top-1 accuracy
Digit classification	MNIST	LeNet	2 Conv and 2 FC	0.9904
Image classification	CIFAR10	Convnet	3 Conv and 2 FC	0.7173
Image classification	ImageNet	AlexNet	5 Conv and 3 FC	0.5748
		NiN	12 Conv	0.5592
		GoogLeNet	2 Conv and 9 IM	0.6846



(a) Weights (b) Data (I) (c) Data (F)

Fig. 1. Accuracy relative to the baseline when the bit width is uniform for all layers.

Target numerical representation: We target a fixed-point representation for all values processed by the networks and study the length required for accurate classification. Fixed-point representations are compatible with integer arithmetic units and conversion from existing numerical data types is relatively straightforward. We parametrize an N-bit fixed point value as having I integer bits and F fractional bits. We study how changing I and F across layers and networks affects the overall network accuracy.

Values studied: We consider both the model values and the layer data outputs/inputs to each layer during classification. We use the terms **weights** to refer to the synaptic weights (after training) and **data** to refer to the output neuron activations of each layer.

DNNs studied: We consider the five most popular neural networks used for image classification which are listed in Table 1 along with their respective datasets [5,15–18]. They range from the relatively simple five layer LeNet to the 22 layer GoogLeNet. We use the models and the pre-trained weights that are available for these networks either through the Caffe distribution or the Caffe Model Zoo [19]. For ImageNet we use the ILSVRC2012 Task 1 dataset [20]. We run each network for 100 batches of 50–100 input images from the validation set of the respective dataset. The last column of Table 1 reports the accuracy achieved on the baseline Caffe implementation, which uses single-precision floating-point values.

Accuracy metric: We use top-1 accuracy (how often the network correctly identifies the precise class of an object) instead of the typical top-5 to increase the sensitivity to reduced precision error. Table 1 reports the baseline top-1 accuracy for the DNNs considered.

Assigning precision: For most networks we consider assigning a particular precision to each layer. We chose this granularity after observing stages within the same layer tend to have the same precision tolerance. For GoogLeNet, we assign a precision to each *inception module*[2] to simplify the analysis and refer to them as *layers* for simplicity.

A layer typically contains a single convolution or fully-connected stage, followed by simpler stages like activation, pooling and local response normalization. The activation function used in all of these networks is rectified linear (ReLU). Table 1 reports the number and type of layers per network.

2.2. Uniform representation across all layers

The results of this section confirm that precision requirements vary *across* networks. Specifically, this section studies the per network, minimum *uniform* representation length. For this analysis, we require that the same representation be used by all layers in the network.

Weights: Fig. 1(a) shows relative accuracy vs. with the number of bits used for the weights. The relative accuracy is the accuracy of the reduced precision network over the accuracy of the full precision network given in Table 1. Weights are real numbers, which can be clipped to the range [−1,1] without affecting accuracy. We fix the integer part to 1 bit and

[2] Inception modules consist of multiple convolutions with varying kernel sizes [18].

only report results when varying the fractional part of the fixed-point representation. Using 10-bit weights is sufficient to maintain accuracy for all networks studied.

Data: Fig. 1(b) and (c) report accuracy as we vary the number of bits used for the integer (b) and fractional (c) portions of the fixed-point representation for the data. Accuracy in Convnet and LeNet persists when the integer portion is at least 8 bits, whereas the other networks require at least 11 bits. Fig. 1(c) shows that most networks need just one fractional bit and some require at most two. These results suggest that a uniform fixed-point representation for the intermediate data values will require a 14-bit fixed-point representation, with 12 integer and 2 fractional bits. LeNet is used for character recognition, which is a relatively simple task compared to image recognition and requires less precision in the data. It can be limited to one integer bit while only reducing accuracy by less that 1%.

2.3. Per layer representation requirements

This section shows that precision requirements vary *within* each network.

Weights: The first column of Fig. 2 shows how DNN accuracy varies when we change the fixed-point representation used for the weights of one layer at time. As weights are typically between -1 and 1, we use a single integer bit (sign bit) and vary the number of fractional bits used by the fixed point representation. The results show that the minimum number of bits needed varies per layer and per network. For example, in LeNet, three bits are sufficient for layer 2, whereas seven bits are needed for layer 3, and in NiN, five bits are needed for layer 2 while nine are needed for layer 3.

Data: The last two columns of Fig. 2 show how DNN accuracy varies when we change the integer and the fractional portions of the data values respectively one layer at time. Focusing on middle column of Fig. 2, the integer portion requirements vary greatly across layers and across networks. The right column of Fig. 2 shows that variation exists in the per layer and per network needs for the fractional portion as well but to a lesser extent for each network. Due to the statistical nature of neural networks, rounding error can sometimes increase accuracy. This is at most an increase of 0.4% which we consider noise.

2.4. Choosing the per layer data representation

So far we studied the effect of changing the representation for one layer at a time. While some loss in fidelity in one layer can be acceptable, this does not suggest that we can simultaneously change the representation for multiple layers and still maintain overall network accuracy. This section studies how accuracy varies as we adjust the representation used by all layers at the same time. The goal is to find the *minimum* length representation possible per layer while maintaining overall network accuracy within acceptable limits. Such a configuration minimizes data traffic while maintaining accuracy.

To explore the design space in a reasonable amount of time we use gradient descent targeting the output accuracy of the network while adjusting the representation length used at each layer. Since the reduced precision error tolerance does not strictly decrease as data propagates through the network, we cannot easily prune the search space. For example in Fig. 2(h) AlexNet layer 5 is more error tolerant than layer 6. As such there is an exponential space of configurations to consider. To make the search tractable, we use slowest gradient descent to approximate the Pareto frontier in the accuracy vs. data traffic space. The algorithm is as follows:

1. Initialize layers to a uniform precision with the baseline accuracy.
2. Create a set of delta configurations by reducing each parameter, integer bits and fractional bits, in each layer by one.
3. Use the highest accuracy configuration to initialize the next iteration.

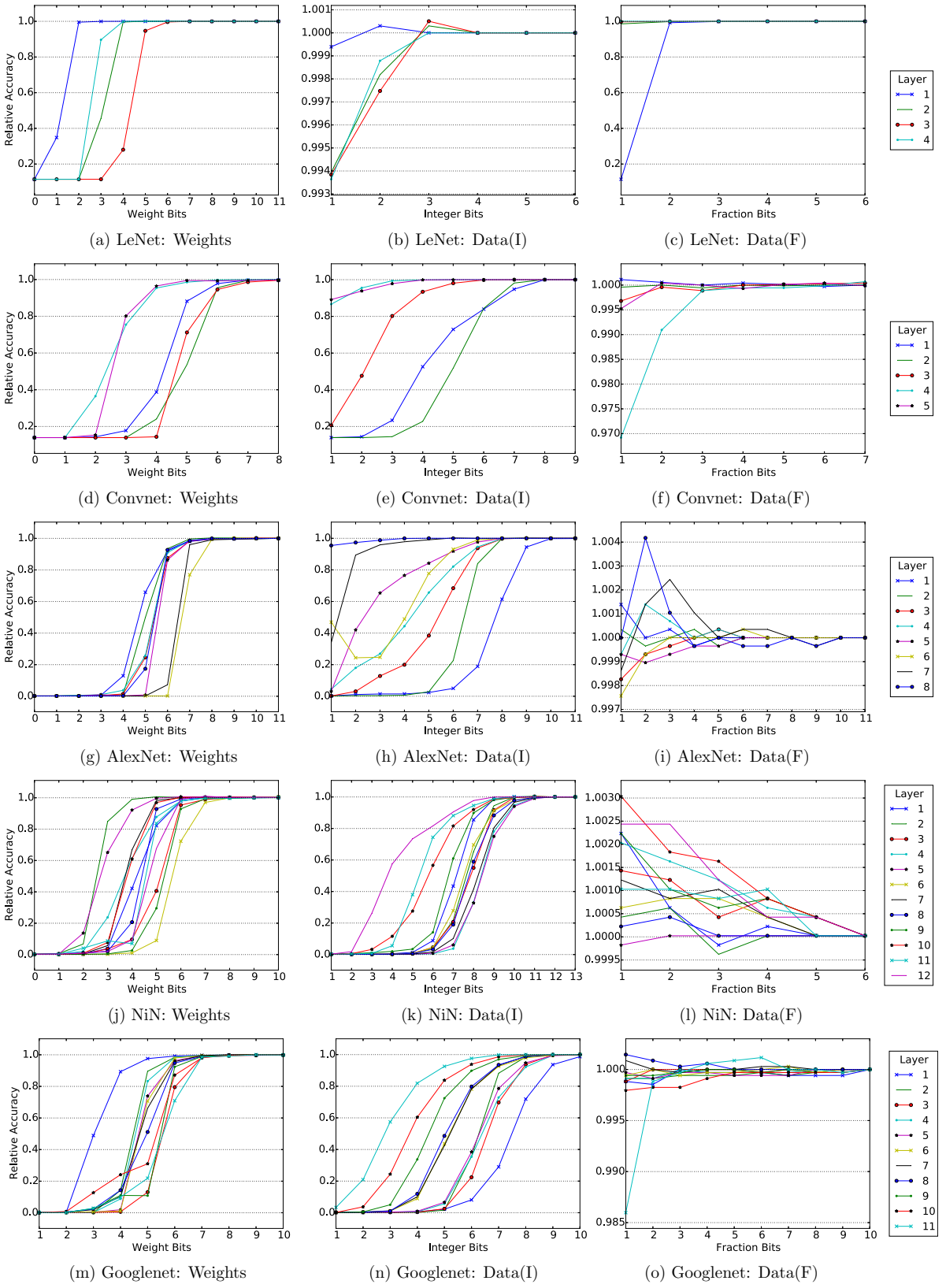
For the more complex networks, AlexNet, NiN and GoogLeNet, running this iterative algorithm is time consuming. To reduce the parameter space we fix the fractional bits to 0, 0 and 2 respectively. These achieve less than 0.1% error in Fig. 2 (right column). LeNet and Convnet are simpler networks and are less tolerant to fractional error so we also vary the fractional bits in our exploration for these networks.

Fig. 3 shows the results of this exploration reporting the resulting traffic and accuracy for several configurations studied. Traffic is calculated as the number of accesses times the number of bits per element (weight or data). Traffic is normalized to the baseline of 16 bits per element. Configurations are assigned to three categories:

1. *uniform* where all layers use the same numerical representation,
2. *mixed* where layers use different numerical representations as found with our iterative algorithm, and
3. *best* which highlight the Pareto frontier of the mixed configurations.

Generally, the best mixed configurations achieve lower bandwidth than uniform configurations for the same accuracy. However, there is one uniform configuration, highlighted in Fig. 3(d), that lies outside the Pareto frontier, demonstrating that our iterative algorithm is not optimal.

Table 2 reports the configurations that offer the minimum bandwidth for mixed networks given a limit on the error relative to the baseline accuracy of the network. We use 1%–10% as a reasonable range of tolerances that we expect for most use cases. Beyond 10% error the plots in Fig. 3 tend to drop off sharply, so there is little traffic reduction for a large increase in error. On average the optimal mixed configurations reduce the traffic by 49% with 1% error tolerance and 54% with 10% tolerance. If we ignore LeNet, the average traffic reduction ranges from 40% (1% tolerance) to 45% (10% tolerance).



(a) LeNet: Weights (b) LeNet: Data(I) (c) LeNet: Data(F)

(d) Convnet: Weights (e) Convnet: Data(I) (f) Convnet: Data(F)

(g) AlexNet: Weights (h) AlexNet: Data(I) (i) AlexNet: Data(F)

(j) NiN: Weights (k) NiN: Data(I) (l) NiN: Data(F)

(m) Googlenet: Weights (n) Googlenet: Data(I) (o) Googlenet: Data(F)

Fig. 2. Accuracy vs. representation length per layer: Weights (left), Data: Integer (center), and Fraction (right) portions.

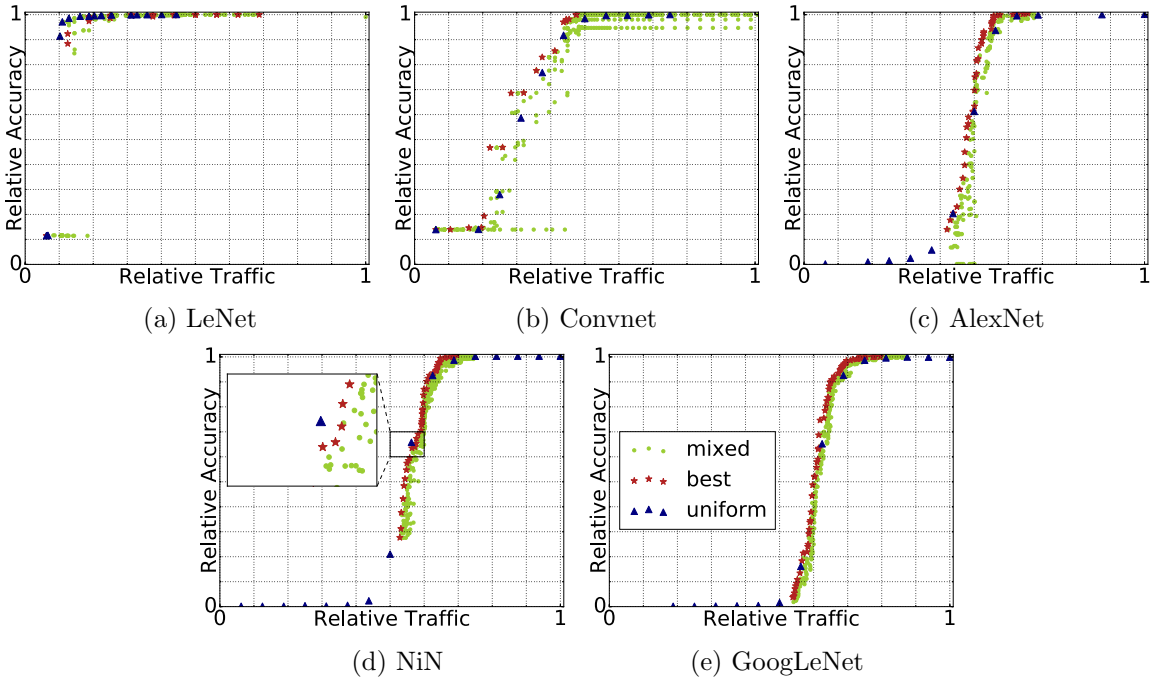


Fig. 3. Design space exploration: Accuracy vs. Traffic for different fixed-point representations per layer, relative to 16-bit fixed-point.

Table 2
Minimum relative traffic (RT) for mixed precision for error tolerance between 1% and 10%. LeNet and Convnet report the integer bits and fractional bits as *I.F*. Fractional bits are fixed for AlexNet, NiN and GoogLeNet and the total bit width is reported.

Tolerance	Bits per layer in I.F	RT	Tolerance	Bits per layer (I+F)	RT
LeNet			**AlexNet** (F = 0)		
1%	1.1-3.1-3.0-3.0	0.16	1%	10-8-8-8-8-8-6-4	0.56
2%	1.1-2.0-3.0-2.0	0.12	2%	10-8-8-8-8-8-5-4	0.56
5%	1.1-1.0-2.0-2.0	0.10	5%	10-8-8-8-7-7-5-3	0.56
10%	1.0-2.1-3.0-2.0	0.10	10%	9-8-8-8-7-7-5-3	0.52
Convnet			**NiN** (F = 0)		
1%	8.0-7.0-7.0-5.0-5.0	0.48	1%	10-10-9-12-12-11-11-11-10-10-10-9	0.64
2%	7.0-8.0-6.0-4.0-4.0	0.44	2%	10-10-9-12-12-11-11-11-10-10-10-9	0.64
5%	7.0-8.0-5.0-3.0-3.0	0.44	5%	10-10-10-11-10-11-11-11-10-9-9-8	0.64
10%	7.0-8.0-5.0-3.0-3.0	0.44	10%	9-10-9-11-11-10-10-10-9-9-9-8	0.60
			GoogLeNet (F = 2)		
			1%	14-10-12-12-12-12-11-11-11-10-9	0.72
			2%	13-11-11-10-12-11-11-11-11-10-9	0.70
			5%	12-11-11-11-11-11-10-10-10-9-9	0.68
			10%	12-9-11-11-11-10-10-10-10-9-9	0.64

3. Proteus

The variability in the representation length needed by each layer and each network can be exploited to improve the performance and energy of DNN implementations. We show how it can be used to reduce the memory traffic by proposing Proteus, or Pr. Pr adds a conversion layer between memory and the computation pipeline so that data can be stored in memory in reduced precision and computed at the system's native precision. The compute units can be general purpose processors, graphics processors or fixed-function accelerators.

We present Pr as an extension to the DaDianNao state-of-the-art DNN accelerator [1]. DaDianNao uses a 16-bit fixed-point format. Our Pr extension supports a fixed-point storage representation whose length varies from 1 to 16 bits. Section 3.1 first describes the baseline accelerator architecture. The sections that follow describe how Pr is incorporated over it.

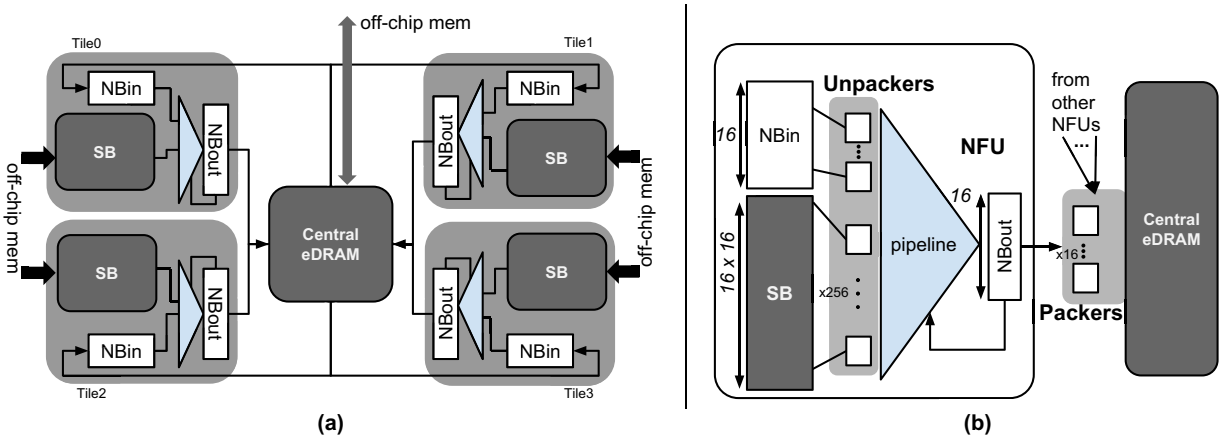


Fig. 4. a) Memory elements of an example accelerator including 4 NFUs. b) Proteus elements in a NFU.

3.1. Baseline accelerator architecture

The baseline accelerator we consider is a derivative of the *DaDianNao* accelerator. The accelerator incorporates 16 **tiles**. Fig. 4 shows an example design with four tiles. Each tile contains a *Neural Functional Unit (NFU)* for computation and a set of buffers. Each NFU is a 256-wide SIMD pipeline [21]. The input neuron activations (**data**) and synaptic **weights** are read from two buffers, *NBin* and *SB*, respectively. The output neuron activations and other intermediate results are stored in the *NBout* buffer.

Each NFU can process 16 input neurons and 256 synapses each cycle. Both synapses and neurons are represented using 16 bits. Every cycle 256 bits are read from NBin and 4096 bits are read from SB. At its output, the NFU can write 16 16-bit results, or 256 bits in total to NBout. NBin and NBout are each 2KB SRAMs and SB is a 2MB eDRAM. The system also contains a shared 4MB central eDRAM to store neuron data. An off-chip DRAM storage provides the initial input and the weights.

3.2. Adding proteus

With PR, data and weights are stored using the storage representation. The data and weights are converted to the internal compute representation just prior to entering the compute units, reducing memory access energy, and data footprint. As Fig. 4(b) shows, PR adds a set of *unpackers* to the front of the pipeline and a set of *packers* to the back of NBout.

3.3. Packing and unpacking data

Conversion between the variable length storage fixed-point representation to the 16-bit compute representation is conceptually straightforward. However, to sustain the NFU wide datapath, PR needs to feed the execution units with a continuous stream of inputs and weights. First considering just the unpackers for the 256 weights in one tile, PR would need to provide a *weight group* of 256 16-bit weights or 512B per cycle to the execution units.

A naive approach would be to pack the reduced precision data into consecutive bits in memory in a linear fashion. However, due to the large width of the pipeline, unpacking this data and aligning it to the appropriate compute unit would require significant lateral data movement and the complexity of selecting data from 4096 possible alignments. The energy overhead of this unpacking networks could easily overwhelm any of the benefits from PR.

To avoid this, PROTEUS opts for a hardware/software co-operative approach where data and weights are laid out in memory in a way that facilitates simple, low-cost and low-energy packers and unpackers.

3.4. The proteus memory layout

PR groups weights first into a set of 256 virtual columns, each corresponding to one input of the 256 execution units. For example, the weights that need to arrive at input port 0 are all packed into virtual column 0. By mapping all weights per port to the same virtual column, each of the unpackers needs to consider only its own virtual column of 16 bits instead of the whole memory block of 512B. Similarity for the neuron data, memory is divided into 16 virtual columns. Fig. 5 shows a simple example of the memory layout for a compute representation of 4 bits and two parallel inputs per unit, where the data elements are packed into 3 bits.

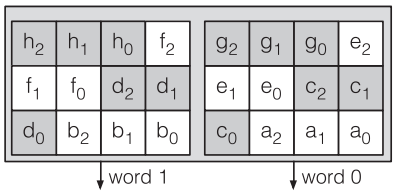


Fig. 5. PROTEUS Layout example: 3-bit values packed into two 4-bit wide virtual columns.

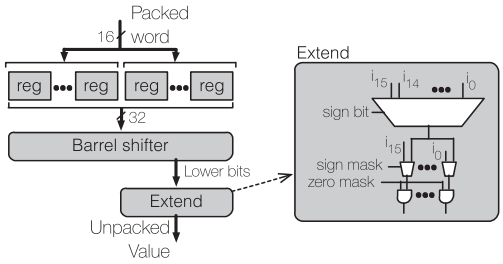


Fig. 6. Unpacker.

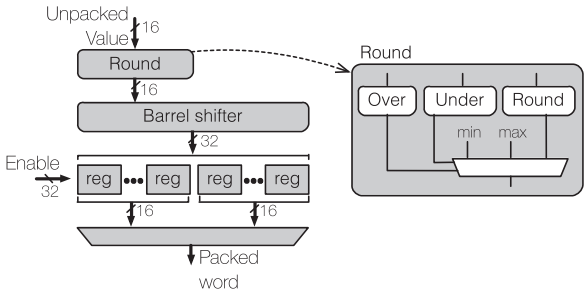


Fig. 7. Packer.

3.5. Packers and unpackers

The *unpacker* and the *packer* convert from and to the storage representation. Fig. 4(b) shows the packers and unpackers integrated into the baseline. At the interface of the central eDRAM there will be 16 packers to pack data before storing. The discussion that follows considers a single unpacker. However, all unpackers operate in tandem performing the same actions on different virtual columns. The data is still read as a 512B memory block as in the baseline design.

Unpacker: Fig. 6 shows the unpacker. Let P denote the storage representation width in bits. To allow a value split between two rows to be recombined, it uses a double width (32-bit) register. When a new 16b word needs to be read in, it is loaded into either the upper or lower word of the register in an alternating fashion. Only once all of the bits have been unpacked from either the upper or lower word does the unpacker read the next word in its place. A circular shifter can recombine values split both between bit 15 and 16 and bit 31 and 0. The 32 bits in the register are rotated to align the desired value to the 16-bit output. Once the desired value is in the right bit position, it is sign-extended, at both the upper and lower bits, to an equivalent 16-bit fixed point value. Since there are many parallel unpackers operating in lock step for each buffer, they all share the same control signals and decoding can be done by a single control block per buffer.

Packer: Fig. 7 shows the packer, which like the unpacker, uses a 32-bit circular shifter and a 32-bit register. In addition, the packer rounds data to the nearest P-bit representation to minimize the approximation error. If the unpacked value is outside the range of the reduced precision then it saturates to the maximum or minimum P-bit value. The rounded unpacked data is then shifted to align with the next available space in the packing register. When loading in to the packing register, only P bits are loaded, so we need a 32-bit mask, *enable*, to specify which bits to load. Once a full 16-bit word has been packed, in either the upper or lower 16 bits, it is written out.

4. Evaluation

This section presents an experimental evaluation of PROTEUS applied to the DaDianNao accelerator. Section 4.1 describes the methodology. Section 4.2 evaluates traffic and power reduction for single image inference. Section 4.3 shows that traffic and power reduction persist in *batching* mode. Section 4.4 reports the area overhead of adding PROTEUS.

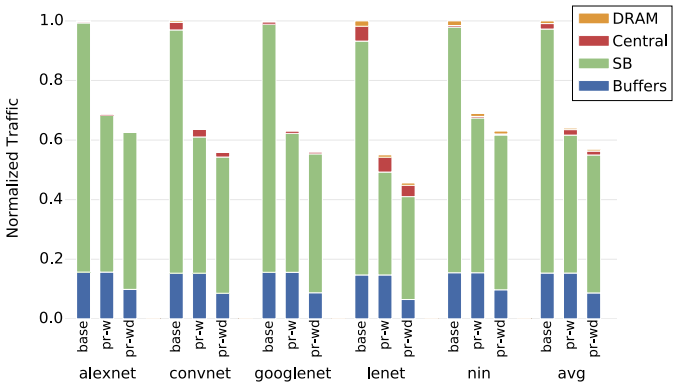


Fig. 8. Single image: Traffic of each network normalized to the baseline.

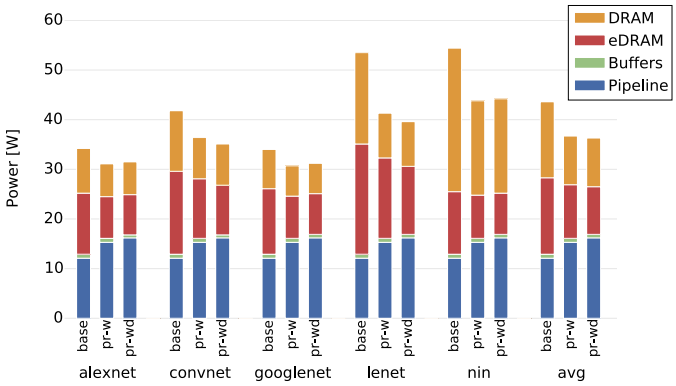


Fig. 9. Single image: Power breakdown for each network.

4.1. Methodology

To model the hardware overheads of PROTEUS, we implemented the baseline NFU pipeline, packers, and unpackers in Verilog. We synthesized the design using the Synopsis Design Compiler vH-2013.03-SP5-2 [22] with the TSMC 65nm library. We modelled the area and power for the main accelerator SRAM buffers, NBin and NBout using CACTI v5.3 [23]. The eDRAM energy was modelled with *Destiny* [24]. The off-chip memory was modelled using DRAMSim2 [25] with 4GB of DDR3 running at 800Mhz. The energy models for the different accelerator components were integrated into a cycle-level model of the accelerator to estimate overall power and execution time. For simplicity we assume the weights of each layer fit on chip. This is true for all layers except AlexNet layer 6, in which case PR would improve performance as well.

We consider two configurations of PR with different area overheads: 1) **PR -W**, where only the weights are packed, requiring only 256 unpackers per tile. 2) **PR -WD**, where weights and data are packed, requiring 272 unpackers per tile and 16 packers at the central eDRAM, as shown in Fig. 4(b).

4.2. Single image

We will first consider the task of classifying a single image. Fig. 8 shows a breakdown of data traffic across the various storage units: the SRAM buffers, eDRAM (SB and central) and off-chip DRAM. The leftmost bar for each network is for the baseline accelerator using 16-bit precision. Traffic is dominated by the large 2MB SBs in each tile, accessing 512B every cycle. The second bar shows PR applied only to the weights (PR -W). This reduces traffic in the SB eDRAM and DRAM. Just compressing the weights reduces traffic by 36% on average. The third bar shows PR applied to both the weights and the data (PR -WD). Now PR also reduces the traffic to the central eDRAM and NBin. When compressing both data and weights PR reduces traffic by 43% on average.

Fig. 9 shows the power breakdown for each DNN for the *baseline* system and PR. The breakdown of average power consumption in the system is 37% for eDRAM, 33% for DRAM, 29% for the pipeline and 2% for the buffers. The next two bars per network in Fig. 9 shows the total power of each network with PR -W and PR -WD. Compressing just the weights yields a power savings of 14.5% on average. Compressing the data as well increases the powers savings to 15.2% on average.

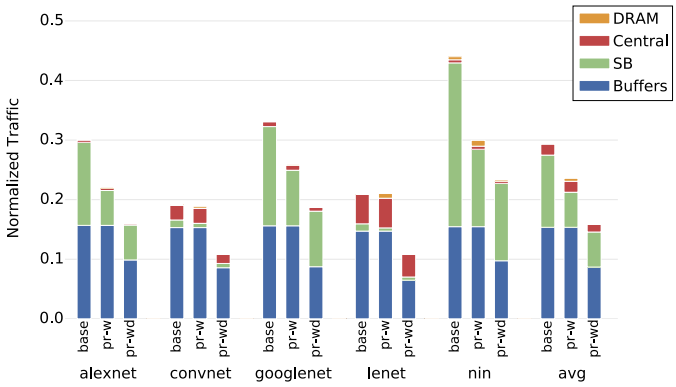


Fig. 10. Batching: Per image traffic of each network normalized to the baseline of the single image case.

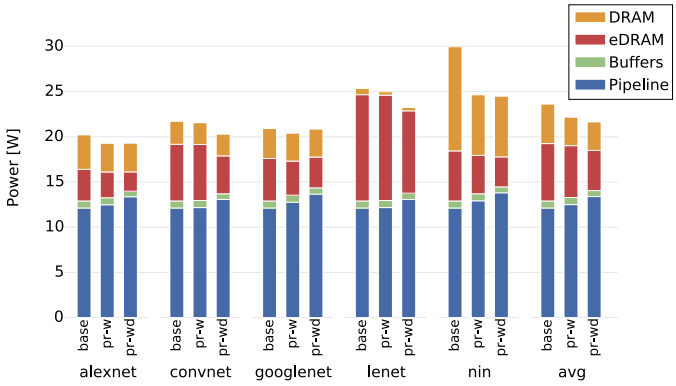


Fig. 11. Batching: Power of each network.

4.3. Batching

Batching is a common strategy for improving efficiency by reusing a set of weights for a batch of multiple images. This amortizes the costs associated with loading the weights, from off-chip and from SB, across multiple images yielding improved energy consumption.

To conserve DRAM energy we chose the largest batch sizes that still allow all neuron data to be kept on chip, based on the size of the central eDRAM. For the larger networks the baseline accelerator can support batches of 3–6 images. Batching will reduce the impact of compression, since it will reduce the relative amount of power consumed in accessing the weights. However, by compressing the data PR is able to fit more images in the central eDRAM, allowing us to increase the batch size and thus the reuse of the weights.

Fig. 10 shows the traffic of each network and memory component normalized to the single image baseline. Batching alone reduces the memory traffic, per image, by 71%. With PR -W, traffic is reduced an additional 5%–76%. When we also compress the data, traffic is reduced an additional 8%–84%. Here compressing data has a bigger impact than compressing the weights since the weight traffic has already been amortized by batching.

Fig. 11 shows the power breakdown. With batching, PR results in power savings of 29% in eDRAM and 27% in DRAM. The overhead in the pipeline is now lower at 11% since it amortizes the dynamic power cost of unpacking the weights. Overall, PR with batching yields power savings of 8% on average.

4.4. Area overhead

The packers and unpackers of PROTEUS increase the area of the pipeline only. The area of one pipeline in the baseline system is 0.94 mm^2, 1.10 mm^2 for PR -W and 1.12 mm^2 for PR -WD, an increase of 17% and 19%, respectively. Taking into account that the pipeline is 12% of the total baseline system area, PR -W and PR -WD only increases the system area by 2% and 2.3%, respectively.

5. Related work

Reduced precision neural networks has been an active topic of research for many years [26–32]. Gupta et al. showed how to train a DNN by using 16-bit fixed-point numbers and stochastic rounding [9]. The resulting network was able to perform a classification task with minimal error with respect to the baseline. They also proposed a hardware accelerator with low-precision fixed-point arithmetic supporting stochastic rounding.

Courbariaux et al. recently trained neural networks using three different representations: floating point, fixed point, and dynamic fixed point. They showed that it is possible not only to test but also train this set of networks with low-precision representations [10]. However, they used a uniform representation for the whole network. In following work they also showed that networks can be trained with binary weights without loss of accuracy [13,14]. For MNIST, they used a fully connected network with more weights than LeNet. Interestingly, the total number of weight bits is comparable: 2.9 million for their network vs 3 million for LeNet with 7 bit weights.

Seide et al. showed how to quantize the gradients using only one bit per value to train a DNN using Stochastic Gradient Descent without loss of accuracy [33]. They used linear quantization and retraining to achieve smaller precisions and improve the accuracy of the network. The same techniques could be applied to Pr. Kyukeon et al. and Anwar et al. quantized the signal (data) and weights in a fully connected network and CNNs and consider different quantization steps per layer [11,12]. In the latter work, they also analyzed the per-layer sensitivity to the number of quantization levels in LeNet but select the number of bits per layer manually. Kim et al. used reduced precision for a custom neural network circuit design [34]. However, this implementation lacks the configurability to run different networks at different precisions.

Han et al. demonstrated how to compress weights using an elaborate process of pruning, quantization, sparse matrix encoding, Huffman coding and retraining [35]. They also designed an accelerator to run these compressed networks [36] which trades off computational efficiency of convolutional layers for significant footprint reduction. Pr yields a more modest footprint reduction but maintain the computational efficiency of the baseline. Recently, Jain et al. proposed a similar technique of bit-packing reduced precision data to reduce footprint and bandwidth, but targeting general purpose processors [37].

6. Conclusion

Classification applications and quality in deep neural networks is currently limited by compute performance and the ability to communicate and store numerical data. An effective technique for improving performance and data traffic is by using lower precision. This work provides a detailed characterization of the per-layer reduced precision tolerance of a wide range of neural networks. We proposed a method for determining precisions, per layers, that offers a good trade off between accuracy and data traffic.

To exploit this, we proposed Proteus, a layered extension over existing hardware that converts data on-the-fly when communicating to and from memory reducing overall data traffic and memory energy. Using simulation, Proteus ' benefits were demonstrated over a state-of-art DNN accelerator. While we presented Proteus as a hardware extension, it is likely that a variation of the approach can be implemented purely in software as well. It may be possible to utilize the precision variability to boost execution bandwidth within the pipeline. The results of this work also motivate further work in exploiting the reduced precision for reducing memory bandwidth and footprint, communication bandwidth, and potentially computation bandwidth. The potential benefits include energy reduction, higher performance and the ability tosupport larger networks.

Acknowledgements

We thank the anonymous reviewers for their comments and suggestions. This work was supported by an NSERC Discovery Grant, an NSERC Discovery Accelerator Supplement and an NSERC CGS-D Scholarship.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.parco.2017.05.003

References

[1] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, O. Temam, DaDianNao: a machine-learning supercomputer, in: Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on, 2014, pp. 609–622, doi:10.1109/MICRO.2014.58.
[2] R. Girshick, J. Donahue, T. Darrell, J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, arXiv preprint arXiv:1311.2524 (2013).
[3] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, Overfeat: Integrated recognition, localization and detection using convolutional networks, CoRR abs/1312.6229 (2013). http://arxiv.org/abs/1312.6229
[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A.L. Yuille, Semantic image segmentation with deep convolutional nets and fully connected CRFs, CoRR abs/1412.7062 (2014). http://arxiv.org/abs/1412.7062
[5] A. Krizhevsky, cuda-convnet: High-performance C++/CUDA implementation of convolutional neural networks, 2011, (https://code.google.com/p/cuda-convnet/). Accessed 2017-05-28.
[6] V.K. Chippa, S.T. Chakradhar, K. Roy, A. Raghunathan, Analysis and characterization of inherent application resilience for approximate computing, in: Proceedings of the 50th Annual Design Automation Conference, in: DAC '13, ACM, New York, NY, USA, 2013, pp. 113:1–113:9, doi:10.1145/2463209.2488873.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, arXiv preprint arXiv:1408.5093 (2014).
[8] I. Buck, NVIDIA's Next-Gen Pascal GPU Architecture to Provide 10X Speedup for Deep Learning Apps, 2015, (http://blogs.nvidia.com/blog/2015/03/17/pascal/). Accessed 2017-05-28.
[9] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, CoRR (2015). arXiv:1502.02551.
[10] M. Courbariaux, Y. Bengio, J.-P. David, Low precision arithmetic for deep learning, CoRR abs/1412.7024 (2014). http://arxiv.org/abs/1412.7024
[11] K. Hwang, W. Sung, Fixed-point feedforward deep neural network design using weights +1, 0, and -1, in: Signal Processing Systems (SiPS), 2014 IEEE Workshop on, 2014, pp. 1–6, doi:10.1109/SiPS.2014.6986082.
[12] S. Anwar, K. Hwang, W. Sung, Fixed point optimization of deep convolutional neural networks for object recognition, in: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 1131–1135, doi:10.1109/ICASSP.2015.7178146.
[13] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, CoRR abs/1511.00363 (2015). http://arxiv.org/abs/1511.00363
[14] Z. Lin, M. Courbariaux, R. Memisevic, Y. Bengio, Neural networks with few multiplications, CoRR abs/1510.03009 (2015). http://arxiv.org/abs/1510.03009
[15] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324, doi:10.1109/5.726791.
[16] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. Burges, L. Bottou, K. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.
[17] M. Lin, Q. Chen, S. Yan, Network in network, CoRR abs/1312.4400 (2013). http://arxiv.org/abs/1312.4400
[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S.E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, CoRR abs/1409.4842 (2014). http://arxiv.org/abs/1409.4842
[19] Y. Jia, Caffe Model Zoo, 2015. https://github.com/BVLC/caffe/wiki/Model-Zoo.
[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, arXiv:1409.0575 [cs] (2014). ArXiv: 1409.0575, http://arxiv.org/abs/1409.0575
[21] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, O. Temam, Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning, in: ACM Sigplan Notices, 49, ACM, 2014, pp. 269–284.
[22] Synopsys, Design Compiler, (http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx).
[23] N. Muralimanohar, R. Balasubramonian, CACTI 6.0: A Tool to Understand Large Caches.
[24] M. Poremba, S. Mittal, D. Li, J. Vetter, Y. Xie, DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2015, 2015, pp. 1543–1546.
[25] P. Rosenfeld, E. Cooper-Balis, B. Jacob, DRAMSim2: A cycle accurate memory system simulator, IEEE Comput. Archit. Lett. 10 (1) (2011) 16–19, doi:10.1109/L-CA.2011.4.
[26] Y. Xie, M.A. Jabri, Training Algorithms for Limited Precision Feed Forward Neural Networks, Technical Rep, 1991.
[27] R. Presley, R. Haggard, A fixed point implementation of the backpropagation learning algorithm, in: Southeastcon '94. Creative Technology Transfer - A Global Affair., Proceedings of the 1994 IEEE, 1994, pp. 136–138, doi:10.1109/SECON.1994.324283.
[28] J. Holt, T. Baker, Back propagation simulations using limited precision calculations, in: Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on, vol. ii, 1991, pp. 121–126 vol.2, doi:10.1109/IJCNN.1991.155324.
[29] A. Strey, N. Avellana, A new concept for parallel neurocomputer architectures, in: European Conference on Parallel Processing, Springer, 1996, pp. 470–477.
[30] D. Larkin, Kinane A., N. OConnor, Neural Information Processing, Towards hardware acceleration of neuroevolution for multimedia processing applications on mobile devices, 2006, pp. 1178–1188.
[31] K. Asanovic, N. Morgan, Using simulations of reduced precision arithmetic to design a neuro-microprocessor, J. VLSI Sig. Proc. (1993) 33–44.
[32] J.L. Holt, J.-N. Hwang, Finite precision error analysis of neural network hardware implementations, IEEE Trans. Comput 42 (1993) 281–290.
[33] F. Seide, H. Fu, J. Droppo, G. Li, D. Yu, 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns., in: INTERSPEECH, 2014, pp. 1058–1062.
[34] J. Kim, K. Hwang, W. Sung, X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks, in: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 7510–7514, doi:10.1109/ICASSP.2014.6855060.
[35] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding, CoRR abs/1510.00149 (2015). http://arxiv.org/abs/1510.00149
[36] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, W.J. Dally, EIE: efficient inference engine on compressed deep neural network, CoRR abs/1602.01528 (2016). http://arxiv.org/abs/1602.01528
[37] A. Jain, P. Hill, S.-C. Lin, M. Khan, M.E. Haque, M.A. Laurenzano, S. Mahlke, L. Tang, J. Mars, Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation, in: Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, in: MICRO-49, IEEE Computer Society, 2016.