Value-Based Deep-Learning Acceleration

Andreas Moshovos
University of Toronto

Jorge Albericio
NVIDIA

Patrick Judd
University of Toronto

Alberto Delmás Lascorz
University of Toronto

Sayeh Sharify
University of Toronto

Tayler Hetherington
University of British Columbia

Tor Aamodt
University of British Columbia

Natalie Enright Jerger
University of Toronto

This article summarizes our recent work on value-based hardware accelerators for image classification using Deep Convolutional Neural Networks (CNNs). The presented designs exploit runtime value properties that are difficult or impossible to discern in advance. These include values that are zero or near zero and that prove ineffectual, have reduced yet variable precision needs, or have ineffectual bits. The designs offer a spectrum of choices in terms of area cost, energy efficiency, and relative performance when embedded in server class installations. More importantly, the accelerators reward advances in CNN design that increase the aforementioned properties.

A few years ago, the idea of computing systems that can converse with us or drive cars seemed the stuff of fiction. Yet today, computing systems are learning to "see," "hear," "read," "write," and interact with the physical world in ways we typically associate with intelligent beings. The technology that drives these advances is machine learning, with most advances due to the sub-branch of deep learning (DL). While conventional programs have to be meticulously constructed to anticipate any possible scenario (which is often an impossible task), DL systems "learn" by example or by trial and error and are ultimately able to handle unanticipated scenarios.

While the foundations of DL date back to the 1950s, it is only in recent years that DL has become practical due to the confluence of three factors: 1) DL algorithmic innovations, 2) the availability of vast amounts of computerized data to learn from, and 3) the advent of sufficiently powerful hardware. These recent DL successes have fueled efforts for broadening the scope and sophistication of DL applications. However, further innovation in DL greatly depends on computing hardware systems with even greater computational power. In the past decades, hardware

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

performance advances were possible at regular intervals, primarily because semiconductor technology scaling enabled us to build hardware with more and faster transistors for the same or lower cost. Unfortunately, this is no longer true due to power constraints.[1]

A viable way to sustain these much-needed performance advances is through hardware accelerators. A hardware accelerator is designed to excel at the processing of a specific algorithm or class of algorithms. It sacrifices generality for greater energy efficiency and performance. Accelerators have been extremely successful in various application domains such as computer graphics and communications, where performance improvements of three orders of magnitude are routinely possible. We believe that DL applications can similarly benefit and have been designing such accelerators targeting DL applications.

In particular, we focus on a core application of DL: image classification. Image classification identifies various objects in an image frame, which is in a 2D projection of the physical world as acquired by an imaging sensor. The building blocks used for this DL application are very similar and often identical to those used for other DL applications. Accordingly, techniques developed for accelerating image classification should be broadly applicable. For example, autonomous driving (AD) is an application area where DL is a key enabling technology and where the needs for higher data storage and processing capacity is evident.[2] Many components of a complete AD system use DL. This article focuses on using DL to detect objects around the vehicle, a key piece of information needed for navigation.

This article summarizes our recent work on value-based hardware accelerators for image classification using Deep Convolutional Neural Networks (CNNs). We provide background on CNNs, and then introduce acceleration concepts, present a state-of-the-art structure-based CNN accelerator, and describe our value-based approach. We explain three value properties and the corresponding value-based accelerator designs. Finally, we comment on informational inefficiency that underlies our DL acceleration approach, as well as many other similar approaches.

CONVOLUTIONAL NEURAL NETWORKS

Figure 1 shows that a CNN is a software pipeline of layers. An input, which is typically a 2D image, is fed into the first layer. The image is treated as a 3D array comprising three 2D images (one per red, blue, and green color plane). Each value is an activation, and the purpose of the CNN is to infer what type of object these activations represent. In its simplest form, the CNN outputs a vector of probabilities with one element per possible object class. This approach can be extended to detect the position of multiple objects or even the trajectory of objects given a sequence of frames. As the figure shows, the output can be an annotated image with bounding boxes identifying various objects of interest such as other vehicles, pedestrians, and street signs.

While there are often many layers, there are only a few layer types. In image classification, the convolutional layers (CVLs) account for more than 90 percent of the execution time on modern graphics processors, the current commodity architecture of choice for CNNs. Fully connected layers (FCLs), a specialized form of CVLs, account for most of the remaining time.

Informally, each CVL applies several filters to the 3D input activation array, producing a 3D output activation array (shown at the bottom of Figure 1). Each output activation is the result of an inner-product of a filter and an equally sized subarray of the input activation array. The filters are 3D arrays of weights, which are pre-determined values that contain the network's "knowledge." The weights are determined during an earlier training phase where the network, starting from randomly selected weight values, is trained by processing several pre-annotated example images. A typical CVL performs hundreds to thousands of inner products, each comprising hundreds to thousands of activation and weight pairs.

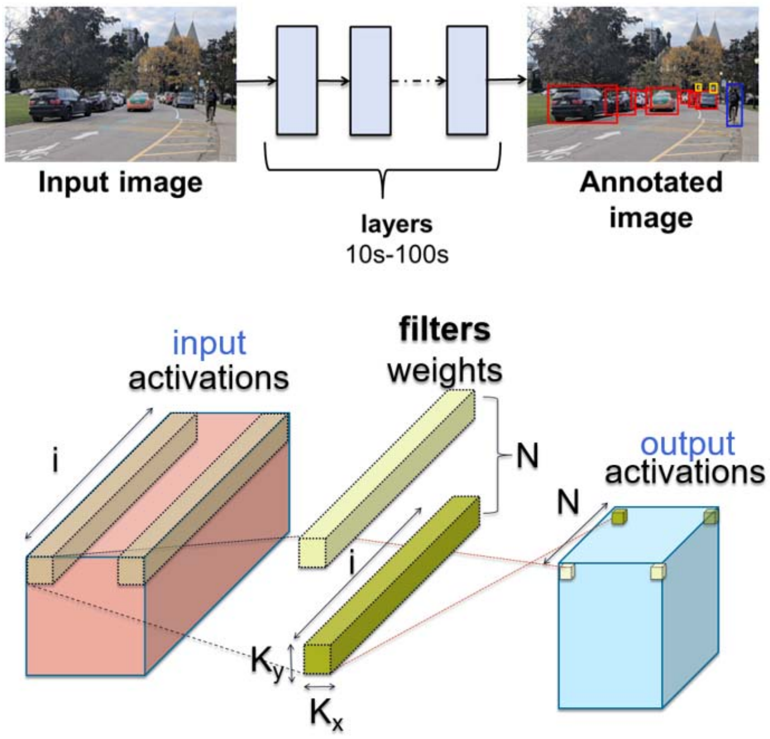


Figure 1. Top, a CNN; Bottom, a CVL.

Formally, a CVL processes and produces 3D activation arrays of real numbers. The layer applies N_f 3D filters in a sliding window fashion using a constant stride S to produce an output 3D array. The input array contains $N_x \times N_y \times N_i$ activations. Each of the N_f filters contains $K_x \times K_y \times N_i$ real valued weights. The output activation array dimensions are $O_x \times O_y \times N_f$; that is, its depth equals the filter count. Each filter corresponds to a desired feature, and the goal of the layer is to determine where in the input activation array these features appear. Accordingly, each constituent 2D array along the n dimension of the output activation array corresponds to a feature. The layer computes the inner product of a filter and a window, a filter-sized, or $K_x \times K_y \times N_i$ subarray of the input activation array. The inner product is then passed through an activation function, such as the Rectified Linear Unit (ReLU) producing an output activation. If $a(y,x,i)$ and $o(y,x,i)$ are respectively input and output activations, $w^f(x,y,i)$ are the weights of filter f, and R is the activation function. The output activation at position (x',y',n) is given by:

$$\underbrace{o(x',y',n)}_{\substack{\text{output}\\ \text{activation}}} = R(\underbrace{\sum_{x=0}^{Kx-1}\sum_{y=0}^{Ky-1}\sum_{i=0}^{Ni-1}\underbrace{w^f(x,y,i)}_{\text{weight}} \times \underbrace{a(x+x'\times S, y+y'\times S, i)}_{\text{input activation}}}_{\text{window}}).$$

The layer applies filters repeatedly over different windows positioned along the X and Y dimensions with stride S, and there is one output activation per window and filter. Accordingly, the output activation array dimensions are $O_x = (N_x - K_x) / S + 1$, $O_y = (N_y - K_y) / S + 1$, and $O_i = N_f$. A fully connected layer can be implemented as a CVL where the filter dimensions match those of the input activation array.

For clarity, we use the term “brick” to refer to a set of 16 elements of a 3D activation or filter array, which are contiguous along the i dimension (for example, $n(x,y,i)...n(x,y,i + 15)$). Bricks will be denoted by their origin element with a B subscript (for example, $n_B(x,y,i)$).

ACCELERATION PRIMER

Being able to process CNNs faster can yield multiple benefits for future ADs: it enables processing higher resolution frames, it enables processing multiple imaging sources, and it reduces reaction times. As power has become the limiting factor in modern hardware, boosting processing speed further requires improving energy efficiency (that is, reducing the amount of energy that is expended to perform each computation).

Therein lies the great opportunity for hardware accelerators. Specifically, existing computing hardware is often general purpose in that it is designed to perform relatively well for numerous applications. Unfortunately, with generality comes inefficiency. For example, consider the simple task of reducing (adding 128 numbers), a subcomputation of an output activation calculation. We should be able to just read these 128 numbers and add them together. However, conventional processors perform a lot more work to achieve the same effect. As Figure 2 shows on the left, a conventional processor implements this reduction using a loop of several instructions. Executing an instruction entails several actions: reading the instruction representation from memory, decoding it into a set of actions, reading the source data operands, performing a calculation, writing the result, and figuring out which instruction to execute next. To reduce 128 values, a processor executes 643 instructions and performs 643 and 128 memory reads for the instructions and the data inputs, respectively.

```
       movia r8, a
       movia r9, 128
       movi r7, 0
loop:  ldw r6, 0[r8]
       add r5, r5, r6
       add r8, r8, 4
       add r7, r7, 1
       bne r7, r9, loop
```

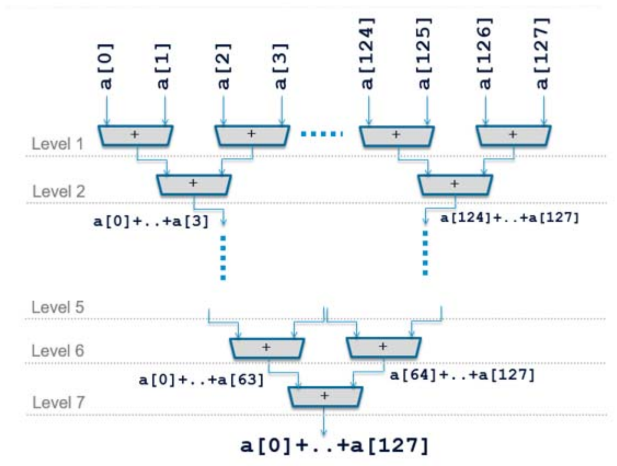


Figure 2: Adding 128 numbers: Left, machine code implementation; Right, structure-based hardware accelerator.

Specialized reduction hardware could instead simply use an adder tree as Figure 2 shows on the right. It would perform just 128 data reads, or even a single wide read, and 128 additions organized in seven levels/steps. This is an example of a computation structure-based accelerator.

The accelerators described in this work go beyond this conventional approach to acceleration by also relying on data content. Examining data content opens up new opportunities for acceleration. For example, what if roughly 50 percent of our numbers are zero, but we do not know in advance which ones? Could we build a better accelerator?

A Structure-Based CNN Accelerator

An example of a state-of-the-art accelerator that relies on the computation structure of neural networks is DaDianNao (DaDN).[3] As Figure 3 shows on the left, a DaDN chip comprises 16 processing tiles, each with its own slice of 2-MB eDRAM weight memory (WM), and a central 4-MB eDRAM activation memory (AM). Each tile contains 16 inner product units (IPUs), each containing 16 multipliers and an adder tree to compute the inner product of a weight and an activation brick each cycle. Each cycle, an input activation brick is broadcast from AM to all IPUs, while each IPU fetches a different weight brick. In total, a tile processes 256 weights and 16 activations per cycle. Larger inner products are computed over multiple cycles and finally are passed through an activation function, *R*, to produce an output activation. Each cycle, the whole chip processes 16 activations and 256 × 16 = 4*K* weights, producing 16 × 16 = 256 partial sums. Processing starts by reading from external memory the first layer's filter weights and the input image. The weights are distributed over the WMs, and the input is stored into AM. While the current layer is being processed, the weights for the next layer can be loaded from off-chip. DaDN is a structure-based accelerator because it implements the layer computations as-is, regardless of the actual values being computed.

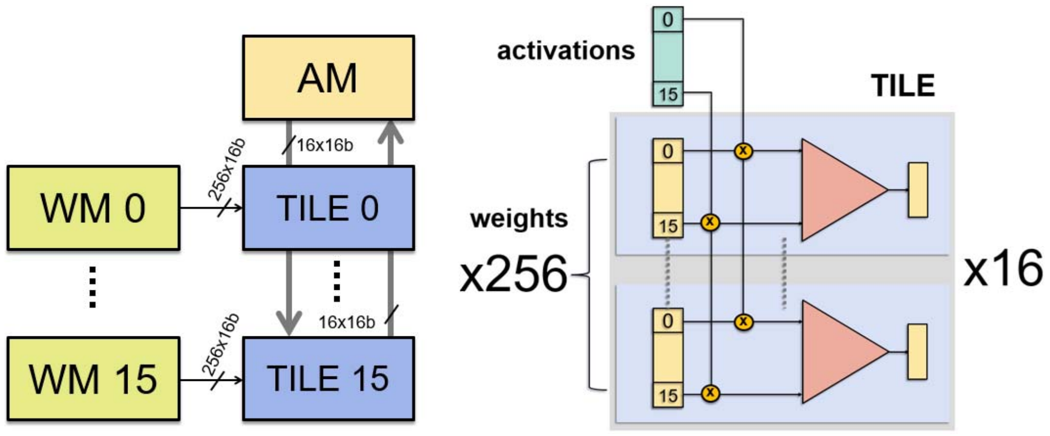


Figure 3. Left, a DaDN overview; Right, a DaDN tile.

The discussion that follows uses DaDN as the baseline design. However, the techniques described are applicable to other accelerator organizations, as well. Because DaDN is a massively data-parallel design, our implementations also target this design style. Specifically, two important properties that we wish to maintain are: 1) wide, regular accesses to the various memories and 2) regular computations across all tiles and IPUs. The first property maintains high utilization of precious on- and off-chip memory bandwidth, while the second allows the use of a common control unit and, thus, avoids the cost of many little independently operating processing units.

OUR APPROACH

We mirror the design philosophy that yielded many successful techniques for improving performance for modern general-purpose cores, which exploit the expected behavior of programs; programs do not behave randomly but rather tend to exhibit specific idiosyncratic behaviors. A well-known example is biased branches, which branch predictors exploit. Accordingly, we target identifying properties in the value stream of CNNs. Additional opportunities exist by co-designing the CNN software and hardware. However, not requiring any modifications to the CNN results in immediate benefits and reduces the burden on the CNN designers.

Methodology

All systems are modelled using the same methodology for consistency. A custom cycle-accurate simulator models execution time. Computation is scheduled such that all designs see the same weight reuse. To estimate power and area, all tile pipeline designs are synthesized with the Synopsys Design Compiler for a TSMC 65 nm library and laid out with Cadence Encounter. Circuit activity is captured with ModelSim and fed into Encounter for power estimation. All SRAM buffers are modelled with CACTI. The eDRAM area and energy are modelled with Destiny.

Overview

Table 1 provides details on the DaDN configuration to which we compare our designs. Table 2 summarizes our various designs and their relative performance, energy efficiency, and area normalized to DaDN.[3] The following sections describe each of these designs in more detail.

Table 1. The DaDN configuration.

Accelerator	Configuration	Performance	Power	Area	Frequency	Tech. Node
DaDN	16-16-16	3.9 Tmul/sec	17.6 Watt	78 mm^2	980 Mhz	65 nm

Table 2. Value-based accelerator characteristics relative to the DaDN configuration in Table 1.

Accelerator	Compared to Table 1 DaDN Configuration	Relative Performance	Relative Energy Efficiency	Relative Area	Layers	Value Property
Cnvlutin[4]	16-16-16	1.6×	1.47×	1.05×	CVL	Ineffectual Activation Values
Stripes[5]	16-16-16	1.9×	1.14×	1.32×	CVL	Per Layer Activation Precision
Tartan[6]	16-16-16	1.9× (CVL) 1.6× (FCL)	1.18× (CVL) 1.06× (FCL)	1.49×	CVL and FCL	Per Layer Activation and Weight Precision
Dynamic Stripes[7]	16-16-16	2.6×	1.54×	1.35×	CVL	Dynamic Activation Precision
Loom[8]	1-8-16	3.25× (CVL) 1.74× (FCL)	2.63× (CVL) 1.41× (FCL)	1.34×	CVL and FCL	Dynamic Activation and Per Layer Weight Precision
Pragmatic[9]	16-16-16	4.3×	1.71×	1.68×	CVL	Ineffectual Activation Bits

INEFFECTUAL ACTIVATIONS

Our first observation is that many activations turn out to be zero at runtime, and even more are close enough to zero that they can be treated as if they were zero. In either case, the energy of the transfers of these ineffectual activations and of the corresponding multiplications and additions can be straightforwardly avoided by temporarily "powering off" the respective links and compute units. While this approach reduces energy, it does not improve performance. What if we could completely eliminate these actions and process in their place some of the other effectual activations?

Figure 4 reports the average total fraction of activation inputs to multiplications that prove ineffectual across all CVLs and across all inputs. On average, 44 percent of all activations are zero. An additional 7 percent of activations can be ignored without affecting accuracy (TOP-1). The position of these ineffectual activations depends on the input data values, and, hence, it would be challenging for a static approach to eliminate the corresponding computations.

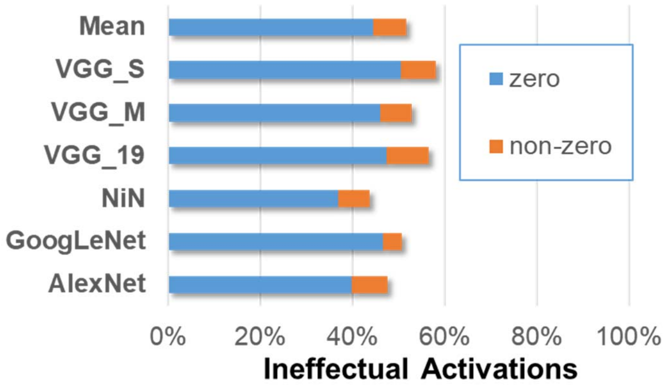


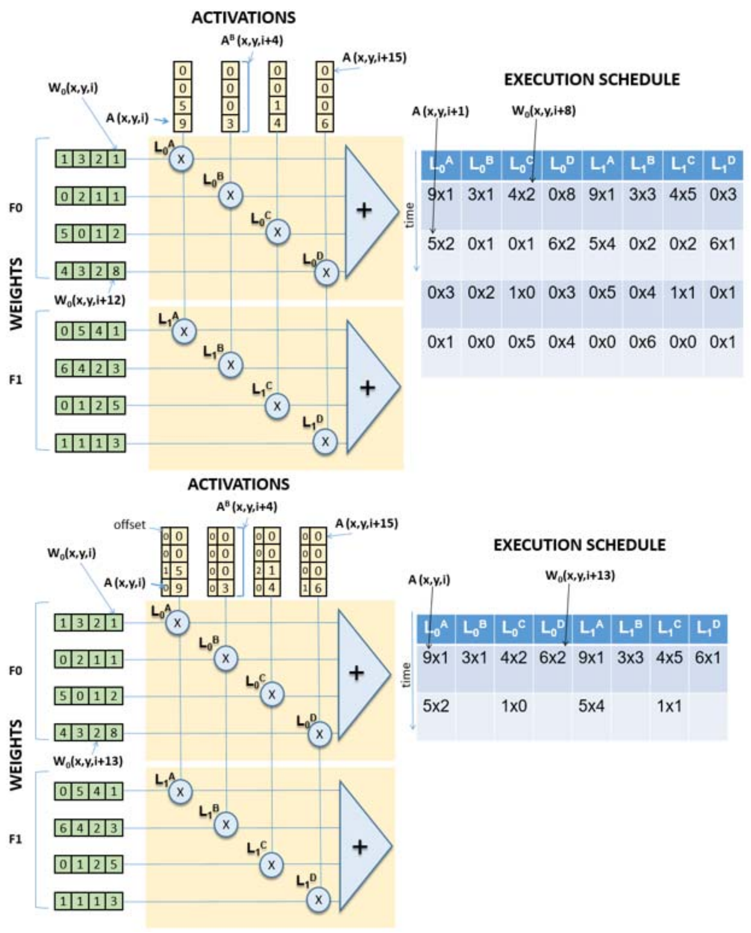
Figure 4. Average fraction of CVL multiplication input activations that prove ineffectual.

But why would a network produce so many zero activations? The answer lies in the nature and structure of CNNs. At a high level, it is convenient to think that CNNs are designed so that each CNN layer attempts to determine whether and where the input contains certain learned "features" such as lines, curves, or more elaborate constructs. The presence of a feature is encoded as a positive-valued activation output and the absence as a zero-valued activation output. It stands to reason that when features exist, most likely, they will not appear all over the input. Moreover, not all features will exist. CNNs detect the presence of features using the CVLs to produce an output encoding the likelihood that a feature exists at a particular position as a number. Negative values suggest that a feature is not present. CVLs are immediately followed by a ReLU layer that clamps negative values to zero.

The Cnvlutin (CNV) Accelerator

Modifying DaDN so that it can seamlessly avoid ineffectual activations is a challenge. When processing all activations, wide accesses to both AM and WM are trivially possible. Moreover, the positions of activation and their corresponding weights are known well in advance and are independent of their values. The straightforward approach to "skipping" the ineffectual activations is of no use; by the time activations are fetched from AM, there is not enough time to test their values and to selectively fetch additional activations and weights to replace those that are ineffectual. Not only would this require several narrow AM and WM accesses, but also several attempts might be required to find a set of 16 effectual activations that can be processed in parallel by all tiles. By that time, the base design that does not attempt to skip ineffectual activations would be further ahead. Often, "dumb but fast" hardware is hard to beat.

The top of Figure 5 shows a simplified DaDN accelerator with two IPUs, each processing four activation and weight pairs per cycle. This DaDN needs four cycles in total to process the 16 products necessary and to produce the two output activations corresponding to filters F0 and F1.



L_0^A	L_0^B	L_0^C	L_0^D	L_1^A	L_1^B	L_1^C	L_1^D
9x1	3x1	4x2	0x8	9x1	3x3	4x5	0x3
5x2	0x1	0x1	6x2	5x4	0x2	0x2	6x1
0x3	0x2	1x0	0x3	0x5	0x4	1x1	0x1
0x1	0x0	0x5	0x4	0x0	0x6	0x0	0x1

L_0^A	L_0^B	L_0^C	L_0^D	L_1^A	L_1^B	L_1^C	L_1^D
9x1	3x1	4x2	6x2	9x1	3x3	4x5	6x1
5x2		1x0		5x4		1x1	

Figure 5. Top, DaDN processes all activations; Bottom, CNV processes only non-zero activations.

The bottom of Figure 5 shows how the equivalent simplified CNV design needs only two cycles. Activations are now augmented with relative offsets. For example, a four-activation brick containing (1,0,4,0) is encoded as ((1,0),(4,2),(0,0),(0,0)) where the (4,2) pair indicates that in the original activation array, 4 was at offset 2 within the brick. CNV decouples the activation lanes, allowing them to proceed independently. By using the offset, the tiles can fetch the corresponding weight from the WM maintaining wide accesses (one activation corresponds to 256 weights in DaDN).

However, how are the offsets generated and the activation values neatly packed into the bricks so that the effectual activations appear first, thus avoiding many narrow AM accesses? Except for the first layer, the input activation array to each layer is the output activation array of the immediately preceding layer. Moreover, several tens to hundreds of cycles are often needed to generate each output activation, given existing layer dimensions. Accordingly, there is plenty of time to encode activations in the desired format while generating them. Albericio et al. describe the process in more detail,[4] but several alternatives exist.

PRECISION

Conventional general-purpose hardware and many accelerators support one or a few precisions for numerical values. For example, DaDN uses 16-bit fixed-point values. However, the precision required by CNNs varies significantly not only across networks but also across the layers of the same network.[10] Nevertheless, because most existing implementations rely on a one-size-fits-all approach, they use the worst-case numerical precision for all values and cannot benefit from the variable precision requirements of CNNs.

Stripes (STR) allows per-layer selection for activations providing a new dimension upon which to improve performance.[5] To do so, STR's execution units are designed so that execution time scales linearly with the length in bits of the numerical precision needed by each layer. Compared to DaDN that uses a 16-bit fixed-point representation, STR would ideally improve performance at each layer by $16/p^L$ where p^L is the layer's required precision length in bits.

Before we touch on the key design decisions in STR, Table 3 reports the precision profile per network, which is the set of the per-layer fixed-point representation lengths needed for activations to maintain the network's classification accuracy.[10] For each network, the per-layer precisions are shown separated with dashes. Overall, the precision needed varies from as much as 14 bits (Layer 1, GoogLeNet) to as little as 5 bits (Layer 3, AlexNet). The Ideal Speedup column reports the speed that is possible over DaDN if performance scales by $16/p^L$ where p^L is the precision used for activations in layer L. Proteus takes advantage of this precision variability to reduce off-chip memory traffic by storing and transferring only as many bits as necessary.[11]

Table 3: Per-CVL activation precision profiles needed to maintain the same TOP-1 accuracy as in the baseline (100 percent).

Network	**Per-Layer Activation Precision in Bits**	**Ideal Speedup**
AlexNet	9-8-5-5-7	2.38
NiN	8-8-8-9-7-8-8-9-9-8-8-8	1.91
GoogLeNet	10-8-10-9-8-10-9-8-9-10-7	1.76
VGG M	7-7-7-8-7	2.23
VGG S	7-8-9-7-9	2.04
VGG – 19	12-12-12-11-12-10-11-11-13-12-1313-13-13-13-13	1.35

STR's Approach

Figure 6 shows (in the top left) a simplified DaDN IP block computing the inner product of vectors (A_0,A_1) (activations) and (B_0,B_1) (weights), where all values are encoded using 2 bits of precision and are shown in binary. The IP uses bit-parallel multipliers and, in a single cycle, computes the pairwise element products of A and B, (1×1,0×3), or (1,0). The two products are then added through a bit-parallel adder to calculate the final inner product $A{\cdot}B$ = (1), which is then truncated to 2 bits. In total, the input bandwidth of this unit is 8 bits per cycle (two activations of 2 bits per cycle and two weights of 2 bits per cycle), and its output bandwidth is 2 bits per cycle.

The top right of Figure 6 shows STR's approach where A's values have been transposed and are now processed bit-serially over two cycles. However, because the upper bits of $A0$ and $A1$ are both zero, both can be represented in just 1 bit of precision. Hence, one cycle is sufficient to calculate the inner product. In general, execution time now becomes proportional to the bit-width of A's values, and, as long as there is enough precision in the partial output buffer and the adders, a range of A precisions can be naturally supported.

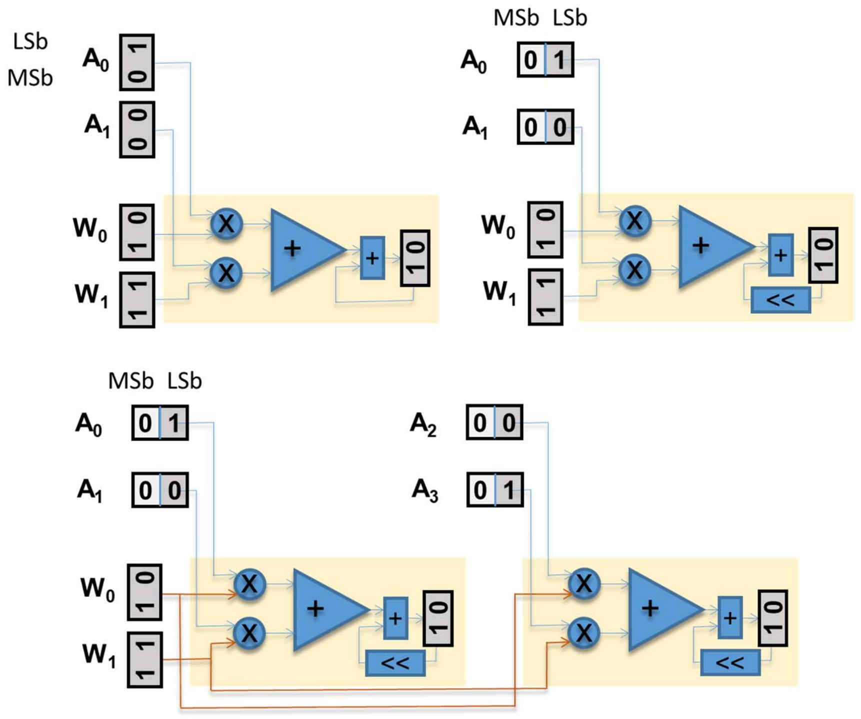


Figure 6. Exploiting precision to boost performance: Top Left, bit-parallel computation; Top Right, bit-serial computation for A_x (execution time depends on A_x's precision); Bottom, bit-serial computation for more activations matching bit-parallel computation's throughput. The same weights are used for both pairs of activations.

However, as described, the unit in the top right of Figure 6 would at best match the performance of DaDN; if both bits of precision are needed, it will be twice as slow. Fortunately, there is abundant parallelism in the CVLs that STR exploits to offer at least the same computational bandwidth as the bit parallel design. As the bottom of Figure 6 shows, STR could process another pair of activations A_2 and A_3 in parallel. In the worst case, this unit would produce two inner products every two cycles, matching the bit-parallel design's throughput. In the best case, it will need just one cycle, which is twice as fast as DaDN.

In our example, two additional activations were sufficient to always match the performance of DaDN, because we assumed that activations used up to 2 bits of precision. Because DaDN uses 16 bits of precision, STR needs to process 16 times as many activations in parallel. The top of Figure 7 shows a portion of an STR tile that replaces a single DaDN IP and that uses exactly the same number of external connections. Instead of using the 256 activation wires to communicate 16 16-bit activations, STR communicates 1 bit from each of 256 activations. Whereas DaDN had one IP multiplying 16 pairs of 16-bit activations and weights, STR has 16 serial inner product units (SIPs), each multiplying 16 pairs of a 16-bit weight and a single-bit activation. Whereas DaDN needed multipliers, STR can use AND gates instead. The same set of 16 16-bit weights is used by all 16 SIPs in the row. However, each SIP is given a different set of 16 activations each corresponding to a different window of the input activation array, as the bottom of Figure 7 shows.

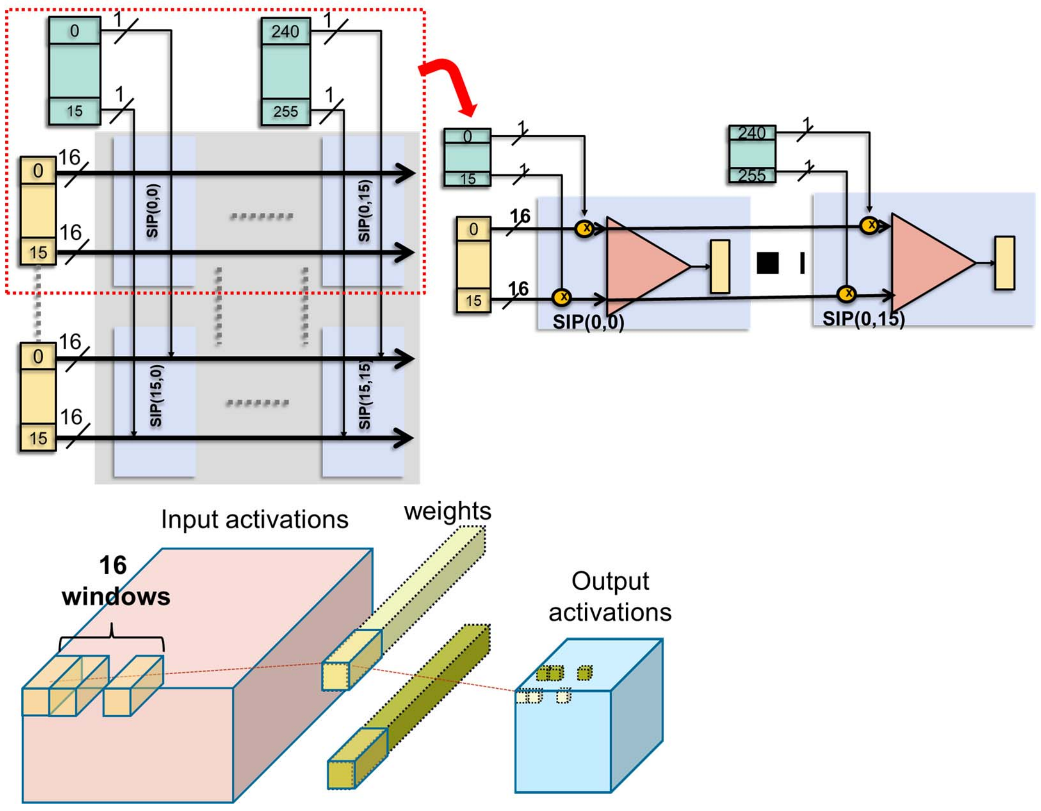


Figure 7. Top, STR's tile detail; Bottom, where STR finds the parallelism it needs.

Dynamic Precision

STR relies on profile-derived precisions per layer. However, in practice, these precisions will probably exceed those necessary for two reasons: 1) the precisions are meant to be sufficient for any possible input, and 2) the precisions are for the full layer. Given a specific input image, it stands to reason that the precisions needed might be narrower. Moreover, at any given point of time, STR processes only 256 activations in parallel. The precision needed to represent these 256 specific values will most likely be narrower than that needed to represent all activations for the layer. Dynamic Stripes extends STR to trim precisions "on the fly" and at every cycle by adjusting to the set of activations that are being processed concurrently.[7]

Fully Connected Layers

It is possible, at an additional area cost, to achieve performance improvements for FCLs, as well. STR cannot improve performance for FCLs because they only contain one window and, thus, at least 16 cycles are needed to load a different set of weights per SIP. As a result, it always takes 16 cycles to process a set of activations and weights regardless of the precision used for the activations. The TARTAN extension to STR takes advantage of the variable precision requirements for weights.[6] In TARTAN, performance for FCLs improves by $16/max(P_w^L, P_a^L)$ where P_w^L and P_a^L are the precisions of the activations and weights respectively for layer L. The key insight behind TARTAN is that the 16 wires that are used during CVL processing to broadcast the same weight to all SIPs along the same row can be used to bit-serially load a different weight to each SIP over P_w cycles for FCLs. The loading of the next set of weights can be overlapped with the processing of the current set.

Exploiting Weight and Activation Precision

So far, our designs are capable of exploiting the precision of activations only for CVLs, or of either activations or weights for FCLs (TARTAN). Loom takes advantage of the precisions of both.[8] In Loom, execution time scales with $256/(P_w^L \, x \, P_a^L)$ and $16/max(P_w^L, P_a^L)$ for CVLs and FCLs, respectively, where P_w^L and P_a^L are the weight and activation precisions needed for layer *L*. However, Loom requires 256× the parallelism to guarantee that it will always be as fast as an equivalent DaDN configuration. For this reason, it is appropriate for smaller scale designs such as those appropriate for mobile or embedded devices. Still, a configuration that is equivalent to just half of a tile of DaDN can saturate a high-bandwidth memory interface offering 512 GB/s. Table 2 reports the characteristics for a Loom configuration that operates at a single-bit granularity. Better energy efficiency is possible when operating at a 2- or 4-bit granularity.

EFFECTUAL BIT CONTENT

The final value property that we exploit is "bit density," which is the fraction of activation bits that are one. Binary multiplication can be broken down into a summation of single-bit multiplications (ANDs). For example, $a \times w$ is calculated as $\sum_{i=0}^{p} a_i \cdot (w \ll i)$, where a_i is the *i*-th bit of *a*.

The multiplier computes *p* terms, each a product of the shifted operand *w* and a bit of *a*, and adds them to produce the final result. With this in mind, any time an activation bit that is zero is multiplied with a weight, it adds nothing to the output activation.

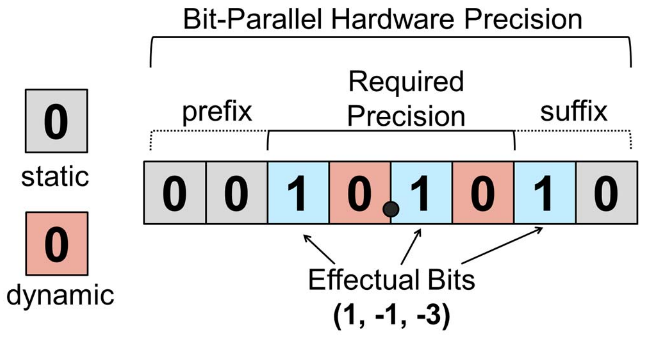


Figure 8: Sources of ineffectual computation with conventional positional representation and fixed-length hardware precision.

We can categorize these zero bits as either statically or dynamically ineffectual. Statically ineffectual bits are those that can be determined to be ineffectual *a priori*. They result from using a data format with more precision than is necessary. In this case, the number 1 might also be statically ineffectual. Figure 8 shows an example illustrating these ineffectual bits using an 8-bit unsigned fixed-point number with 4 fractional and 4 integer bits. Assume that we know ahead of time that our data only needs 5 bits, then we have 3 statically ineffectual bits as a prefix and suffix to the required precision. While $10.101_{(2)}$ requires just five bits, two dynamically generated zero bits appear at positions 0 and -2. In total, 5 ineffectual bits will be processed generating five ineffectual terms. STR and its variants do exploit these statically ineffectual bits. Dynamic Stripes can exploit some of the dynamically ineffectual zero bits, as well. No STR variant can exploit the dynamically ineffectual zero bits that appear in between bits that are one.

The non-zero bits can instead be encoded with their corresponding exponents (1,-1,-3). While such a representation might require more bits, which is undesirable for storage, dynamically generating the exponents and only computing the non-zero terms might benefit performance and energy efficiency. For the networks studied, only 7.6 percent and 28 percent of the activation bits are effectual for the 16-bit fixed-point and 8-bit quantized[12] representations, respectively. These results point to a potential performance improvement of nearly 13× over DaDN. Even if we assume that the zero-valued activations can be removed, the corresponding fractions of effectual bits remain low at 19 percent and 41 percent, respectively.

Pragmatic

Pragmatic (PRA) is an accelerator whose goal is to process only the effectual (non-zero) bits of activations. PRA processes the activations bit-serially while compensating for the loss in computation bandwidth by exploiting the abundant parallelism of CVLs, similar to STR. However, PRA skips the activation bits that are zero, a task that required the confluence of several design choices and techniques to realize. Albericio et al. describe the design in more detail.[9] Configurations of PRA that use less activation lanes per tile are 7.8× faster than an equivalent DaDN configuration.

CONCLUSION

Hardware acceleration of DL is an active area of research; unfortunately, we cannot fully review this area due to limited space. However, most recent advances exploit various forms of informational inefficiency in CNNs and other DNNs.

At the value level, many activations and weights are ineffectual. Han et al. retrain NNs to eliminate computations with zero-valued activations and ineffectual weights.[13] Other work shows that much smaller networks than the originally proposed ones can maintain accuracy,[14] suggesting that networks are often over-provisioned. SCNN avoids computations with both ineffectual weights and activations.[15] Recent evidence suggests that, for a given storage budget, weight pruning large networks offers better accuracy than naively scaling down networks to size.[16] However, the occurrence of ineffectual activations appears to be an intrinsic property of CNNs, as their neurons are designed to detect the presence of relevant features in their input. Combining ineffectual weight skipping with STR or PRA offers benefits for all activations and would be interesting to explore further.

Informational inefficiency also manifests in excess of precision, which STR exploits. Various forms of quantization also exploit this phenomenon,[12] whereas other designs hardwire different per-layer precisions.[17] At the extreme end of the spectrum are “binarized” networks[18] that use binary weights and/or activations. Where such networks are possible, they are preferable due to their reduced area, power, and complexity.

In all, the aforementioned body of work suggests that existing networks exhibit informational inefficiency at various levels and for various reasons. Whether these inefficiencies are best exploited statically, dynamically, or both is an open question. Furthermore, it remains to be seen which forms of inefficiency will persist as networks evolve.

Our designs work with out-of-the-box NNs, thus offering immediate benefits. More importantly, they open up new opportunities and incentives for NN designers, providing a safe path towards innovation while offering rewards for even small advances. For example, STR, if deployed, can accelerate innovation in low-precision NN design with an eye towards binary and ternary networks. This is because it enables experimentation with the whole spectrum of precision choices while also delivering excellent performance for full-precision networks. This will incentivize the ML community to further invest in this direction, delivering immediate, proportional rewards. Eventually, if extremely low-precision networks take over, more efficient hardware platforms can be deployed.

REFERENCES

1. H. Esmaeilzadeh et al., “Dark silicon and the end of multicore scaling,” *38th International Symposium on Computer Architecture* (ISCA '11), 2011, pp. 365–376; https://dl.acm.org/citation.cfm?id=2000108.

2. M. Bojarski et al., “End to End Learning for Self-Driving Cars,” *arxiv.org*, 2016; https://arxiv.org/abs/1604.07316.
3. Y. Chen et al., “DaDianNao: A Machine-Learning Supercomputer,” *47th IEEE/ACM International Symposium on Microarchitecture* (MICRO), 2014; http://ieeexplore.ieee.org/document/7011421/.
4. J. Albericio et al., “Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing,” *43rd ACM/IEEE International Symposium on Computer Architecture* (ISCA), 2016; http://ieeexplore.ieee.org/document/7551378/.
5. P. Judd et al., “Stripes: Bit-serial deep neural network computing,” *49th IEEE/ACM International Symposium on Microarchitecture* (MICRO), 2016; http://ieeexplore.ieee.org/document/7783722/.
6. A. Delmas et al., “Tartan: Accelerating Fully-Connected and Convolutional Layers in Deep Learning Networks by Exploiting Numerical Precision Variability,” *arxiv.org*, 2017; https://arxiv.org/abs/1707.09068.
7. A. Delmas et al., “Dynamic Stripes: Exploiting the Dynamic Precision Requirements of Activation Values in Neural Networks,” *arxiv.org*, 2017; https://arxiv.org/abs/1706.00504.
8. S. Sharify et al., “Loom: Exploiting Weight and Activation Precisions to Accelerate Convolutional Neural Networks,” *arxiv.org*, 2017; https://arxiv.org/abs/1706.07853.
9. J. Albericio et al., “Bit-pragmatic deep neural network computing,” *50th IEEE/ACM International Symposium on Microarchitecture* (MICRO-50 '17), 2017, pp. 382–394; https://dl.acm.org/citation.cfm?id=3123982.
10. P. Judd et al., “Reduced-Precision Strategies for Bounded Memory in Deep Neural Nets,” *arxiv.org*, 2015; https://arxiv.org/abs/1511.05236.
11. P. Judd et al., “Proteus: Exploiting Numerical Precision Variability in Deep Neural Networks,” *Workshop On Approximate Computing* (WAPCO), 2016; https://wapco.e-ce.uth.gr/2016/papers/SESSION2/wapco2016_2_2.pdf.
12. P. Warden, “How to Quantize Neural Networks with TensorFlow,” *Pete Warden's Blog*, 2016; https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/.
13. S. Han et al., “EIE: Efficient Inference Engine on Compressed Deep Neural Network,” *arxiv.org*, 2016; https://arxiv.org/abs/1602.01528.
14. F.N. Iandola et al., “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,” *arxiv.org*, 2016; https://arxiv.org/abs/1602.07360.
15. A. Parashar et al., “SCNN: An accelerator for compressed-sparse convolutional neural networks,” *44th ACM/IEEE International Symposium on Computer Architecture* (ISCA), 2017; https://www.computer.org/csdl/proceedings/isca/2017/4892/00/08192478-abs.html.
16. M. Zhu and S. Gupta, “To prune, or not to prune: exploring the efficacy of pruning for model compression,” *arxiv.org*, 2017; https://arxiv.org/abs/1710.01878.
17. J. Kim, K. Hwang, and W. Sung, “X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks,” *IEEE International Conference on Acoustics, Speech and Signal Processing* (ICASSP), 2014; http://ieeexplore.ieee.org/document/6855060/.
18. M. Courbariaux, Y. Bengio, and J.P. David, “BinaryConnect: Training Deep Neural Networks with binary weights during propagations,” *arxiv.org*, 2015; https://arxiv.org/abs/1511.00363.

ABOUT THE AUTHORS

Andreas Moshovos is a professor in the Electrical and Computer Engineering Department at the University of Toronto. His research interests are in architecting highly efficient and high-performance computing hardware. Moshovos has a PhD in computer science from the University of Wisconsin-Madison. He is a senior member of the IEEE and a Fellow of the ACM. Contact him at moshovos@eecg.toronto.edu.

Jorge Albericio is a Senior Deep Learning Architect at NVIDIA. He has a PhD in systems engineering and computing from the University of Zaragoza. He was a postdoctoral fellow at the University of Toronto from 2013 to 2016, where he worked on branch prediction, approximate computing, and hardware accelerators for machine learning. He is a member of the IEEE. Contact him at jorge.albericio@gmail.com.

Patrick Judd is a fourth year PhD candidate at the University of Toronto. His research interests include computer architecture, machine learning, and approximate computing. His research focuses on the design of hardware accelerators for deep neural networks that exploit approximation for improved performance and energy efficiency. Judd is a student member of the IEEE. Contact him at patrick.judd@mail.utoronto.ca.

Alberto Delmás Lascorz is a third year PhD candidate at the University of Toronto, where he focuses on hardware design for machine-learning accelerators. His research interests include computer architecture, deep learning, and embedded and reconfigurable systems. Delmás Lascorz previously studied computer engineering at the University of Zaragoza. He is a student member of the IEEE. Contact him at a.delmaslascorz@mail.utoronto.ca.

Sayeh Sharify is a third year PhD candidate at the University of Toronto. Her research interests include computer architecture, machine learning, embedded systems, and reconfigurable computing. She designs hardware accelerators for machine-learning algorithms. Sharify previously studied computer engineering at Sharif University of Technology. She is a student member of the IEEE and the ACM. Contact her at sayeh@ece.utoronto.ca.

Tayler Hetherington is a final-year PhD candidate in computer engineering at the University of British Columbia and is currently working at Oracle Labs. His research interests include computer architecture, specifically general-purpose GPUs, machine-learning accelerators, and system software. He is a student member of the IEEE. Contact him at taylerh@ece.ubc.ca.

Tor Aamodt is a professor in the Department of Electrical and Computer Engineering at the University of British Columbia. His research interests include architecture of general-purpose GPUs and machine-learning accelerators. Aamodt has a PhD in electrical and computer engineering from the University of Toronto. He is a member of the IEEE and ACM. Contact him at aamodt@ece.ubc.ca.

Natalie Enright Jerger is the Percy Edward Hart Professor of Electrical and Computer Engineering at the University of Toronto. Her research interests include computer architecture, approximate computing, interconnection networks, and hardware acceleration of machine learning. Enright Jerger has a PhD in electrical engineering from the University of Wisconsin-Madison. She is a senior member of the IEEE and ACM. Contact her at enright@ece.utoronto.ca.