2018 51st Annual IEEE/ACM International Symposium on Microarchitecture

Diffy: a Déjà vu-Free Differential Deep Neural Network Accelerator

Mostafa Mahmoud
Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada
mostafa.mahmoud@mail.utoronto.ca

Kevin Siu
Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada
kcm.siu@mail.utoronto.ca

Andreas Moshovos
Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada
moshovos@eecg.toronto.edu

***Abstract*—We show that Deep Convolutional Neural Network (CNN) implementations of computational imaging tasks exhibit spatially correlated values. We exploit this correlation to reduce the amount of computation, communication, and storage needed to execute such CNNs by introducing *Diffy*, a hardware accelerator that performs *Differential Convolution*. Diffy stores, communicates, and processes the bulk of the activation values as *deltas*. Experiments show that, over five state-of-the-art CNN models and for HD resolution inputs, *Diffy* boosts the average performance by 7.1× over a baseline value-agnostic accelerator [1] and by 1.41× over a state-of-the-art accelerator that processes only the effectual content of the raw activation values [2]. Further, *Diffy* is respectively 1.83× and 1.36× more energy efficient when considering only the on-chip energy. However, *Diffy* requires 55% less on-chip storage and 2.5× less off-chip bandwidth compared to storing the raw values using profiled per-layer precisions [3]. Compared to using dynamic per group precisions [4], *Diffy* requires 32% less storage and 1.43× less off-chip memory bandwidth. More importantly, *Diffy* provides the performance necessary to achieve real-time processing of HD resolution images with practical configurations. Finally, *Diffy* is robust and can serve as a general CNN accelerator as it improves performance even for image classification models.**

***Index Terms*—neural networks, deep learning, differential convolution, computational imaging, accelerator**

I. Introduction

The successes of Deep Neural Networks (*DNN*) in high-level classification applications such as in image recognition [5], [6], [7], object segmentation [8], [9], [10] and speech recognition [11] are well known. However, DNNs recently achieved state-of-the-art output quality also in a wide range of Computational Imaging (*CI*) and in low-level computer vision tasks that traditionally were dominated by analytical solutions. These tasks include image denoising [12], [13], [14], demosaicking [15], sharpening [16], deblurring [17], [18], [19], and super-resolution [20], [21], [22], [23], [24]. These are essential tasks for virtually all systems that incorporate imaging sensors such as mobile devices, digital cameras, medical devices, or automation systems. Such embedded devices are typically cost-, power-, energy-, and form-factor-constrained. Accordingly, one of the goals of this work is to investigate whether DNN-based computational imaging can be deployed on such devices. While the emphasis of this work is on such devices, interest is not limited to only to them. For example, DNN-based computational imaging benefits also scientific applications such as telescope imaging with input images of up to 1.5 billion pixels [25], [26], automation applications in manufacturing pipelines, or even in server farms. Thus, there are also applications where higher cost and energy can be acceptable for better quality.

Due to the high computational and data supply demands of DNNs, several DNN accelerators have been proposed to boost performance and energy efficiency over commodity Graphics Processing Units (GPUs) and processors, e.g., [1], [2], [27], [28], [29], [30], [31], [32]. These accelerators have taken advantage of the computation structure, the data reuse, the static and dynamic ineffectual value content, and the precision requirements of DNNs.

As these past successes demonstrate, identifying additional runtime behaviors in DNNs is invaluable as it can inform further innovation in accelerator design. Accordingly, the first contribution of this work is that it shows that DNNs for Computational Imaging exhibit high spatial correlation in their runtime-calculated value stream. It further shows that this property can have tangible practical applications by presenting *Diffy* a practical hardware accelerator that exploits this spatial correlation to transparently reduce 1) the number of bits needed to store the network's values on- and off-chip, and 2) the computations that need to be performed. Combined these reductions increase performance and energy efficiency benefits over state-of-the-art designs.

To date CNN acceleration efforts have focused primarily on image classification CNNs. While image classification CNNs extract features and identify correlations among them, computational imaging DNNs, or *CI-DNNs*, perform *per-pixel prediction*. That is, for each input pixel the model predicts a corresponding output pixel. As a result, their structure and behavior are different. First, while DNN models generally include a variety of layers, the per-pixel prediction models are fully convolutional which favors specialized designs for convolutional neural networks (CNNs). Second, these CI-DNN models exhibit significantly higher spatial correlation in their runtime calculated values. That is, the inputs (activations) used during the calculation of neighboring outputs tend to be close in value. This is a known property of images, which these models preserve throughout their layers. Third, these models naturally scale with the input resolution whereas classification

978-1-5386-6240-3/18/$31.00 ©2018 IEEE
DOI 10.1109/MICRO.2018.00020

models are resolution-specific.

To take advantage of the spatial correlation in their values, we introduce *Differential Convolution* which operates on the differences, or the *deltas*, of the activations rather than on their absolute values. This approach greatly reduces the amount of work that is required to execute a CI-DNN. We demonstrate that differential convolution can be practically implemented by proposing *Diffy*, a CI-DNN accelerator that translates the reduced precision and the reduced effectual bit-content of these deltas into improved performance, reduced on- and off-chip storage and communication, and ultimately, improved energy efficiency. While *Diffy* targets CI-DNNs, it benefits also other models — we experiment with image classification and image segmentation – albeit to a lesser extend. This shows that *Diffy* is robust and not CI-DNN specific.

In summary the contributions and findings of this work are:

- We study an emerging class of CNN models that performs per-pixel prediction showing that they exhibit strong spatial correlation in their value stream.
- We present *Differential Convolution* (*DC*) which exploits the preceding property of CI-DNNs to reduce the work necessary to compute convolutions.
- We propose *Diffy*, a practical DC-based architecture that boosts performance and energy efficiency for CI-DNNs and other convolutional neural networks (CNNs).
- We propose to store values as deltas both off- and on-chip reducing the amount of storage and communication needed, or equivalently boosting the effective capacity of on- and off-chip storage and communication links.
- For a set of state-of-the-art CI-DNNs we show that a *Diffy* configuration that can compute the equivalent of 1K $16\times16b$ multiply-accumulate operations per cycle boosts performance by $7.1\times$ and $1.41\times$ over a baseline value-agnostic accelerator (*VAA*) and a state-of-the-art value-aware accelerator (*PRA*) respectively. This *Diffy* configuration processes HD frames (1920×1080) at 3.9 up to 28.5 FPS (frames/sec) depending on the target application. By comparison, *VAA* achieves 0.7 to 3.9 FPS, and *PRA* 2.6 to 18.9 FPS. Compared to using dynamic per group precisions for the raw values, *Diffy* reduces on-chip storage by 32% and off-chip traffic by $1.43\times$.
- We compare *Diffy* with SCNN when executing CI-DNNs and under various assumptions about weight sparsity and show that *Diffy* consistently outperforms SCNN [32] (e.g., by $4.5\times$ with 50% weight sparsity).
- Layout results show that *Diffy* is $1.83\times$ and $1.36\times$ more energy efficient compared to *VAA* and *PRA*.
- Further, we show that *Diffy* scales much more efficiently and enables real-time processing of HD frames with considerably less resources.
- We study performance with practical on- and off-chip memory hierarchies showing that our delta value compression schemes can greatly reduce on- and off-chip storage footprint and traffic.
- We show that *Diffy* benefits image classification CNNs as well improving performance on average by $6.1\times$ and by $1.16\times$ compared to *VAA* and *PRA* respectively. Most of the benefits appear at the earlier layers of these networks where *Diffy* proves to be over $2.1\times$ faster than *PRA*.

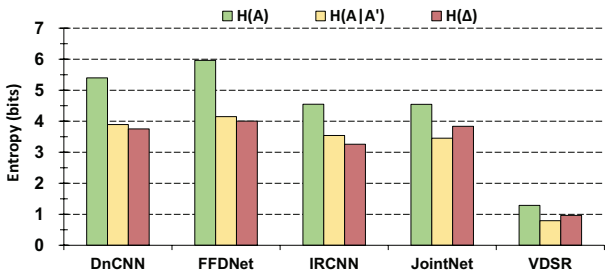


Fig. 1: Information content: Entropy of raw activations $H(A)$, conditional entropy $H(A|A')$ (see text) and entropy of deltas $H(\Delta)$

II. Motivation

Ideally, to avoid redundant computation and communication, we should process only the new, or *essential*, information carried by the data in hand. Fig. 1 presents the first set of evidence that compared to the raw values, the deltas of adjacent values more compactly convey the essential information content of the CI-DNNs of Table I. The figure presents the per network *entropy* $H(A)$ of the activations (runtime values, see Section II-A), the conditional entropy $H(A|A')$ of the activations A given their adjacent along the X-axis activation A', and finally the entropy $H(\Delta)$ of the activation deltas along the X-axis. These measurements were collected over all input datasets detailed in Table II. While $H(A)$ represents the average amount of information contained within the activation values, $H(A|A')$ measures the amount of *new information* carried by A if we already know A'. $H(\Delta)$ shows how much of the redundant information can be removed from A if we replaced the activations with their deltas.

The measurements show that there is considerable redundancy in information from an activation to the next with the potential to compress the encoded information by a factor of at least $1.29\times$ for IRCNN and by up to $1.62\times$ for VDSR. For some networks, $H(\Delta)$ further compresses the information compared to $H(A|A')$ while for others it does not. However, on average over all the models the potential to compress the underlying information with $H(A|A')$ and $H(\Delta)$ is nearly identical at $1.41\times$ and $1.4\times$ respectively.

Next we review the operation of the convolutional layers (Section II-A) so that we can explain how using deltas can in principle reduce computation, communication and data footprint (Section II-B). We finally motivate *Diffy* by reporting the spatial locality in CI-DNNs (Section II-C), and the potential of delta encoding to reduce computation (Section II-D), and communication and data storage (Section II-E).

A. Background: Convolutional Layers

A convolutional layer takes an input feature map *imap*, which is a 3D array of activations of size $C\times H\times W$ (channels,

height, and width), applies K 3D filter maps, or *fmaps*, of size $C \times H_F \times W_F$ in a sliding window fashion with stride S and produces an output map, or *omap*, which is a 3D array of activations of size $K \times H_O \times W_O$. Each output activation is the inner product of a filter and a *window*, a sub-array of the imap of the same size as the filter. Assuming a, o and w^n are respectively the imap, the omap and the n-*th* filter, the output activation $o(n,y,x)$ is computed as the following inner product $\langle w^n, \cdot \rangle$:

$$o(n,y,x) = \sum_{k=0}^{C-1} \sum_{j=0}^{H_F-1} \sum_{i=0}^{W_F-1} w^n(k,j,i) \times a(k,\ j+y \times S,\ i+x \times S) \quad (1)$$

Input windows form a grid with a stride S. As a result the omap dimensions are respectively K, $H_O = (H - H_F)/S + 1$, $W_O = (W - W_F)/S + 1$. In the discussion that follows we assume without loss of generality that $S = 1$ which is the common case for CI-DNNs. However, the concepts apply regardless.

B. Revealing Redundant Information and Work

Given the abundant reuse in the convolutional layers, it is beneficial to transform the input activations from their raw value space R to some space D where: 1) the operations performed on R can still be applied seamlessly on D, and 2) the representation of values in D is compressed leading to less communication and computation. One such transformation is delta encoding where adjacent activations are represented by their differences. First, deltas are subject to the distributive and associative properties of multiplication and addition, the main operations of convolution. Second, if the raw values are correlated enough delta encoding is a compressed and more space- and communication-efficient representation of the values.

Multiplications account for the bulk of the computational work in CI-DNNs. For this reason, strong spatial correlation in the imaps presents an opportunity to reduce the amount work needed. To understand why this is so, let us consider the multiplication $a \times w$ of an activation a with a weight w. If a is represented using p bits, the multiplication amounts to adding p terms where the i-*th term* is the result of "multiplying" the i-*th* bit of the multiplier a with the shifted by i bit positions multiplicand w:

$$a \times w = \sum_{i=0}^{i=p} a_i \cdot (w \ll i) \quad (2)$$

It is only those bits of a that are 1 that yield effectual work. Using modified Booth encoding, we can further reduce the *effectual terms* as long as we allow both addition and subtraction.

Since convolution layers process the imap using overlapping windows, a weight w that was multiplied with an activation a for some window I, during the processing of the adjacent (along the H or W dimension) window I' will be multiplied with the adjacent activation a'. Thus, rather than calculating $a' \times w$ directly we could instead calculate it relatively to $a \times w$:

$$a' \times w = (a \times w) + (a' - a) \times w = (a \times w) + (\Delta_a \times w) \quad (3)$$

When adjacent activations are close in value, calculating $a' \times w$ from scratch will be just a *Déjà vu* of $a \times w$ repeating almost the same long multiplication. However, their difference Δ_a will be relatively small with typically fewer effectual terms to process compared to a or a'. Given that we already calculated $a \times w$ this approach will reduce the amount of work needed overall. Representing the imap using deltas can also reduce its footprint and the amount of information to communicate to and from the compute units. This will be possible as long as the deltas can be represented using a shorter datatype than the original imap values.

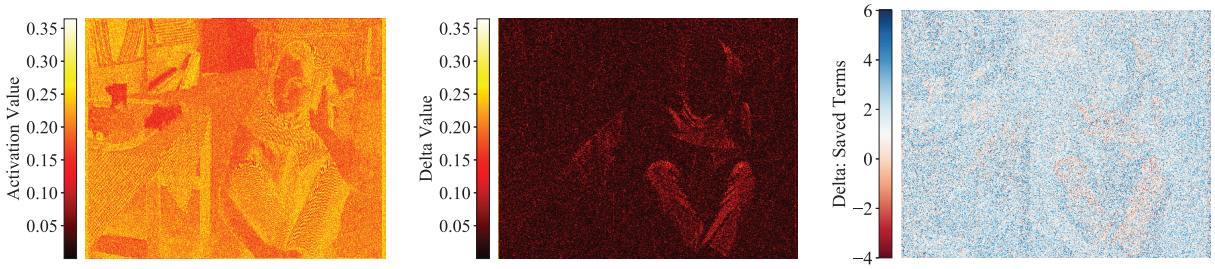
C. Spatial Correlation in CI-DNNs imaps

Fig. 2a shows a heatmap of the raw imap values from conv_3, the third convolutional layer of DnCNN, while denoising the *Barbara* image. Even though this is an intermediate layer, the image is still discernible. More relevant to our discussion, Fig. 2b shows a heatmap of the differences, or the *deltas* between adjacent along the X-axis activations. The heatmap reveals a strong correlation. It is only around the edges in the original image that deltas peak. Fig. 2c shows the possible reduction in effectual terms if we were to calculate the omap *differentially*. In this scheme we calculate only the first window along each row using the raw activation values. All subsequent windows are calculated differentially as will be detailed in Section III-C. For the specific imap, the average number of terms is 3.65 and 1.9 per activation and delta respectively. Thus processing windows differentially, has the potential to reduce the amount of work by 1.9×. Savings are higher, reaching up to 6 terms in homogeneous color areas. However, deltas do not always yield less terms than the raw values. In areas with rapid changes in colors like edges deltas may have up to 4 more terms compared to the raw activations. Fortunately, in typical images the former is by far the dominant case.

Fig. 3 shows the cumulative distribution of the number of effectual terms per activation and delta. The distribution is measured over all the CI-DNN models (Table I) and over all images (Table II). The figure shows that there is significant potential for reduction in the amount of computations needed if deltas are processed instead of the raw imap as deltas contain considerably fewer effectual terms per value. The figure also shows that the sparsity of the raw imap values — the fraction of the values that are zero — is 43% and it is higher at 48% for the deltas. Thus, processing the deltas improves the potential performance benefit of any technique exploiting activation sparsity.

D. Computation Reduction Potential

Fig. 4 compares the work that has to be performed by three computation approaches. (1) *ALL*: is the baseline *value-agnostic* approach which processes all product terms, (2) Raw_E: a value-aware approach that processes only the effectual imap terms, and (3) Δ_E: a value-aware approach that processes only the effectual terms of the imap deltas. The figure reports the reduction in work as a speedup normalized over *ALL*.



(a) Activation values (b) Deltas along the X-Axis (c) Reduction in effectual terms per imap activation

Fig. 2: The imap values of CI-DNNs are spatially correlated. Thus, processing deltas instead of raw values reduces work. All results are with the *Barbara* image as input.

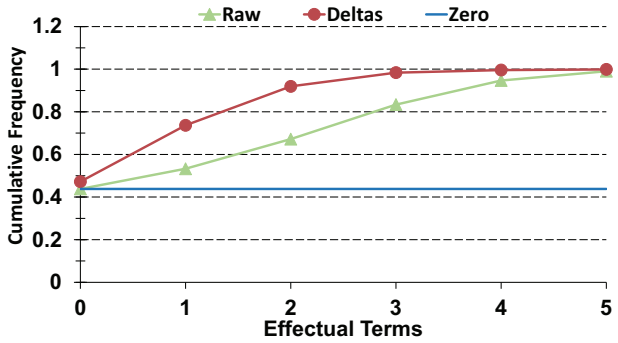


Fig. 3: Cumulative distribution of the number of effectual terms per activation/delta over all considered CI-DNNs and datasets. Average sparsity is shown for raw activations.

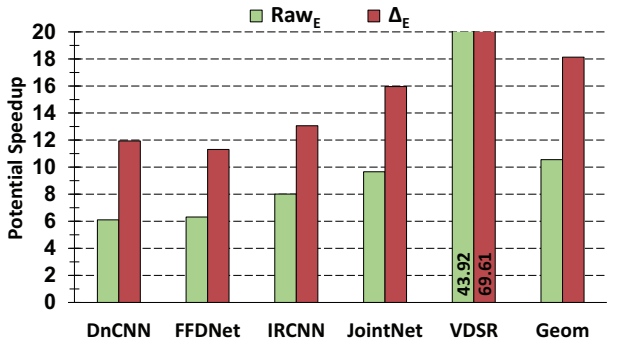


Fig. 4: Potential speedups when processing only the effectual terms of the imaps (Raw_E) or of their deltas (Δ_E). Speedups are reported over processing all imap terms.

On average, Δ_E needs to process 18.13× less terms than *ALL*. All CI-DNNs benefit with Δ_E with the work savings over *ALL* ranging from 11.3× for FFDNet to 69.6× for VDSR. VDSR exhibits much higher imap sparsity (77%) than the other models (around 23% is typical) which explains its much higher potential. Compared to Raw_E, Δ_E can ideally reduce work by 1.72× on average. The least potential for work reduction compared to Raw_E is 1.59× for VDSR and the maximum is 1.95× for DnCNN. This suggests that *Diffy*'s approach has the potential to boost performance and energy efficiency even over architectures that target the effectual terms of the imaps.

E. Off-Chip Footprint and Communication Reduction Potential

As Section IV shows, the imaps of CI-DNNs occupy a lot more space than their fmaps. The latter tend to be small in the order of a few tens of KB whereas the imaps scale proportionally with the input image resolution and dominate off-chip bandwidth needs. Fig. 5 compares the normalized amount of off-chip storage needed for the imaps of all layers for six approaches: 1) *NoCompression*: where all imap values are stored using 16b, 2) *RLEz*: where the imap values are Run-Length Encoded such that each non-zero value is stored as a 16b value and a 4b distance to the next non-zero value, 3) *RLE*: where the imap values are Run-Length Encoded such that each value is stored as a 16b value and a 4b distance to the next different value, 4) *Profiled*: where the imap values are stored using a profile-derived precision per layer [3], 5) Raw_{D16}: where the imap values are stored using dynamically detected precisions per group of 16 activations [4], [33], and 6) $Delta_{D16}$: where we store the imap values using dynamically detected precisions for their deltas.

RLEz captures activation sparsity and offers little benefit except for VDSR. *RLE* performs slightly better since it further exploits repetitive values. While *Profiled* can reduce the off-chip footprint to $47\% - 61\%$ of *NoCompression*, Raw_{D16} can further compress it down to $9.7\% - 38.6\%$. Our $Delta_{D16}$ can reduce off-chip footprint to only $8\% - 30\%$.

These results do not take into account the metadata needed for each scheme. Moreover, the number of bits that need to be communicated depends on the dataflow and the tiling approach used. Thus we defer the traffic measurements until Section IV-C.

III. Diffy

We first describe our baseline value-agnostic accelerator that resembles *DaDianNao* [1]. This well understood design is an appropriate baseline not only because it is a well optimized data-parallel yet value-agnostic design but also because it is widely referenced and thus can enable rough comparisons with

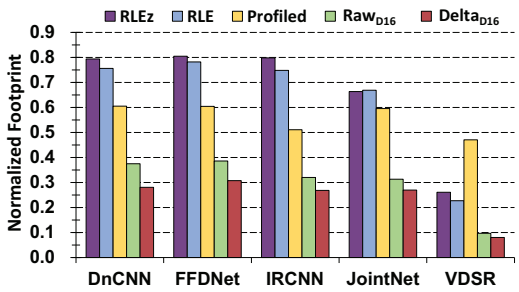


Fig. 5: Off-chip footprint with three compression approaches normalized to a fixed precision storage scheme.

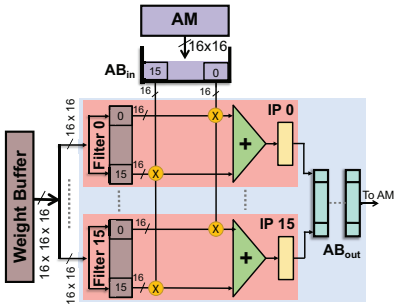


Fig. 6: A tile of our Baseline Value-Agnostic Accelerator (VAA).

the plethora of accelerator designs that have emerged since and continue to appear.

Diffy builds on top and modifies the *Bit-Pragmatic* accelerator (*PRA*) whose execution time is proportional to the number of effectual terms of the imap [2]. Since *Diffy* targets processing deltas which we have shown to have fewer effectual terms than the raw activation values, *PRA*'s processing approach can be adapted to translate this phenomenon to performance improvement. However, *PRA* has been designed to process raw activation values and needs to be modified to enable delta processing. Before we describe *Diffy* we first review *PRA*'s tile design. At the end, we implement the additional functionality *Diffy* needs with only a modest investment in extra hardware which is a major advantage for any hardware proposal. We expect that the proposed techniques can be incorporated to other designs and this work serves as the necessary motivation for such followup investigations. That said, demonstrating the specific implementation is *essential* and *sufficient*.

A. Baseline Value-Agnostic Accelerator

Figure 6 shows a tile of a data-parallel value-agnostic accelerator *VAA* that is modeled after DaDianNao [1]. A *VAA* tile comprises 16 inner product units (IPs) operating in parallel over the same set of 16 activations each producing a partial output activation per cycle. Each cycle, each IP reads 16 weights, one per input activation, calculates 16 products, reduces them via an adder tree, and accumulates the result into an output register. A per tile *Weight Memory* (WM) and an *Activation Memory* (AM) respectively store fmaps and imaps/omaps. The WM and the AM can supply 16×16

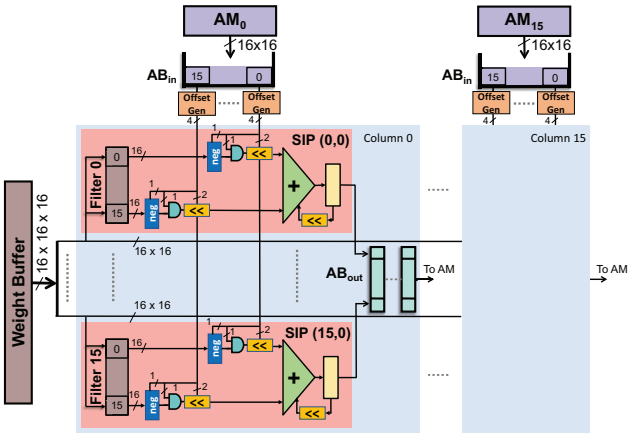


Fig. 7: A tile of *PRA*. The AM is partitioned across columns.

weights and 16 activations per cycle respectively. Each AM can broadcast its values to all tiles. Only one AM slice operates per cycle and all tiles see the same set of 16 activations. The AM slices are single-ported and banked. For each layer, half of the banks are used for the imaps and the other half for the omaps. Depending on their size, WM and AM can be banked and weight and activation buffers can be used to hide their latency. An output activation buffer collects the results prior to writing them back to AM. Both activations and weights are read from and written to an off-chip memory. The number of filters, tiles, weights per filter, precision, etc., are all design parameters that can be adjusted as necessary. For clarity, in our discussion we assume that all data per layer fits in WM and AM. However, the evaluation considers the effects of limited on-chip storage and of the external memory bandwidth.

B. Value-Aware Accelerator

Our *Diffy* implementation builds upon the *Bit-Pragmatic* accelerator (*PRA*) [2]. *PRA* processes activations "bit"-serially, one effectual term at a time. *Offset generators* convert the activations into a stream of effectual powers of two after applying a modified Booth encoding. *PRA* multiplies a weight with a power of two or *oneffset* each cycle using a shifter. The oneffset's sign determines whether to add or subtract the shifted weight. *PRA* always matches or exceeds the throughput of an equivalent *VAA* by processing concurrently 16 activation windows while reusing the same set of weights (8 windows would have been sufficient).

Figure 7 shows a *PRA* tile comprising a grid of 16×16 Serial IP units (SIPs). Each SIP column corresponds to a different window. Each cycle, the tile broadcasts a set of 16 activation oneffsets to each SIP column for a total of 256 activation oneffsets per cycle. *PRA* restricts the distance of concurrently processed oneffsets to better balance area and performance. Accordingly, each oneffset needs 4b, 2b for the power of 2, a sign bit and a valid bit. Each SIP has a 16-input adder tree and instead of 16 multipliers it has 16 shifters. Each of these shift the 16b weight input as directed by the oneffset. All SIPs along the same row share the same set of 16 weights.

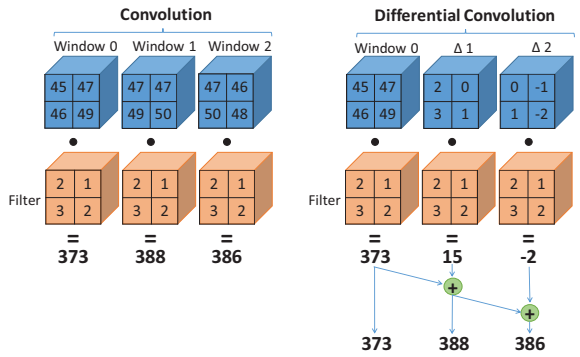


Fig. 8: Differential Convolution with inner product result propagating from a column to the next.

While *VAA* processes 16 activations per cycle, *PRA* processes 256 activations oneffset-serially. The dataflow we use for *VAA* processes $\{a(c,x,y)\cdots a(c+15,x,y)\}$ — a *brick* $a^B(c,x,y)$ in *PRA*'s terminology — concurrently, where $(c \text{ MOD } 16) = 0$. *PRA* processes $\{a^B(c,x,y), a^B(c,x+1,y), \cdots, a^B(c,x+15,y)\}$ — a *pallet* in PRA's terminology — concurrently and over multiple cycles.

C. Differential Convolution

Formally, given an output activation $o(n,y,x)$ that has been computed *directly* as per Eq. (1) and exploiting the distributive and associative properties of multiplication and addition, it is possible to compute $o(n,y,x+1)$ *differentially* as follows:

$$o(n,y,x+1) = o(n,y,x) + \langle w^n, \Delta_a \rangle \tag{4}$$

where Δ_a are the element-wise deltas of the imap windows corresponding to $o(n,y,x+1)$ and $o(n,y,x)$:

$$\begin{aligned} \Delta_a(k,j,i) \quad = \quad & a(k, j+y\times S,\ i+(x+1)\times S) \\ & -a(k, j+y\times S,\ i+x\times S) \end{aligned}$$

In the above, S is the stride between the two imap windows. The above method can be applied along the H or the W dimensions, and in general through any sequence through the imap. A design can choose an appropriate ratio of output calculations to calculate directly as per Eq. (1) or differentially as per Eq. (4) and a convenient dataflow.

Fig. 8 shows an example of differential convolution as opposed to direct convolution. It applies a 2×2 filter on three consecutive activation windows. While direct convolution uses the raw activations for all the three windows, differential convolution uses the raw activations just for the first window of a row (*Window 0*) and the deltas for the subsequent windows along the row. All three windows are computed *concurrently*. At the end, to reconstruct the correct outputs for windows 1 and 2, differential convolution adds the result of the previous window in a cascaded fashion. This latter phase is overlapped with the differential processing of additional windows.

D. Delta Dataflow

In the designs we evaluate we choose to calculate only the leftmost per row output directly and the remaining outputs along the row differentially. We do so since this is compatible with designs that buffer two complete window rows of the imap on-chip. This dataflow strategy reduces off-chip bandwidth when it is not possible to store the full imap on-chip.

Timing-wise, *Diffy* calculates each output row in *two phases* which are pipelined. During the first phase, *Diffy* calculates the leftmost per row output in parallel with calculating the $\langle w^n, \Delta_a \rangle$ terms for the remaining outputs per row. During a second phase, starting from the leftmost output, *Diffy* propagates the direct components in a cascaded fashion. A single addition per output is all that is needed. Given that the bulk of the time is needed for processing the leftmost inner-product and the $\langle w^n, \Delta_a \rangle$ terms, a set of adders provides sufficient compute bandwidth for the second phase. Each phase can process the whole row, or part of the row to balance the number of adders and buffers needed.

E. Diffy Architecture

Fig. 9 and Fig. 10 show how *Diffy* modifies the *PRA* architecture by introducing respectively a *Differential Reconstruction Engine* (*DR*) per SIP and a *Delta Output Calculation engine* ($Delta_{out}$) per tile. As Fig. 9 shows the per SIP DR engines reconstruct the original output while allowing the SIPs to calculate output activations using either deltas or raw values. The reconstruction proceeds in a cascaded fashion across the tile columns and per row as follows: Let us consider the processing of the first 16 windows of a row. The imap windows are fed into the 16 columns as deltas except for the very first window of the row which is fed as raw values. The SIPs in columns 1-15 process their assigned windows differentially while the SIPs of column 0 do so normally. When the SIPs of column 0 finish computing their current *output* brick, they pass it along through their AB_{out} to the SIPs of column 1. The column 1 SIPs can then update their differential outputs to their normal values. They then forward their results to the column 2 SIPs and so on along the columns. All subsequent sets of 16 windows per row can be processed differentially including the window assigned to column 0. This is possible since column 15 from the previous set of 16 windows can pass its final output brick back to column 0 in a round robin fashion. Since processing the next set of 16 windows typically takes hundreds of cycles, there is plenty of time to reconstruct the output activations and pass them through the activation function units. The multiplexer per *DR*: 1) allows the first window of a row, which is calculated using the raw imap values, to be written unmodified to AB_{out}, 2) allows intermediate results to be written to AB_{out} if needed to support other dataflow processing orders, and 3) makes it possible to revert to normal convolution for those layers whose performance might be negatively affected by differential convolution.

We have investigated two schemes for calculating the deltas. The first stores imaps raw in AM and calculates the deltas as the values are read out. We do not present this scheme for two reasons: First, it recomputes deltas anytime values are

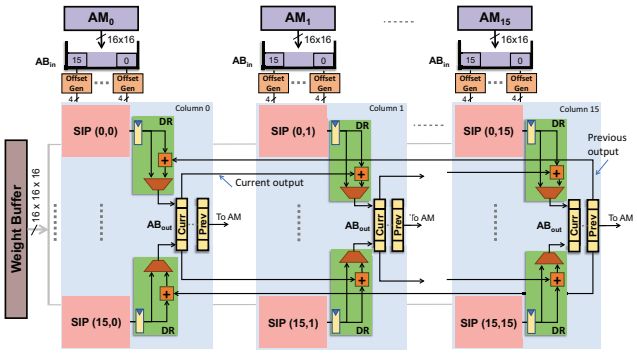


Fig. 9: Diffy tile reconstructing differential convolution output.

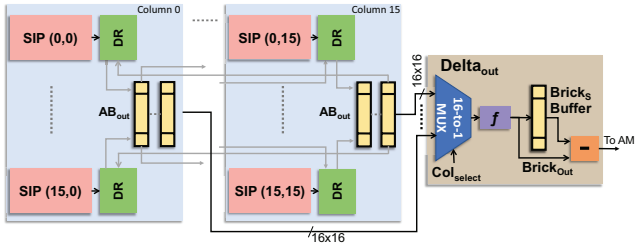


Fig. 10: *Diffy*'s $\text{Delta}_{\text{out}}$ engine computing deltas for next layer.

read from AM. Second, it does not take advantage of deltas to reduce on-chip storage and communication. We instead present the scheme where the deltas are calculated at the output of each layer, once per omap value and stored as such in AM.

Fig. 10 shows the architecture of the $Delta_{out}$ engine that *Diffy* uses to write the output bricks of the current layer back to the AM in the delta format. $Delta_{out}$ computes the delta bricks of columns 0 to 15 one at a time to reuse the hardware. Assuming the next layer's stride is S_{next}, computing the delta brick for column Col_{out} is done in two steps: 1) reading the output brick with stride S_{next} to the left of Col_{out} from the corresponding AB_{out}, passing it through the activation function f then storing it in the $Brick_s$ buffer. This process might need to wrap around to read an output brick corresponding to a previous pallet depending on the stride S_{next}. 2) Reading the brick of column Col_{out} from the corresponding AB_{out}, passing it through the activation function f and computing the delta brick using element-wise subtractors before writing the results to the AM. The 16-to-1 multiplexer controls from which AB_{out} to read at each step (the multiplexer is implemented as a read port across the AB_{out}s of a row). For example, if $S_{next} = 2$ and we want to compute the delta brick for column $Col_{out} = 0$, we need the output brick of column 14 which belongs to the previous pallet of output bricks. Thus, for step 1, the selection lines Col_{select} are set to $(Col_{out} - S_{next})$ MOD 16, while for step 2 they are set to Col_{out}. Each AB_{out} can store up to 4 output bricks corresponding to 4 consecutive output pallets. Thus, *Diffy* can handle any stride up to 48 which is far beyond what is needed by the current models.

F. Memory System

For per-pixel models it is imap/omap storage and traffic that dominates for the following reasons: 1) The models naturally scale with the input image resolution which is typically much higher than that used by most image classification models that are publicly available (for example, ImageNet models process image frames of roughly 230×230 pixels). 2) While all layers maintain the input resolution, most intermediate layers increase the number of channels. 3) The fmaps are comparatively small, do not increase with resolution, and may be dilated (e.g., a 3×3 filter expanded to 9×9 by adding zero elements). Accordingly, it may not be reasonable to assume that the imaps/omaps can fit on chip and off-chip bandwidth becomes a major concern. Similarly, it may not be reasonable to assume that we can fit the fmaps on chip for the full model and an effective dataflow that utilizes fmap and imap/omap reuse is essential.

Diffy opts for an off-chip strategy that reads each weight or input activation once per layer, and that writes each output activation at most once per layer. For this purpose the AM is sized to accommodate enough input rows to fit two complete rows of windows plus two output rows. This way *Diffy* can process the windows of one row from on-chip (which requires buffering a row of output activations), while loading the activations for the next row of windows from off-chip memory, while also simultaneously writing the previous row of output activations to off-chip memory (which requires buffering another row of output activations). For the fmaps, the WM is sized to be as large as to hold all fmaps that will be processed concurrently. This depends on the number of fmaps per tile and the number of tiles. To completely hide the loading of the next set of fmaps we need the buffer to also have space to load the next set of fmaps for the same layer or for the first set of the next layer. Section IV-C shows that the AM and WM memories needed are reasonable for today's technologies and demonstrates that delta encoding can further reduce their size. If smaller WM and AM are desired, off-chip bandwidth will increase. Yang *et al.,* present an algorithm for determining energy efficient blocking dataflows which can be adapted for our purposes [34]. To reduce off-chip traffic *Diffy* encodes activations as deltas using a dynamically detected precision per group of activations instead of the raw values. Dynamic precision detection is done similarly to Dynamic Stripes [33]. Activations are stored in groups of 16. Each group contains a 4-bit header indicating the number of bits for all activations per group. On-chip, when compression is used the activations are stored in virtual columns as in Proteus [3] and a separate virtual column contains the precisions per group.

IV. Evaluation

Table I details the CI-DNNs we study. DnCNN [13], FFDNet [14] and IRCNN [12] are state-of-the-art image denoising DNN models that rival the output quality of non-local similarity-based methods such as BM3D [48] and WNNM [49]. JointNet performs demosaicking and denoising. VDSR is a 20-layer DNN model that delivers state-of-the-art quality single-image super-resolution [20]. Some of these models

TABLE I: CI-DNNs studied.

	DnCNN	FFDNet	IRCNN	JointNet	VDSR
Application	Denoising			Demosaicking + Denoising	Super-resolution
Conv. Layers	20	10	7	19	20
ReLU Layers	19	9	6	16	19
Max Filter Size (KB)	1.13	1.69	1.13	1.13	1.13
Max Total Filter Size per Layer (KB)	72	162	72	144	72

TABLE II: Input Datasets Used

Dataset	Samples	Resolution	Description
CBSD68	68	481×321	test section of the Berkeley data set [35], [36]
McMaster	18	500×500	CDM Dataset, modified McMaster [37]
Kodak24	24	500×500	Kodak dataset [38],
RNI15	15	370×280 – 700×700	noisy images covering real noise such as from the camera or from JPEG compression [14], [39], [40]
LIVE1	29	634×438 – 768×512	widely used to evaluate super-resolution algorithms [41], [42], [43], [44]
Set5+Set14	5 + 14	256×256 – 720×576	images used for testing super-resolution algorithms [20], [45], [46].
HD33	33	1920×1080	HD frames depicting nature, city and texture scenes [47]

TABLE III: Profile-derived per layer activation precisions.

Network	Profile-derived per layer precisions
DnCNN	9-9-10-11-10-9-10-9-10-10-9-9-9-9-9-9-9-9-9-11-13
FFDNet	10-9-10-10-10-10-10-10-9-9
IRCNN	9-9-9-8-7-8-8
JointNet	9-9-10-9-9-9-9-9-9-10-9-9-9-9-10-8-10-10-11
VDSR	9-10-9-7-7-7-7-7-7-8-8-6-7-8-7-7-7-7-9-8

VAA*, *PRA* and *Diffy			
Tiles	4	WM/Tile	$8KB \times 16$ Banks
Filters/Tile	16	$AB_{in/out}$/Tile	$2KB$
Weights/Filter	16	Off-chip Memory	4GB LPDDR4-3200
Tech Node	65nm	Frequency	1GHz
Diffy			
AM/Tile		$8KB \times 16$ Banks = 128KB	
VAA* and *PRA			
AM/Tile		$16KB \times 16$ Banks = 256KB	

TABLE IV: *VAA*, *PRA* and *Diffy* configurations.

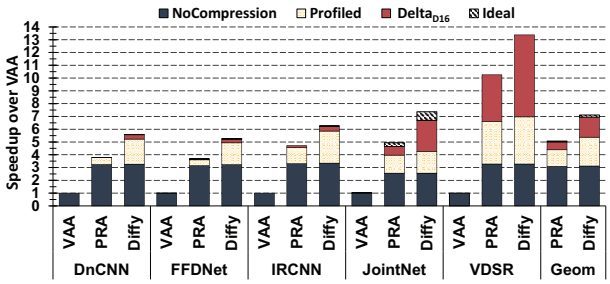


Fig. 11: Speedup of *PRA* and *Diffy* over *VAA*.

can be retrained to perform other image tasks. For example, the same DnCNN network architecture can be trained to perform single-image super-resolution and JPEG deblocking as well [15] and IRCNN can be trained to perform in-painting and deblurring [12]. We followed the methodology of Judd *et al.* [31] to find the minimum per layer precisions such that the output quality remains within 1% relative to that of the floating point representation for the two quality metrics used to evaluate the original DNN models: 1) Signal-to-noise ratio (*SNR*), and 2) Structural similarity (*SSIM*). Table III reports the resulting per layer precisions.

A custom cycle-accurate simulator models the performance of all architectures. Table IV reports the default configurations. For area and power consumption all designs were implemented in Verilog and synthesized through the Synopsys Design Compiler [50]. Layout was performed using Cadence Innovus [51] and for a 65nm TSMC technology. For power estimation we used Mentor Graphics ModelSim to capture circuit activity and used that as input to Innovus. We use CACTI to model the area and power consumption of the on-chip SRAM memories and buffers as an SRAM compiler is not available to us. The accelerator target frequency is set to 1GHz given CACTI's estimate for the speed of the buffers. We study the effect of the off-chip memory technology node by considering nodes from DDR3-1600 and up to HBM2.

A. Relative Performance

Since the design space is large, we start by investigating performance for HD images and with a DDR4-3200 off-chip memory interface. First we look at relative performance. Fig. 11 shows the performance of *PRA* and of *Diffy* normalized to *VAA* while taking into account the following off-chip compression schemes: a) no compression (*NoCompression*), b) encoding using a per layer profile-derived precision (*Profiled*) [3], c) per group dynamic precision for groups of 16 activations ($Delta_{D16}$) and where for activations we store deltas, and d) infinite off-chip bandwidth (*Ideal*). We do not show performance results with *RLEz* and *RLE* since they are less effective than the other schemes as Section IV-C will show.

Off-chip memory is not a bottleneck for *VAA* and thus its performance is unaffected by compression (its energy efficiency of course will). Compression however enables *PRA* and *Diffy* to deliver their full performance benefits. Specifically, *PRA* can ideally improve performance by 5.1× over *VAA*. However, depending on the model, *PRA* needs the *Profiled* or the $Delta_{D16}$ compression to avoid stalling noticeably due to off-chip memory transfers. The $Delta_{D16}$ scheme is necessary for JointNet and VDSR. Under this scheme *PRA* achieves almost the ideal speedup with 5× over *VAA*.

Diffy outperforms *VAA* and *PRA* by 7.1× and 1.41× respectively on average. As with *PRA* it needs $Delta_{D16}$ compression

to avoid stalling noticeably for off-chip memory. It is only for JointNet that off-chip memory stalls remain noticeable at about 8.2%. Benefits with both *PRA* and *Diffy* are nearly double for VDSR compared to the other models. VDSR exhibits high activation sparsity in the intermediate layers and requires shorter precisions compared to the other models. In general, the benefits are proportional to but lower than the potential measured in Section II. There are two reasons why the full potential is not achieved: Underutilization due to the number of filters available per layer, and cross-lane synchronization due to imbalance in the number of effectual terms per activation. The latter is the most significant and discussed in Section IV-E.

Fig. 12 reports a per layer breakdown of lane utilization for *Diffy* in the following categories: a) useful cycles, b) idle cycles which may be due to cross-lane synchronization or due to filter underutilization, and c) stalls due to off-chip delays. Utilization varies considerably per layer and per network. Off-chip delays noticeably appear only for certain layers of FFDNet and JointNet. For FFDNet these layers account for a small percentage of overall execution time and hence do not impact overall performance as much as they do for JointNet. Utilization is very low for VDSR. This is due to cross lane synchronization since VDSR has high activation sparsity and the few non-zero activations dominate execution time. The first layer of all networks incurs relatively low utilization because the input image has 3 channels thus, with the dataflow used, 13 out of the 16 available activation lanes are typically idle. FFDNet is an exception since the input to the first layer is a 15-channel feature map: the input image is pre-split into 4 tiles stacked along the channel dimension with 3 extra channels describing the noise standard deviation of each of the RGB color channels. Also the last layer for all networks exhibits very low utilization. This layer produces the final 3-channel output and has only 3 filters and thus with the dataflow used can keep only 3 of the 64 filter lanes busy. Allowing each tile to use its activation locally could enable *Diffy* to partition the input activation space across tiles and to improve utilization. Moreover, idle cycles could be used to reduce energy. However, these explorations are left for future work.

The per-layer relative speedup of *Diffy* over *PRA*, not shown due to the limited space, is fairly uniform with a mean of 1.42× and a standard deviation of 0.32. *Diffy* underperforms *PRA* only on a few noncritical layers in JointNet and VDSR and there by at most 10%. We have experimented with a *Diffy* variant that uses profiling to apply Differential Convolution selectively per layer and only when it is beneficial. While this eliminated the per layer slowdowns compared to *PRA*, the overall improvement was negligible and below 1% at best.

B. Absolute Performance: Frame Rate

Since our interest is on HD resolution processing, Fig. 13 reports detailed measurements for this resolution. The figure shows the FPS for *VAA*, *PRA*, and *Diffy* for the considered off-chip compression schemes. The results show that *Diffy* can robustly boost the FPS over *PRA* and *VAA*. The achievable FPS varies depending on the image content with the variance being ±7.5% and ±15% of the average FPS for *PRA* and *Diffy* respectively. While *Diffy* is much faster than the alternatives, it is only for JointNet that the FPS is near the real-time 30 FPS rate. For real-time performance to be possible more processing tiles are needed. Section IV-G explores such designs. With the current configuration, *Diffy* is more appropriate for user-interactive applications such as photography with a smartphone.

TABLE V: Minimum on-chip memory capacities vs. encoding scheme.

Mem.	Type	Baseline	Profiled	Raw_{D16}	Delta_{D16}
AM	SRAM	964KB	782KB	514KB	348KB
WM	SRAM	324KB			

C. Compression and Off-Chip Memory

We study the effect of delta encoding on on-chip storage and off-chip traffic. Table V reports the total on-chip storage needed for fmaps and imaps/omaps. The total weight memory needed for these networks is 324KB, which can be rounded up to 512KB or to 128KB per tile for a four tile configuration. Since our $Delta_{D16}$ scheme targets activations, the table reports the total activation memory size needed for three storage schemes that mirror the compression schemes of Fig. 5. Our $Delta_{D16}$ can reduce the on-chip activation memory or boost its effective capacity. Without any compression the AM needs to be 964KB. *Profiled* reduces the storage needed by 19% to 782KB, whereas Raw_{D16} reduces AM by 46% to 514KB. If we were to round all these up to the next power of two sized capacity, they would all lead to a 1MB AM, however, the fact remains that with Raw_{D16} the effective capacity will be higher enabling the processing of larger models and/or higher resolutions. Finally, using our proposed $Delta_{D16}$ reduces AM to just 348KB. This is a 55% and a 32% reduction over *Profiled* and Raw_{D16} respectively. We can round this up to 512KB as needed. Regardless of the rounding scheme used, our $Delta_{D16}$ compression considerably reduces the on-chip AM capacity that is required. For the rest of the evaluation we round up the AM capacity to the nearest higher power of two.

Fig. 14 reports off-chip traffic normalized to *NoCompression*. Taking the metadata into account, the benefit of the *RLEz* and *RLE* schemes is significant only for VDSR due to its high activation sparsity. These schemes prove ineffective for CI-DNNs while they were shown to be effective for classification models [32]. *Profiled* reduces off-chip traffic to about 54%. Using dynamic per group precisions reduces off-chip traffic further to 39% with a group of 256 (Raw_{D256}) and to about 28% with the smaller group sizes of 16 (Raw_{D16}) or 8 (Raw_{D8}) — the overhead due to the metadata increases as the group size decreases. Storing activations as deltas with per group precision ($Delta_{D16}$) further reduces off-chip traffic resulting to just 22% of the uncompressed traffic, an improvement of 27% over Raw_{D16}. Since off-chip accesses are two orders of magnitude more expensive than on-chip accesses, this reduction in off-chip traffic should greatly improve overall energy efficiency. While using a group size of 16 ($Delta_{D16}$) reduces traffic considerably compared to using a group size of 256 ($Delta_{D256}$) the metadata

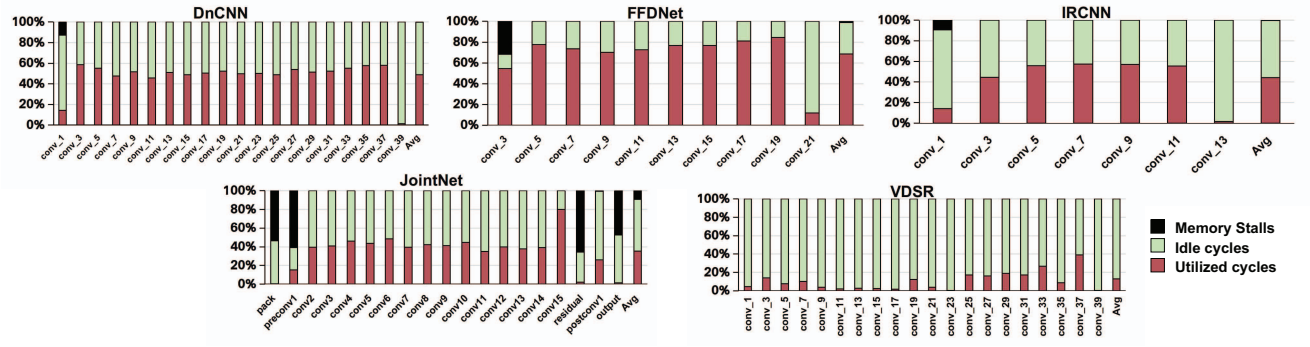


Fig. 12: Execution time breakdown for each network showing the useful work cycles, idle cycle and memory stalls.

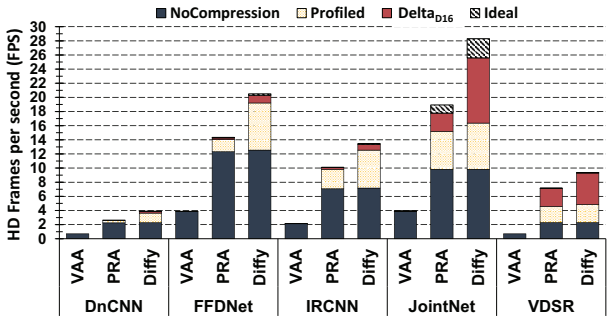


Fig. 13: HD frames per second processed by *VAA*, *PRA* and *Diffy* with different compression schemes.

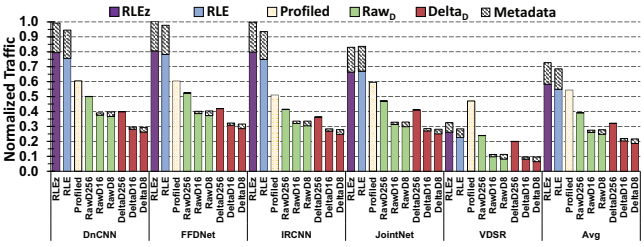


Fig. 14: Compression schemes: Off-chip traffic normalized to no compression.

overhead prevents further reduction with the smaller group size (*Delta*$_{D8}$). In the rest of the evaluation we restrict attention to *Delta*$_{D16}$ for on-chip and off-chip encoding of imaps/omaps.

Finally, we study the effect of the off-chip memory on overall performance by considering six memory technologies ranging from the now low-end LPDDR3-1600 up to the high-end HBM2. We do so to demonstrate that our off-chip compression scheme is essential to sustain the performance gains with realistic off-chip memory. Fig. 15 shows the performance of *Diffy* normalized to *VAA* with the compression schemes shown as stacked bars. Without any compression all models require at least an HBM2 memory to incur no slowdown. JointNet and VDSR are the most sensitive since even with *Profiled* and the high-end LPDDR4X-4267, they sustain 77% and 68% of their maximum performance respectively. The other networks perform within 2.5% of their peak. For any of

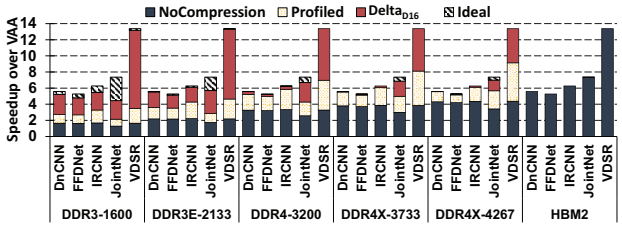


Fig. 15: Performance of *Diffy* with different off-chip DRAM technologies and activation compression schemes.

the less capable memory nodes, the performance slowdowns are much more noticeable. Our *Delta*$_{D16}$ allows all networks to operate at nearly their maximum for all memory nodes starting from LPDDR4-3200 where only JointNet incurs an 8.2% slowdown. Even with the LPDDR3E-2133 node, performance with *Delta*$_{D16}$ is within 2% of the maximum possible for all networks except JointNet for which it is within 22% of maximum. With a 2-channel LPDDR4X-4267 memory system *Diffy* sustains only 87% and 65% of its maximum performance for JointNet and VDSR under no compression. With *Profiled*, two channels of LPDDR4X-3733 are sufficient to preserve 94% and 98% of their performance respectively. Finally, with *Delta*$_{D16}$ and a dual-channel LPDDR3E-2133 memory system, VDSR incurs no slowdowns while JointNet performs within 5% of its maximum.

D. Power, Energy Efficiency, and Area

Table VI reports a breakdown of power for the three architectures. While both *PRA* and *Diffy* consume more power, their speedup is higher than the increase in power, and thus they are 1.34× and 1.83× more energy efficient than *VAA*. Even if *Diffy* used the same uncompressed on-chip 1MB AM, it will still be 1.76× energy efficient with 1.57× the area of the baseline *VAA*. Moreover, these measurements ignore the off-chip traffic reduction achieved by *Diffy*. As off-chip accesses are orders of magnitude more expensive than on-chip accesses and computation, the overall energy efficiency for *Diffy* will be higher. The power and area of the compute cores of *Diffy* is higher than *PRA* due to the additional DR engines.

TABLE VI: Power [W] consumption breakdown for *Diffy* vs. *PRA* and *VAA*.

	Diffy		*PRA*		*VAA*	
	Power	**%**	**Power**	**%**	**Power**	**%**
Compute	11.77	86.67	10.80	82.75	1.90	54.42
AM	0.79	5.85	1.36	10.45	1.36	38.96
WM	0.37	2.73	0.27	2.07	0.08	2.29
AB_{in}+AB_{out}	0.15	1.12	0.15	1.16	0.15	4.33
Dispatcher	0.25	1.85	0.25	1.92	-	-
Offset Gens.	0.21	1.58	0.21	1.64	-	-
$Delta_{out}$	0.03	0.21	-	-	-	-
Total	**13.58**	100%	**13.05**	100%	**3.50**	100%
Normalized	**3.88**×		**3.73**×		**1**×	
Energy Efficiency	**1.83**×		**1.34**×		**1**×	

TABLE VII: Area [mm^2] breakdown for *Diffy* vs. *PRA* and *VAA*.

	Diffy		*PRA*		*VAA*	
	Area	**%**	**Area**	**%**	**Area**	**%**
Compute	15.50	53.05	14.49	40.19	3.36	14.26
AM	6.05	20.70	13.93	38.62	13.93	59.11
WM	6.05	20.70	6.05	16.77	6.05	25.67
AB_{in}+AB_{out}	0.23	0.77	0.23	0.62	0.23	0.96
Dispatcher	0.37	1.28	0.37	1.04	-	-
Offset Gens.	1.00	3.42	1.00	2.77	-	-
$Delta_{out}$	0.02	0.09	-	-	-	-
Total	**29.22**	100%	**36.07**	100%	**23.56**	100%
Normalized	**1.24**×		**1.53**×		**1**×	

Table VII reports a breakdown of area for the architectures. Since *Diffy* uses $Delta_{D16}$ for the AM, its overall overhead over *VAA* is lower than *PRA*, furthermore, its area overhead is far less than its performance advantage. Even if *Diffy* were to use the same AM as *PRA* or *VAA*, it would be 1.76× more energy efficient using 1.57× the area of *VAA*. Thus, it would still be preferable over a scaled up *VAA* (performance would not scale linearly for *VAA*— it would suffer more from underutilization and would require wider WMs).

E. Sensitivity to Tiling Configuration

Fig. 16 reports performance for different tile configurations T_x where x is the number of terms (weight×activation products) processed concurrently per filter. So far we considered only T_{16}. When both *VAA* and *Diffy* are configured to process one term per filter at a time (T_1), the average speedup of *Diffy* over *VAA* increases from 7.1× to 11.9×. When multiple terms are concurrently processed, the execution time is determined by the activation with the largest number of effectual terms which induces idle cycles for other activation lanes having fewer effectual terms. The T_1 configuration eliminates these stalls and

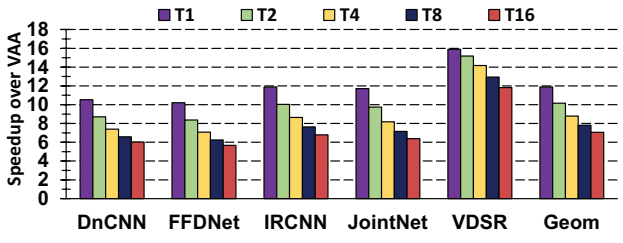


Fig. 16: Performance sensitivity to the number of terms processed concurrently per filter.

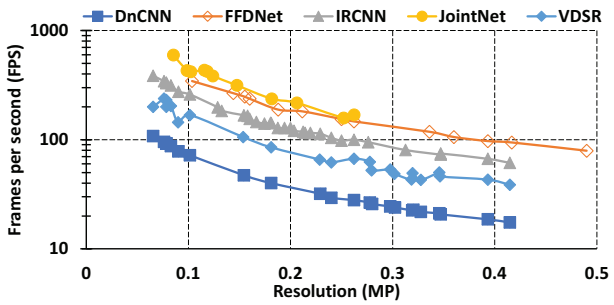


Fig. 17: Frames per second processed by *Diffy* as a function of image resolution. HD frame rate is plotted in Fig. 13.

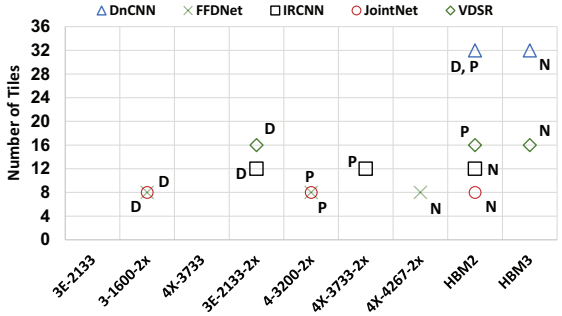


Fig. 18: Tiles needed for *Diffy* to sustain real-time HD processing along with the needed memory system for each compression scheme: *D*, *P* and *N* for $Delta_{D16}$, *Profiled* and *NoCompression*.

closes most of the gap between potential and actual speedup for all models except for VDSR due to its high activation sparsity.

F. Absolute Performance: Low Resolution Images

Since there are applications where processing lower resolution images is sufficient and for completeness we report absolute performance measured as the number of frames processed per second for a range of low resolutions. Each DNN model is run on *Diffy* with a subset of the datasets listed in Table II excluding HD33. Fig. 17 shows that for lower resolution images, *Diffy* can achieve real-time processing for all models except for DnCNN when used with resolutions above 0.25 mega-pixel (*MP*). Even for DnCNN, *Diffy* can process 19 FPS for 0.4MP frames. These results suggest that *Diffy*, with its current configuration, can be used for real-time applications where processing lower resolution frames is sufficient.

G. Scaling for Real-time HD Processing

To achieve real-time 30 FPS processing of HD frames, *Diffy* needs to be scaled up as reported in Fig. 18. For each model and each compression scheme we report the minimum configuration needed, that is the number of tiles (y-axis) and external memory system (x-axis). The x-axis reports the off-chip memory configuration as *v-r-x* where *v* is the LPDDR version, *r* is the transfer rate and *x* is the number of channels. DnCNN is the most demanding requiring 32 tiles and an HBM2 under $Delta_{D16}$ and *Profiled* or HBM3 otherwise. Then VDSR needs 16 tiles however a dual-channel DDR3E-2133 is

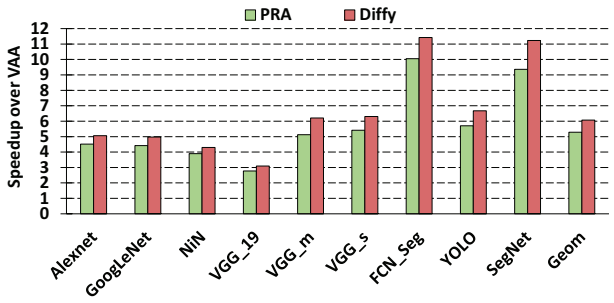


Fig. 19: *PRA* and *Diffy* speedup for classification DNNs.

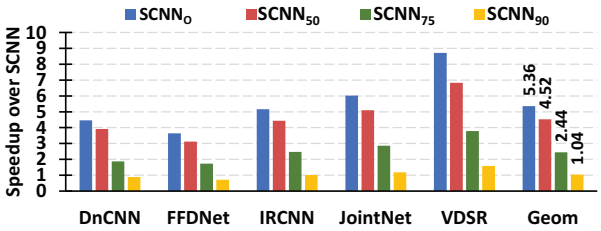


Fig. 20: Speedup of *Diffy* over SCNN.

sufficient with the $Delta_{D16}$ compression due to the activations sparsity of that model. FFDNet and JointNet need 8 tiles with a dual-channel DDR3-1600 while IRCNN needs 12 with a dual-channel DDR3E-2133 under our $Delta_{D16}$ compression.

H. Classification DNN Models

While *Diffy* targets CI-DNNs it can execute any CNN. Since classification remains an important workload we run several well known ImageNet classification models on *VAA*, *PRA* and *Diffy*. The figure also includes: FCN_Seg, a per-pixel prediction model performing semantic segmentation which is another form of classification where pixels are grouped into areas of interest, YOLO V2 [52], and SegNet [9] which are detection models that detect and classify multiple objects in an image frame. Fig. 19 reports the resulting performance. *Diffy*'s speedup is 6.1× over *VAA*, an improvement of 1.16× over *PRA*. While modest, the results show that differential convolution not only does not degrade but also benefits performance for this class of models. Accordingly, *Diffy* can be used as a general CNN accelerator.

I. Performance Comparison with SCNN

Fig. 20 shows the speedup of *Diffy* over SCNN [32] under various assumptions about weight sparsity where $SCNN_O$, $SCNN_{50}$, $SCNN_{75}$ and $SCNN_{90}$ refer to SCNN running unmodified, 50%, 75% and 90% randomly sparsified versions of the models respectively.

On average, *Diffy* consistently outperforms SCNN even with the 90% sparse models. Specifically, *Diffy* is 5.4×, 4.5×, 2.4× and 1.04× faster than SCNN for the four sparsity assumptions respectively. We believe that even a 50% weight sparsity for these models is optimistic since in the analytical models these CI-DNNs mimic each pixel value depends on a large neighborhood of other pixel values. Moreover, activation sparsity is lower for these models compared to the classification models used in the original SCNN study.

V. Related Work

Due to space limitations we limit attention to the most relevant works. We have already compared to *PRA* and SCNN. Another accelerator that could potentially benefit from differential convolution is *Dynamic Stripes* whose performance varies with the precision of the activations [33]. Since deltas are smaller values than the activations, their precision requirements will be lower as well. While Dynamic Stripes does not perform as well as *PRA* it is a simpler and lower cost design.

CBInfer is an algorithm that reduces inference time for CNNs processing video frames by computing only convolutions for activation values that have changed across frames [53]. While CBInfer targets changes in values *temporally* across frames, *Diffy* targets changes in values *spatially* within frames. CBInfer is limited to frame-by-frame video processing, whereas *Diffy* can work for more general computational imaging tasks. Unlike CBInfer which requires additional storage to store the previous frame values, *Diffy* can potentially reduce the needed storage and bandwidth through delta compression. Moreover, CBInfer is a software implementation for graphics processors. However, the two concepts could be potentially combined. Similarly, Euphrates exploits the motion vectors generated based on the cross-frame temporal changes in pixel data, to accelerate CNN-based high-level semantic-level tasks later on along the vision pipeline [54]. Instead of performing an expensive CNN inference for each frame, Euphrates extrapolates the results for object detection and tracking applications based on the motion vectors naturally generated by the early stages of the pipeline. Our approach is complementary as it operates within each frame. Moreover we target lower-level computational imaging tasks.

IDEAL accelerated the BM3D-based image denoising algorithm enabling real-time processing of HD frames [55]. While IDEAL's performance is 5.4× over DaDianNao [1] running JointNet, *Diffy* boosts the performance by 7.25× over DadianNao for that specific DNN model.

VI. Summary

We studied an emerging class of Deep Learning Networks for computational imaging tasks that rival conventional analytical methods. We demonstrated that they exhibit spatial correlation in their value stream and proposed an accelerator that converts this property into communication, storage, and computation benefits boosting performance and energy efficiency. Our accelerator makes it more practical to deploy such Deep Learning Networks enabling real-time processing. Diffy enables differential processing and motivates approaches where computation can be performed on deltas. While we have studied this approach for inference and for imaging based Deep Learning models, future work can investigate its application on other model architectures and in training.

REFERENCES

[1] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014.

[2] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3123939.3123982

[3] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, N. E. Jerger, and A. Moshovos, "Proteus: Exploiting numerical precision variability in deep neural networks," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: ACM, 2016. [Online]. Available: http://doi.acm.org/10.1145/2925426.2926294

[4] A. Delmas, S. Sharify, P. Judd, M. Nikolic, and A. Moshovos, "DPRed: Making typical activation values matter in deep learning computing," *CoRR*, vol. abs/1804.06732, 2018. [Online]. Available: http://arxiv.org/abs/1804.06732

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.81

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. [Online]. Available: http://arxiv.org/abs/1411.4038

[9] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, Dec 2017.

[10] G. Lin, A. Milan, C. Shen, and I. D. Reid, "RefineNet: Multi-path refinement networks for high-resolution semantic segmentation," *CoRR*, vol. abs/1611.06612, 2016. [Online]. Available: http://arxiv.org/abs/1611.06612

[11] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," *CoRR*, vol. abs/1412.5567, 2014. [Online]. Available: http://arxiv.org/abs/1412.5567

[12] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep CNN denoiser prior for image restoration," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, July 2017.

[14] K. Zhang, W. Zuo, and L. Zhang, "FFDNet: Toward a fast and flexible solution for CNN based image denoising," *CoRR*, vol. abs/1710.04026, 2017. [Online]. Available: http://arxiv.org/abs/1710.04026

[15] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, "Deep joint demosaicking and denoising," *ACM Trans. Graph.*, vol. 35, Nov. 2016. [Online]. Available: http://doi.acm.org/10.1145/2980179.2982399

[16] S. Zhang and E. Salari, "A neural network-based nonlinear filter for image enhancement," *International Journal of Imaging Systems and Technology*, vol. 12, 2002. [Online]. Available: http://dx.doi.org/10.1002/ima.10009

[17] S. Nah, T. H. Kim, and K. M. Lee, "Deep multi-scale convolutional neural network for dynamic scene deblurring," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[18] M. Hradiš, J. Kotera, P. Zemčík, and F. Šroubek, "Convolutional neural networks for direct text deblurring," in *Proceedings of BMVC 2015*. The British Machine Vision Association and Society for Pattern Recognition, 2015. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php?id=10922

[19] C. J. Schuler, M. Hirsch, S. Harmeling, and B. Schlkopf, "Learning to deblur," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, July 2016.

[20] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," *CoRR*, vol. abs/1511.04587, 2015. [Online]. Available: http://arxiv.org/abs/1511.04587

[21] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, Feb 2016.

[22] W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[23] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-recursive convolutional network for image super-resolution," *CoRR*, vol. abs/1511.04491, 2015. [Online]. Available: http://arxiv.org/abs/1511.04491

[24] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[25] L. Jenner, "Hubble's high-definition panoramic view of the Andromeda galaxy," https://www.nasa.gov/content/goddard/hubble-s-high-definition-panoramic-view-of-the-andromeda-galaxy, Jan. 5 2015.

[26] J. E. Krist, "Deconvolution of hubble space telescope images using simulated point spread functions," in *Astronomical Data Analysis Software and Systems I*, vol. 25, 1992.

[27] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016.

[28] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernndez-Lobato, G. Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016.

[29] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, Jan 2017.

[30] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016. [Online]. Available: https://doi.org/10.1109/ISCA.2016.30

[31] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," *IEEE Computer Architecture Letters*, vol. 16, Jan 2017.

[32] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017.

[33] A. Delmas, P. Judd, S. Sharify, and A. Moshovos, "Dynamic Stripes: Exploiting the dynamic precision requirements of activation values in neural networks," *CoRR*, vol. abs/1706.00504, 2017. [Online]. Available: http://arxiv.org/abs/1706.00504

[34] X. Yang, J. Pu, B. B. Rister, N. Bhagdikar, S. Richardson, S. Kvatinsky, J. Ragan-Kelley, A. Pedram, and M. Horowitz, "A systematic approach to blocking convolutional neural networks," *CoRR*, vol. abs/1606.04209, 2016. [Online]. Available: http://arxiv.org/abs/1606.04209

[35] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, 2001.

[36] S. Roth and M. J. Black, "Fields of experts: a framework for learning image priors," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, June 2005.

[37] L. Zhang, X. Wu, A. Buades, and X. Li, "Color demosaicking by local directional interpolation and nonlocal adaptive thresholding," *Journal of Electronic imaging*, vol. 20, 2011.

[38] R. Franzen, "Kodak lossless true color image suite," http://r0k.us/graphics/kodak, Nov. 15 1999.

[39] M. Lebrun, M. Colom, and J.-M. Morel, "The noise clinic: a blind image denoising algorithm," *Image Processing On Line*, vol. 5, 2015.

[40] "Neat image," https://ni.neatvideo.com/home.

[41] C. Dong, Y. Deng, C. Change Loy, and X. Tang, "Compression artifacts reduction by a deep convolutional network," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[42] H. R. Sheikh, Z. Wang, L. Cormack, and A. C. Bovik, "Live image quality assessment database release 2," http://live.ece.utexas.edu/research/quality/subjective.htm.

[43] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Transactions on Image Processing*, vol. 15, Nov 2006.

[44] C.-Y. Yang, C. Ma, and M.-H. Yang, "Single-image super-resolution: A benchmark," in *Proceedings of European Conference on Computer Vision*, 2014.

[45] M. Bevilacqua, A. Roumy, C. Guillemot, and M. line Alberi Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," *Proceedings of the British Machine Vision Conference*, 2012.

[46] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," in *Curves and Surfaces*, J.-D. Boissonnat, P. Chenin, A. Cohen, C. Gout, T. Lyche, M.-L. Mazure, and L. Schumaker, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[47] "Photography blog," 2017. [Online]. Available: http://www.photographyblog.com

[48] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-D transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, vol. 16, Aug 2007.

[49] S. Gu, L. Zhang, W. Zuo, and X. Feng, "Weighted nuclear norm minimization with application to image denoising," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.366

[50] Synopsys, "Design compiler graphical," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html.

[51] Cadence, "Innovus implementation system," https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/hierarchical-design-and-floorplanning/innovus-implementation-system.html.

[52] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. [Online]. Available: http://arxiv.org/abs/1612.08242

[53] L. Cavigelli, P. Degen, and L. Benini, "CBinfer: Change-based inference for convolutional neural networks on video data," in *Proceedings of the 11th International Conference on Distributed Smart Cameras*, ser. ICDSC 2017. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3131885.3131906

[54] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: Algorithm-soc co-design for low-power mobile continuous vision," *CoRR*, vol. abs/1803.11232, 2018. [Online]. Available: https://arxiv.org/abs/1803.11232

[55] M. Mahmoud, B. Zheng, A. D. Lascorz, F. Heide, J. Assouline, P. Boucher, E. Onzon, and A. Moshovos, "IDEAL: Image denoising accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3123939.3123941