3D-based Video Recognition Acceleration by Leveraging Temporal Locality

Huixiang Chen[*], Mingcong Song[*], Jiechen Zhao[*], Yuting Dai[‡], Tao Li[*]
[*]IDEAL Lab, University of Florida; [†]Guizhou University
{stanley.chen, songmingcong, jiechen.zhao}@ufl.edu, yutingdai90@gmail.com, taoli@ece.ufl.edu

Abstract

Recent years have seen an explosion of domain-specific accelerator for Convolutional Neural Networks (CNN). Most of the prior CNN accelerators target neural networks on image recognition, such as AlexNet, VGG, GoogleNet, ResNet, etc. In this paper, we take a different route and study the acceleration of 3D CNN, which are more computational-intensive than 2D CNN and exhibits more opportunities. After our characterization on representative 3D CNNs, we leverage differential convolution across the temporal dimension, which operates on the temporal delta of imaps for each layer and process the computation bit-serially using only the effectual bits of the temporal delta. To further leverage the spatial locality and temporal locality, and make the architecture general to all CNNs, we propose a control mechanism to dynamically switch across spatial delta dataflow and temporal delta dataflow. We call our design temporal-spatial value aware accelerator (TSVA). Evaluation on a set of representation NN networks shows that TSVA can achieve an average of 4.24× speedup and 1.42× energy efficiency. While we target 3D CNN for video recognition, TSVA could also benefit other general CNNs for continuous batch processing.

1. Introduction

The end of Moore's law [1] and Dennard scaling [2], and the consequently dark silicon phenomenon [3] has led to the end of rapid improvement of general-purpose program performance. Instead of improving the general-purpose computing, domain-specific architecture is considered the most effective way. Recent years, neural networks (NN) are gaining lots of interests as they achieve state-of-the-art performance on many tasks [4], including image recognition [5], speech recognition [6], video understanding [7] and analytics [8][9], autonomous driving [10][11]. Towards the domain-specific acceleration for better performance and energy efficiency, various hardware accelerators have been designed [12][13][14][15][16][17][18][19][20].

Given the recent fast development of neural network on image recognition, the deep learning-based approaches have also significantly improved video recognition performance. Action recognition from videos would require capture context from the entire video (i.e. a sequence of frames) rather than just capturing information from each frame [21]. Video recognition also received numerous attention from computer vision community [22][7][23][24][25][26], with different datasets developed for different domains [27][28][29]. Deep 3-dimensional convolution neural networks (3D-CNN) have demonstrated their outstanding classification performance in video recognition.

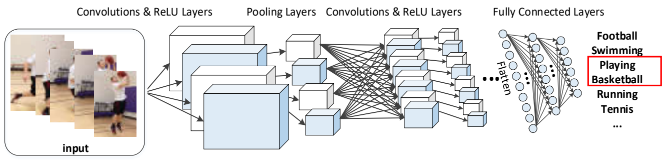


Figure 1. A real 3D CNN Model for video action recognition.

Video-based 3D CNN inferences the activity based on a sequence of frames extracted directly from the video. It involves the identification of different actions across video clips (i.e. a sequence of frames) where the action may or may not be performed throughout the entire duration of the video [21]. It has been tough for the following reasons: (1) High computational cost. For instance, a simple 2D convolution network for image classification for 101 classes has just ~5M parameters, whereas the same architecture inflated to a 3D structure results in ~33M parameters [21]. It also takes 3 to 4 days to train a 3D convolutional neural network on UCF101 datasets [27] and about two months on Sports-1M [7]. (2) Capturing long context action involves capturing spatiotemporal context across frames [21]. There is a local as well as global context (motion information) which needs to be captured for robust predictions.

The algorithm complexity and memory bandwidth demand of 3D CNN put great pressure on the processing speed. Different acceleration options have been leveraged to accelerate 3D CNN, including FPGAs [30][31], ASIC [32]. 3D CNN exhibits some characteristics that challenge the option of directly applying 2D acceleration option infeasible: (1) working set size exceeds on-chip memory; (2) 3D CNN is much more computation-intensive than memory-intensive based on the Roofline model [33]; (3) 3D CNN exhibits more reuse opportunities: it also has temporal dimensional data reuse besides spatial reuse.

Since video processing and analytics is popular in people's daily life, and video traffic is predicted to account for most of the internet traffic in the future [34]. Accelerating 3D CNN for video recognition will shed light on the development of future video analytics, saving power and making users' life easier.

ISCA '19, June 22–26, 2019, Phoenix, AZ, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6669-4/19/06…$15.00
https://doi.org/10.1145/3307650.3322260

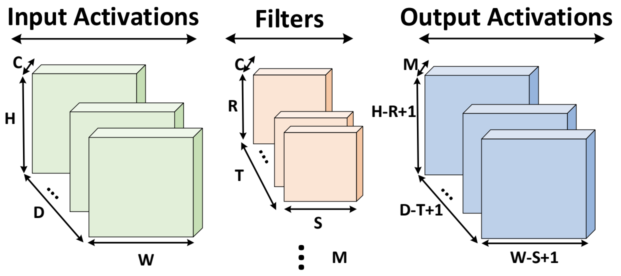


Figure 2. Illustration of 3D CONV. The illustration shows when stride=1, which is the most common case.

```
for m = 0 to M
  for d = 0 to $D_{out}$ (temporal information: number of frames)
    for h = 0 to $H_{out}$ (Row)
      for w = 0 to $W_{out}$ (Column)
        for c = 0 to C
          for t = 0 to T
            for r = 0 to R
              for s = 0 to S do
                Filter=F[m][c][t][r][s]
                Input=In[c][d*stride+t][h*stride+r][w*stride+s]
                Output[m][d][h][w] += Filter * Input
```

Listing 1. 3D Convolution in 3D CNN.

Thus, this paper explores the acceleration architecture of 3D CNN. 2D CNN accelerator mainly focuses on designing dataflow that exploits the spatial data reuse and across filters effectively. For 3D CNN, there is one more dimension: the temporal dimension, which indicates the data reuse across continuous frames. Since the change of consecutive frames sending to a 3D CNN are not arbitrary, there are large data overlaps, which exist in not only the first layer as continuous input frames, but also the input activations maps (imaps) within the intermediate layers of 3D NN. Towards this, we first conduct characterization and prove that: (1) there is large redundancy between imaps across the temporal dimension; (2) quantization of input activations could further increase the data overlap.

Towards this opportunity, we then leverage differential convolution across the temporal dimension, which operates on the temporal delta of imaps for each layer, and process the computation only using the effectual bits of the temporal delta. While using only the temporal delta dataflow could not benefit all layer's execution of 3D CNN, and it only benefits 3D CNN, we then propose a control mechanism to switch between spatial delta dataflow and temporal delta dataflow. We call this temporal-spatial aware accelerator (TSVA). It could not only cause higher speedup, but also makes the acceleration architecture general to all CNNs. In summary, we make the following contributions:

First, we study a very important CNN: 3D CNN for video recognition. Our characterizations show that large data overlap exists between *imaps* across the temporal dimension, and quantization of *imaps* could increase the overlap. This implies architectural optimization opportunities.

Second, we leverage the data overlap across the temporal dimension and process the computation of effectual items for 3D CNN efficiently. Then to further leverage both the spatial locality and temporal locality that makes our architecture general to all CNNs, we propose a dynamic control mechanism to switch between spatial delta dataflow and temporal delta dataflow dynamically.

Third, we evaluate across multiple 3D CNNs and some state-of-the-art 2D CNNs by comparing it with other NN accelerators [12][13][35], and show that our accelerator achieves an average of ×4.24 speedup and ×1.42 energy efficiency.

The rest of this paper is organized as follows: Section 2 describes the necessary background for 3D CNN, and characterization the data overlap. Section 3 introduces the architecture design. Section 4 introduces the benefits of our architecture towards other general 2D CNNs. Section 5 evaluates our design. Section 6 concludes the paper.

2. Background and Motivation

2.1 3D Convolution Operation

The computation of 3D CONV is more complicated than that of 2D CONV, which can be described as follows:

$$O[m][h][w][c] = \sum_{c=0}^{C}\sum_{t=0}^{T}\sum_{r=0}^{R}\sum_{s=0}^{S} F[m][c][t][r][s] * In[c][d * stride + t][h * stride + r][w * stride + s]$$

Where F and In represents the $M\times C\times R\times S\times T$ dimension filter and $C\times D\times H\times W$ dimension input activation map. In each layer, a set of C input feature maps of size $D\times H\times W$ are convolved by M sets of $C\times Ksize$ filters ($Ksize = R\times S\times T$), outputting M feature maps of size $D_{out}\times H_{out}\times W_{out}$ while D_{out}=floor((D-T)/*stride*) + 1, H_{out}=floor((H-R)/*stride*) + 1, W_{out}=floor((W-S)/*stride*) + 1. D is the temporal dimension which indicates the number of consecutive frame sequences sending to the neural network for video recognition. For example, D is 16 for C3D network for the first layer's input. 2D convolution is the special case of 3D when D equals to 1 and T equals to 1. The computational complexity of the 3D convolution is much higher than 2D convolution: $2\times M\times C\times H\times W\times R\times S$ for 2D convolution, $2\times M\times C\times D\times H\times W\times R\times S\times T$ for 3D convolution. The pseudocode of 3D CONV is shown in Listing 1 and illustrated in Figure 2 with stride equals to 1, which indicates the most common case (D_{out}=D-T+1, H_{out}=H-R+1, W_{out}=W-S+1).

The computation pattern of the FC layers in 3D CNN is a matrix-vector multiplication, which is the same as the FC layers in 2D CNN. CONV layer in 3D CNN is more computation-intensive and occupies more percentage than in 2D CNN. For example, C3D (including RELU) occupies over 99.6% of the computation time [31]. (2) the amount of intermediate data (i.e. psums) is much larger than the weights (356.7MB versus 66.5MB for C3D, even though C3D is still a small-scale network compared to other 3D CNN [36][37].

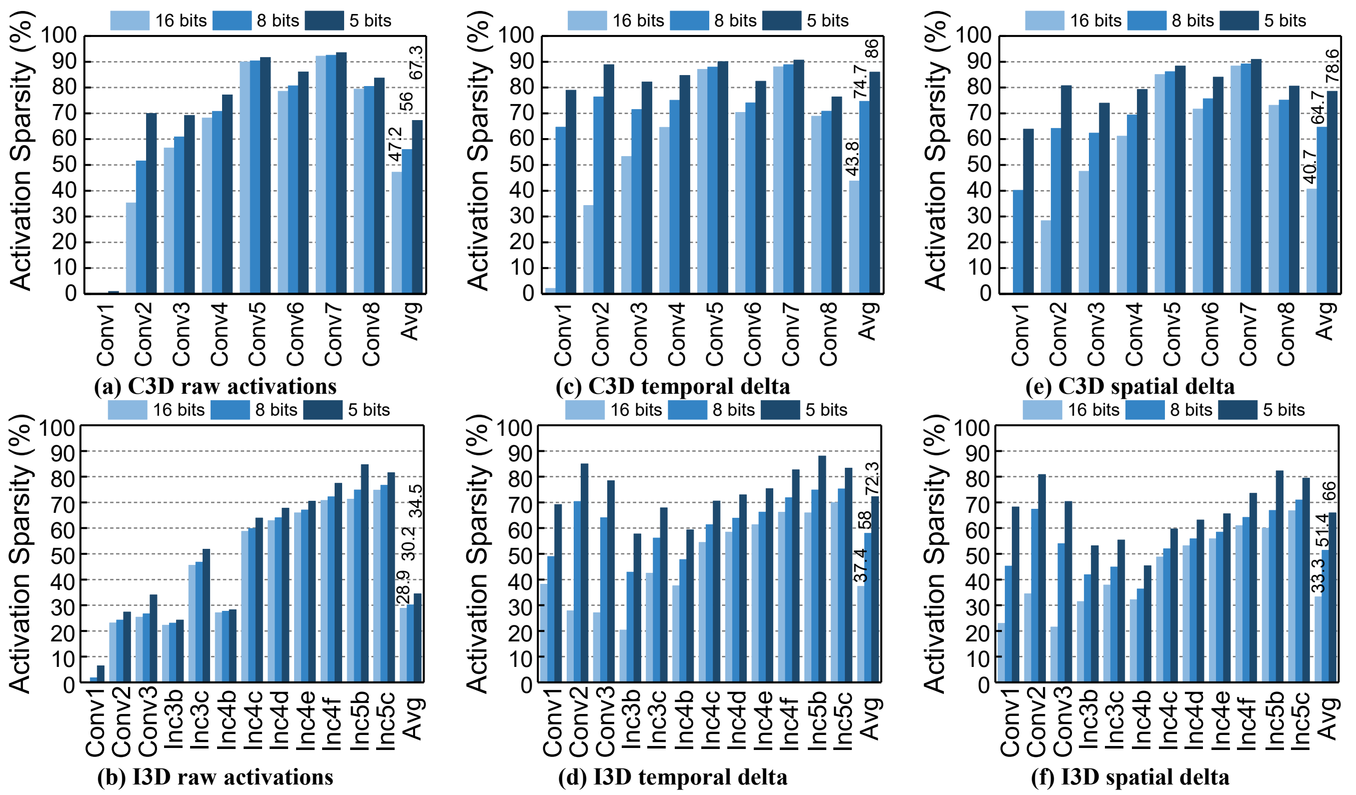


Figure 3. Sparsity of input activations (a, b), temporal delta (c, d), and spatial delta (e, f). "Avg" is the sparsity across all layers; inc stands for inception module [36]. The sparsity across all layers are denoted in the Figures.

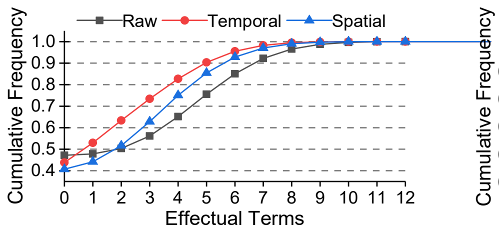
A typical CNN accelerator consists of three levels of hierarchy: (1) external memory, (2) on-chip buffers, (3) processing engines (PEs) and register files. The basic flow is to fetch data and weights from the external memory to on-chip buffer, and feed them into registers and PEs. After the PE computation completes, results are transferred back to on-chip buffers and to the external memory if necessary, which will be used as the next layer inputs.

2.2 Activation Sparsity in 3D CNN

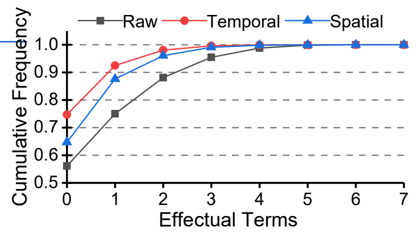
Sparsity in a layer of CNN is defined as the fraction of zeros in the layer's weight and input activation matrices. Weight sparsity mainly comes from the pruning (unstructured [38][39] or structured [40]) process during training. There are already plenty of works on weight pruning algorithms [39][41][42] and hardware accelerators [16][43][12]. Activation sparsity occurs dynamically during inference and is highly dependent on the data being processed. To measure the activation sparsity, we execute two typical 3D CNNs: C3D [23] and I3D [37]. We instrument code to inspect the activation sparsity of the CONV layers. Quantizing [44][45][46][35] the input activations within a specific range could reduce the computation overhead with the inference accuracy decreases within the minimal range. 16 bits storage of the activations is the baseline, while weights are always stored as 16 bits in our paper based on the well-recognized 16-bit multiplier [13]. Prior works have shown that linear quantization [39] to 8 bits [47][48][49][50] does not cause accuracy loss, and further quantizing the input activations to fewer bits (more or equal than 5) cause less than 1% accuracy loss. Our experiment also verified that, we run C3D with a pre-trained network model (we didn't do retraining) and then evaluate the accuracy using UCF101 [27] datasets. The accuracies are 88.3% for 8 bits, 87.7% for 5 bits (0.7% accuracy loss), and 84.5% when further quantizing to 4 bits (4.3% accuracy loss in this case), which shows that quantizing the imaps less than 5 bits could cause intolerable accuracy loss. I3D also exhibits similar trends. Thus, we choose 16, 8 and 5 bits to analyze the input activation characteristics.

Figure 3(a) and 3(b) shows the activation sparsity across the CONV layers, referenced to as the y-axis. (1) First, 3D CNN has similar activation sparsity characteristics with 2D NN [51][17]. For 16 bits, the activation sparsity is 47.2% for C3D and 28.9% for I3D. (2) Activation sparsity varies across layers, with sparsity typically low in the earlier layers and increase gradually in later layers. (3) With quantization, the activation sparsity increases from 47.2% (16 bits) to 56% (8bits), and then 67.3% (5 bits) for C3D, and 28.9% (16bits) to 30.2% (8bits), and then 34.5% (5 bits) for I3D.

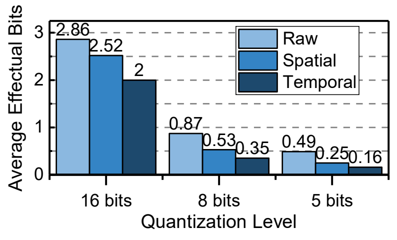
Prior works have leveraged the activation sparsity to design hardware accelerators [19][51], either by prediction-based [19] or speculation-based method [51]. However, none of them considered combining it with temporal locality: the high degree of the input activation redundancy across the temporal dimension. Since 3D CNN adds the temporal dimension D (D is 16 consecutive frames for C3D and 64 for I3D), which provides acceleration opportunities. Next, we



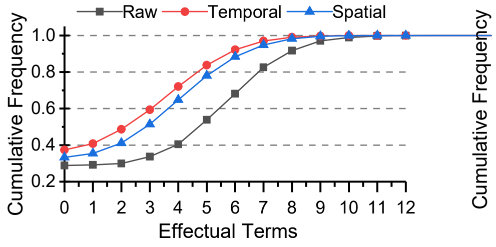
(a) C3D 16 bits quantization



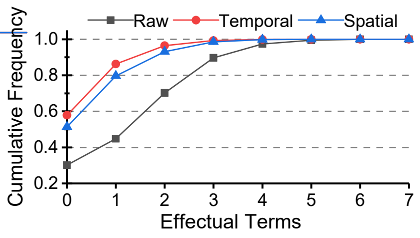
(c) C3D 8 bits quantization



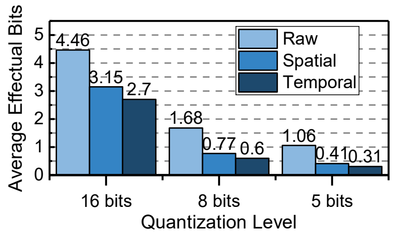
(e) C3D average effectual bits comparison



(b) I3D 16 bits quantization



(d) I3D 8 bits quantization



(f) I3D average effectual bits

Figure 4. Cumulative distribution of the number of effectual terms per activation/delta over C3D and I3D.

characterize the temporal locality and corresponding effectual bits.

2.3 Input Activation Temporal Locality

Video is a sequence of consecutive frames. There are large overlaps between these consecutive frames, which is the temporal locality. The input activations *In*[*D*][*C*][*H*][*W*] are represented as: *In*[*d*][*C*][*H*][*W*], *In*[*d*+1][*C*][*H*][*W*], *In*[*d*+2][*C*][*H*][*W*]…, *d*=(0,1,2…..*D*-1). We define the temporal delta T_delta as:

T_delta = *In*[*d*+1][*C*][*H*][*W*]-*In*[*d*][*C*][*H*][*W*]

If an element in *T_delta* is zero, the input activations in the same position of *In*[*d*+1][*C*][*H*][*W*] and *In*[*d*][*C*][*H*][*W*] are the same. Thus, the sparsity of *T_delta* represents the overlap/redundancy between consecutive input activations (*In*[*d*+1][*C*][*H*][*W*] and *In*[*d*][*C*][*H*][*W*]).We characterize the sparsity *T_delta* across all the layers, which are shown in Figure 3(c) and 3(d).

(1) First, the sparsity of temporal delta tends to be more compared to the raw activations across layers, especially when the bits are quantized smaller. Specifically, the first few layers tend to have higher sparsity (especially after quantization) comparing to raw activations, while the latter layers tend to be a little lower. Note that the reason for the low sparsity for the first two CONV layers is caused by data normalization. Removing normalization would give higher sparsity for 16 bits: 56% on average, with first two layers 56% and 49.6%; while normalization has less effect on 8- and 5-bits quantization, with 77% and 87% respectively. Second, the temporal delta sparsity dramatically increases with quantization, i.e. quantization makes consecutive imaps more redundant. Third, compared to raw activation, temporal delta tends to be sparser with quantization: 56% (8 bits) and 67.3% (5 bits) for C3D raw activations to 74.7% (8 bits) and 86% (5 bits) for C3D temporal delta for example.

Related work [52] leveraged the spatial data locality of adjacent CONV window to design DNN accelerators. We further analyze the characteristics of spatial delta (which are defined as *S_delta*=*In*[*D*][*C*][*H*][*w*+1]-*In*[*D*][*C*][*H*][*w*], *w*=(0,1,2,…..W-1).), and show that temporal delta tends to be more sparse compared to spatial delta, as shown in Figure 3(e) and 3(f).

(2) Another insight is that temporal delta needs less effectual bits than raw activations. Let's first give the definition of effectual bits: a multiplication *a*×*w* of an activation *a* with a weight *w*. If *a* is represents by *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*-th bit positions multiplicand *w*:

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

It is only those bits of a are 1 that yield effectual work. The effectual bits are defined as the bits in *a* that is 1. Figure 4(a)-(d) shows the cumulative distribution of the number of effectual terms per raw activation, temporal delta spatial delta for 16- and 8-bits quantization respectively across all neural network layers. The distribution is measured over all input data. These figures show that there is significant potential for reduction in the number of computations needed if temporal delta is processed using bit-serial multiplier [46][35] as temporal deltas contain considerably few effectual terms per value. Figure 4(e) and (f) show the average effectual bits under different quantization levels: temporal delta contains the lowest effectual bits on average, which indicates potential speedup. For example, under 8-bit quantization, average effectual bits for raw activations and temporal delta is 0.87 and 0.35 for C3D, and 1.68 and 0.6 for I3D, which indicates ×2.49 and ×2.8 potential speedup. To be specific, assuming that *a* is an activation of *In*[*d*][*C*][*H*][*W*], *a'* is the corresponding in the same position of *In*[*d*+1][*C*][*H*][*W*]. Rather than calculating *a'*×*w* directly, we could instead calculate it relative to *a*×*w:*

$a' \times w = (a \times w) + (a' - a) \times w = (a \times w) + (\Delta a \times w)$. Note that if we change *d* to *w*,

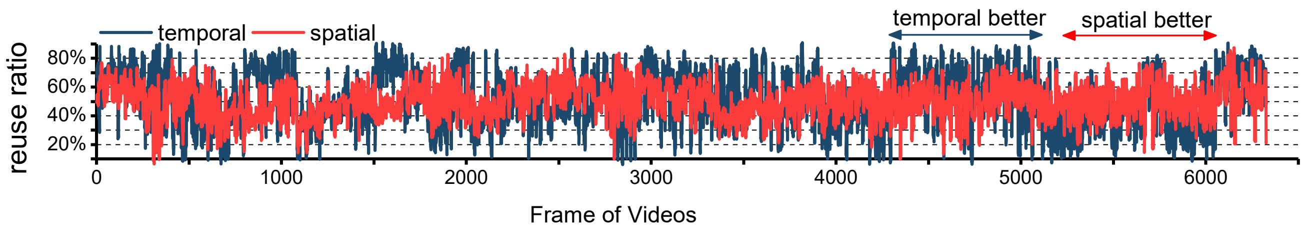


Figure 5. Reuse ratio across different input frames of UCF-101.

S_delta=*In*[*D*][*C*][*H*][*w*+1]-*In*[*D*][*C*][*H*][*w*], *w*=(0,1,2,…..W-1). This is called S-Flow, which could also be applied in 2D CONV.

(3) Third, even though on average temporal delta tends to have sparser and contains less effectual bits than spatial delta, it still varies across different inputs. We argue that dynamic control logic to change between spatial delta dataflow (S-Flow) and temporal delta dataflow (T-Flow) should be leveraged. First, figure 5 shows the computation reuse ratio across spatial dimension and temporal dimension for the different input in UCF-101 [27] datasets. As discussed previously, temporal delta on average has better sparsity and less effectual bits on average, but the benefits also vary across different inputs. Some inputs benefit more from T-Flow (most), while others benefit from S-Flow. Second, in some layers, *D* is very small (ranging from 2-4 for the last four layers of C3D). If directly using T-Flow, makes the PE along the row (as shown later in our design) underutilized, since the temporal/spatial dimension is unrolled to 8 (P_d or P_w) in our design. Thus, to maximize the benefits, a dynamic control mechanism to switch between spatial and T-Flow is needed, varying across different inputs and layers.

Prior works leveraged the temporal locality of consecutive frames in continuous mobile vision to design accelerators [53][54]. However, they do not consider the additional effectual bit savings. Besides, they only consider the temporal locality of *imaps* of the 1st CONV layer. We leveraged a tile-based computation dataflow that can process temporal delta bit-serially only on effectual bits. Besides, we leverage a dynamic control mechanism to switch between T-Flow and S-Flow to maximize the benefits. This can not only be applied to 3D CNNs, but also to other general 2D CNNs.

3. Architecture Design

3.1 Baseline Accelerator

We first describe our baseline accelerator bit-parallel value-agnostic accelerator (BVA) modeled after *DianNao* [12] and *DaDiaNao* [55]. We choose this because this is a well-understood design and inspires many new accelerators later. Figure 6 shows a tile of BVA, which has a weight buffer (WB) which provides 16×16 weights per cycle, one per weight lane. The tile also has an input activation memory (NB_{in}) which provides 16 neurons per cycle, one per activation lane, and an output activation buffer (NB_{out}) which can accept 16 output activations per cycle. We use P_c and P_m to denote the number of input activations and weights to be processed in each cycle, P_c=16, P_m=16 here in each tile. In each cycle, 16 activations are broadcasted to 16 filter lanes. Only one AM slice (16-activation slice) operates per cycle and all tiles see the same set of 16 activations. The neuron buffer (NB) is single-ported and banked, which is the last-level on-chip SRAM. Half of the banks are used for the *imaps* and the other half for the *omaps*. Both activations and weights are read and written to the off-chip memory. The number of filters (P_m), tiles (Figure 6 is just one tile), weights per filter (P_c), precision *B* (Figure 6 shows the precision as 16-bit) are all design parameters that are configurable as necessary.

BVA features a dataflow that performs P_c input feature maps and P_m output feature maps in parallel. At each clock cycle, it handles P_c imaps and P_m omaps, one activation of each output feature map, and one weight of each kernel. It is used not only in *DianNao* [12] and *DaDianNao*, but also in many state-of-the-art FPGA accelerators [56][57][58]. We name this as the MIFM-MOFM dataflow.

For 3D convolution, there are some challenges when directly applying the baseline accelerator: (1) Put every data including *imaps*, *omaps*, and weights on chip is infeasible since 3D CNN has two more dimensions that make the working dataset exceed the on-chip memory easily. Actually, this is a problem for not only 3D CNN, but also 2D CNN [59]. Loop tiling should be leveraged. (2) *DaDianNao* [55] is based on the premise that the storage needed for the input activations are far less than the weights. However, for 3D CNN, this is not the case. For example, the storage requirements of input activations exceed the weight size for the first four CONV layer. Memory fragmentation overheads will be larger if applying a uniform on-chip buffer design and partition. (2) Since this dataflow is value agnostic, there is no way to leverage the temporal dimension for 3D CNN, and the

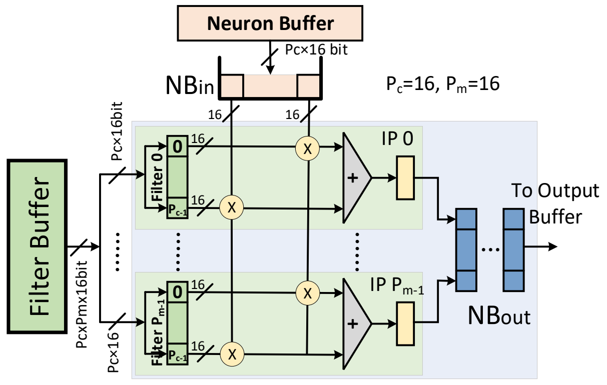


Figure 6. Baseline bit-parallel value-agnostic accelerator.

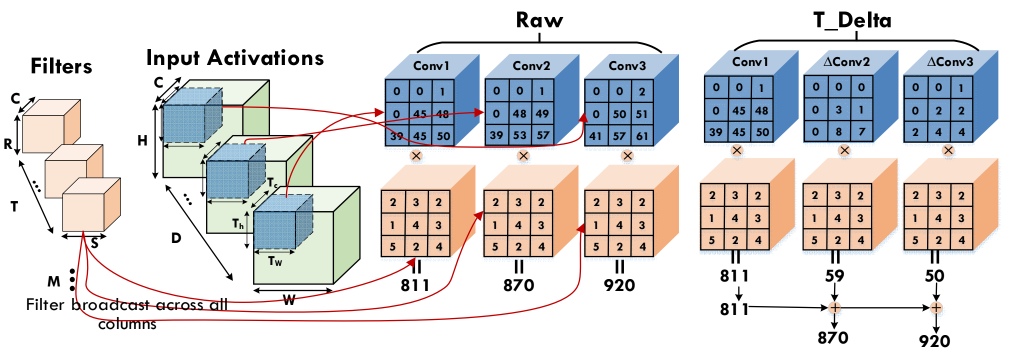


Figure 7. Differential convolution of the temporal delta output activation propagating across the temporal (D) dimension from one column to the next, each column corresponds to one *i*. *i*=0,1,2…*D*-1.

effectual bits reduced by the temporal delta. These factors are needed to be considered for accelerator design.

3.2 Delta Value-Aware Accelerator

In Section A-E, we first discuss our design using T-Flow, and later we add the design about flexibility.

A. Architectural Factors to be Considered

Loop Tiling. Since the on-chip buffer is not enough to store the feature maps and weights for 3D CNN. The efficient way should be to tile the data so that each tile fits the on-chip buffer. We use T_x to represent the tile of a parameter X. When the *imap* does not fit on chip, the input activation of [C, D, H, W] should be broken into tiles of [T_c, T_d, T_h, T_w]. When not all M filters fit in on-chip memory, the [M, C, T, R, S] should be broken into tiles of [T_m, T_c, T, R, S]. T, R, and S are generally small values (that ranges between 1 and 7 [23][36]) and not considered for tiling. Since Section 2 already characterizes that the data reuse along the D dimension is high. To ensure the high performance along with the temporal delta calculation, we want the data along the D dimension to fit into the on-chip buffer as large as possible. For example, if the input activation buffer is 108 KB (which the total SRAM of Eyeriss [15]). CONV2 layer in C3D has the largest input activation size: C=64, D=16, H=56, W=56 (activations stored using 1 Byte), choosing a tile factor of T_c=64, T_d=16, T_h=10, T_w=10 is more favorable than T_c=64, T_d=4, T_h=20, T_w=20. We cannot store all the input activations and weights on-chip. As discussed previously, we opt T_d as close as D. While the first tile of $In[T_h][T_w]$ and the next tile have overlap along H-dimension or W-dimension (referred as "data halo" [60]), we opt to take advantage of the overlap and do not re-fetch the overlapped region in the H- and W-dimension.

Loop Unrolling/Parallelization. While tiling ensures loading the data on-chip partially with better data reuse, loop unrolling ensures some dimension can be spatially expanded across PEs to ensure the parallelizing multiply-and-adds. We leverage MIFM-MOFM dataflow that unrolls along the C-dimension and M-dimension, and extend it by unrolling along the D-dimension with unroll factors P_c, P_m, and P_d respectively. T_c, T_m, T_d, T_h, T_w denotes the tiling factor to define the on-chip buffer size, P_c, P_m, and P_d are not necessarily equal to T_c, T_m, and T_d. But we are sure: $P_c \le T_c, P_m \le T_m, P_d \le T_d$. For example, as discussed in Section A, T_d should be as close as D to maximize temporal reuse. However, P_d should not since D ranges as long as 64 in I3D, if we choose the largest one, there will be no design space for P_c and P_m due to area limitation. As for P_c and P_m, *DianNao* [12] has P_c=P_m=16 with area 3.02 mm^2, *DaDianNao* [13] has P_c=16, P_m=256 with area of 67.7 mm^2 in total. These parameters can be configurable based on chip area, design purposes, and using scenarios.

B. Temporal Delta Convolution

After tiling and unrolling, we consider the temporal delta convolution on-chip: for a given output activation $out(m, d, h, w)$, it is possible to compute $out(m, d+1, h, w)$ differentially using the equation as follows:

$$out(m,d+1,h,w) = out(m,d,h,w) + CONV(filter[m], \Delta In)$$

ΔIn refers to the element-wise deltas of the *imap* windows corresponding to $out(m,d+1,h,w)$ and $out(m,d,h,w)$:

$$\Delta In(m,k,i,j) = In(c, k+stride*(d+1), i+stride*h, j+stride*w) - In(c, k+stride*d, i+stride*h, j+stride*w)$$
$$c = 0,1,2\ldots.C$$

stride is the stride between two adjacent *imap* window. Figure 7 shows an example of differential convolution, which applies a 3×3 on three convolution windows along the temporal dimension (D). While all columns on the T-Flow architecture shares the same filters, each column processes one convolution activation window along the D-dimension. Direct convolution directly applies to the raw activations; however, differential convolution only computes the raw activations for the first window and temporal deltas on the rest. All three convolution windows are computed concurrently. Then the differential convolution is constructed in a cascaded fashion, as shown in the rightmost part of the figure. First, the $\Delta CONV2$'s output activation is calculated as 59, and the $\Delta CONV3$'s output activation is calculated as 50. The output activation of CONV2 and CONV3 is calculated by adding 811+59=870, and 870+50=920.

C. Basic Processing Engine (PE)

The basic PE builds upon the *Bit-Pragmatic* accelerator (PRA) [35], which processed only effectual bits of input activations bit-serially, as shown in Figure 8. After loading P_c×8 bits (P_c=16, activations stored as 8bits) input activations,

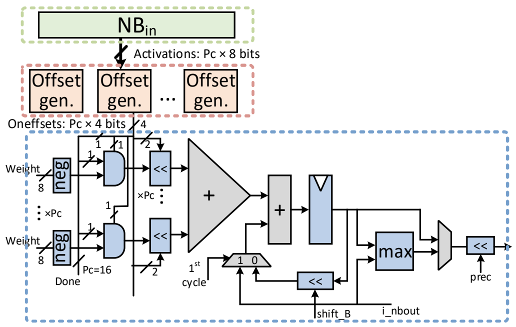


Figure 8. Basic bit-serial PE.

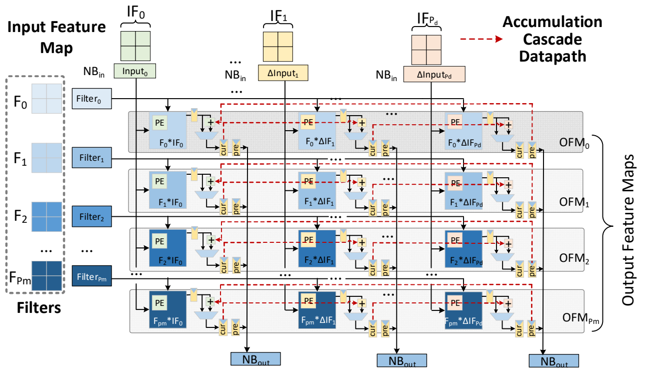


Figure 9. Scale-out USPE array, one Processing Unit (PU).

the offset generators convert the activations into a stream of effectual powers of two after applying a modified booth coding. PRA multiplies a weight of the power of two or *oneffset* each cycle using a shifter. To rate match the throughput of bit-parallel baseline, we unroll P_d to 8 since the activations are stored in 8-bit. In that way, $P_c \times P_d$ activations can be computed concurrently in multiple bit-serial cycles. PRA uses an optimized two-stage shifter scheme, during which each *oneffset* consists of 4 bits: 2-bit for the one-offset, 1 sign bit and 1 valid bit. Each bit-serial PE has P_c-input adder tree and P_c-shifters (instead of multipliers). Each of them shifts the weight directed by the offset. Multiplying activations by filters is implemented by feeding the activations bit-serially using the generated *oneffsets*, and weights bit-parallelly.

D. Scale-out Design

We present the scale-out design as shown in Figure 9, where the PE is duplicated in two dimensions to improve the processing throughput. Since PE uses a bit-serial multiplier to perform multiplications only on the effectual bits, it takes multiple cycles to accomplish one multiplication. Even though the average is only 2-3 cycles, in the worst case, it can still take up to 8 cycles (when one input activation is 11111111) to calculate one product. To maintain the comparable performance of the baseline bit-parallel accelerator, we need to simultaneously process P_d output activation (only the first one is raw output activation, the rest are output activation of the temporal delta) along the *D*-dimension. As shown in Figure 9, the vertical parallelism granularity is the number of *omaps* P_m, the horizontal parallelism granularity is P_d. The parallelism of P_c has already been discussed in Section C, which is hidden inside the PE itself. The red arrow shows the cascaded accumulation along the temporal dimension.

E. T-Flow

The parallelism of the temporal dimension offers a multitude of options for processing neurons in parallel. We opt to process P_d windows in parallel using a neuron brick to from the window in the same row and column pointwise, so that the accelerator can process $P_c \times P_m$ output neurons in parallel. For example, for a layer with stride 1 in the *D*-dimension, the accelerator can process P_d neuron bricks[1] $InB(c, d, h, w)$, $InB(c, d{+}1, h, w) \ldots InB(c, d{+}P_d{-}1, h, w)$. Where each column processes the same convolutional window along the *D*-dimension. The first CONV window is calculated using the raw activations, while the rest along the column is computed using the temporal delta. We conservatively unroll P_d to 8 to compensate for the cycle loss by bit-serial processing.

In this architecture, we only need to calculate the first imaps in the *D*-dimension and the remaining outputs along *D* differentially. We do this so that we can buffer the data along *D* completely using the height T_h and width T_w. Once all the input activations along $T_h \times T_w$ are consumed, the next sets of $T_h \times T_w$ input activations are loaded on-chip. The on-chip SRAM is double-buffered so that the on-chip MAC computation can be pipelined with the loading of the next set of data from off-chip.

The accelerator computes each output in the temporal dimension in two phases: (1) first, T-Flow calculates the first imap using raw data in parallel with computing the rest using the temporal delta; (2) second, the data are propagated in cascaded fashion with just a single addition per output. Since the first phase needs tens of hundreds of cycles, the second phase just takes several cycles to finish using a set of adders. The second phase can pipeline with the computation of the next set of windows along the temporal dimension. The data path of the second phase for cascaded accumulation is shown in Figure 9 as the red arrow.

We use the scheme of store "the first column input activations as raw, and the rest as deltas" way. Since storing all values along the *D*-dimension as raw activations incur recomputing overhead every time activations are read out. Besides, this can save additional off-chip memory traffic. Each column's NB_{out} refers to the output neuron brick of next layer. We use a SIMD engine, $Delta_{out}$, containing (1) a P_d-to-1 multiplexer, (2) ALUs to perform activation function to generate the activation deltas of the next layer, and (3) output activation neuron brick buffer. Computing the delta bricks of the next layer contains two phases: (1) read out the output bricks to the stride ($Stride_{next}$) left to NB_{out}, going through the NB_{out}, and store them into the output brick buffer of the

[1] The neuron brick $InB(c, d, h, w)$ equals to $InB(c, d, h, w)$, $InB(c{+}1, d, h, w) \ldots InB(c{+}P_c{-}1, d, h, w)$.

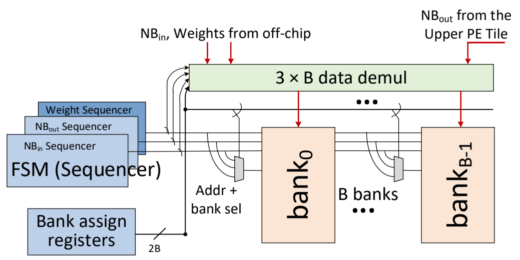


Figure 10. Configurable last-level SRAM, on-chip buffer. Blue blocks are programmable. The bank assign logic outputs a 2B-bit vector to indicate whether each bank belongs to input, output or weight.

engine; (2) reading the output brick of current column's NB$_{out}$, passing through the activation function, and use the SIMD-adder to perform element-wise subtractions to generate delta of next layer, and then store them into the last-level on-chip SRAM. The delta values can be stored as dynamic precisions [61] of groups to reduce off-chip energy.

Supporting S-Flow: note that all the above discussions from Section A-E refer to temporal delta dataflow. To support spatial delta dataflow, we just need to replace all the above discussions relating to d to w. That is, processing the spatial delta (instead of the temporal delta) of different input CONV window along W-dimension. The loop unrolling fac tor is P_c, P_m and P_w. P_d is replaced with P_w, which both indicates the PE columns in the scale-out design in Figure 9. Next, let's talk about configurability.

F. Configurability

Configurable last-level on-chip SRAM: The goal of providi ng buffer configurable on the last level on-chip SRAM is to consider that the optimal tile size (input activations, output activations, and weights) varies across layers. I.e. Since the on-chip SRAM is fixed after design decision is made, for one layer, the on-chip buffer should be allocated more to the activations and less to the weights, and for other layers, the on-chip buffer should be allocated more to the weights and less to the activations. Configurability allows that and minimize the internal fragmentation. The design is only used in the last-level SRAM. Banks are allocated to each data type contiguously, and "Bank Assign" registers are configured at each layer start time to denote the range of banks used to store each data type. The parallel demultiplexer is used to index into and read/write data into/out of each group of banks. The output (read) mux is replicated for each of the three types reads one word per access, there are no bank conflicts. Programmable FSMs (Sequencers) are used to generate address patterns into each group of banks. High-order address bits, along with the bank assignment registers, determine which bank is responsible for each type. We configure this with 16 banks, and banking does not show very little overhead [62]. The traffic between the last-level SRAM and the tiles of PUs (as shown in Figure 9) are implemented through NoC. The NoC manages the data delivery between buffer and PE arrays. It is implemented using broadcast style network.

Dynamic switch between S-Flow and T-Flow: The difference between T-Flow and S-Flow [52] is: (1) the former is unrolled into P_c, P_m, and P_d (current number of frames unrolled in temporal dimension), the latter is also unrolled into P_c, P_m, and P_w (concurrent number of adjacent CONV window unrolled in spatial dimension W). (2) The former opts to accommodate enough *imaps* across the D-dimension by making T_d as large as possible; while the latter accommodates enough *imaps* by making T_w as large as possible.

The priority of loading data on-chip along which dimension first is decided by the control logic of the on-chip buffer, which is decided by the loop order [d, h, w, m, c]. Thus, the control logic of the on-chip buffer decides either the order of loading which on dimension first complete on-chip.

Loop order determines the sequential data loading order of the CONV loops. There are two kinds of loop order: intra-tiling and inter-tiling loop orders. Inter-tiling loop orders determine the data movement from off-chip memory to on-chip buffer. The intra-tiling loop order determines the pattern of data movement from the on-chip buffer to PEs. We consider the inter-tiling loop orders, which indicates the dynamic FSM (sequencer) in Figure 10. To dynamically switch between S-Flow and T-flow, we leverage a dynamic FSM to generate addresses into the on-chip buffer.

The sequencer is used to generate address into the on-chip buffer, count how many MACCs to perform, and when to read/write psums relative to performing MACs, when processing all tiles are done. When the loop order of loading data on-chip changes, loop bounds and memory access to each tile changes. To ensure the control flexibility, the FSM is programmed by setting two sets of configurable registers that denote the loop bounds and loop steps and use loop counter to iterate between the specified loop orders. I.e. the loop bound for [D, H, W, M, C] is [0->D, 0->H, 0->W, 0->M, 0->C], while the loop step is [T_d, T_h, T_w, T_m, T_c]. The FSM walks through the loop using the loop bounds and accumulates a step into an output register. For the L-level (L=5) loop, the user specifies bounds $b_0,\ldots b_{L-1}$, and steps $s_0,\ldots s_{L-1}$. And the iteration indexes are $i_0,\ldots,i_{D-1}$. The data loading order behaves like the software iterations. When entering each state, the FSM outputs the current value in the output register and one of the steps s_j is added to that register for the next iteration (similar to [32][63]). j equals to which loop is currently terminating. Different loop orders and steps give different address sequences need by the on-chip buffer.

There are two different kinds of loop orders and steps to choose for the registers, which are determined by the S-Flow and T-Flow. We leverage the offline software optimization framework [64] that pre-analyzes 3D CNNs and finds the optimal tiling and inter-tiling loop order of S-Flow and T-Flow, with S-Flow has restrictions that W=T_w, T-Flow has restrictions that D=T_d. We use two signals: "temporal_signal" and "temporal_flow" signal to choose whether to use T-Flow or S-Flow. Before entering the CONV layer, the input is first profiles to compare the reuse ratio of spatial delta and temporal delta: if the temporal delta is sparser, "temporal_signal" is set on, else set off. Then for each layer, if temporal_signal is set on, and D>=the number of PE columns, temporal_flow signal is set on and set the optimal T-Flow's loop order and tiling in the on-chip buffer's sequencer, else always set temporal_flow signal to off, indicating to use S-Flow and set the optimal S-Flow's loop order and tiling in the sequencer. With this control flow, we ensure the optimal off-chip loading energy efficiency.

The temporal_flow signal also controls the unrolling factors. The computation order for S-Flow is p_w->p_m->p_c, and for T-Flow is p_d->p_m->p_c. To achieve that, we also change the data address generation pattern from the on-chip buffer to PEs. For T-Flow, different columns of PEs in the scale-out design are loaded with the inputs corresponding to different CONV windows in the *D*-dimension; for S-Flow, different columns of PEs are loaded with the inputs corresponding to different CONV windows in the *W*-dimension. Besides, 2D CONV is directly supported using S-Flow by setting temporal_flow to off.

Note that under our control flow, only two cases can happen for all layer's execution: (1) the former layers use T-Flow, the latter layers use the S-Flow. In this case, during the layer that switches from T-Flow to S-Flow in the next layer, the spatial delta value of the next layer needs to be computed separately. (2) all layers use S-Flow. In this case, all inputs are stored as spatial delta.

4. Benefits to Other CNNs

Our TSVA dataflow could not only benefit 3D CNNs, but also 2D CNN inferences. First, for those workloads running under continuous mobile vision scenario, pixel changes across continuous frames are not arbitrary. Also, in most cases, several continuous frames contain the same objects to be detected. Thus, it is possible to send multiple frames into neural networks and fuse several imaps (with *D* larger than 8) running concurrently into the neural networks using T-Flow if the users care more about throughput instead of latency. In that case, lots of redundant computations can be saved. Based on the statistics published by Google, at least 29% of Google's data center workloads are sequence processing. Besides, S-Flow can be directly applied to 2D CNN.

5. Evaluation

This section evaluates the performance and energy consumption of TSVA. First, we present our evaluation methodology. Second, we present our speedup by TSVA. Then, we present the area information, power breakdown, energy efficiency, and off-chip memory energy savings.

A. Evaluation Methodology

We have developed a cycle-accurate simulator to model the performance of all architectures. We model an accelerator of four tiles. The designs are implemented in Verilog and synthesized through the Synopsys Design Compiler [65], with the TSMC 65nm library. We use CACTI [62] to model the area and power consumption of the on-chip SRAM memories. The accelerator frequency is set at 1GHz. We model a DRAM with 4GB LPDDR4-2400 channels, the DRAM energy is estimated using information in Micron technote [66]. We

Table I. Parameters for the accelerator

# of Tiles	4
SRAM Banks	64KB×16
Technology	65nm
Off-chip Memory	4GB LPDDR4-2400
Filters/Tile	16
Weights/Filter	16
Frequency	1GHz
NB_{in}/Tile	2KB
NB_{out}/Tile	2KB

compare our design with state-of-the-art schemes, including the BVA tile design by *DianNao* [12] and *DaDianNao* [13], bit-serial accelerator *Bit-pragmatic* [35]. Since our accelerator only targets inference, as prior work claimed that 8 bits is already good enough for inference [50]. We store the weights/activations using 8 bits, both for the baseline and our design. We further compare the speedup when the activations are quantized to 5 bits.

The hardware unrolling parameters for the baseline BVA design is 4 PU tiles with each P_c=16, P_m=16. To match the performance, for bit-serial design, we choose a 4 PU tiles with each PU's P_c=16, P_m=16, with P_d=8. There are total 64KB×16 banks, which works as the last-level on-chip SRAM to store input, output, and weights. The data dispatcher, offset generator, engine for calculating and cascading delta value, and control logic also has some overhead that need to be considered. The default configurations are reported in Table I.

B. Evaluated Networks

We evaluated the following 3D CNNs based on their popularity and state-of-the-art research: (1) C3D [23], one of the most widely used 3D CNN for video recognition by Facebook. To evaluate C3D, we use videos from UCF101 [67] benchmark suite; (2) I3D [36], since it currently holds state-of-the-art results on Kinects [36] video dataset. (3) 3D ResNet-50 [37], a 3D version of ResNet-50. We also use UCF101 dataset for 3D Resnet-50; (4) YOLO [68] and SSD [69], two most widely used object detection neural networks. The public object detection benchmarks (Pascal VOC or COCO datasets) cannot be used since they mainly contain standalone images. Instead, we capture a series of videos and extract image sequences. Each image is manually annotated with bounding boxes and labels. We fuse 8 frames as a batch for every layer's execution. (i.e. *D*=8, *T*=1).

C. Speedup

We first compare the speedup of TSVA with the baselines. Note that the baseline BVA also use 8-bit multiplier. The bit-serial always unroll and processes different neuron bricks along the *W*-dimension (P_w). The speedup is shown in Figure 11, which mainly demonstrates the speedup caused by reduced effectual bits processed. On average, the speedup is 1.35× towards bit-serial, and 4.24× towards BVA. The speedup is caused by the dynamic control between spatial delta and temporal delta to maximize the delta's effectual bit saving in computation. To further shows the benefits of TSVA, we quantize the input activations to 5-bits to consider

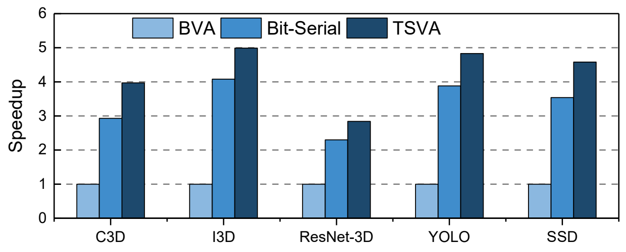


Figure 11. Speedup of TVSA.

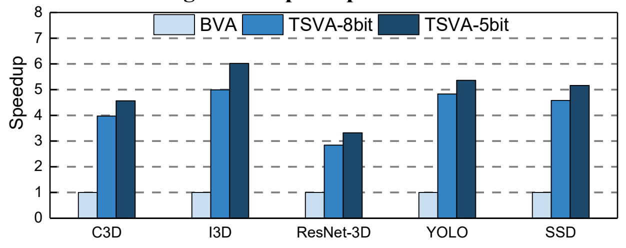


Figure 12. Speedup with 5-bit quantization enabled.

the speedup with a little loss of NN accuracy. As shown in Figure 12, it further brings a speedup of an additional 15.3% using the same architecture. The additional speedup is caused by the decreased effectual bits saving by quantization.

E. Area and Energy Efficiency

We also evaluate the area information of TSVA. The re sults are shown in Figure 13. Since we leveraged an 8-bit par allel multiplier instead of a 16-bit multiplier in the baseline, the PE area greatly decreases and consumes much less area percentage than the on-chip SRAM subsystem. The con trol logic for S-Flow and T-Flow switching added is not significant, comparing to the computation PE and the SRAM on-chip buffer. Overall, the area is ×1.64 more than the BVA accelerator, mainly increased by the computation logic. TSVA is 22.3 mm^2, while BVA is 13.58 mm^2 in total.

Figure 14 reports a breakdown of the power for the TSVA and BVA. While TSVA consumes more power than BVA (around ×2.98), the speedup is higher than the increase in power consumption, which results in an overall ×1.42 more energy efficient than the baseline. Note that this is only the on-chip computation energy consumption, the off-chip memory transmission consumption is orders or magnitude than on-chip memory accesses.

While storing the deltas using dynamic precisions can save the off-chip DRAM access energy greatly, we also show that our configurable buffer that determines the optimal loop order could save another 38%, 33%, and 27% of off-chip DRAM access energy for the three 3D CNNs respectively.

6. Related Work

Neural network accelerators. The huge computational requirements and applicability of deep neural networks have prompted researchers to design numerical accelerators, ei ther on FPGA [70][57] [56][58][71], ASIC [12][13][14][72] [15] [50][60], or leveraging efficient memory technologies [73][74]. The representative works are *DianNao* [12], *DaDianNao* [13] utilize large global buffer as shared storage to minimize the DRAM access energy consumption. *Eyeriss* [15] proposes a row stationary dataflow by exploiting local data

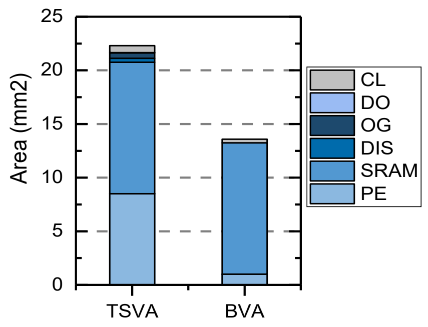


Figure 13. Area breakdown [mm^2] TSVA vs BVA. CL: control logic, DO: Delta$_{out}$: the SIMD engine for delta calculation of net layer., OG: Offset generator, DIS: Dispatcher.

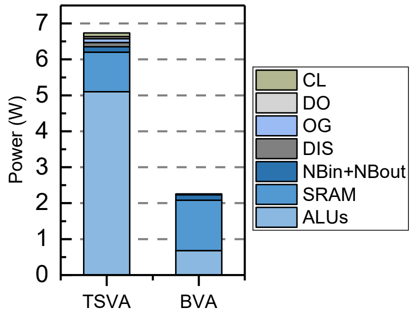


Figure 14. Power breakdown. CL: control logic, DO: Delta$_{out}$, OG: Offset generator, Dis:Dispatcher.

reuse for both filters and input activation maps. Several recent works that leverage bit-serial multiplier [46][35][19] and bit-serial cache [75].

3D CNN accelerators. Prior work on 3D CNN acceleration also lies in either FPGA [31][76], or ASIC [32]. Hedge at al. [32] designed a flexible 3D CNN accelerator that features flexible buffer and loop control. We believe that bit-serial processing will benefit more.

Accelerators that exploiting spatial and temporal locality. There are several research works that leverage the spatial [52] and temporal locality [53][54][45] to accelerate the execution of CNN. *Diffy* takes advantages of the spatial correlation between adjacent convolution windows and introduces "Differential Convolution" architecture and store the spatial delta values both off- and on-chip reducing the amount of storage and communication needed. We can leverage the benefits of both the spatial and temporal delta. Zhu et al. [53]leverage motion information of continuous frames using motion estimation to improve the execution of DNN. The same idea was proposed [54] by implementing an embedded vision accelerator as a co-processor for DNN at the same time. Riera et al. [45] propose an input reuse design that can leverage the temporal locality within layers. Our design is different since we only need to cache the input and output activations of one layer, instead of all layers.

7. Conclusion

In this paper, we propose TSVA, an accelerator for 3D CNN and other NNs under continuous frame batch processing scenario. The evaluation shows that TSVA achieves ×4.24 speedup and ×1.42 energy efficiency. While the video based inference is more and more important in people's daily life, we believe that our work will benefit more in the future.

ACKNOWLEDGEMENT

We thank all the anonymous reviewers for invaluable and insightful comments to make this paper better. This work is supported in part by NSF grants 1900713, 1822989, 1822459, 1527535, 1423090, and 1320100.

References

[1] Gordon E. Moore. Cramming more components onto integrated circuits. Proc. IEEE, vol. 86, no. 1, pp. 82–85, 1998.

[2] Robert Dennard, Fritz Gaensslen, Wha-Nien Yu, Leo Rideout, Ernest Bassous, and Andre Le Blanc. Design of Ion-Implanted MOSFET ' S with Very Small Physical Dimensions. JSSC, vol. 9, no. 5, pp. 257–268, 1974.

[3] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In ISCA, 2011.

[4] Mingcong Song, Yang Hu, Huixiang Chen, and Tao Li. Towards Pervasive and User Satisfactory CNN across GPU Microarchitectures. In HPCA, 2017.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In NIPS, 2012.

[6] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick Legresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. In ICML, 2016.

[7] Andrej Karpathy and Thomas Leung. Large-scale Video Classification with Convolutional Neural Networks. In CVPR, 2014.

[8] Kevin Hsieh. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In OSDI, 2018.

[9] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable Adaptation of Video Analytics. In SIGCOMM, 2018.

[10] Shih-chieh Lin, Yunqi Zhang, Chang-hong Hsu, Matt Skach, E Haque, Lingjia Tang, and Jason Mars. The Architectural Implications of Autonomous Driving: Constraints and Acceleration. In ASPLOS, 2018.

[11] Mariusz Bojarski, Davide Del Testa, Prasoon Goyal, Jiakai Zhang, Daniel Dworakowski, Lawrence D Jackel, Bernhard Firner, Mathew Monfort, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars, CoRR abs/1604.07316, pp. 1–9, 2016.

[12] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In ASPLOS, 2014.

[13] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, and Ninghui Sun. DaDianNao: A Machine-Learning Supercomputer. In MICRO, 2014.

[14] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. ShiDianNao: Shifting Vision Processing Closer to the Sensor. In ISCA, 2015.

[15] Yu-hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In ISCA, 2016.

[16] Mark A Horowitz, William J Dally, and C V May. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In ISCA, 2016.

[17] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In ISCA, 2017.

[18] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-Neuron-Free Deep Convolutional Neural Network Computing. In ISCA, 2016.

[19] Mingcong Song, Jiechen Zhao, Yang Hu, Jiaqi Zhang, and Tao Li. Prediction based Execution on Deep Neural Networks. In ISCA, 2018.

[20] Mingcong Song, Jiaqi Zhang, Huixiang Chen, and Tao Li. Towards Efficient Microarchitectural Design for Accelerating Unsupervised GAN-Based Deep Learning. In HPCA, 2018.

[21] Deep Learning for Videos: A 2018 Guide to Action Recognition: http://blog.qure.ai/notes/deep-learning-for-videos-action-recognition-review.

[22] Karen Simonyan and Andrew Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. In NIPS, 2014.

[23] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3D convolutional networks. In ICCV, 2015.

[24] Shuiwang Ji, Ming Yang, and Kai Yu. 3D Convolutional Neural Networks for Human Action Recognition., TPAMI, vol. 35, no. 1, pp. 221–231, 2013.

[25] Nicolas Ballas, Hugo Larochelle, and Aaron Courville. Describing Videos by Exploiting Temporal Structure. In ICCV, 2015.

[26] Yi Zhu, Zhenzhong Lan, Shawn Newsam, and Alexander Hauptmann. Hidden Two-Stream Convolutional Networks for Action Recognition. In CoRR abs/1704.00389, 2017.

[27] Khurram Soomro, Amir Roshan Zamir, Mubarak Shah, and Action Recognition. UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild, CoRR abs/1212.0402, no. November, 2012.

[28] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, Juan Carlos Niebles, and Universidad Norte. ActivityNet: A Large-Scale Video Benchmark for Human Activity Understanding. In CVPR, 2015.

[29] Mathew Monfort, Bolei Zhou, Sarah Adel Bargal, Alex Andonian, Tom Yan, Kandan Ramakrishnan, Lisa Brown, Quanfu Fan, Dan Gutfruend, Carl Vondrick, and Aude Oliva. Moments in Time Dataset : one million videos for event understanding, CoRR abs/1801.03150, pp. 1–11, 2018.

[30] Hongxiang Fan, Ho-cheung Ng, Shuanglong Liu, Zhiqiang Que, Xinyu Niu, and Wayne Luk. Reconfigurable Acceleration of 3D-CNNs for Human Action Recognition with Block Floating-Point Representation. In FPL, 2018.

[31] Junzhong Shen, You Huang, Zelong Wang, Yuran Qiao, Mei Wen, and Chunyuan Zhang. Towards a Uniform Template-based Architecture for Accelerating 2D and 3D CNNs on FPGA. In FPGA, 2018.

[32] Kartik Hegde, Rohit Agrawal, Yulun Yao, and Christopher W Fletcher. Morph: Flexible Acceleration for 3D CNN-based Video Understanding. In MICRO, 2018.

[33] Samuel Williams, Andrew Waterman, David Patterson, Soda Hall, and U C Berkeley. Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures, Commun. ACM, vol. 52(4), no. April, pp. 65–76, 2009.

[34] Visual Networking Index, Cisco Vni, Cisco Vni, and V N I Forecast Highlights. Cisco Visual Networking Index: Forecast and Methodology, 2016–2021, 2017.

[35] Jorge Albericio, Patric Judd, Alberto Delmas Lascorz, Sayeh Sharify, and Andreas Moshovos. Bit-Pragmatic Deep Neural Network Computing. In MICRO, 2017.

[36] Andrew Zisserman. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In CVPR, 2017.

[37] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet? In CVPR, 2018.

[38] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. In NIPS, 2015.

[39] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In ICLR, 2016.

[40] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning Structured Sparsity in Deep Neural Networks. In NIPS, 2016.

[41] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism. In ISCA, 2017.

[42] Xingyu Liu, Jeff Pool, Song Han, and William J. Dally. Efficient Sparse-Winograd Convolutional Neural Networks. In ICLR, 2018.

[43] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-X: An Accelerator for Sparse Neural Networks. In MICRO, 2017.

[44] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. Reduced-Precision Strategies for Bounded Memory in Deep Neural Nets, CoRR abs/1511.05236, 2015.

[45] Antonio González Marc Riera, Jose-Maria Arnau. Computation Reuse in DNNs by Exploiting Input Similarity. In ISCA, 2018.

[46] Patrick Judd, Jorge Albericio, and Andreas Moshovos. Stripes: Bit-Serial Deep Neural Network Computing. In MICRO, 2016.

[47] K Guo, L Sui, J Qiu, S Yao, S Han, Y Wang, and H Yang. Angel-Eye: A Complete Design Flow for Mapping CNN onto Customized Hardware. In ISVLSI, 2016.

[48] Young H Oh, Quan Quan, Daeyeon Kim, Seonghak Kim, Jun Heo, Sungjun Jung, Jaeyoung Jang, and Jae W Lee. A Portable, Automatic Data Quantizer for Deep Neural Networks. In ICS, 2018.

[49] S Migsacz. 8-Bit Inference With Tensort. In GPU Technology Conference, 2017.

[50] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In ISCA, 2017.

[51] Vahideh Akhlaghi, Amir Yazdanbakhsh, Kambiz Samadi, Rajesh K Gupta, and Hadi Esmaeilzadeh. SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks. In ISCA, 2018.

[52] Mostafa Mahmoud, Kevin Siu, and Andreas Moshovos. Diffy: a D ´ ej ` a vu-Free Differential Deep Neural Network Accelerator. In MICRO, 2018.

[53] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision. In ISCA, 2018.

[54] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. EVA^2: Exploiting Temporal Redundancy in Live Computer Vision. In ISCA, 2018.

[55] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. Cambricon: An Instruction Set Architecture for Neural Networks. In ISCA, 2016.

[56] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing CNN Accelerator Efficiency Through Resource Partitioning. In ISCA, 2016.

[57] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In FPGA, 2015.

[58] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-Layer CNN Accelerators. In MICRO, 2016.

[59] Doug Burger, Stephen W. Keckler, Kathryn S. McKinley, Mike Dahlin, Lizy K. John, Calvin Lin, Charles R. Moore, James Burrill, Robert G. McDonald, and William Yoder. Scaling to the end of silicon with EDGE architectures. IEEE Comput., vol. 37, no. 7, pp. 44–55, 2004.

[60] Angshuman Parashar, Antonio Puglielli, Joel Emer, and William J Dally. SCNN : An Accelerator for Compressed-sparse Convolutional Neural Networks.

[61] Alberto Delmas, Patrick Judd, Sayeh Sharify, and Andreas Moshovos. Dynamic Stripes: Exploiting the Dynamic Precision Requirements of Activation Values in Neural Networks, CoRR abs/1706.00504, 2017.

[62] CACTI 6.5: https://github.com/HewlettPackard/cacti.

[63] Hardik Sharma, Jongse Park, Benson Chau, and Vikas Chandra. Bit Fusion : Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks. In ISCA, 2018.

[64] Xuan Yang, Jing Pu, Blaine Burton Rister, Nikhil Bhagdikar, Stephen Richardson, Shahar Kvatinsky, Jonathan Ragan-kelley, Ardavan Pedram, and Mark Horowitz. A Systematic Approach to Blocking Convolutional Neural Networks, CoRR abs/1606.04209, 2016.

[65] Synopsis Design Compiler: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html.

[66] Micro Tech Note: https://www.micron.com/support/tools-and-utilities/power-calc.

[67] UCF101 Action Recognition Benchmark: http://crcv.ucf.edu/data/UCF101. php.

[68] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In CVPR, 2015.

[69] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In ECCV, 2016.

[70] Yu Cao, Sarma Vrudhula, and Jae-sun Seo. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In FPL, 2017.

[71] Kalin Ovtcharov, Olatunji Ruwase, Joo-young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating Deep Convolutional Neural Networks Using Specialized Hardware, 2015.

[72] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark Horowitz, and Bill Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In ISCA, 2016.

[73] Ping Chi, Shuangchen Li, and Cong Xu. PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In ISCA, 2016.

[74] Convolutional Neural and Network Accelerator. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In ISCA, 2016.

[75] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, and Reetuparna Das. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. In *ISCA*, 2018.

[76] Hongxiang Fan, Xinyu Niu, and Wayne Luk. F-C3D: FPGA-based 3-Dimensional Convolutional Neural Network. In *FPL*, 2017.