AivoTTA: An Energy Efficient Programmable Accelerator for CNN-Based Object Recognition

Jos IJzerman*, Timo Viitanen*, Pekka Jääskeläinen*, Heikki Kultala*, Lasse Lehtonen*, Maurice Peemen†, Henk Corporaal†, Jarmo Takala*
* Tampere University of Technology, Tampere, Finland
† Eindhoven University of Technology, Eindhoven, The Netherlands
josijzerman89@gmail.com, timo.2.viitanen@tut.fi, pekka.jaaskelainen@tut.fi, heikki.kultala@tut.fi, lasse.lehtonen@phnet.fi, m.c.j.peemen@tue.nl, h.corporaal@tue.nl, jarmo.takala@tut.fi

ABSTRACT

Battery driven intelligent cameras used, e.g., in police operations or pico drone based surveillance require good object detection accuracy and low energy consumption at the same time. Object recognition algorithms based on Convolutional Neural Networks (CNN) currently produce the best accuracy, but require relatively high computational power. General purpose CPU and GPU implementations of CNN-based object recognition provide flexibility and performance, but this flexibility comes at a high energy cost. Fixed function hardware acceleration of CNNs provides the best energy efficiency, with a trade-off in reduced flexibility. This paper presents AivoTTA, a flexible and energy efficient CNN accelerator with a SIMD Transport-Triggered Architecture that is programmable in C and OpenCL C. The proposed accelerator makes use of smart memory access patterns and fusion of layers to greatly reduce the number of memory transfers and improve energy efficiency. The accelerator was synthesized using 28 nm ASIC technology for different supply voltages and clock frequencies. The most power efficient design points consume 11.3 mW for an object recognition network running 16 GOPS at 400 MHz. The maximum clock frequency is 1.4 GHz. With the maximum clock, the accelerator consumes 116 mW for an effective 57 GOPS. To the best of our knowledge, it is the most energy efficient compiler programmable CNN accelerator published.

1. INTRODUCTION

For intelligent camera systems, the main problems are the large data storage and transfers, and the need for human intervention [6]. Convolutional Neural Networks (CNNs) are state-of-the-art algorithms for visual classification and other pattern recognition tasks. These networks make human intervention obsolete by offering comparable recognition accuracy [4]. In addition, artificial neural networks can be processed near the sensor, reducing the need for central processing and allowing for collaborative scene analysis with minimized data transfers.

Considering recent developments in micro air vehicles such as robotic insects [36, 23, 5], in combination with cheap lensless cameras [35], we are entering an age of ubiquitous visual sensing. This gives rise to a demand in flexible, low energy visual classification solutions for decentralized processing.

Low energy footprint is the most important factor in CNN driven object recognition on battery powered devices such as smartphones, watches or glasses [34, 32], or micro air vehicles. Hardware accelerators for CNNs have been the focus of attention in recent years as a low-energy, high performance alternative to CPUs and GPUs.

Different CNNs have many different parameters such as the number of layers and their connectivity, the number of feature maps and the size of the weight kernels. In order for a CNN accelerator to cover a wide range of application domains, it must have the flexibility to support different network parameters. However, flexibility always brings overheads visible as higher energy cost and larger chip area making the combination of flexibility and a low energy footprint contradictory goals.

Recent CNN accelerators mostly use finite state machines (FSM) [9, 12, 31] with configurable parameters to support different network characteristics, offering a generalized order of processing for a wide variety of networks. However, this approach cannot utilize flexible ways to optimize memory access patterns [27], layer fusion [1] or other techniques that change the order of operations in favor of energy efficiency and low memory transfer costs. Different networks require custom implementations of these techniques or a more flexible accelerator with software programmability.

Programmable co-processors often utilize static VLIW-style architectures [24, 3, 30] to take advantage of the instruction level parallelism that is visible at compile time. However, a well-known contribution to VLIW energy consumption is its register file complexity [10]. For every parallel functional unit, additional register file ports have to be added, increasing the power consumption and hardware complexity of the accelerator implementation.

In this article we propose AivoTTA, a programmable CNN accelerator design that utilizes the Transport-Triggered Architecture [10] (TTA), which has been shown to reduce the register file bottleneck of VLIW designs. The bottleneck is reduced by exposing more of the architecture to the programmer's or compiler's control, which trades energy effi-

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

ciency benefits to less dense instruction words [18].

Later in the article we demonstrate that by using a TTA instead of a traditional VLIW, we achieve a high degree of functional flexibility at a lower energy cost. This is because on a TTA-based accelerator, the register file complexity does not increase at the same rate as the number of functional units, enabling better scalability. The result is a scalable accelerator that can be programmed in higher level languages for a variety of CNN classification tasks at a low energy cost.

The accelerator is programmable in C and OpenCL C, utilizes wide SIMD instructions for computational efficiency and has small local memories to statically exploit the available data locality in the networks for increased power efficiency.

2. RELATED WORK

As CNNs have been one of the hottest topics in the past years, there has been a lot of recent work regarding their acceleration. Overall, CPU implementations offer limited performance, while GPU [8] acceleration offer performance at high power consumption, making them unsuitable for many embedded applications. Implementations on FPGAs [14, 37, 16, 29] offer great flexibility with limited performance.

Fixed function ASIC implementations offer both computational performance and low power consumption at a cost of limited flexibility. The DianNao [9] accelerator focuses on efficiently executing the computational primitives on its Neural Function Units, but does not offer support for exploiting data locality of 2-dimensional convolution. DianNao uses an FSM to support different network configurations.

ShiDianNao [12] uses modified DianNao Neural Function Units to better exploit the data locality. It maps the entire CNN in SRAM and greatly improves energy efficiency by eliminating DRAM. Compared to GPU implementations ShiDianNao consumes about 4700 times less energy and is designed to be embedded near the sensor. The functionality of this accelerator is also managed by an FSM.

NeuFlow [13, 31] is an accelerator aimed at embedded visual applications. It has a reconfigurable dataflow architecture that is configured by a control unit. Its behavior is defined by configuration parameters written to its control unit.

Origami [7] is another energy efficient accelerator. An important focus point of the design is reducing external memory access and alleviating the memory bottleneck of other designs. It exploits data locality to use its memory bandwidth more efficiently. Origami is a fixed function accelerator and does not make use of instruction control based processing.

The *Neuro Vector Engine* (NVE) [30] is a low power CNN accelerator that uses high level XML network descriptions to generate VLIW code [28]. The programmability of the accelerator allows for many advanced data locality optimizations that offset the costs of programmability. The proposed accelerator allows for the same level of programmable flexibility while improving efficiency by using the exposed data path TTA paradigm.

There are also various commerical SoCs (CEVA XM6 [3], Movidius Myriad 2 [24]) targeting general vision applications. These systems contain VLIW cores with vector units, but little is published about their performance and efficiency.

For these wide VLIW cores, the register file complexity limits power efficiency and scalability. In our case, the transport triggered architecture based design enables simpler single ported register files.

```
for(o=0; o<No; o++){       //output feature maps
  for(m=0; m<Nm; m++){     //row in feature map
    for(n=0; n<Nn; n++){ //column in feature map
      acc = bias[o];       //initialise with bias
      for(i=0; i<Ni; i++){     //input feature maps
        for(k=0; k<Nk; k++){    //row in kernel
          for(l=0; l<Nl; l++){ //column in kernel
            acc += in[i][S*m+k][S*n+l] *
                weight[o][i][k][l]; }}}
      out[o][m][n] = Sigmoid(acc); }}}
```

Figure 1: Pseudo code of a single layer in a convolutional neural network. Subsampling and feature extraction are merged.

Google's Tensor Processing Unit (TPU) [20] is a neural network accelerator designed for use in data centers and cloud computing. It features a matrix multiply unit that can offer 92 TOPS and contains a large on-chip memory.

AivoTTA provides a software programmable control with power efficiency improved over standard VLIW designs. Its power-efficiency is closer to fixed function accelerators than with the previously published programmable accelerators. Unlike other designs, the proposed one has an exposed data path which allows additional compiler-based optimizations.

3. CONVOLUTIONAL NEURAL NETWORKS

CNNs are the state-of-the-art machine learning algorithms for visual recognition. Our accelerator focuses on feed forward inference with CNNs. Training of the network is assumed to be done offline in order to save resources on embedded platforms. A CNN consists of layers that transform the representation to a higher abstraction level, starting with the raw pixels of an input image. Simple features are extracted from the raw image in the first layer and these features are combined into more complex representations later in the network. The last layer combines the highest order features and classifies objects at a location in the image.

In contrast to deep neural networks (DNNs), CNNs reuse synaptic weights per output feature map. This allows all weights to be simultaneously stored in a local memory. This reflects the ability of CNNs to detect features anywhere in the image [21].

Fig. 1 shows pseudo-code of a CNN layer; it contains a number of parameters that define the number of neurons and their connectivity. Every layer generates a number of output feature maps based on parameter Nn, each output feature map has dimensions *Nm*No*. Every neuron has a bias that is shared among all neurons on an output feature map. Every neuron performs a 3-dimensional convolution based on parameter *Ni* (the number of input feature maps) and parameters *Nk* and *Nl*, the vertical and horizontal dimension of the input window on a single input feature map.

The parameter *S* defines the subsampling factor, the stepsize between consequtive convolution windows. It allows us to perform a data reduction and scale down the output feature maps. In some networks, subsampling and feature extraction are performed in separate layers. Merging these together reduces the computational load for comparable detection accuracy [25]. Figure 2 shows the data and computational reduction that is the result of layer merging.

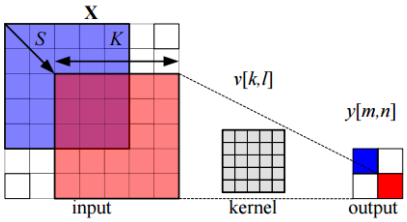


Figure 2: Merged convolution and 2x2 subsampling. Instead of computing every possible neuron and then reducing the result, we only compute a single neuron in a 2x2 window.

```
for(o=0; o<No; o++){ //output feature maps
  for(m=0; m<Nm; m++){ //row in feature map
    for(n=0; n<Nn; n+=Np){ //column in feature map
      for(p=0;p<Np;p++){ //parallel PEs
        acc[p] = bias[o];} //initialise with bias
      for(i=0; i<Ni; i++){ //input feature maps
        for(k=0; k<Nk; k++){ //row in kernel
          for(l=0; l<Nl; l++){ //column in kernel
            for(p=0;p<Np;p++){ //parallel PEs
              acc[p] += in[i][S*m+k][S*(n+p)+l] *
                  weight[o][i][k][l]; }}}}
      for(p=0;p<Np;p++){ //parallel PEs
        out[o][m][n+p] = Sigmoid(acc[p]); }}}}}
```

Figure 3: Pseudo code of a tiled loop with Np processing elements that compute adjacent neurons in parallel.

4. DATA LOCALITY

Recent work on low power CNN acceleration has placed a focus on smart memory access patterns and data reuse in an effort to reduce memory accesses and power consumption [29, 12, 30]. CNNs consist of overlapping convolution windows that can be computed independently and there is a great potential for data reuse. There are several techniques to exploit the different types of data locality that are present in these networks, but the effectiveness depends on the actual network parameters and connectivity.

Neurons that map to the same output feature map share the same weight kernel and adjacent neurons have overlapping receptive fields. An intuitive way of exploiting data locality is to compute a number of overlapping neurons in parallel; reusing data amongst multiple neurons and reducing the total amount of data transferred from memory.

Neurons associated with different output feature maps in the same layer do not share weight kernels, but their receptive fields can have full overlap with neurons of different output feature maps. We refer to this type of locality as *output feature map locality* and it has the potential for more reuse of input data, but it does not benefit from weight kernel sharing.

A more high level reordering of operations for data reuse is *layer fusion* [1], a technique to exploit data locality across different layers. In layer fusion, instead of processing each layer to completion before starting the next, multiple layers are processed for a small region of the input image at a time, such that the intermediate results can be kept in small on-chip memories. This can significantly reduce memory traffic.

Using any combination of these techniques changes the order of operations in order to reduce memory transfers. The specific network parameters determine the benefits of these techniques. As a result, it is desirable to have a flexible, programmable accelerator design where computations can be reordered in different ways to take advantage of the data locality specific to each network.

Parameters Nk and Nl in Figure 1 represent the convolution window size on a single input feature map. Bigger windows increase the overlap and favour exploiting of this type of data locality. The subsampling parameter S increases the step size between subsequent windows and reduces the number of windows calculated. This reduces the available data locality. Parameters Ni and No denote the number of input and output feature maps respectively. A layer can be fully or sparsely connected and this influences the available data locality. A high number of ouptut feature maps and a dense connectivity increase the benefits of exploiting output feature map locality.

5. MAPPING PRINCIPLES AND PROCESSING

A good mapping of convolution windows to computational primitives is important. Spreading the computation of a single neuron among multiple processing elements would mean that the ideal number of processing elements becomes dependent on the kernel size. A small kernel might leave processing elements unutilized and a big kernel would need to combine partial results.

Since kernel sizes vary across different network layers, an alternative is to map processing elements to individual neurons and perform the computations in time. This approach makes the utilization of these units invariant to the kernel size. An additional advantage is the flexibility to deal with irregular weight kernels that may be the result of intra-kernel weight pruning, a technique used to remove insignificant weights from kernels to improve performance [2].

Fig. 3 shows the network layer pseudo-code discussed earlier with the addition of several p loops. Instead of computing all windows in sequence, we tile the loop to process Np horizontally adjacent pixels in parallel.

In addition to speeding up computation, we exploit the data locality of overlapping receptive fields of the neurons processed in parallel. All neurons processed in parallel share the same weight kernel and their convolution windows overlap horizontally. For a layer with 5x5 kernels and no subsampling, 32 parallel processing elements operating this way reduce the weight data loaded by 97% and pixel data loaded by 78% compared to sequential processing.

Fig. 4 shows how data locality is exploited in processing a single row of pixels on four separate convolution windows sharing the same kernel. Four vector multiply/add operations are performed each with a separate weight from the first row of the weight kernel. Each vector of pixels that we multiply overlaps with the previous vector and the results of each multiplication are accumulated. Without subsampling, the vector of pixels that is multiplied with a weight is adjacent in memory.

We can apply layer fusion [1] by tiling loops n and m. The tiling size of n and m is different per layer and depends on the amount of data needed by the next layer. We can calculate the minimal fusion tile size as shown in Fig. 5. Parameters H, W and D denote the height, width and depth (number of output feature maps) of a fusion tile.

Given p parallel accumulators that calculate p horizontally adjacent neurons, the fusion tile of the last layer (Y) has a width of p (to utilize the full vector width), a height of 1 and a depth depending on the number of output fea-

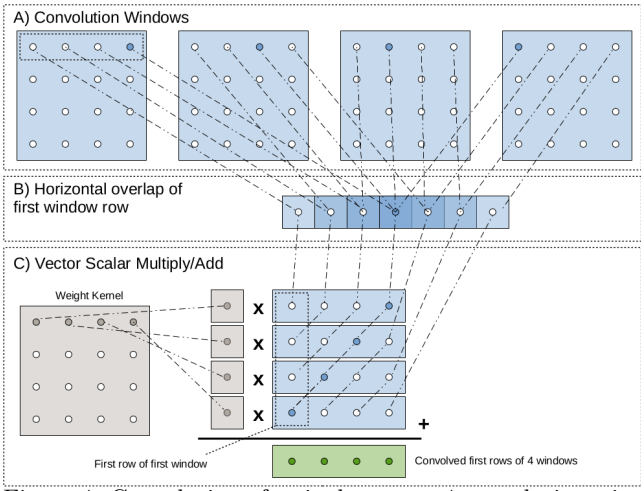


Figure 4: Convolution of a single row on 4 convolution windows without subsampling. A) 4 separate convolution windows. B) The overlap of the first row of these 4 windows. C) Multiplying a vector of pixels with a single weight, shifting additional values into the pixel vector and accumulating the results per neuron.

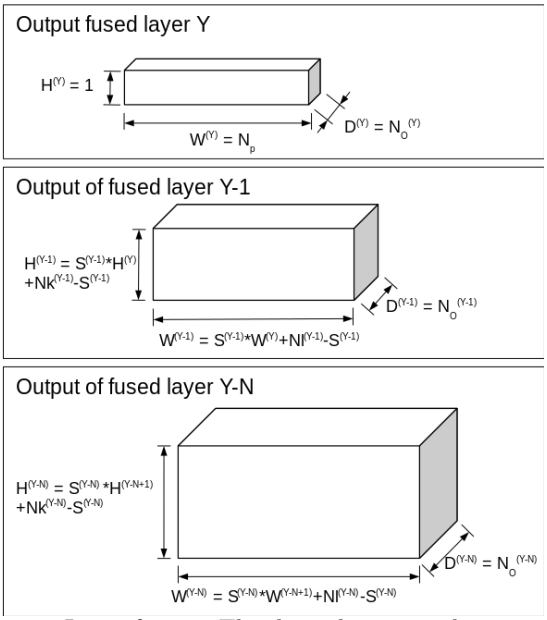


Figure 5: Layer fusion. The desired output tile size at the last layer (layer Y), determines the minimal required tile sizes of the preceeding layers.

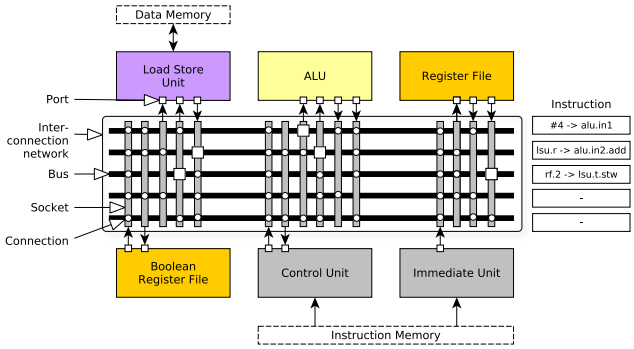


Figure 6: Example of a TTA processor. The current instruction being processed is shown on the right, it contains 5 slots; each describing a move operation on one of the 5 buses.

ture maps. The dimensions of the preceding layers' fusion tiles are dependent on the required tile size of the next layer, kernel size, subsampling factor and number of feature maps.

Computing all fusion tile sizes gives us the minimum memory size required to avoid all external memory accesses, except for loading the initial image. The order of processing for these fusion tiles opens up additional design decisions. After the first fusion tile is computed on all layers, we can pick a second tile that overlaps with the previous one and buffer data. Alternatively overlapping neurons can be recalculated.

6. ACCELERATOR ARCHITECTURE

6.1 Architecture Template

The proposed accelerator is based on a Transport Triggered Architecture [10]. The difference between TTA and traditional VLIW is that data transports over the internal buses are exposed to the programmer. Instead of programming operations and implicitly triggering the buses, data movements are programmed explicitly. Computations are done as soon as the data arrives on the triggering port of a functional unit. The set of computation operations is defined by the operations available on the functional units.

A TTA processor allows for extra software optimizations compared to VLIW, such as software bypassing [18] and operand sharing. By programming data moves instead of operations, we can determine register bypassing at compile time allowing the compiler to explicitly move results from one FU to another or keep operands on FU input ports for multiple computations. Allowing data to be moved from one functional unit to another greatly simplifies the architecture, reduces the required number of general purpose registers [17], and allows for more instruction level parallelism by explicitly avoiding registers.

A TTA processor consists of a control unit, register files, load store units, functional units, and transport buses. An example TTA is shown in Fig. 6. The instructions contain a slot for each bus and the instruction width is dependent on the number of buses and the number of connections to functional units. Units can be connected to a subset of buses to reduce the instruction size.

6.2 Function Units

The most critical unit in the proposed accelerator is the vector multiply/add unit (MADD). This unit is responsible for performing convolution. It is designed to compute 32 convolutions on the same output feature map. This allows weight sharing. The MADD unit takes a vector of 32 8-bit pixels and either multiplies every element with a single 16-bit signed value or multiplies elementwise with a vector of 32 16-bit weights. The result of these operations are added to the 32 bit accumulator inputs that are stored in a vector of 32 elements.

Given that a full crossbar shuffle is very expensive, we include a limited operation set for rearranging vectors, picked to support the CNN mapping techniques described in the previous Section, namely, vector shift, interleave and broadcast. The *vector shift unit* is used to shift a 32 × 8b vector left while inserting elements from a 32b scalar on the right side. Five outputs are produced, corresponding to shift lengths 0..4. The shift operation is used to efficiently generate inputs to the MADD unit, while reusing loaded data for several multiplications, and avoiding the need to support expensive unaligned loads.

The *vector deinterleave unit* takes two 32 × 8b input vectors, and arranges their even and odd elements into separate output vectors. This operation is used to implement 2x2 subsampled layers. Finally, the *vector broadcast unit* outputs a vector with all elements set to an input scalar value. In this work, it is mainly used to initialize accumulators.

After a vector of convolutions is completed the result is stored in a 1024-bit vector containing 32 32-bit elements. We apply an activation function on these values and reduce them to a 32-element vector of 8-bit values. Activation functions are expensive to evaluate, so we include hardware support for their piecewise linear approximation. The approximation is computed with the MADD unit, but an *interpolation unit* is included to generate the inputs of the MADD based on table lookup. One vector input of the interpolation unit is used as the lookup table which encodes the levels and slopes of each segment.

Table 1 lists all functional units and register files. Besides the special functional units already described, the accelerator contains three basic arithmetic units. One ALU that can perform basic arithmetic and logic operations, a unit that can perform scalar multiplication and addition, and a scalar add unit. These three units can all perform addition, since this operation is common for address calculation and loop indexing. The vector broadcast unit is used to broadcast biases to an entire vector and the vector compare unit is used in the last layer to perform detections.

6.3 Memory System

The accelerator contains 3 on-chip SRAMs with separate memories for weights, data and instructions. An off-chip DRAM is assumed to store the raw image frame buffer. Since weight kernels are shared amongst output feature maps, all weights and biases fit in a single SRAM. The data memory is used to buffer part of the network locally and to reduce the number of DRAM accesses.

The image DRAM, weight SRAM and data SRAM are each accessed with a separate *load-store unit* (LSU). All SRAMs are optimized with multiple banks to reduce the energy consumption per access. The total on-chip SRAM is 128 kB: 32 kB instruction memory, 64 kB weight memory and 32 kB data memory.

Table 1: Functional units and register files.

Unit	Operations
ALU	Basic arithmetic & logic
Add	Add
Mul/Add	Mul, Add
Immediate	Immediate value from instruction
General Control Unit	Jump, Call, Loop buffer
Data SRAM LSU	ld32, ld256, st32, st256
Weight SRAM LSU	ld16
DRAM LSU	ld32, ld256, st32, st256
Vector Multiply/Add	MADD scalar, MADD vector
Vector Shift	Vector shift
Vector deinterleave	Vector deinterleave
Word deinterleave	Word deinterleave
Vector Broadcast	Broadcast32x32
Vector Compare	GreaterThan32x32, Vector Reduce
Interpolation	Interpolation
8x1024-bit register file	Single ported read/write
8x256-bit register file	Single ported read/write
8x32-bit register file	Single ported read/write
16x32-bit register file	Single ported read/write

In case of TTAs, the register files are the second lowest level in the data memory hierarchy with the function unit port registers being the lowest. One of the most interesting features of the proposed accelerator is its simplified register files enabled by the TTA programming model. All register files on the accelerator have only a single read and a write port, making them area economical and power efficient.

In the design, there are three different register files: the scalar register file contains 24 32-bit registers, the pixel vector register file has 8 256-bit registers, and the widest vector register file 8 1024-bit registers for intermediate convolution results. The 256-bit registers are used to store vectors of 32 8-bit pixels and the 1024-bit registers are used to store 32 32-bit accumulators. By using eight 1024-bit registers we can interweave the processing of 256 neurons; either on the same or different output feature maps.

6.4 Control Hardware and Interconnection Network

The proposed accelerator has an instruction based control. A TTA instruction consists of a separate *move slot* field for each bus. Each move slot specifies a *move* which transports data from a source to a destination on the corresponding bus. Sources and destinations are encoded in separate fields. The length of each field depends on the bus connectivity, e.g., how many different operations can be triggered, or RF registers written to, on a given bus.

The interconnection network of the proposed accelerator consists of seven transport buses. Buses 0–2 are used for general-purpose scalar computation, such as control flow and address computations. The function units connected to these units make up the equivalent of a miniature RISC machine with RFs, ALUs, LSUs and a control unit. Buses 3–5 are scalar buses optimized for supplying kernel weights from the weight memory. Bus 6 is used to supply 256-bit input vectors to the MADD unit, and bus 7 handles the 1024-bit output accumulator vectors from MADD. In total, the instruction word is 64 bits long.

The accelerator contains a hardware loop buffer. Up to

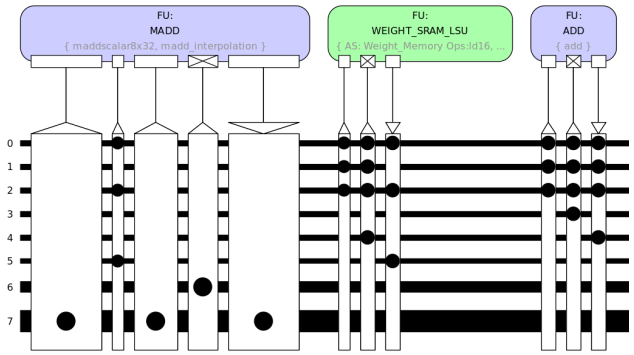


Figure 7: Weight Data Path.

64 instructions can be stored in the buffer and it keeps track of the loop counter in hardware. The loop buffer reduces the number of cycles by more efficiently executing loops and also minimizes accesses to the instruction memory in code with iterations, which are typical with CNNs.

6.5 Implementation

The accelerator was designed using the TTA-based co-design environment (TCE) toolset [19], which includes, e.g., a graphical processor editor, a retargetable compiler, a cycle-accurate simulator, and a processor RTL generator. In this work the accelerator is programmed in C, but OpenCL C is also supported by the toolflow.

7. EVALUATION

The accelerator was synthesized using the Synopsys design compiler using topographical synthesis. We used the 28 nm FDSOI (low power) technology and tested different target clock frequencies up to 1.4 GHz. In addition, we synthesized the design for 65 nm technology to provide more meaningful comparison with other accelerators that utilize this older technology. The SRAM memory costs were estimated with CACTI [22].

Two different CNN applications were used to benchmark the proposed accelerator: A face detection [15], and a speed sign detection network [26]. Both networks perform recognition tasks on 720p images and consist of 4 layers.

The parameters of our evaluation networks are listed in Table 2. The face detection network has fewer connections between the layers and it is executed a total of 4 times, once on the normal input image and three times on downscaled versions of the input; this allowed the network to detect faces of different sizes. For speed sign detection we used a much denser connected network without input downscaling.

Table 3 details the available data locality per layer of the face and speed sign networks respectively. Using the total number of multiplications per layer and the total size of each layer's input feature maps, we can calculate the total number of operations per pixel. This gives an theoretical upper bound to the available data locality within each network.

Examining Table 3, we note that the speed sign detection network has a much higher potential for data reuse. The available data locality of each layer is much bigger than the face detection network and the weighted average over all layers is 6 times bigger. The third layer of the speed sign network accounts for over 80% of the total computational load of the entire network. Due to the large number of output feature maps and dense connectivity of this layer, a pixel is used by 965 computations on average.

Depending on the target supply voltage and clock frequency, the silicon area of AivoTTA varies between 0.32 mm and 0.37 mm. The three on-chip SRAMs (128 kB) require 0.21 mm^2 of the total area. Fig. 9 shows the layout of the chip for 0.6V at 400 MHz.

Table 4 lists the total number of instructions per application and the utilization of the vector unit and loop buffer. The power consumption of the face detection network is higher due to the more intensive memory usage and lower instruction buffer utilization. The speed sign network gets half of its instructions from the loop buffer and has a vector utilization of 65%, while the face detection network has a 36% and 38% utilization for loop buffer and vector unit respectively.

Both networks have different memory bandwidth characteristics. The face detection network generates 5MB external memory traffic per frame, while speed sign detection network uses 2.3MB. Most memory traffic is kept in the local SRAM memories, with the instruction memory accounting for ca. half of the local traffic.

The power consumption of both networks is comparable. Figs. 10a and 10b show the power consumption of the face and speed sign detection respectively for different clock frequencies. The power includes the core and on-chip memories, but excludes DRAM. For lower frequencies, the memory power consumption can amount to as much as 85% of total. For the highest frequencies per voltage level the power consumption of the SRAMs drops to 40-45%.

Table 5 shows the power breakdown of the accelerator at 400 MHz. The vector multiply/add unit is responsible for 28.6% of the total power consumption, followed by the instruction, weight and data SRAM. All other functional units combined reach only 7.1% of the total power consumption with 0.8 mW.

From this table we can also deduce the cost of the accelerator's programmability and assess somewhat the benefits of the TTA design choice. The instruction SRAM, loop buffer, fetch & decode logic combined make up 22.7% of the total power consumption: 2.49 mW. We consider this a small power cost of a programmable accelerator, which is more flexible and allowing C and OpenCL C programs to be directly compiled greatly reduces the design time. The interconnection network consumes an additional 0.81 mW. The register files together only consume 0.41 mW due to their simplicity.

Both graphs show the energy cost per frame converging.

Table 2: Network parameters for speed sign and face detection CNNs.

Face Detection	No	Ni	Nm	Nn	Nk	Nl	S
Layer 1	4	1	638	358	6	6	2
Layer 2	14	4	317	177	4	4	2
Layer 3	14	14	312	172	6	6	1
Layer 4	1	14	312	172	1	1	1
Speed Sign	No	Ni	Nm	Nn	Nk	Nl	S
Layer 1	6	1	638	358	6	6	2
Layer 2	16	6	317	177	6	6	2
Layer 3	80	16	313	173	5	5	1
Layer 4	8	80	312	172	1	1	1

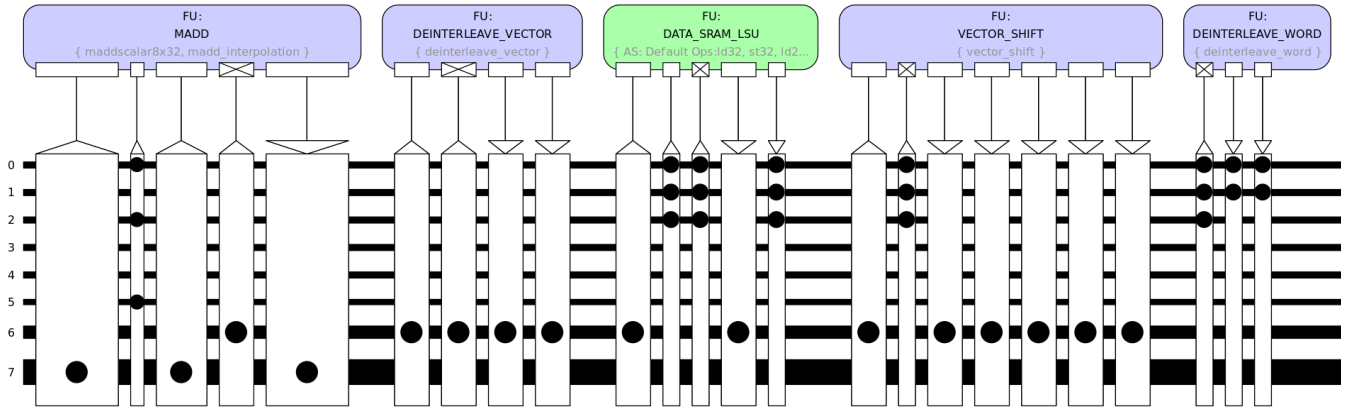


Figure 8: Pixel Data Path.

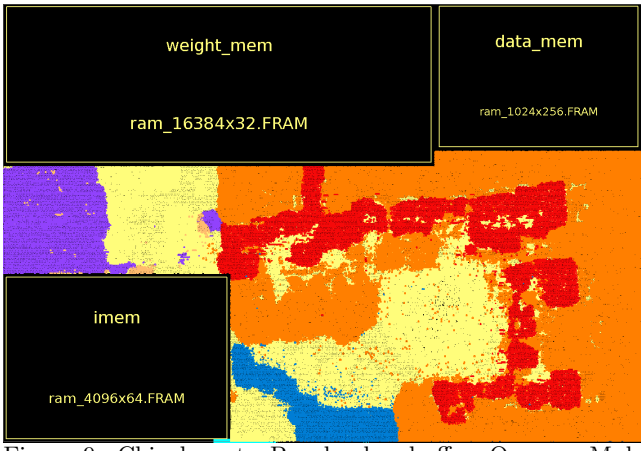


Figure 9: Chip layout. Purple: loopbuffer; Orange: Multiply/Add unit; Red: 1024-bit register file; Blue: 256-bit register file; Yellow: other units, buses, etc.

For example, the speed sign detection network consumes just less than 5 mJ per frame at 1.1V for any configuration between 8 and 23 frames per second (500-1400 MHz).

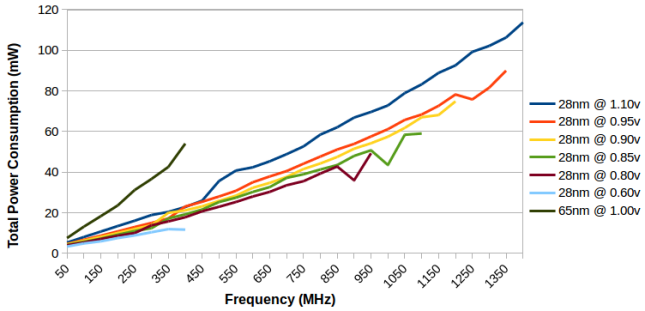
A measure of energy efficiency is the GOPS/W, listed in Figs. 11a and 11b. We use the effective number of GOPS and only consider the vector multiply/add operations on both networks. Arithmetic operations from other functional units are ignored. These results show big differences between the networks. The computational efficiency depends greatly on the TCE compiler's ability to generate efficient code. The speed sign network has a much higher utilization due to network density and better data locality. Table 4

Table 3: Data locality per layer of the face detection network.

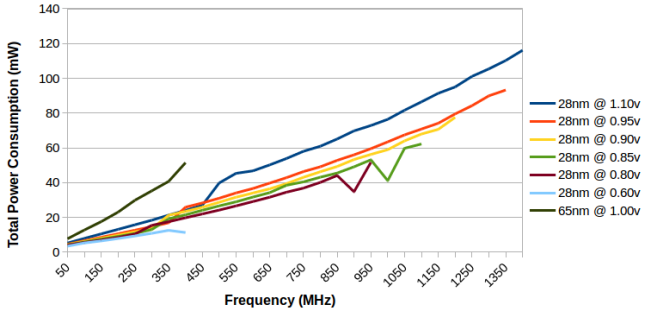
Network	Face detection		Speed sign detection	
Layer	% of total ops.	ops. / pixel	% of total ops.	ops. / pixel
1	41.8	35.7	4.6	53.5
2	22.8	19.7	11.3	88.4
3	34.4	36.0	80.9	965.1
4	1.0	1.0	3.2	8.0
Total	100.0	23.6	100	143.2

Table 4: Vector unit and loop buffer utilization

	Face	Speed Sign
Total instructions (10^6)	21.2	58.4
Vector MADD instructions (10^6)	8.15	38.0
Instructions from loop buffer (10^6)	7.6	29.2
Vector unit utilization	38.5%	65.2%
Loopbuffer utilization	36.1%	50.0%



(a) Face detection network.

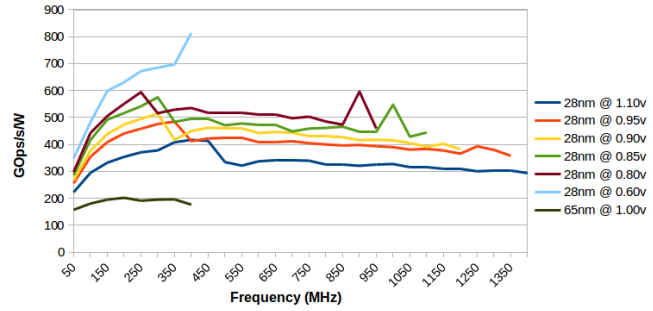


(b) Speed sign detection network.

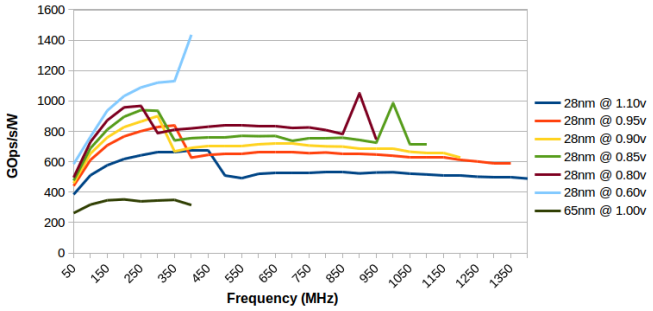
Figure 10: Total power consumption

Table 5: Power breakdown 0.6V 400 MHz

Module	Power (mW)	(%)
Data SRAM	1.32	11.7
Weight SRAM	1.99	17.6
Instruction SRAM	2.16	19.1
Multiply/Add Unit	3.23	28.6
Other Functional Units	0.80	7.1
Loopbuffer, Fetch & Decode Logic	0.58	5.1
Interconnection Network	0.81	7.2
Register Files	0.41	3.6
Total	11.3	100



(a) Face detection network.



(b) Speed sign detection network.

Figure 11: Energy efficiency (GOPS/W).

shows that while running the speed sign network, 65% of all cycles contain a multiply/add operation, compared to 39% for face detection.

The best energy efficiency is found at 28nm, 400MHz, 0.6V. With this synthesis target, the processor can run inference for the speed sign detection network at 7FPS. The highest reachable single-core performance for this network is 23FPS. The face detection network has performances of 19FPS and 66FPS, respectively.

A sparse network contains less exploitable data locality and cannot be run as efficient as a more dense network. For sparse networks, it becomes more difficult for the compiler to efficiently schedule a multiply/add operation every cycle. Since there are less operations per pixel (Table 3), the compiler has to schedule more operations to load data in the background, increasing the scheduling complexity. Additionally, the face detection network loads multiple scaled images from DRAM and spends more cycles on buffering this data on local SRAM.

Compiler optimizations can greatly improve the computational efficiency. Manual analysis of the compiled code of each layer's inner loops show that many opportunities for improvement remain.

The main contributors to the low energy usage are the use of local banked memories and the TTA programming model, which enable simple control unit and register files. We almost exclusively use local memories and avoid costly external memory transfers by using layer fusion and other techniques that change the order of operations. The TTA instruction overhead is relatively low since instruction decoding is simple and the use of an instruction buffer reduces the instruction loads by up to 50%.

7.1 Comparison

Table 6 shows a number of ASIC CNN accelerators and several implementations of the proposed accelerator with different clock frequencies and voltages. Since different technologies are difficult to compare, we also synthesized our design for 65 nm CMOS technology.

The proposed accelerator has less computational resources than most of our comparison targets. The only exception is the NVE, which uses 16-element vectors instead of our 32.

The different accelerators listed in the table employ different techniques of parallelizing computation. Both our design and the NVE perform convolution in time and map computational resources to different neurons. This makes utilization of the functional units invariant to convolution window size. Other accelerators such as the DianNao variations and Origami, perform convolution in space, mapping computational resources to different pixels on the receptive fields of a neuron. This makes the computational efficiency dependent on the window size and can negatively impact utilization if hardware choices and network parameters do not match.

The most efficient configuration has a low power budget. At 400 MHz the proposed accelerator consumes 11 mW for 1434 GOPS/W. Designs with higher clock frequencies trade an increase in performance for a decrease in power efficiency. However, the energy cost per frame for both the dense and sparse network remain stable.

External memory bandwidth is a major constraint for CNN accelerators. All power budgets listed exclude DRAM power consumption. Using layer fusion, our design reduces the number of external accesses to a minimum; only the first layer reads from DRAM. The NVE uses the exact same speed sign detection network as was used for our benchmarking and lists the number of external accesses. Our implementation of this network makes 3.2 x 10^5 external accesses, a factor of 56 fewer compared to the 1.8 x 10^7 DRAM read operations of the NVE. The NVE uses a much smaller local memory and so has to use DRAM more intensively. For the same reason the NVE has a smaller chip area even though it uses a bigger technology.

When we express the bandwidth efficiency in GOPS/GB, we notice significant differences between our dense and sparse network. The speed sign detection network loads a total of 2.3 MB from DRAM for an efficiency of 1081 GOPS/GB. The face detection network has a much bigger memory footprint due to the fact that it loads multiple rescaled images for a total load of 5 MB. Combined with the fact that this network is more sparse and performs fewer computations, the bandwidth efficiency drops to 102.4 GOPS/GB. NeuFlow and Origami report 50 and 521 GOPS/GB respectively. It should be noted that the only design with a clearly better reported bandwidth economy is the server-oriented Tensor Processing Unit [20], which has a very large on-chip memory.

The accelerator designed in [33] avoids most multiplica-

Table 6: Comparison with state-of-the-art

	mW	V	Tech. (nm)	MHz	GOPS	GOPS /W	GOPS/W Core only	On-chip Mem.	GOPS /GB	Control	mm^2	GOPS /mm^2
Neuflow [31]	600	1.0	45	400	294	490	N/A	75 kB	50	Dataflow	12.5	23.5
Origami [7]	93	0.8	65	189	55	803	N/A	43 kB	521	Config	1.31	42
NVE [30]	54	N/A	40	1000	30	559	595	19 kB	125	VLIW	0.26	115
DianNao [9]	485	N/A	65	980	452	931	N/A	44 kB	N/A	FSM	3.02	149.7
ShiDianNao [12]	320	N/A	65	1000	128[1]	400[1]	470[1]	288 kB	N/A	FSM	4.86	26.3
SoC [11]	51	0.575	28	200	79	N/A	1542	6 MB	N/A	Config	34	2.3
DNPU [33]	35	0.765	65	50	73	2100[2]	N/A	290 kB	N/A	FSM	16	4.5
TPU [20]	40000	N/A	28	700	46000[3]	1150[3]	N/A	28 MB	1352[3]	CISC	<331	>139
AivoTTA (proposed)	11	0.6	28	400	16	1434	2030	128 kB	1081	TTA	0.36	44.4
	35	0.8	”	900	36	1049		”		”	0.36	100
	41	0.85	”	1000	40	986		”		”	0.36	112
	93	0.95	”	1350	54	587		”		”	0.36	150
	116	1.1	”	1400	57	489		”		”	0.35	163
	41	1.0	65	350	14	349		”		”	1.43	9.8

[1] Indicates theoretical peak performance, as opposed to measurements on a real network.
[2] Q-table lookup of precomputed multiplication results.
[3] Results with 16-bit arithmetic. TPU can also operate at 8-bit for double performance.

tions by performing lookups in a Q-table containing precomputed multiplication results. It contains a custom instruction set-based controller that configures finite state machines on lower level convolution cores.

[11] presents a chip with multiple DSP cores for general vision processing, a big on-chip memory, and a co-processor subsystem with multiple convolution accelerators. The paper only lists power consumption for the co-processor, excluding on-chip memory. Control of the convolution accelerators is handled through a set of configuration registers that configure a stream switch, allowing the reuse of data streams. Multiple accelerators can be grouped or chained together.

8. CONCLUSION

The available instruction level and data level parallelism, and the regular structure of convolutional neural networks makes them very suitable for VLIW-style processors. In this paper, we demonstrated that a transport triggered architecture with wide SIMD operations allows for additional energy savings by simplifying the architecture and allowing more compile time optimizations. By exploiting data locality and using local memories, the number of external memory accesses could be minimized.

The proposed accelerator is scalable to match energy budgets and performance requirements of the application. Most importantly, the accelerator is software programmable, allowing memory access patterns and computational ordering to be easily customized for specific networks. The costs of programmability were offset by the efficient transport triggered architecture and optimized data locality.

A compiler programmable accelerator offers more flexibility than fixed function accelerators in exchange for an increase in power consumption. FPGA solutions offer even more flexibility but cannot compete with the power efficiency of the proposed accelerator. The compiler is designed to adapt to changes in the architecture, allowing for performance portability in the form of scalability.

The efficiency of the accelerator is strongly dependent on the density of the network, the available data locality and the compiler's ability to exploit this locality. Our dense example network achieves 1434 GOPS/W, while the sparse example network performs at 813 GOPS/W. Both networks can perform object detection at a cost of less than 2 mJ per frame, allowing for battery powered, mobile object recognition in ubiquitous visual sensing applications.

In the future, we plan to improve the TCE compiler to further increase vector MADD utilization, and allow compilation for architectures with more parallel units. In early, manually scheduled results, this would give some improvement in computational efficiency. Moreover, we are working on optimizing the design for FPGA soft core use.

9. REFERENCES

[1] M. Alwani, H. Chen, M. Ferdman, and P. Milder. Fused-layer CNN accelerators. In *Proc. Int. Symp. Microarch.*, pages 1–12, 2016.

[2] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *ACM J. Emerging Tech. in Comput. Sys.*, 13(3):32, 2017.

[3] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma. Always-on vision processing unit for mobile applications. *IEEE Micro*, 35(2):56–66, 2015.

[4] A. Blanton, K. C. Allen, T. Miller, N. D. Kalka, and A. K. Jain. A comparison of human and automated face verification accuracy on unconstrained image sets. In *Proc. Conf. Computer Vis. and Pattern Recog. Workshops*, June 2016.

[5] R. Brockers, M. Hummenberger, S. Weiss, and L. Matthies. Towards autonomous navigation of miniature UAV. In *Proc. Conf. Computer Vis. and Pattern Recog. Workshops*, pages 645–651, 2014.

[6] C. Bobda. *Distributed Embedded Smart Cameras: Architectures, Design and Applications*. Springer, 2014.

[7] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini. Origami: A convolutional

network accelerator. In *Proc. Great Lakes Symp. VLSI*, pages 199–204, 2015.

[8] L. Cavigelli, M. Magno, and L. Benini. Accelerating real-time embedded scene labeling with convolutional networks. In *Proc. Design Automation Conf.*, pages 1–6, 2015.

[9] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, pages 269–284, 2014.

[10] H. Corporaal. *Microprocessor Architectures: From VLIW to TTA*. John Wiley & Sons, Inc., 1997.

[11] G. Desoli, N. Chawla, T. Boesch, S. p. Singh, et al. A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems. In *Proc. Int. Solid-State Circuits Conf.*, pages 238–239, 2017.

[12] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. ShiDianNao: Shifting vision processing closer to the sensor. In *Proc. Int. Symp. Computer Architecture*, pages 92–104, 2015.

[13] C. Farabet, B. Martini, B. Corda, P. Akselrod, et al. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *Proc. Conf. Computer Vis. and Pattern Recog. Workshops*, pages 109–116, 2011.

[14] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. CNP: An FPGA-based processor for convolutional networks. In *Int. Conf. Field Programmable Logic and Applications*, pages 32–37, 2009.

[15] C. Garcia and M. Delakis. Convolutional face finder: a neural architecture for fast and robust face detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(11):1408–1423, 2004.

[16] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello. A 240 G-ops/s mobile coprocessor for deep neural networks. In *Proc. Conf. Computer Vis. and Pattern Recog. Workshops*, pages 696–701, 2014.

[17] J. Hoogerbrugge and H. Corporaal. Register file port requirements of transport triggered architectures. In *Proc. Int. Symp. Microarch.*, pages 191–195, 1994.

[18] P. Jääskeläinen, H. Kultala, T. Viitanen, and J. Takala. Code density and energy efficiency of exposed datapath architectures. *J. Signal Process. Syst.*, 80(1):49–64, July 2015.

[19] P. Jääskeläinen, T. Viitanen, J. Takala, and H. Berg. *HW/SW Co-design Toolset for Customization of Exposed Datapath Processors*, pages 147–164. Springer International Publishing, 2017.

[20] N. P. Jouppi, C. Young, N. Patil, P. David, et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. Int. Symp. Computer Architecture*, pages 1–12, 2017.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[22] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi. CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *Proc. Int. Conf. Computer-Aided Design*, pages 694–701, 2011.

[23] K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood. Controlled flight of a biologically inspired, insect-scale robot. *Science*, 340(6132):603–607, 2013.

[24] Movidius. Always-on vision processing unit for mobile applications.

[25] M. Peemen, B. Mesman, and H. Corporaal. Efficiency optimization of trainable feature extractors for a consumer platform. In *Proc. Int. Conf. Adv. Concepts for Intelligent Vis. Systems*, pages 293–304, 2011.

[26] M. Peemen, B. Mesman, and H. Corporaal. Speed sign detection and recognition by convolutional neural networks. In *Proc. Int. Automotive Congress*, pages 162–170, 2011.

[27] M. Peemen, B. Mesman, and H. Corporaal. Inter-tile reuse optimization applied to bandwidth constrained embedded accelerators. In *Proc. Design, Automation & Test in Europe Conf.*, pages 169–174, 2015.

[28] M. Peemen, W. Pramadi, B. Mesman, and H. Corporaal. VLIW code generation for a convolutional network accelerator. In *Proc. Int. Workshop on Software and Compilers for Embedded Systems*, pages 117–120, 2015.

[29] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal. Memory-centric accelerator design for convolutional neural networks. In *Proc. IEEE Int. Conf. Computer Design*, pages 13–19, 2013.

[30] M. Peemen, R. Shi, S. Lal, B. Juurlink, B. Mesman, and H. Corporaal. The neuro vector engine: Flexibility to improve convolutional net efficiency for wearable vision. In *Proc. Design, Automation & Test in Europe Conf.*, pages 1604–1609, 2016.

[31] P. H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello. Neuflow: Dataflow vision processing system-on-a-chip. In *IEEE Int. Midwest Symp. Circuits and Systems*, pages 1044–1047, 2012.

[32] Samsung electronics. Samsung gear2 tech specs.

[33] D. Shin, J. Lee, J. Lee, and H. J. Yoo. 14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In *Proc. IEEE Int. Solid-State Circuits Conf.*, pages 240–241, 2017.

[34] T. Starner. Project glass: An extension of the self. *IEEE Pervasive Computing*, 12(2):14–16, 2013.

[35] D. G. Stork and P. R. Gill. Optical, mathematical, and computational foundations of lensless ultra-miniature diffractive imagers and sensors. *Int. J. Advances in Systems and Measurements*, 7(3):4, 2014.

[36] R. Wood, B. Finio, M. Karpelson, K. Ma, N. Pérez-Arancibia, P. Sreetharan, H. Tanaka, and J. Whitney. Progress on 'pico' air vehicles. *Int. J. Robotics Research*, 31(11):1292–1302, 2012.

[37] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proc. Int. Symp. Field-Programmable Gate Arrays*, pages 161–170, 2015.