The University of Toronto has filed **patent applications** for the mentioned technologies.
**Deep Learning: Where Time Goes?**

**Time:** ~ 60% - 90% $\rightarrow$ inner products

Convolutional Neural Networks: e.g., Image Classification
Deep Learning: Where Time Goes?

Time: ~ 60% - 90% → inner products
SIMD: Exploit Computation Structure

DaDianNao
4K terms/cycle

x256

x16
Our Approach

Improve by Exploiting Value Properties

Maintaining:
- Massive Parallelism
- SIMD Lanes
- Wide Memory Accesses
- No Modifications to the Networks
Longer Term Goal

One Architecture to Rule them All
Value Properties to Exploit? Many $\sim 0$ values
Value Properties to Exploit? Varying Precision Needs
Our Results: Performance

- **Accuracy**
  - CNVLUTIN (ISCA'16): 100% 1.60x
  - TARTAN + STRIPES (MICRO'16): 100% 2.08x
  - PRAGMATIC (Arxiv + ICLR Workshop): 99% 4.3x

vs. DaDianNao which was ~300x over GPUs
Our Results: Memory Footprint and Bandwidth

• Proteus:

44% less memory bandwidth + footprint
Roadmap

- Avoiding computations with ~0
- Performance from precision
- Performance from zero bits
- Reducing footprint and bandwidth
#1: Skipping Ineffectual Activations

\[
\begin{align*}
0 \times A \\
\sim 0 \times A
\end{align*}
\]
Many ineffectual multiplications
Many Activations and Weights are \textbf{Intrinsically} Ineffectual (zero)

- Zero Activations:
  - Runtime calculated
  - \textbf{None} that are \textbf{always} zero

- Zero Weights:
  - Known in Advance
  - Not pursued in this work

\textbf{45\% of Runtime Values are zero}

\% Stable for any Input

\textbf{None always zero}
Many ineffectual multiplications
Many more ineffectual multiplications
Beating Fast and “Dumb” SIMD is Hard

On-the-fly ineffectual product elimination
Performance + energy
Optional: accuracy loss + performance
No Accuracy Loss

+52% performance
-7% power
+5% area

Can relax the ineffectual criterion
better performance: 60%
even more w/ some accuracy loss
Deep Learning: Convolutional Neural Networks

image

layers
10s-100

Beaver
maybe
Deep Learning: Convolutional Networks

- > 60% -- 90% of time in Convolutional Layers
Why are there so many zero neurons?

Hypothesis:

Filter = feature

\( N \)
SIMD: Exploit Computation Structure

DaDianNao
4K terms/cycle

x256

x16
Skipping Ineffectual Activations: Key Challenge

• Processing all Activations:
  • All Lanes operate in lock step
Naïve Solution: No Wide Memory Accesses

- 16 independent narrow activation streams
Removing Zeroes: At the output of each layer

Layer $i$ \[\rightarrow\] encode \[\rightarrow\] Neuron Mem \[\rightarrow\] Layer $i + 1$

Beaver
#1: Partition NM in 16 Slices over 16 Banks

*Processing order does not matter*
#2: Fetch and Maintain One Container per Slice

Container: **up to 16 non-zero neurons**
#3: Keep Neuron Lanes Supplied with One Neuron Per Cycle
#4: When a container is exhausted, get the next one within the slice
Maintaining Wide Accesses But Skipping Zeros

**Container**: stores only non-zeros

**Encoding**: Value, 4-bit offset

Could use 1 extra bit: encoded vs. raw
ZFNAf: Enabling the Skipping of Ineffectual Neurons

- Zero-Free Neuron Array Format:
  - Only non-zero neurons + offsets
  - Brick-level
Cnvlutin: No Accuracy Loss

- Speedup comparison between CNV and CNV + Pruning for models: alex, google, nin, vgg19, cnnM, cnnS, geo

- Better performance indicated by higher speedup values
Loosening the Ineffectual Neuron Criterion

Open Questions:
Are these robust? How to find the best?

Treat Neurons close to zero as zero
#2: Exploiting Precision
Another Property of CNNs

16 bits

Operand Precision Required Fixed? 16 bits?
CNNs: Precision Requirements Vary

Operand Precision Required: Fixed, Varies
5 bits to 13 bits
Execution Time = 16 / P

Performance + Energy Efficiency + Accuracy Knob
Devil in the Details: Carefully chose what to serialize and what to reuse → same input wires as baseline
SIMD: Exploit Computation Structure

DaDianNao
4K terms/cycle

x16
Stripes Bit-Serial Engine
Compensating for Bit-Serial’s Compute Bandwidth Loss

- **Each Tile:**
  - 16 Windows Concurrently – 16 neurons each
  - 16 Filters
  - 16 partial output neurons
No Accuracy Loss

+192% performance

-57% energy

+32% area

More performance w/ accuracy loss

* W/O Older: LeNet + Covnet
Stripes: Performance Boost

![Graph showing speedup comparison between STR and Ideal for various models including lenet, conv, alex, nin, google, vggM, vggS, vgg19, and geo. The y-axis represents speedup, ranging from 0.0 to 3.0, and the x-axis lists the model names. The bar for STR is blue, and the bar for Ideal is green.](image)
• Each Tile:
  • No Weight Reuse
  • Cannot Have 16 Windows
• No Weight Reuse
• Cannot Have 16 Windows
TARTAN: Accelerating Fully-Connected Layers

- **Bit-Parallel Engine**
  - $V$: activation
  - $I$: weight
  - Both 2 bits
Bit-Parallel Engine: Processing one Activation x Weight

• **Cycle 1:**
  • *Activation: a1 and Weight: W*
Bit-Parallel Engine: Processing Another Pair

• Cycle 2:
  - Activation: $a_2$ and Weight: $W$

• $a_1 \times W + a_2 \times W$ over two cycles
• 2 x 1b activation inputs
• 2b or 2 x 1b weight inputs
• Cycle 1: load 2b weight into BRs
TARTAN: Weight x 1\textsuperscript{st} bit of Two Activations

- Cycle 2: Multiply W with bit 1 of activations a1 and a2
• Cycle 3: multiply $W$ with 2nd bit of $a1$ and $a2$
• Load new $W'$ into BR

• 3-stage pipeline to do 2: 2b activation x 2b weight
TARTAN: Fully-Connected Layers: Loading Weights

• What is different? Weights cannot be reused

• Cycle 1: Load first bit of two weights into Ars

Bit 1 of Two Different Weights
• Cycle 2: Load 2\textsuperscript{nd} bit of w1 and w2 into ARs

• Bit 2 of Two Different Weights

• Loaded Different Weights to Each Unit
Cycle 3: Move AR into BR and proceed as before over two cycles

5-stage pipeline to do:
- TWO of (2b activation x 2b weight)
TARTAN: Result Summary

• Bit-Serial TARTAN
  • 2.04x faster than DaDiannao
  • 1.25x more energy efficient at the same frequency
  • 1.5x area overhead

• 2-bit at-a-time TARTAN
  • 1.6x faster over DaDiannao
  • Roughly same energy efficiency
  • 1.25x area overhead
Operand Information Content Varies
• Want to do $A \times B$
• Let’s look at $A$

• Which bits really matter?
- Only 8% of bits are non-zero once precision is reduced
- 15%-10% otherwise
Zero Bit Content: 8-bit Quantized (Tensorflow-like)

- Only 27% of bits are non-zero
Pragmatic Concept: Bit-Parallel Engine

(a) Bit-Parallel Unit
• Simply Modify Stripes?
• Too Large + Cross Lane Synchronization
BIG = 3.7x area overhead just for the datapath
Solution to #1? 2-Stage Shifting

- Process in groups of Max N Difference
- Example with N = 4

- Some opportunity loss, much lower area overhead
- Can skip groups of all zeroes
Solution to #1? 2-Stage Shifting

- Process in groups of Max N Difference
- Example with N = 4

Some opportunity loss, much lower area overhead
Lane Synchronization

- Different # of 1 bits
- Lanes go out of sync
- May have to fetch up to 256 different activations from NM

Keep Lanes Synchronized:

- No cost: All lanes
- Extra register for weights:
  - *Allow columns to advance by 1*
  - *Some cost but much better performance*
Speedup and Energy Efficiency vs. DaDianNao

![Chart showing speedup and energy efficiency for various models compared to DaDianNao. The chart includes categories for Alexnet, NiN, Google, VGGM, VGGS, VGG19, and Avg, with different bars representing different configurations like STR-16b, 16b, 8bQ, IE-16b, IE-8bQ, and EE 16b. The x-axis represents different models, and the y-axis represents the speedup and energy efficiency scores.]
No Accuracy Loss
+310% performance
- 48% Energy
+ 45% Area
Better w/ 8-bit Quantization
4.3x with Encoding
Reducing Memory Footprint and Bandwidth
Proteus

• Operand Precision Required Varies

Proteus: Store in reduced precision in memory
Less Bandwidth, Less Energy
• Weights (synapses) and Data (activations/neurons)

Layered Extension:
Compatible with Existing Systems
Conventional Format: Base Precision

Data Physically aligns with Unit Inputs
Conventional Format: Base Precision

Need Shuffling Network to Route Synapses

4K input bits → Any 4K output bit position
Proteus’ Key Idea: Pack Along Data Lane Columns

Local Shufflers: 16b input 16b output
Much simpler
44% less memory bandwidth
What’s Next

• Training
• Prototype
  • *Design Space: lower-end confs*
• Unified Architecture
  • *Dispatcher + Compute*
  • *Other Workloads: Comp. Photo*
• General Purpose Compute Class
A Value-Based Approach to Acceleration

• More properties to discover and exploit
  • E.g., Filters do overlap significantly

• CNNs one class
  • Other networks
  • Use the same layers
  • Relative importance different

• Training