Scalable High-Performance Architecture for Convolutional Ternary Neural Networks

Frédéric Pétrot, Adrien Prost-Boucle, Alban Bourge

July 7th 2017

Current context

- Software and computers : floating point
- Integers on 16b, 8b, or binary

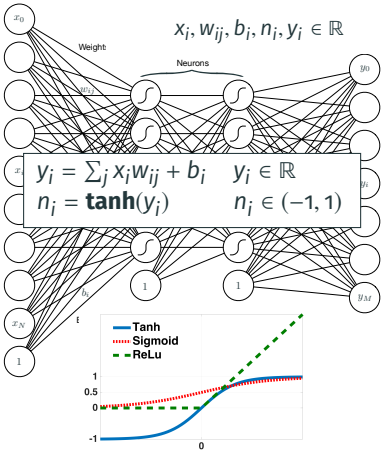
Objectives

- High energy efficient inference for AI tasks
- Without sacrifiying "too much" accuracy

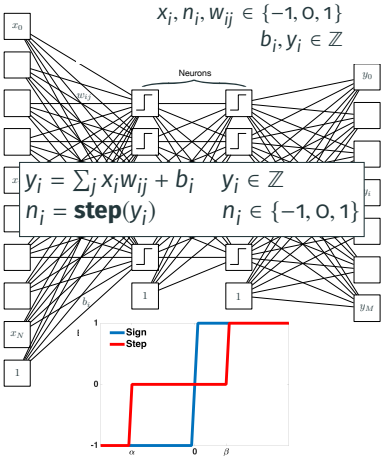
Solution

- Ternarize $\{-1, 0, 1\}$ weights and activations [a]
- Sweet spot between resource usage and accuracy

a. *"Perhaps the prettiest number system of all is the balanced ternary notation."* Donald Knuth, The Art of Computer Programming, Volume 2 : Seminumerical algorithms.



Back-propagation



?

	NN	Teacher	Student
parameters	$\mathbb{R}$	$\mathbb{R}$	$\{-1, 0, 1\}$
neuron input	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{Z}$
activation function	any	any with $(-1, 1)$	2-threshold step
neuron output	$\mathbb{R}$	stochastic firing $\{-1, 0, 1\}$	$\{-1, 0, 1\}$

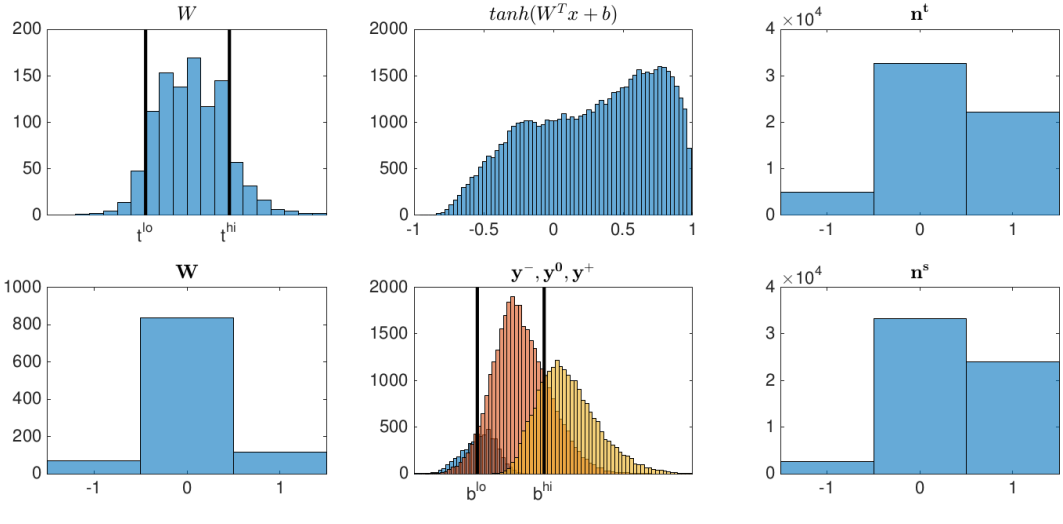
Teacher

$$\rho = \mathbf{tanh}(y_i)$$

$$n_i = \begin{cases} -1 \text{ with prob. } -\rho \text{ if } \rho < 0 \\ 1 \text{ with prob. } \rho \text{ if } \rho > 0 \\ 0 \text{ otherwise} \end{cases}$$

Student

$$n_i = \begin{cases} -1 \text{ if } y_i < \mathbf{b}^{lo} \\ 1 \text{ if } y_i > \mathbf{b}^{hi} \\ 0 \text{ otherwise} \end{cases}$$


W
$tanh(W^T x + b)$
$\mathbf{n^t}$
$\mathbf{W}$
$\mathbf{y^-, y^0, y^+}$
$\mathbf{n^s}$
$\times 10^4$
t^{lo}
t^{hi}
b^{lo}
b^{hi}

Classification Performance - Error Rates (%)

	MNIST	CIFAR10	SVHN	GTRSB	CIFAR100
Fully Discretized					
TNN	1.67	**12.11**	**2.73**	**0.98**	48.40
TrueNorth	7.30	16.59	3.34	3.50	**44.36**
Bitwise NN	**1.33**				
Partially Discretized					
Binarized NN	0.96	10.15	2.53		
BC	1.29	9.90	2.30		
TC	1.15	12.01	2.42		
TWN	0.65	7.44			
EBP	2.20				
XNOR-Net		9.88			
DoReFa-Net			2.40		

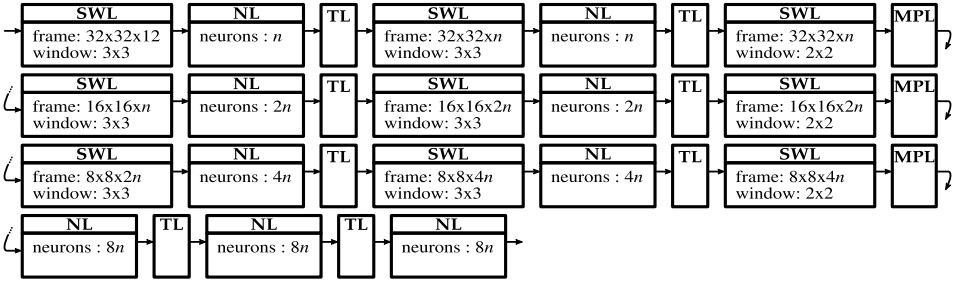
Multiple networks

- VGG-like networks
- two geometries : NN-64 and NN-128
- multiple parallelism inside network (ranging 1 to 64)
- tradeoff area/throughput/energy

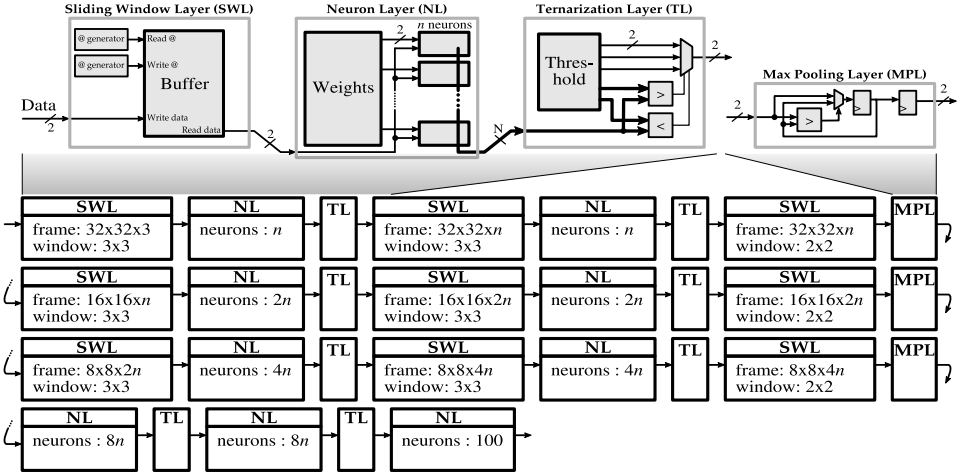
Automation

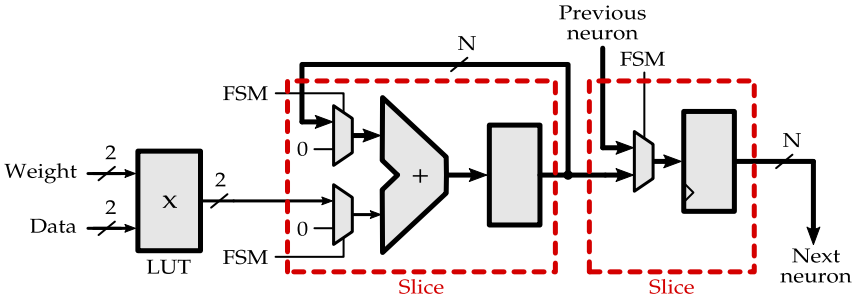
- *handmade* generic hardware building blocks
- automatically generated networks
- customizable home-made tools (old school `C` and `tcl`)

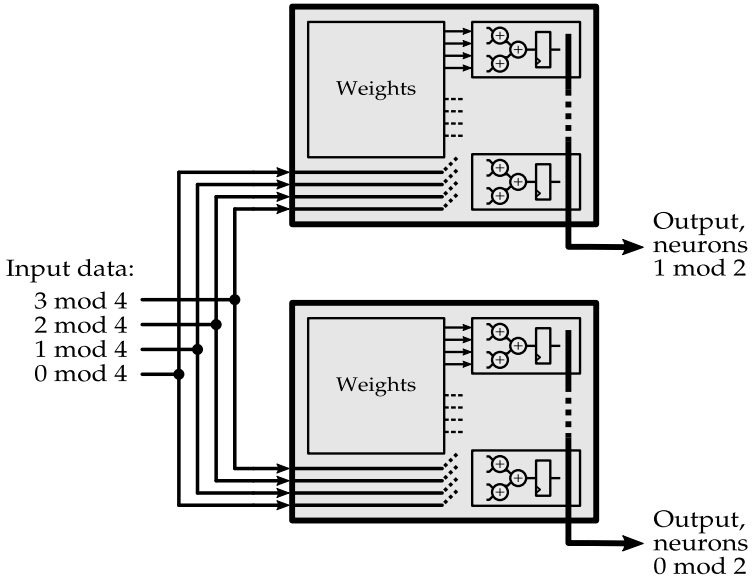
- 12 theoretical layers
- 29 physical layers (+30 "glue" fifos)
- NN64 : 1930 neurons, 3.5 mega parameters
- NN128 : 3850 neurons, 14 mega parameters



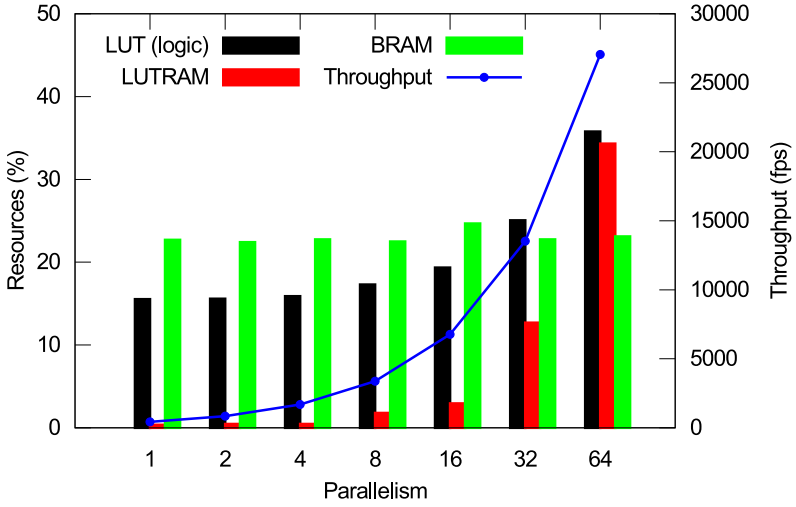
Avantage of ternary : *parameters fit in on-chip memory*

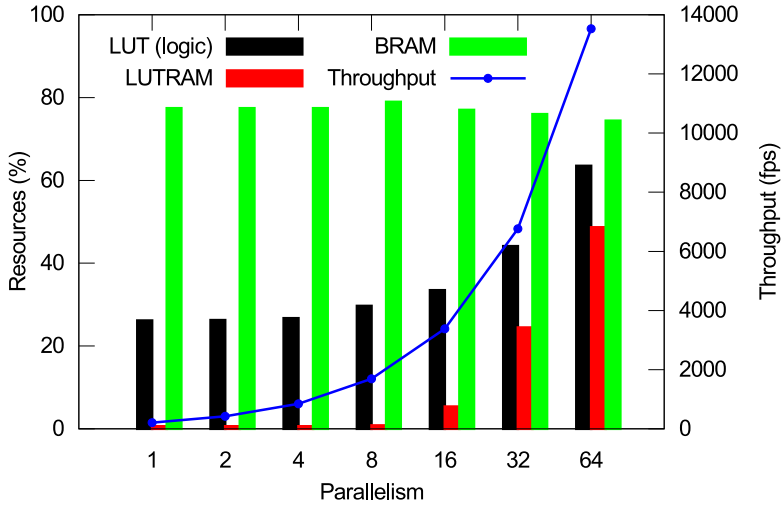

Sliding Window Layer (SWL)
Neuron Layer (NL)
Ternarization Layer (TL)
Max Pooling Layer (MPL)
@ generator
@ generator
Read @
Write @
Buffer
Write data
Read data
Data
2
Weights
n neurons
N
Thres-
hold
SWL
frame: 32x32x3
window: 3x3
NL
neurons : n
TL
SWL
frame: 32x32xn
window: 3x3
NL
neurons : n
TL
SWL
frame: 32x32xn
window: 2x2
MPL
SWL
frame: 16x16xn
window: 3x3
NL
neurons : 2n
TL
SWL
frame: 16x16x2n
window: 3x3
NL
neurons : 2n
TL
SWL
frame: 16x16x2n
window: 2x2
MPL
SWL
frame: 8x8x2n
window: 3x3
NL
neurons : 4n
TL
SWL
frame: 8x8x4n
window: 3x3
NL
neurons : 4n
TL
SWL
frame: 8x8x4n
window: 2x2
MPL
NL
neurons : 8n
TL
NL
neurons : 8n
TL
NL
neurons : 100

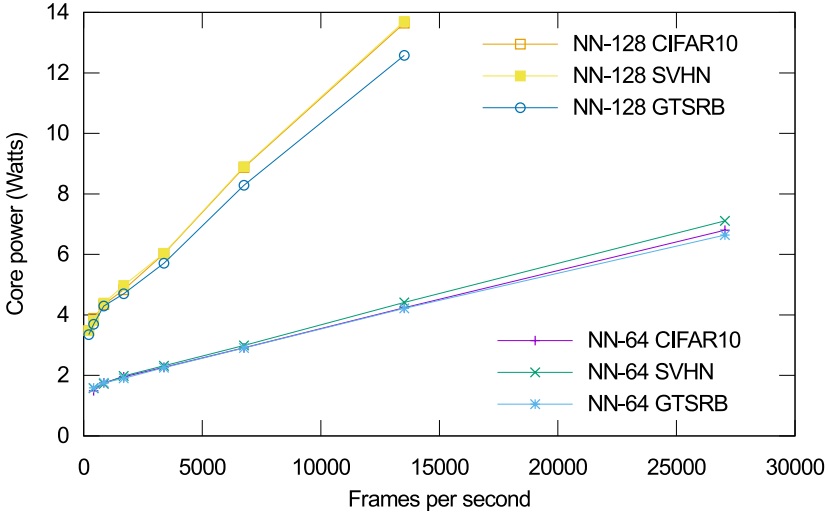

Previous
neuron
N
FSM
FSM
Weight
2
Data
2
X
LUT
2
0
0
+
FSM
N
Next
neuron
Slice
Slice


Weights
Output,
neurons
1 mod 2
Input data:
3 mod 4
2 mod 4
1 mod 4
0 mod 4
Weights
Output,
neurons
0 mod 2

NN size	Par. level	Parallelism per layer (in/out)											
		*NL*1	*NL*2	*MPL*1	*NL*3	*NL*4	*MPL*2	*NL*5	*NL*6	*MPL*3	*NL*7	*NL*8	*NL*9
64	1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1
	2	1 / 1	**2 / 1**	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1
	4	1 / 1	**4 / 1**	1 / 1	1 / 1	**2 / 1**	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1
	8	1 / 1	**8 / 1**	1 / 1	**2 / 1**	**4 / 1**	1 / 1	1 / 1	**2 / 1**	1 / 1	1 / 1	1 / 1	1 / 1
	16	**1 / 2**	**16 / 2**	**2 / 1**	**4 / 1**	**8 / 1**	1 / 1	2 / 1	**4 / 1**	1 / 1	1 / 1	1 / 1	1 / 1
	32	**2 / 4**	**32 / 4**	**4 / 1**	**8 / 2**	**16 / 2**	**2 / 1**	**4 / 1**	**8 / 1**	1 / 1	1 / 1	1 / 1	1 / 1
	64	**3 / 8**	**64 / 8**	**8 / 2**	**16 / 4**	**32 / 4**	**4 / 1**	**8 / 2**	**16 / 2**	**2 / 1**	1 / 1	1 / 1	1 / 1
128	1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1
	2	1 / 1	**2 / 1**	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1
	4	1 / 1	**4 / 1**	1 / 1	1 / 1	2 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1
	8	1 / 1	**8 / 1**	1 / 1	**2 / 1**	**4 / 1**	1 / 1	1 / 1	**2 / 1**	1 / 1	1 / 1	1 / 1	1 / 1
	16	**1 / 2**	**16 / 2**	**2 / 1**	**4 / 1**	**8 / 1**	1 / 1	**2 / 1**	**4 / 1**	1 / 1	1 / 1	1 / 1	1 / 1
	32	**1 / 4**	**32 / 4**	**4 / 1**	**8 / 2**	**16 / 2**	**2 / 1**	**4 / 1**	**8 / 1**	1 / 1	1 / 1	1 / 1	1 / 1
	64	**2 / 8**	**64 / 8**	**8 / 2**	**16 / 4**	**32 / 4**	**4 / 1**	**8 / 2**	**16 / 2**	**2 / 1**	1 / 1	1 / 1	1 / 1


LUT (logic)
LUTRAM
BRAM
Throughput
Resources (%)
Throughput (fps)
Parallelism
0
10
20
30
40
50
0
5000
10000
15000
20000
25000
30000
1
2
4
8
16
32
64


LUT (logic)
LUTRAM
BRAM
Throughput
Resources (%)
Throughput (fps)
Parallelism
0
20
40
60
80
100
0
2000
4000
6000
8000
10000
12000
14000
1
2
4
8
16
32
64



Measured on a VC709 board using PMBus and RTC

Dataset	Platform	NN Arch.	Input quant.	Weight quant.	%err	fps	Power (W)	fps/W	Target
CIFAR10	**This work**	NN-64	3 ch, 8 bits	2 bits	13.29	**27043**	6.80	3976	VC709
	This work	NN-128	3 ch, 8 bits	2 bits	**10.61**	13526	13.64	992	VC709
	FINN	NN-64	24 bits	1 bit	19.90	21900	3.6	**6080**	ZC706
	BCNN	NN-128	3 ch, 6 bits	1 bit	12.20	6218	8.2	758	xc7vx690t
	BNN	NN-128	3 ch, 20 bits	1 bit	11.32	168	4.7	35.8	ZedBoard
	TNN	NN-128	12 ch, 2 bits	2 bits	12.11	1695	9.58	178	VC709
SVHN	**This work**	NN-64	3 ch, 8 bits	2 bits	2.40	**27043**	7.08	3820	VC709
	This work	NN-128	3 ch, 8 bits	2 bits	**2.30**	13526	13.70	987	VC709
	FINN	NN-64	24 bits	1 bit	5.10	21900	3.6	**6080**	ZC706
	TNN	NN-64	12 ch, 2 bits	2 bits	2.73	3390	4.8	709	VC709
GTSRB	**This work**	NN-64	3 ch, 8 bits	2 bits	1.05	**27043**	6.64	**4073**	VC709
	This work	NN-128	3 ch, 8 bits	2 bits	**0.80**	13526	12.57	1076	VC709
	TNN	NN-128	12 ch, 2 bits	2 bits	0.98	1695	9.58	178	VC709

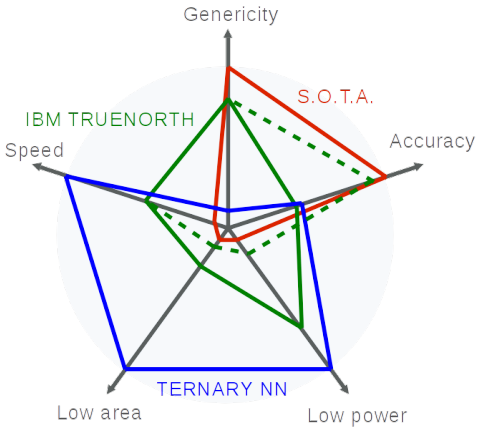
Up to 8.36 TMAC/s and 0.66 TMAC/s/W

By using :

- Simple ternary approach
- Novel training algorithm
- Specialized Hardware

We achieve :

- High area & power efficiency
- High speed
- Good accuracy



Training
→ https ://github.com/slide-lig/tnn-train
Designs
→ http ://tima.imag.fr/sls/research-projects/tnn-fpga-implementation