ALL PROGRAMMABLE
ANY MEDIA
5G
4K/8K
ANY STANDARD
ANY MACHINE
ANY NETWORK
5G Wireless • SDN/NFV • Video/Vision • ADAS • Industrial IoT • Cloud Computing

XILINX
ALL PROGRAMMABLE™

NTNU
Norwegian University of
Science and Technology

THE UNIVERSITY OF
SYDNEY

FINN: A Framework for Fast, Scalable Binarized Neural Network Inference

Yaman Umuroglu (XIR & NTNU), *Nick Fraser (XIR & USydney), Giulio Gambardella (XIR), Michaela Blott (XIR), Philip Leong (USydney), Magnus Jahre (NTNU), Kees Vissers (XSJ)*

Executive Summary

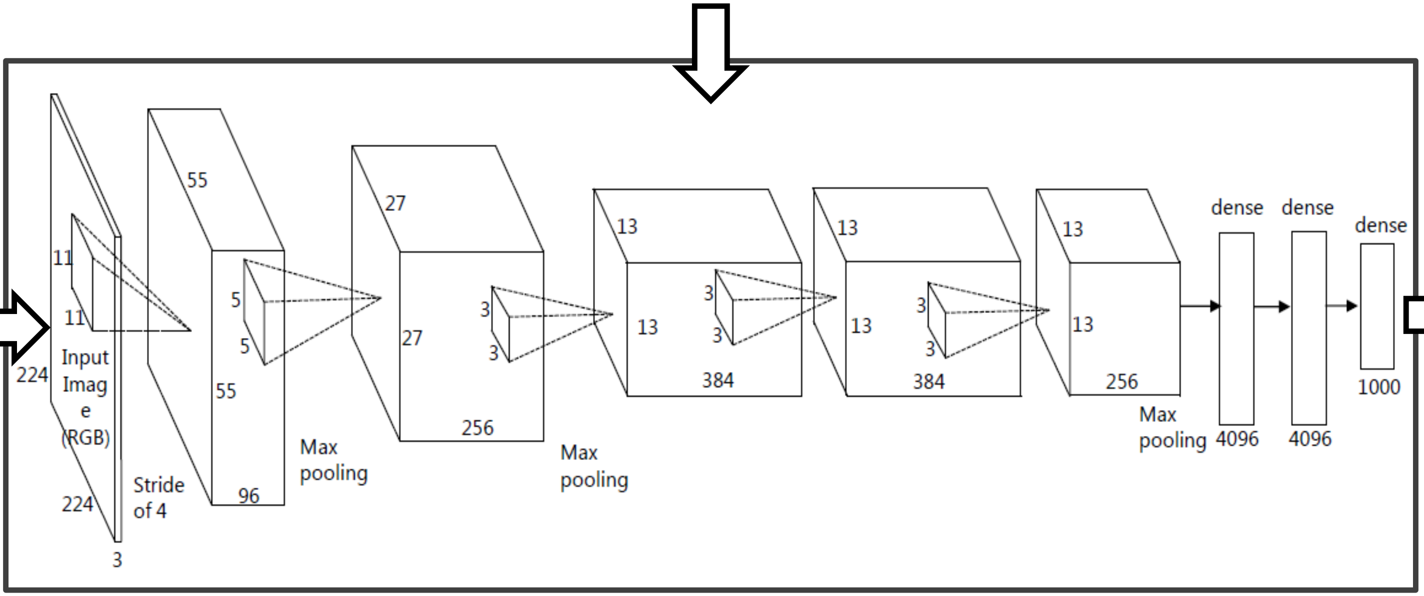
- FPGAs can do **trillions of binary operations per second** & binarized neural nets can put this to good use.
- Inference accelerators that classify **10ks to Ms of images per second, at < 25 W, on today's hardware.**

- Introduction
- Framework & Architecture
- Experimental Evaluation
- Conclusions

Inference with Convolutional Neural Networks (CNNs)

off-chip, tens of megabytes of **floating point** weight data
(from training)

image to be
classified

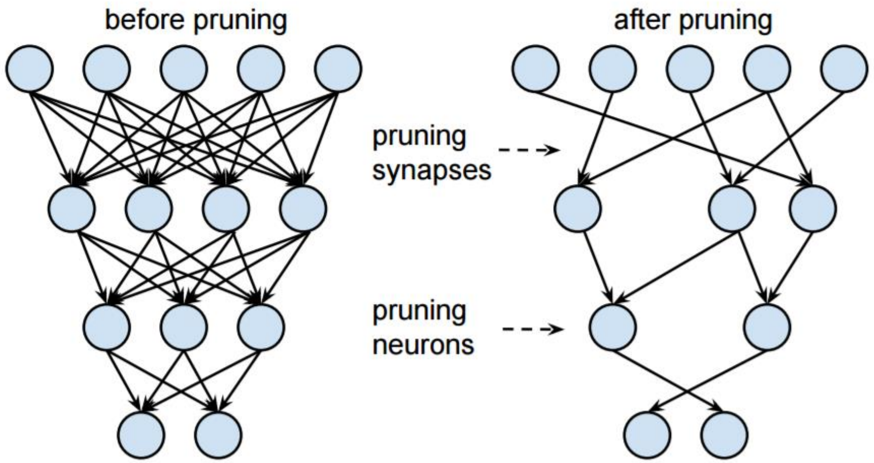


billions of **floating point** multiply-accumulate ops

(up to several joules of energy)

«cat»

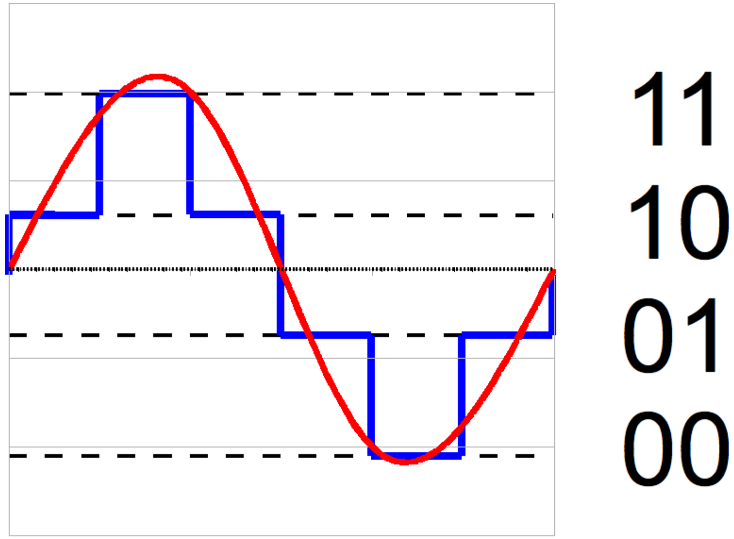
Some Emerging Alternatives for Energy-Efficient Inference



Synapse and neuron pruning

Sparse, irregular computation -- difficult to process efficiently

(Han et al., Learning both Weights and Connections for Efficient Neural Networks)



Quantization

Regular computation, smaller datapaths, fewer bits per weight

Binarized Neural Networks (BNNs)

- **The extreme case of quantization**
  - Permit only two values: +1 and -1
  - Binary weights, binary activations
  - Trained from scratch, not truncated FP

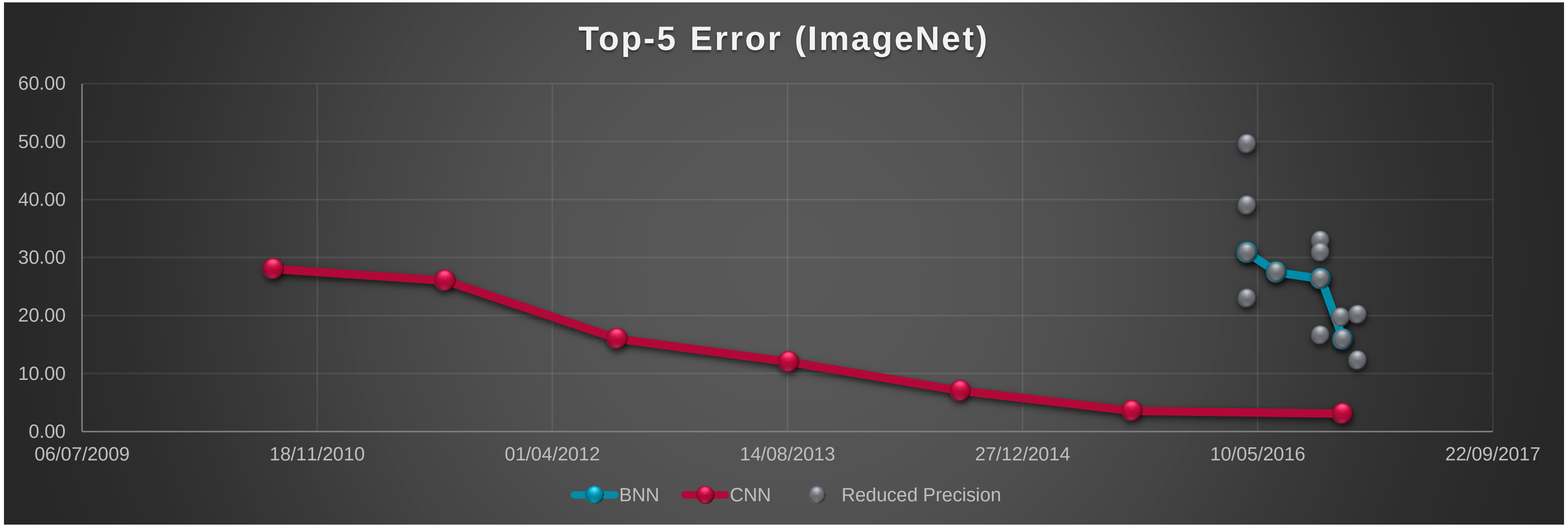
- **Courbariaux and Hubara et al. (NIPS 2016)**
  - Open source training flow
  - Standard “deep learning” layers
    - Convolutions, max pooling, batch norm, fully connected…
  - Competitive results on three smaller benchmarks

	MNIST	SVHN	CIFAR-10
Binary weights & activations	**0.96%**	**2.53%**	**10.15%**
FP weights & activations	**0.94%**	**1.69%**	**7.62%**
BNN accuracy loss	*-0.2%*	*-0.84%*	*-2.53%*

% classification error
(lower is better)

Accuracy of BNNs on ImageNet

Published Results for FP CNNs, BNNs and Extreme Reduced Precision NNs



- BNNs are new and accuracy results are improving rapidly

Potential of BNNs on FPGAs

- ***Much* smaller datapaths**
  - Multiply becomes XNOR, addition becomes popcount
  - No DSPs needed, everything in LUTs
  - Lower cost per op = more ops every cycle, **trillions** of ops per second
- ***Much* smaller weights**
  - Large networks can fit entirely into on-chip memory (OCM)
  - More bandwidth, less energy compared to off-chip

Xilinx UltraScale+ MPSoC ZU19EG
(Vivado HLS, conservative estimates)

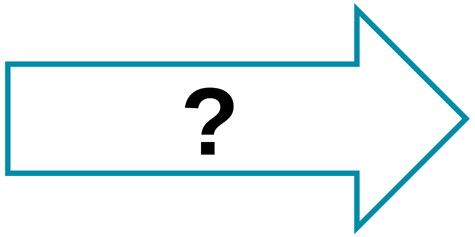
Precision	Peak GOPS	On-chip weights
1b	~66 000	~70 M
8b	~4 000	~10 M
16b	~1 000	~5 M
32b	~300	~2 M

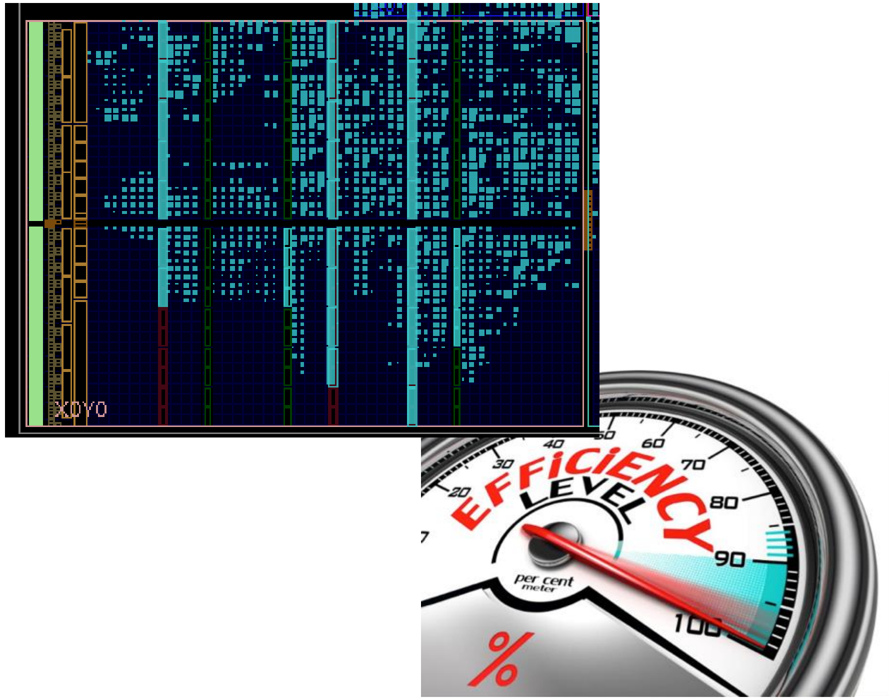
200x

30x

= potential for **blazing fast** inference with **large BNNs** *on today's hardware*

How do we exploit this potential?

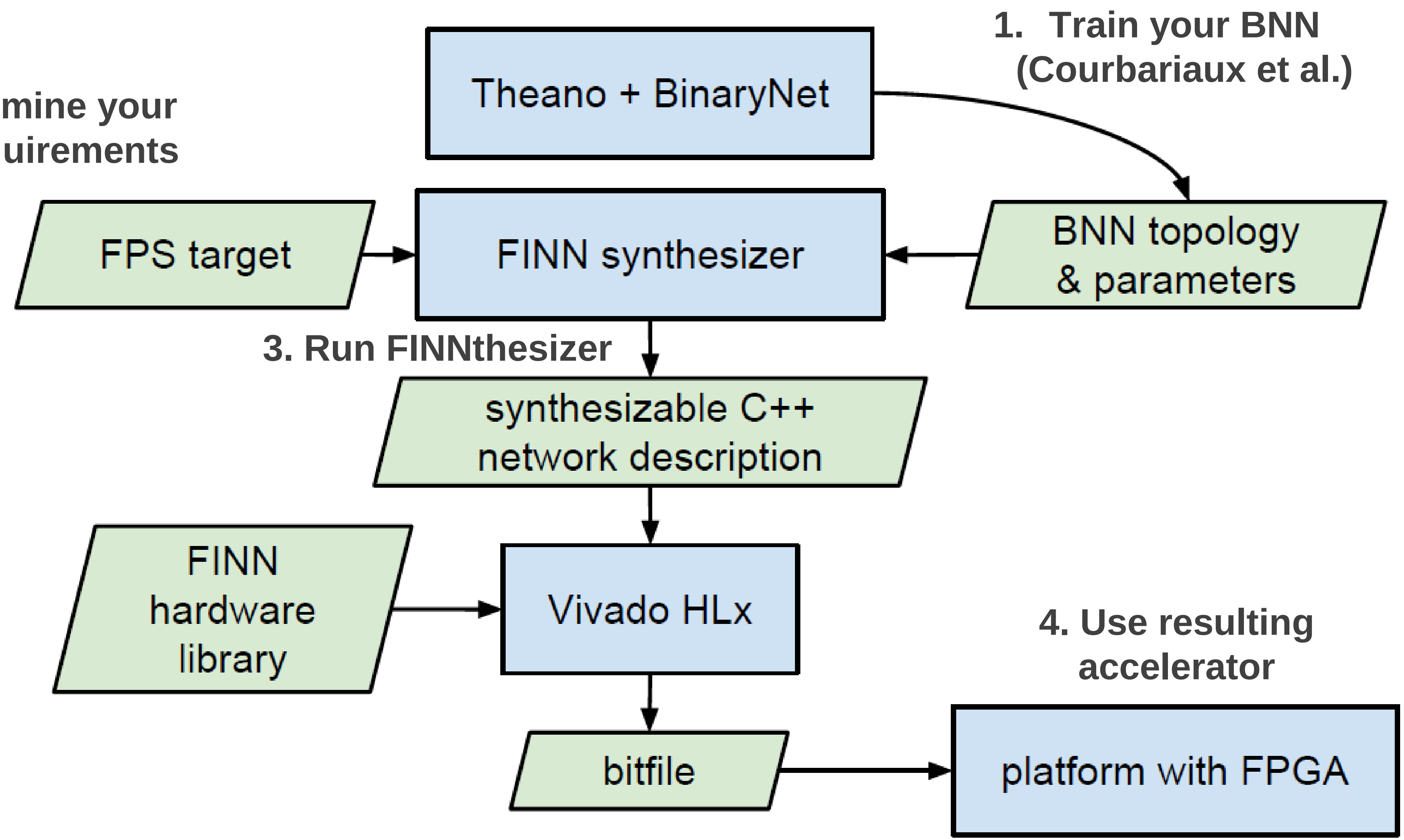




FINN, a Framework for Fast, Scalable Binarized Neural Network Inference

- Introduction
- Framework & Architecture
- Experimental Evaluation
- Conclusions

FINN at a glance

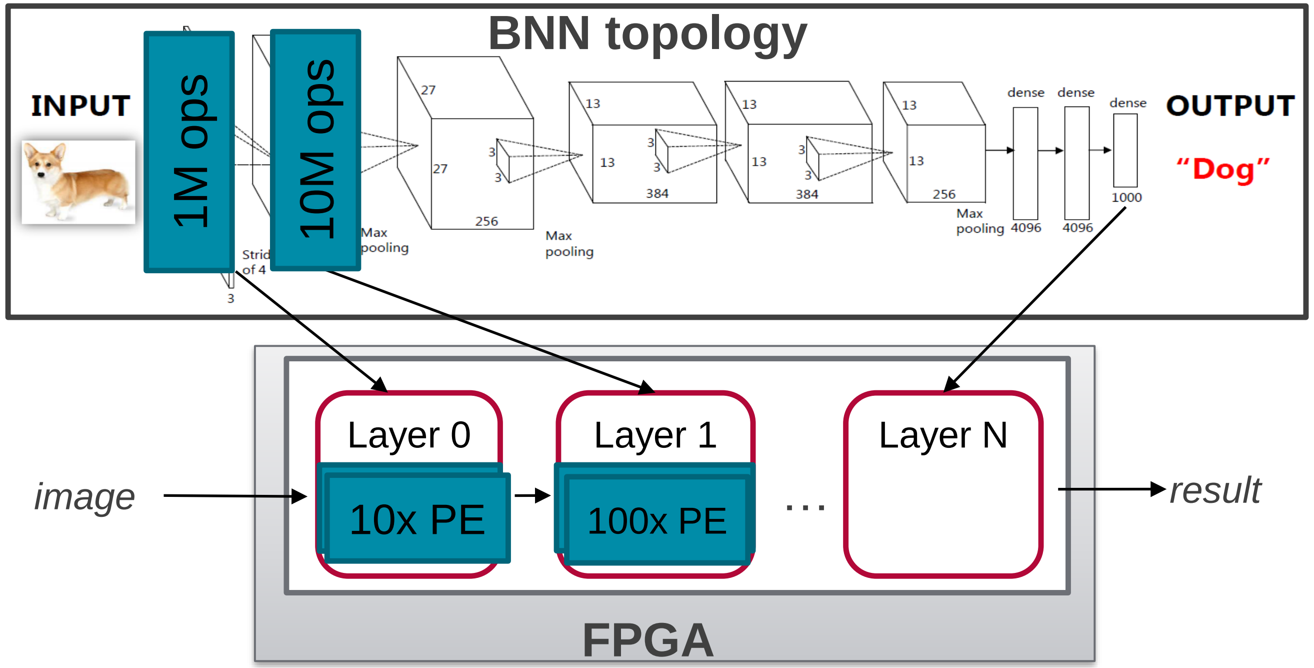


1. Train your BNN (Courbariaux et al.)
2. Determine your FPS requirements
Theano + BinaryNet
FPS target
FINN synthesizer
BNN topology & parameters
3. Run FINNthesizer
synthesizable C++ network description
FINN hardware library
Vivado HLx
4. Use resulting accelerator
bitfile
platform with FPGA

FINN Design Principles

- **One size does not fit all**
  - Generate tailored hardware for network and use-case
- **Stay on-chip**
  - Higher energy efficiency and bandwidth
- **Support portability and rapid exploration**
  - Vivado HLS (High-Level Synthesis)
- **Simplify with BNN-specific optimizations**
  - Exploit “compile time” optimizations to simplify the generated hardware
  - E.g. batchnorm and activation => thresholding. See details in the paper

Heterogeneous Streaming Architecture



1x FPS

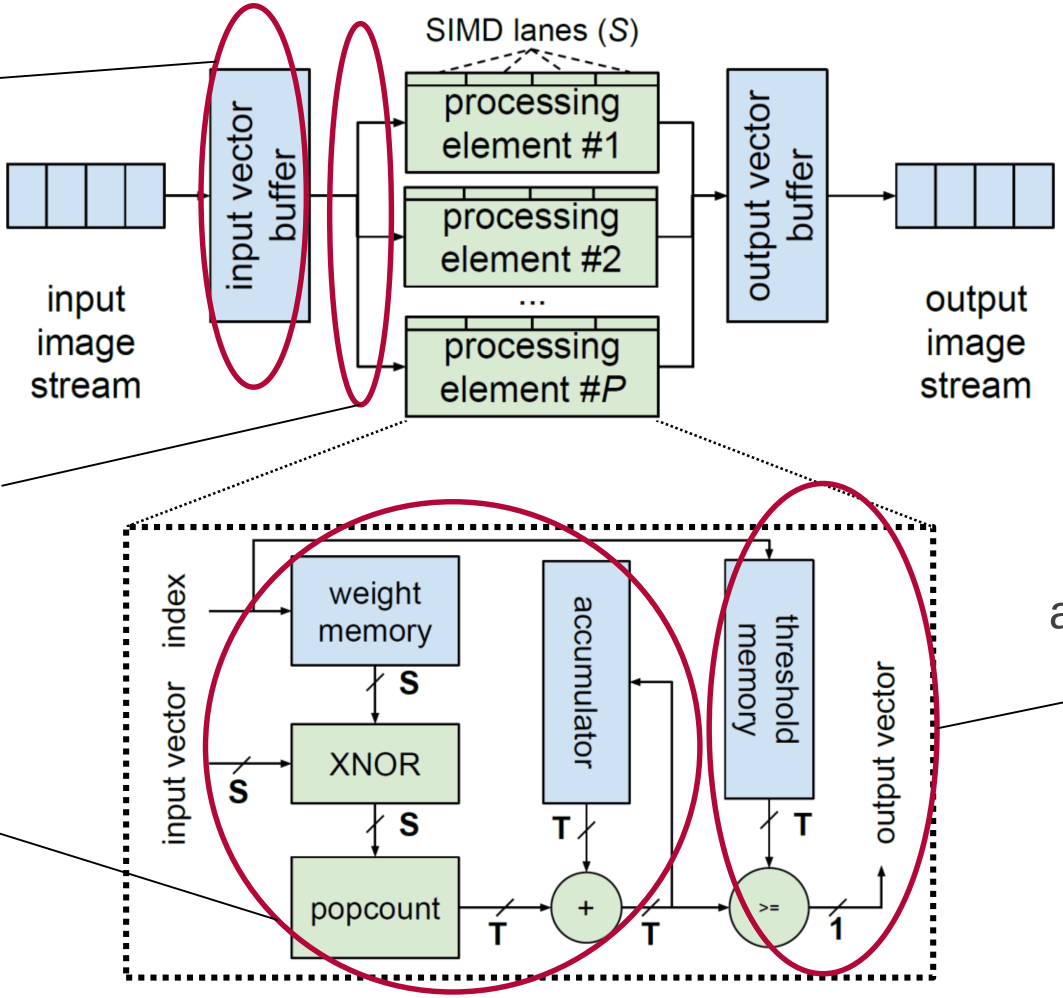
10x FPS

- **One hardware layer per BNN layer, parameters built into bitstream**
  - – Both inter- and intra-layer parallelism
- **Heterogeneous: Avoid "one-size-fits-all" penalties**
  - – Allocate compute resources according to FPS and network requirements
- **Streaming: Maximize throughput, minimize latency**
  - – Overlapping computation and communication, batch size = 1

The Matrix-Vector Threshold Unit (MVTU)

- **Core computational element of FINN, tiled matrix-vector multiply**
- **Computes a (P) row x (S) column chunk of matrix every cycle, per-layer configurable tile size**

buffer input for reuse
(neuron folding)



broadcast S bits of
input every cycle

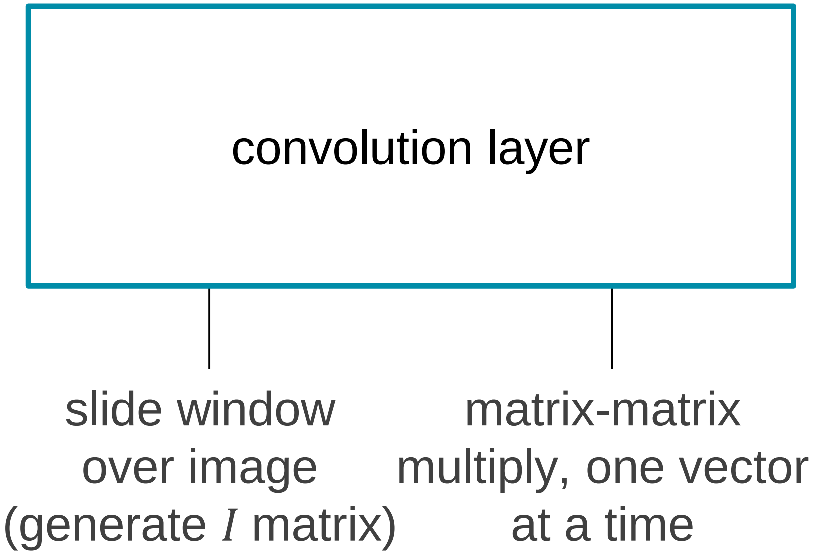
apply thresholding and send
to next layer

multiply-accumulate, but with
XNOR and popcount instead

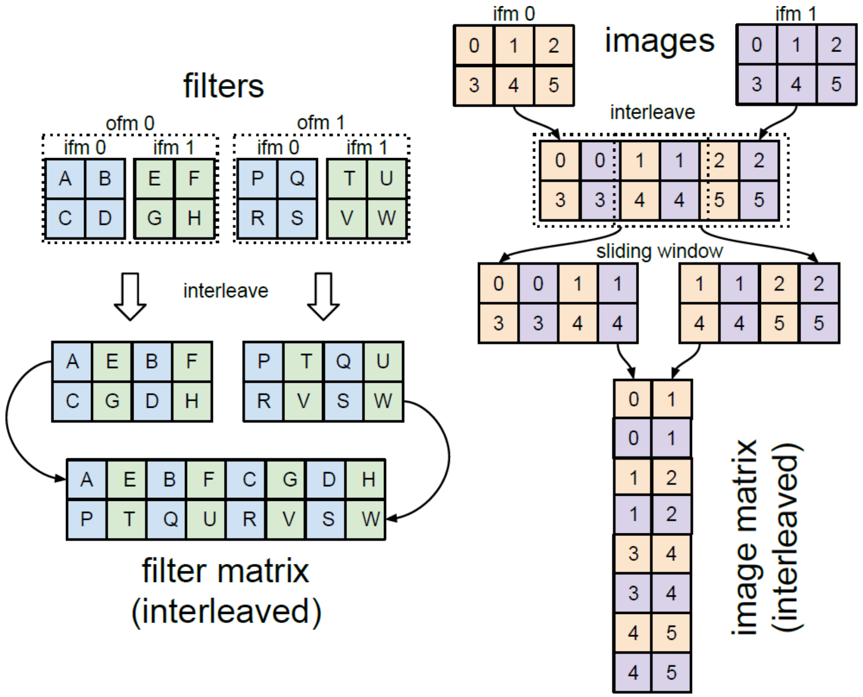
Convolutional Layers

- **Lower convolutions to matrix-matrix multiplication, $W \cdot I$**
  - W : filter matrix (generated offline)
  - I: image matrix (generated on-the-fly)

- **Two components:**



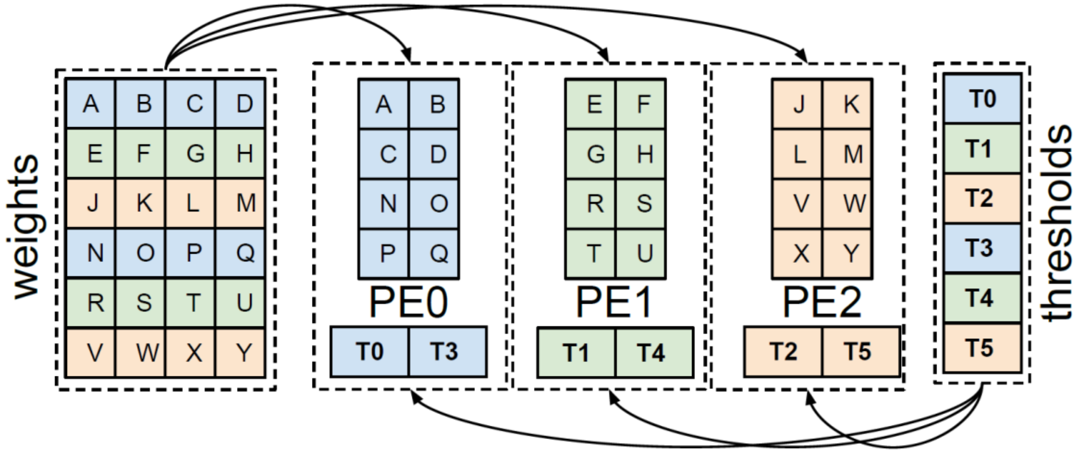
convolution layer
slide window over image (generate I matrix)
matrix-matrix multiply, one vector at a time



Folding

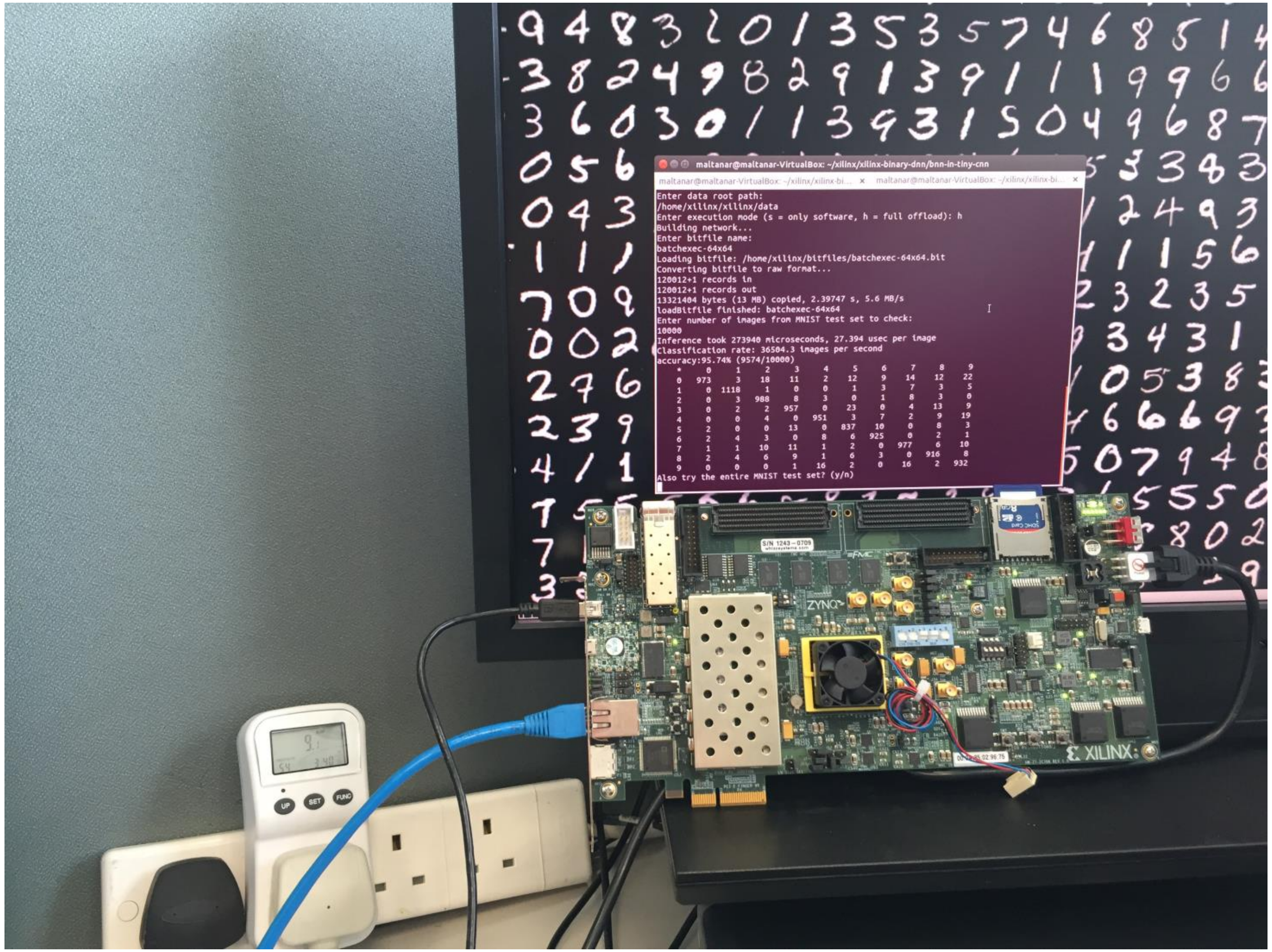
- **Time-multiplex (or *fold*) real neurons onto hardware neurons**
  - Control folding via number of PEs and SIMD lanes in each layer

- **Folding computed by FINNthesizer to satisfy FPS requirements**
  - FPS for one layer = clock frequency / folding factor
  - FPS of streaming system = minimum FPS of any layer
  - FINNthesizer will balance folding factors to match FPS across layers



- Introduction
- Framework & Architecture
- Experimental Evaluation
- Conclusions

Experimental Setup



```
maltanar@maltanar-VirtualBox: ~/xilinx/xilinx-binary-dnn/bnn-in-tiny-cnn
Enter data root path:
/home/xilinx/xilinx/data
Enter execution mode (s = only software, h = full offload): h
Building network...
Enter bitfile name:
batchexec-64x64
Loading bitfile: /home/xilinx/bitfiles/batchexec-64x64.bit
Converting bitfile to raw format...
120012+1 records in
120012+1 records out
13321404 bytes (13 MB) copied, 2.39747 s, 5.6 MB/s
loadBitfile finished: batchexec-64x64
Enter number of images from MNIST test set to check:
10000
Inference took 273940 microseconds, 27.394 usec per image
Classification rate: 36504.3 images per second
accuracy:95.74% (9574/10000)
    *     0     1     2     3     4     5     6     7     8     9
    0   973     3    18    11     2    12     9    14    12    22
    1     0  1118     1     0     0     1     3     7     3     5
    2     0     3   988     8     3     0     1     8     3     0
    3     0     2     2   957     0    23     0     4    13     9
    4     0     0     4     0   951     3     7     2     9    19
    5     2     0     0    13     0   837    10     0     8     3
    6     2     4     3     0     8     6   925     0     2     1
    7     1     1    10    11     1     2     0   977     6    10
    8     2     4     6     9     1     6     3     0   916     8
    9     0     0     0     1    16     2     0    16     2   932
Also try the entire MNIST test set? (y/n)
```

ZC706 development platform:
Z7045 All-Programmable SoC
2 ARM Cortex-A9 cores
218k LUTs, 545 BRAMs

- **10000 test images in PS DDR**
  - Streamed in-out via DMA
- **FINN-generated accelerator on PL**
  - Running at 200 MHz
- **ARM core:**
  - launches accelerator
  - measures time
  - verifies results
- **PMBus and wall power monitoring**
  - Idle wall power ~7 W

Test Networks & Scenarios

«CNV-fix»

BNN Topology:

- **SFC:** small fully-connected, 0.6 MOP per image
- **LFC:** large fully-connected, 5.8 MOP per image
- **CNV:** convolutional, 112.5 MOP per image

- SFC & LFC on MNIST, binarized inputs and outputs
- CNV on CIFAR-10 and SVHN, 8-bit inputs, 16-bit outputs

Scenario:

- **fix:** assume I/O bound, achieve 9000 FPS
- **max:** achieve as high FPS as possible

Results – Maximum Throughput

Prototype	FPS	GOPS	BRAM	LUT	Latency [us]	Power [W]
SFC-max	**12.3 M**	8 265	4.5	91 131 (42%)	**0.31**	21.2
LFC-max	**1.5 M**	9 085	396	82 988 (38%)	**2.44**	22.6
CNV-max	**21.9 k**	2 465	186	46 253 (21%)	**283**	11.7

Unprecedented classification rates

Ultra-low latency
For robotics, AR, UAVs

Results – 9k FPS target

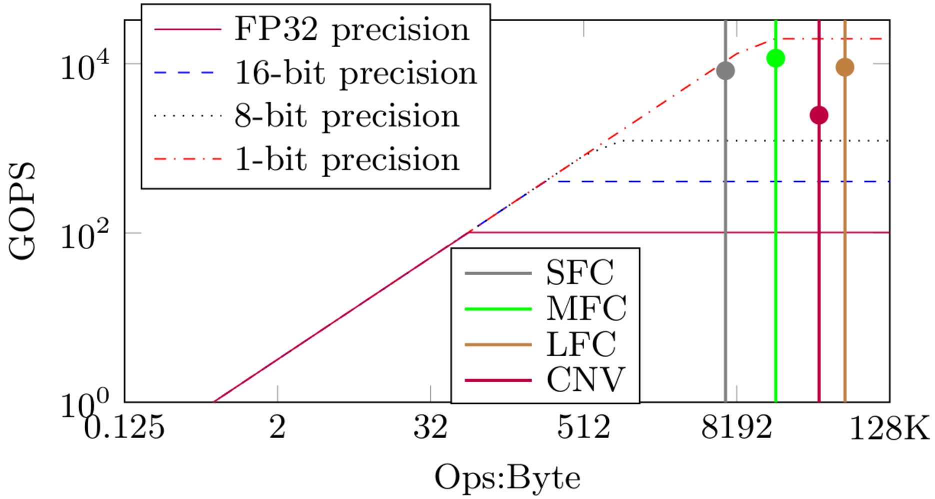
12 kFPS with ~1-3 W over idle power

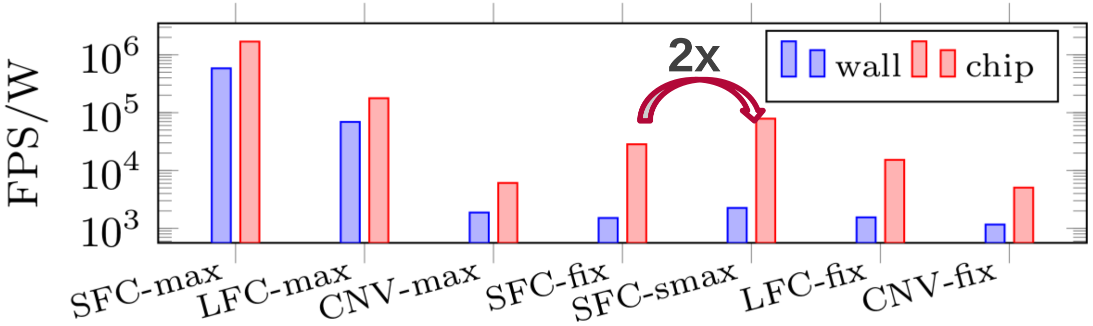
Prototype	FPS	GOPS	BRAM	LUT	Latency [us]	Power [W]
SFC-fix	12.2 k	8	16	**5 155 (3%)**	240	**8.1**
LFC-fix	12.2 k	71	114.5	**5 636 (3%)**	282	7.9
CNV-fix	11.6 k	1 306	152.5	29 274 (13%)	550	**10**

FPS goal exceeded (integer folding factors)

Scalability to small footprints

Results – Other Highlights





- **Up to 58% of roofline performance estimate**
  - SFC-max: DRAM bandwidth-bound
  - LFC-max: resource bound (BRAM)
  - CNV-max: architecture bound (SWU)

- **Massive but slow-clock parallelism: good energy efficiency**
  - Use 250 kHz clock for 12M FPS prototype: 15 kFPS on MNIST with 0.2 W chip power
  - Observed that slowed-down SFC-max 2x more energy efficient than SFC-fix

Comparison to Prior Work

- **How to compare neural network accelerators across precisions and devices?**
  - Accuracy, images per second, energy efficiency

		Accuracy	FPS	Power (chip)	Power (wall)	kFPS / Watt (chip)	kFPS / Watt (wall)	Precision
FINN	MNIST, SFC-max	95.8%	12.3 M	7.3 W	21.2 W	1693	583	1
	MNIST, LFC-max	98.4%	1.5 M	8.8 W	22.6 W	177	269	1
	CIFAR-10, CNV-max	80.1%	21.9 k	3.6 W	11.7 W	6	2	1
	SVHN, CNV-max	94.9%	21.9 k	3.6 W	11.7 W	6	2	1
Prior Work	MNIST, Alemdar et al.	97.8%	255.1 k	0.3 W	-	806	-	2
	CIFAR-10, TrueNorth	83.4%	1.2 k	0.2 W	-	6	-	1
	SVHN, TrueNorth	96.7%	2.5 k	0.3 W	-	10	-	1

Max accuracy loss: ~3%

10 – 100x better performance

CIFAR-10/SVHN energy efficiency comparable to TrueNorth ASIC

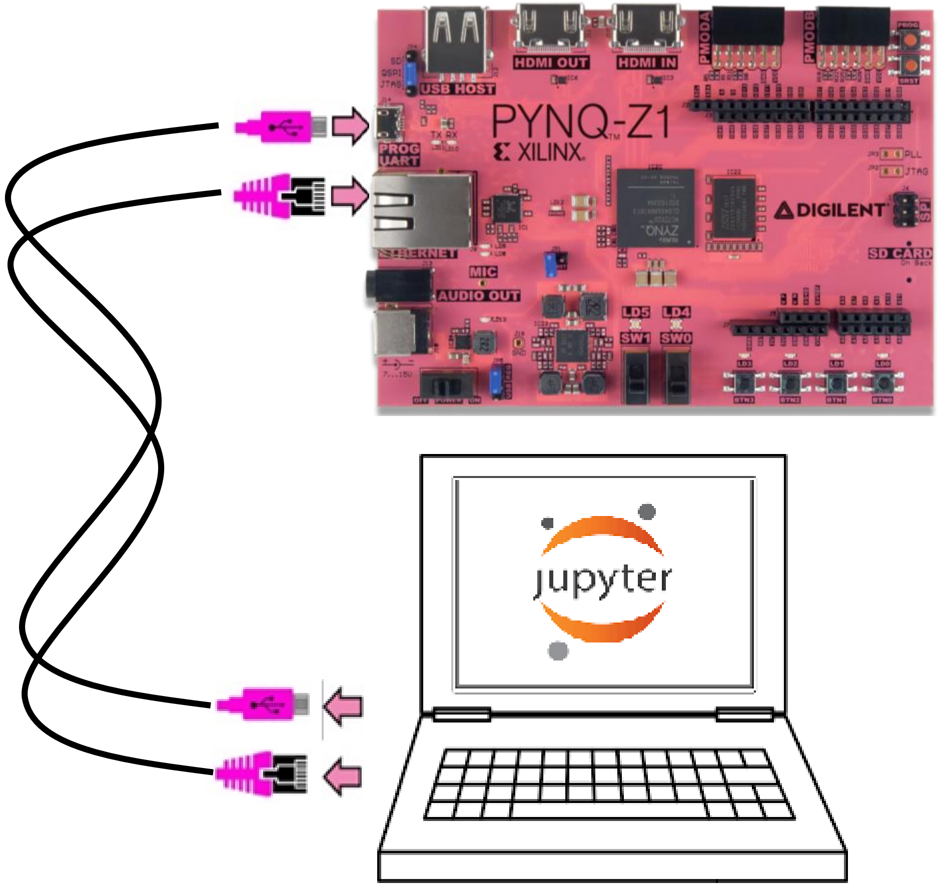
- Introduction
- Framework & Architecture
- Experimental Evaluation
- Conclusions

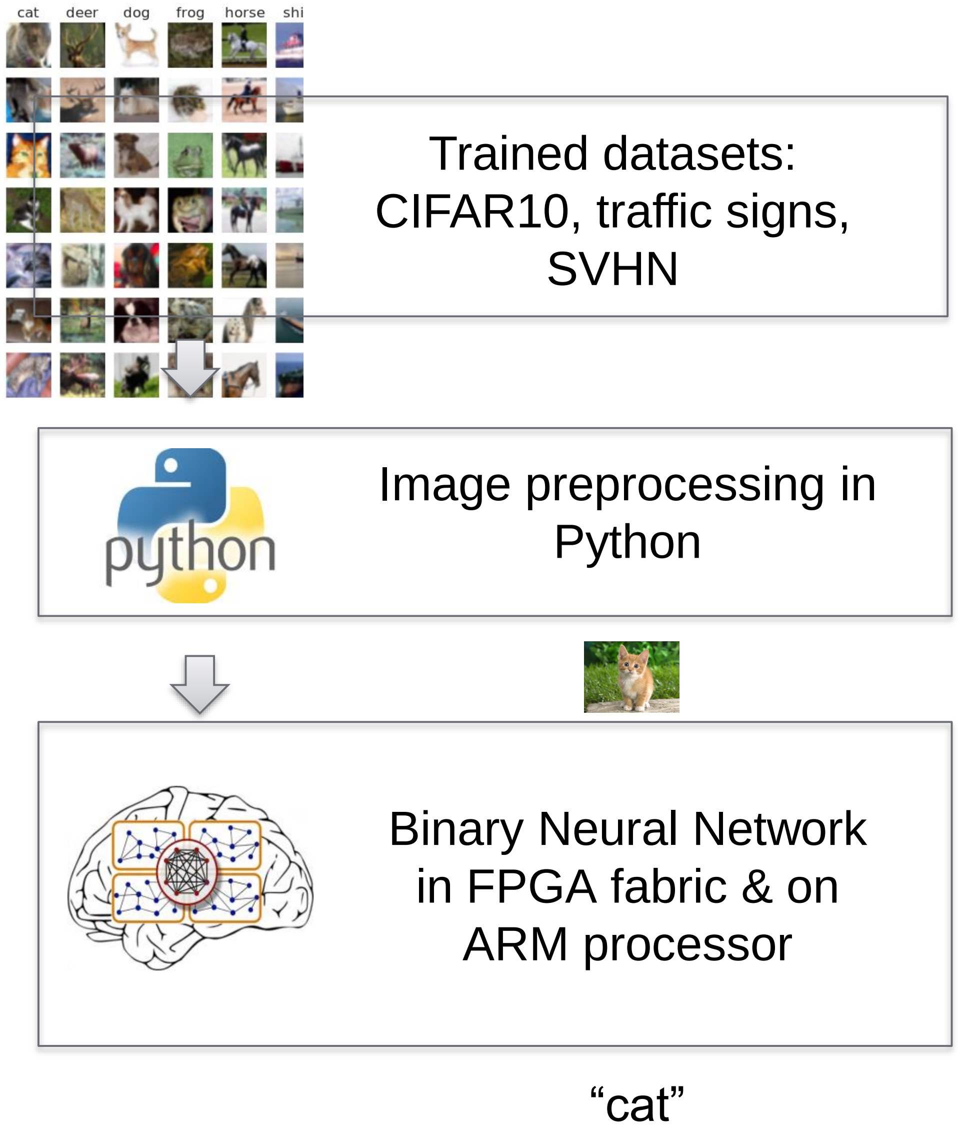
Conclusions

- FPGAs can do **trillions of binary operations per second.**
- FINN can build BNN inference accelerators that classify **10Ks to Ms of images per second**, at < 25 W, on **today's hardware.**

- Future work:
  - Non-binary low precision and mixed precision
  - Support external memory when parameters don't fit in OCM
  - BNNs on ImageNet

BNN – Demo on Xilinx's Python Productivity Kit PYNQ





Trained datasets: CIFAR10, traffic signs, SVHN

Image preprocessing in Python

Binary Neural Network in FPGA fabric & on ARM processor

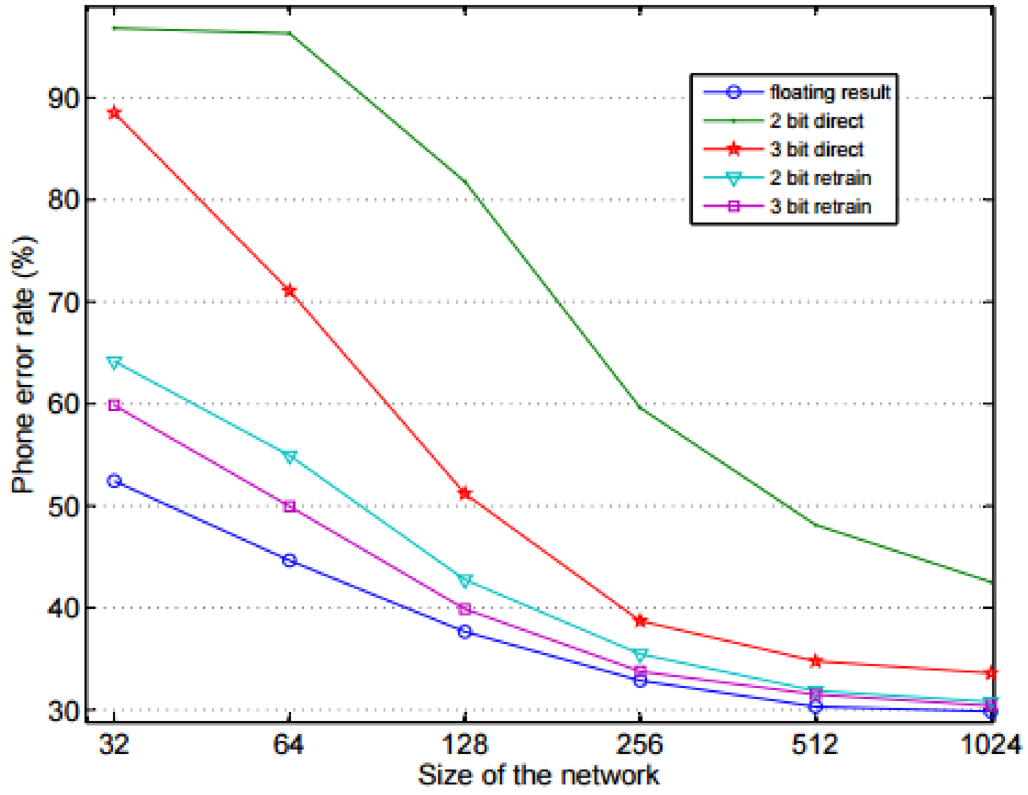
"cat"

Come see the demo at the Xilinx booth!
Open source release coming soon

- Thank You!

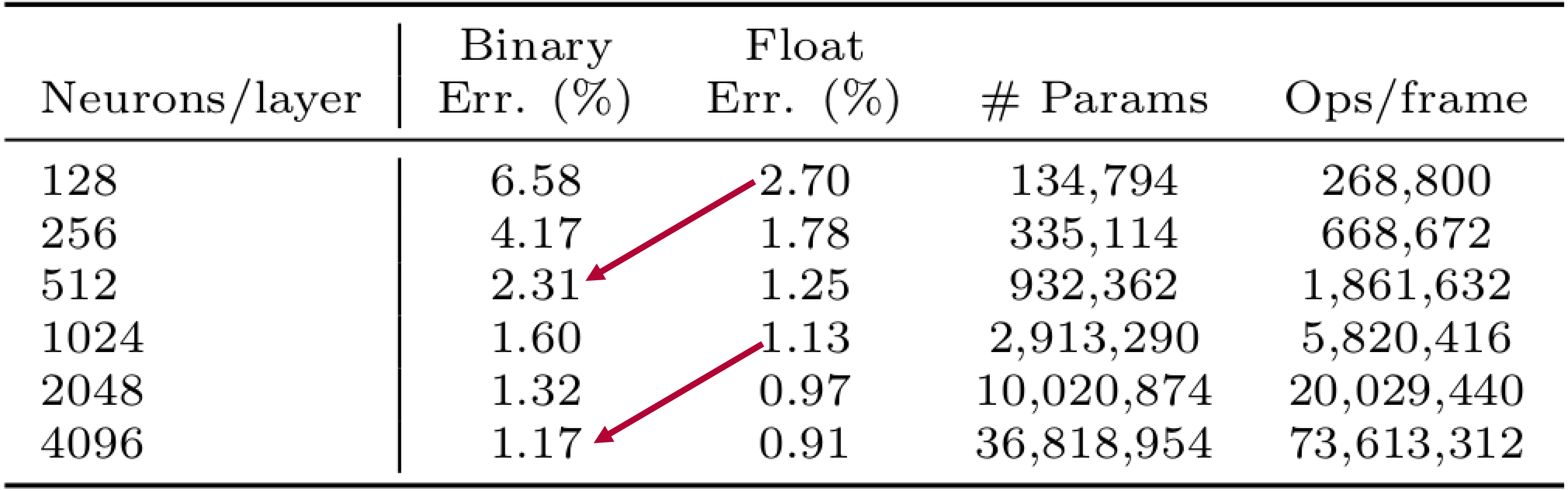
Redundancy and Quantization

- **Evidence of redundancy in trained networks**
  - sparsification, low-rank approximations, fault tolerance…
- **Reduced precision (quantization)**
  - Restrict weights and/or activations to *Q*-bit values
  - HW benefits: Low-bitwidth datapaths, regular compute
- **Sung et al: Quantization works well when…**
  - …the network is "big enough"
  - …the network is aware of quantization during (re)training



"(…) the performance gap between the floating-point and the retrain-based ternary (+1, 0, -1) weight neural networks (…) almost vanishes in fully complex networks (…)"
(Sung et al, Resiliency of Deep NNs Under Quantization)

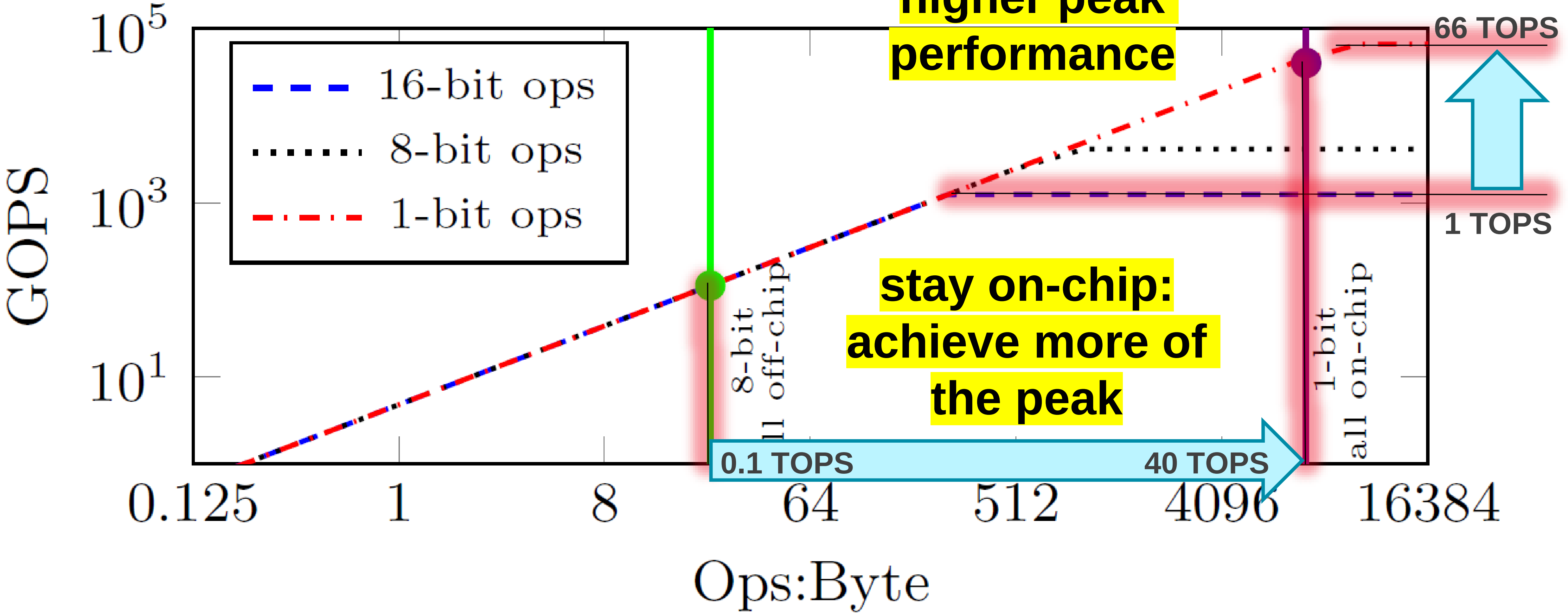
# Neurons versus Accuracy – Float and Binarized



Neurons/layer	Binary Err. (%)	Float Err. (%)	# Params	Ops/frame
128	6.58	2.70	134,794	268,800
256	4.17	1.78	335,114	668,672
512	2.31	1.25	932,362	1,861,632
1024	1.60	1.13	2,913,290	5,820,416
2048	1.32	0.97	10,020,874	20,029,440
4096	1.17	0.91	36,818,954	73,613,312

~2x binary neurons give approximately the same accuracy (for MNIST)

Potential of BNNs on FPGAs



fewer LUTs/op: higher peak performance

stay on-chip: achieve more of the peak

FINN Synthesizer («FINNthesizer»)

- **Inputs:**
  - BNN topology (JSON) and trained parameters (NPZ)
  - Desired frames per second (FPS)
- **Apply BNN-specific compute transformations**
  - Simplifications enabled by the value-constrained nature of BNNs
  - Popcount, batchnorm-activation as threshold, maxpool as OR (details in paper)
- **Compute «folding factors» to meet FPS goal**
- **Output:**
  - C++ (Vivado HLS) description of desired architecture

Top Level

```
void DoCompute(ap_uint<64> * in, ap_uint<64> * out) {
#pragma HLS DATAFLOW
   stream<ap_uint<64> > memInStrm("memInStrm");
   stream<ap_uint<64> > InStrm("InStrm");
          .
          .
          .
   stream<ap_uint<64> > memOutStrm("memOutStrm");

   Mem2Stream<64, inBytesPadded>(in, memInStrm);
   StreamingMatrixVector<L0_SIMD, L0_PE, 16, L0_MW, L0_MH, L0_WMEM, L0_TMEM>
          (InStrm, inter0, weightMem0, thresMem0);
   StreamingMatrixVector<L1_SIMD, L1_PE, 16, L1_MW, L1_MH, L1_WMEM, L1_TMEM>
          (inter0, inter1, weightMem1, thresMem1);
   StreamingMatrixVector<L2_SIMD, L2_PE, 16, L2_MW, L2_MH, L2_WMEM, L2_TMEM>
          (inter1, inter2, weightMem2, thresMem2);
   StreamingMatrixVector<L3_SIMD, L3_PE, 16, L3_MW, L3_MH, L3_WMEM, L3_TMEM>
          (inter2, outstream, weightMem3, thresMem3);
    StreamingCast<ap_uint<16>, ap_uint<64> >(outstream, memOutStrm);
    Stream2Mem<64, outBytesPadded>(memOutStrm, out);
}
```

Stream definitions

Move image in from PS memory

Layer instantiation
connected by streams

Move results to PS memory

MVTU

```
for (unsigned int nm = 0; nm < neuronFold; nm++) {
    for (unsigned int sf = 0; sf < synapseFold; sf++) {
#pragma HLS PIPELINE II=1
          ap_uint<SIMDWidth> inElem;
          if (nm == 0) {
             inElem = in.read();
             inputBuf[sf] = inElem;
          } else {
             inElem = inputBuf[sf];
          }
          for (unsigned int pe = 0; pe < PECount; pe++) {
#pragma HLS UNROLL
             ap_uint<SIMDWidth> weight = weightMem[pe][nm * synapseFold + sf];
             ap_uint<SIMDWidth> masked = ~(weight ^ inElem);
             accPopCount[pe] += NaivePopCount<SIMDWidth, PopCountWidth>(masked);
          }
    }
    ap_uint<PECount> outElem = 0;
    for (unsigned int pe = 0; pe < PECount; pe++) {
#pragma HLS UNROLL
          outElem(pe, pe) = accPopCount[pe] > thresMem[pe][nm] ? 1 : 0;
          accPopCount[pe] = 0;          // clear the accumulator
    }
```

Folding

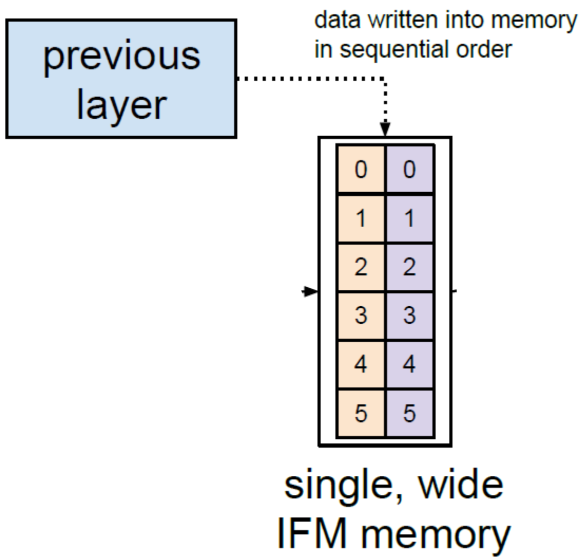
Reading
Inputs or consume
internal (when folded)

Indexing weight and
threshold memory
binary MAC

Batchnorm & activation
as threshold

Convolution: Sliding Window Unit (SWU)

- **Buffer incoming images in a single, #IFM-wide memory**
- **Read out addresses corresponding to sliding window location**
- **Preserve produce-consume order to minimize buffering**



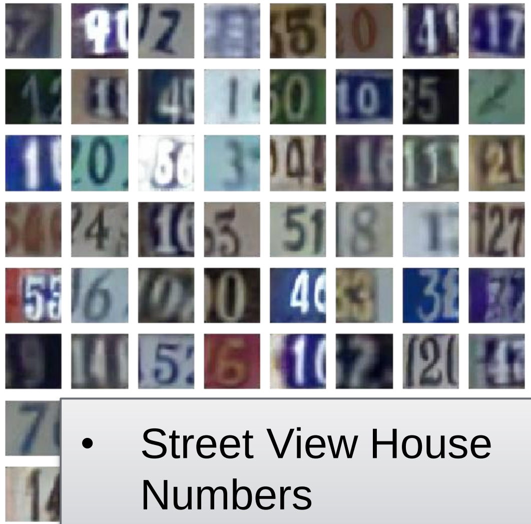
previous layer

data written into memory in sequential order

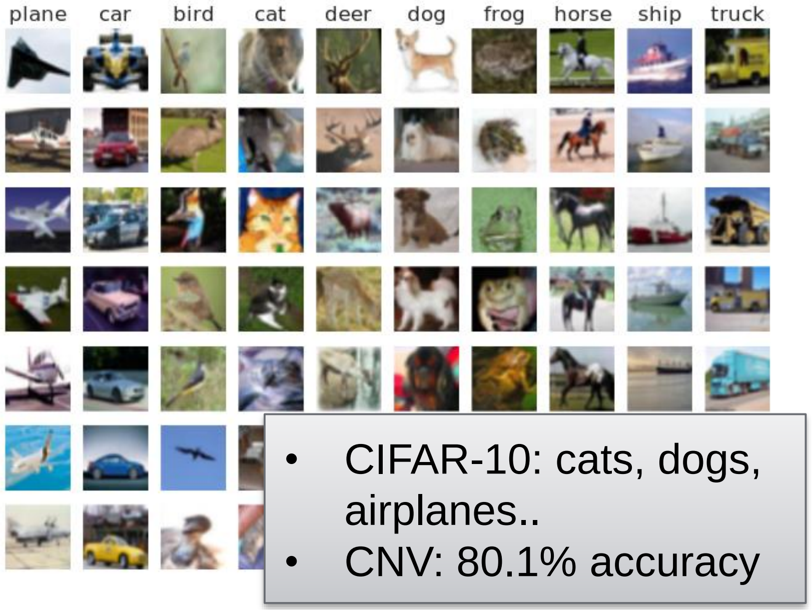
single, wide IFM memory

Input Data

- MNIST handwritten digits
- LFC: 98.4% accuracy



- Street View House Numbers
- CNV: 94.9% accuracy

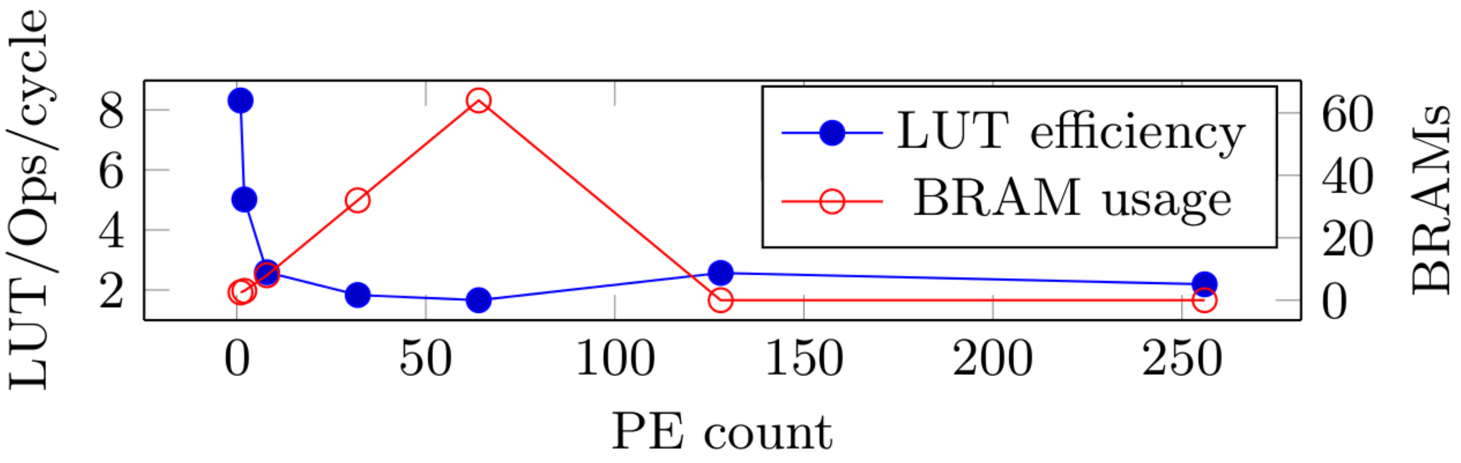


- CIFAR-10: cats, dogs, airplanes..
- CNV: 80.1% accuracy

Results – Efficiency

- **Runtime utilization: Operators busy 70-90% of the time**

- **LUT (instead of BRAM) storage if many PEs**
  - Fixed amount of work divided between more workers
  - Complex mapping problem, multi-dimensional tradeoff between performance/area


LUT/Ops/cycle
PE count
BRAMs
LUT efficiency
BRAM usage