

FPGA-based Machine Learning for FPGA-based Machine Learning for Electronic Measurement and Instrumentation

Philip Leong (梁恆惠) | Computer Engineering Laboratory
School of Electrical and Information Engineering,
The University of Sydney



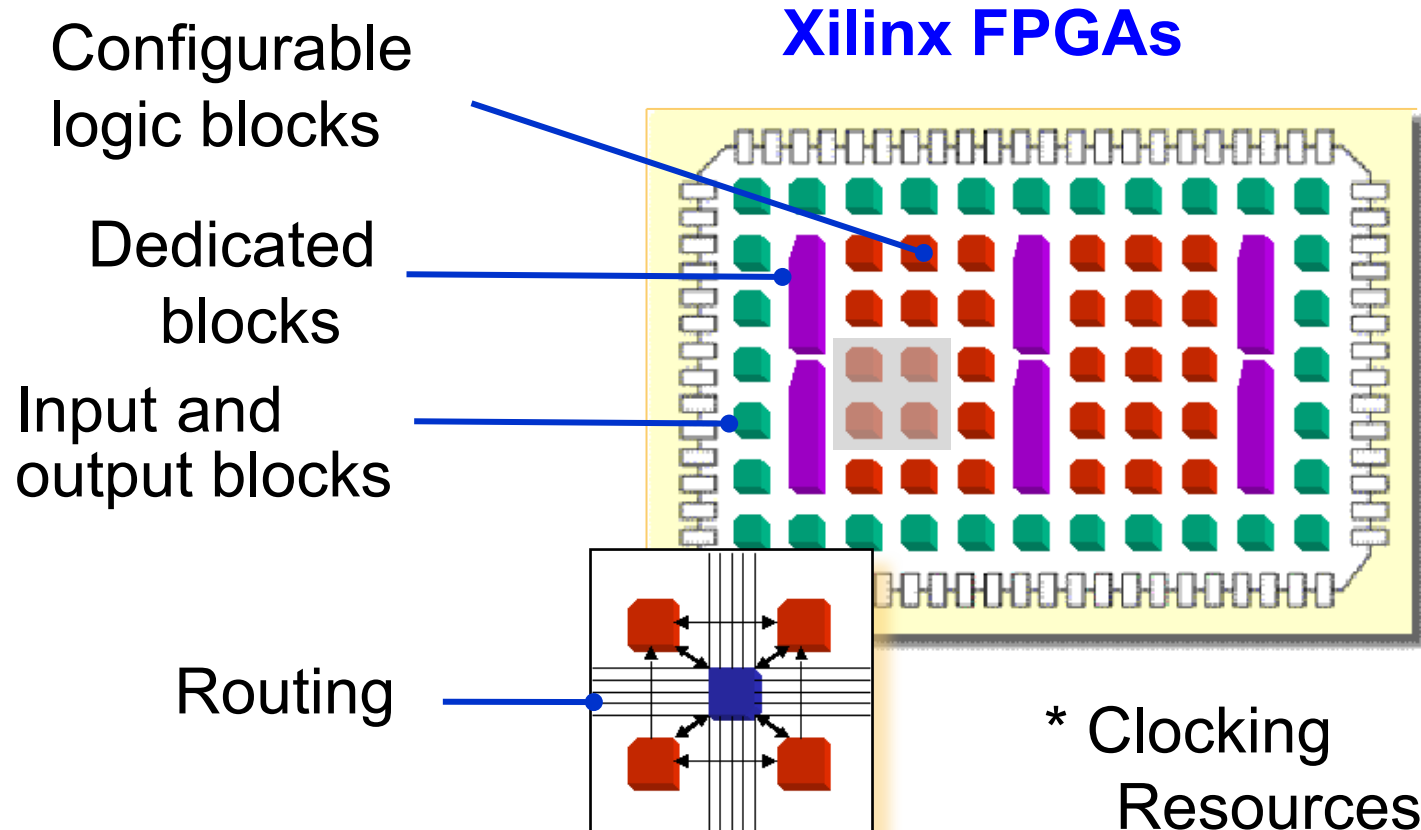
THE UNIVERSITY OF
SYDNEY

- › Focuses on how to use parallelism to solve demanding problems
 - Novel architectures, applications and design techniques using VLSI, FPGA and parallel computing technology
- › Research
 - Reconfigurable computing
 - Machine learning
 - Nanoscale interfaces



User-customisable integrated circuit

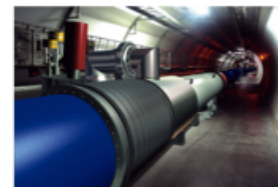
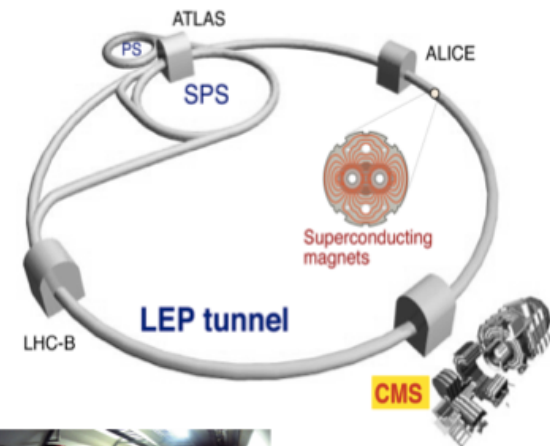
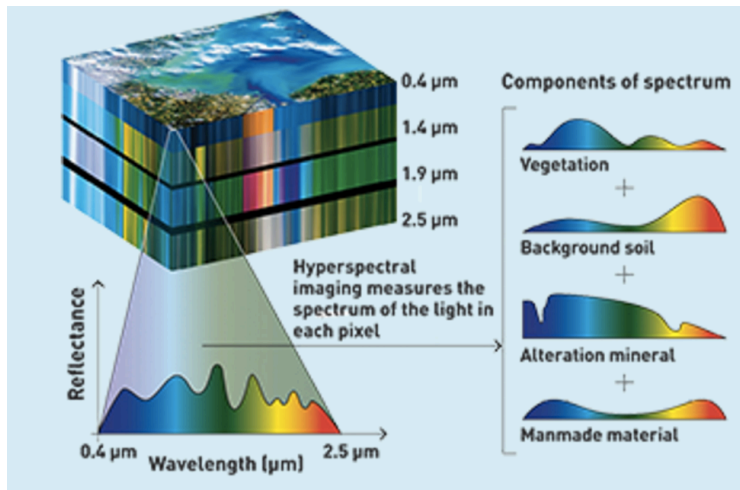
- > Dedicated blocks: memory, transceivers and MAC, PLLs, DSPs, ARM cores



- › **Instrumentation of the future** can use FPGA-based ML to interpret and process data **adaptively in real-time**
 - › Offer new capabilities
 - › e.g. trigger oscilloscope or spectrum analyzer on an anomaly
 - › e.g. battery power, high data rates, supercomputer performance in small form factor
- › There are many applications that could benefit
 - › test equipment
 - › network monitors
 - › prognostics and health management

Challenges in measurement and control are becoming feasible

- › Significant improvements in ML algorithms but cannot keep up with sources e.g. hyperspectral imager or wireless transceiver
- › Need extremely high throughput
- › In control applications we need low latency e.g. triggering data collection in Large Hadron Collider
- › Need very low latency



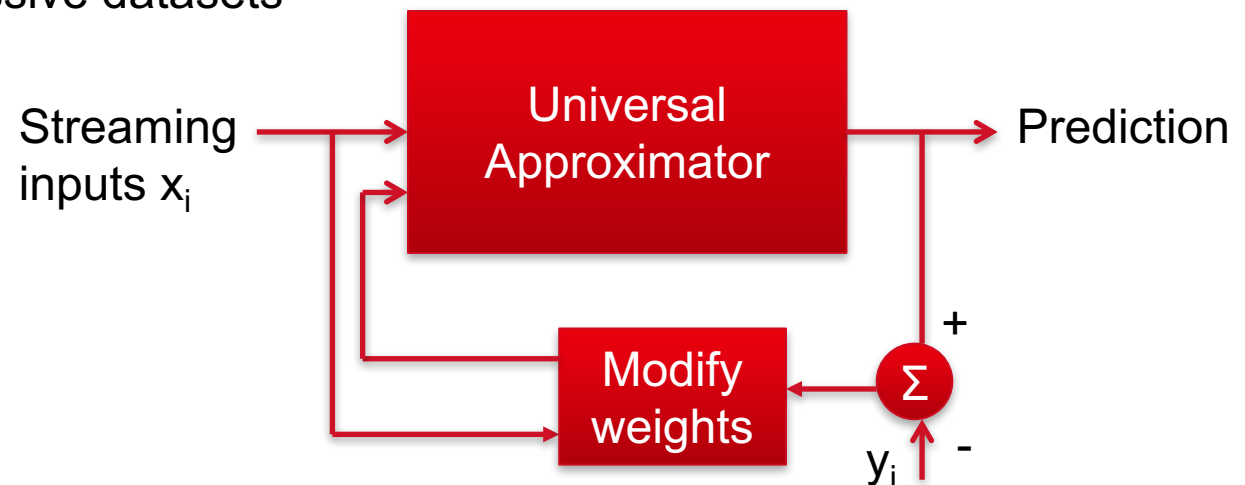
Improvements in throughput and latency enable new applications!

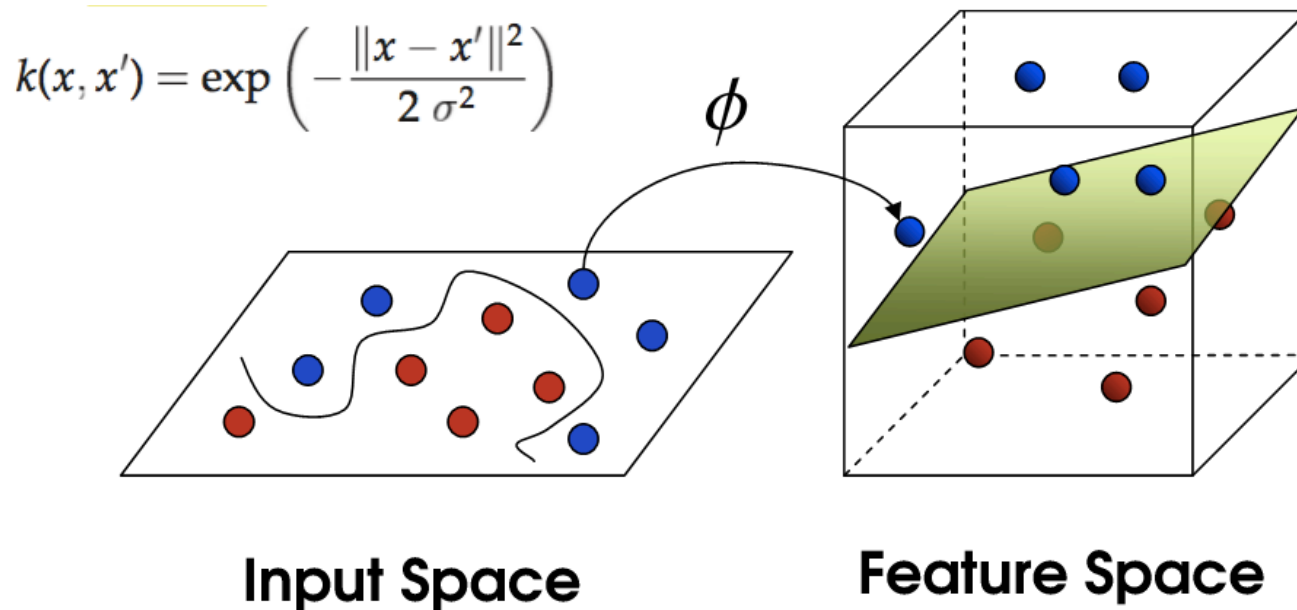
- › FPGAs offer an opportunity to provide ML algorithms with higher throughput and lower latency through
 - **E**xploration – easily try different ideas to arrive at a good solution
 - **P**arallelism – so we can arrive at an answer faster
 - **I**ntegration – so interfaces are not a bottleneck
 - **C**ustomisation – problem-specific designs to improve efficiency
- › **Describe our work on implementations of ML that use these ideas (and can be applied to instrumentation)**

- › **Exploration (Online kernel methods)**
- › **Parallelisation**
- › **Integration**
- › **Customisation**

Examples are KLMS and KRLS

- › Traditional ML algorithms are batch based
 - Make several passes through data
 - Requires storage of the input data
 - Not all data may be available initially
 - Not suitable for massive datasets
- › Our approach: online algorithms
 - Incremental, inexpensive state update based on new data
 - Single pass through the data
 - Can be high throughput, low latency





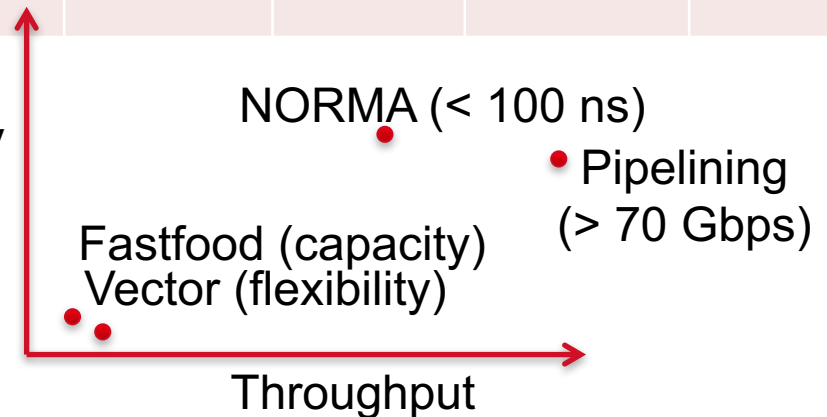
- › Choose high dimensional feature space (so easily separable)
- › Use kernel trick to avoid computing the mapping (fast)
- › Do regression/classification using

$$f(x_i) = \sum_{j=1}^N \alpha_j \kappa(x_i, v_j)$$

| Impl. | M | N | Latency (cycles) | Fmax (MHz) | Time (ns) | CPU (ns) | Speed Up |
|---|----|-----|------------------|------------|-------------|----------|----------|
| Vector - Flexible (Stratix 5) | 8 | 127 | 4396 | 157 | 28000 | 141000 | 5.1x |
| Pipeline - Throughput (Virtex 7) | 8 | 16 | 207 | 314 | 3.18 | 940 | 296x |
| Braided - Latency (Virtex 7) | 8 | 200 | 13 | 127 | 7.87 | 4025.8 | 511x |
| FASTFOOD - Capacity (Kintex Ultrascale) | 1k | 16k | 1694 | 500 | 3388 | 580000 | 171x |

- > M – Input dimension
- > N – Dictionary Size (or Sliding Window Size for KRLS)

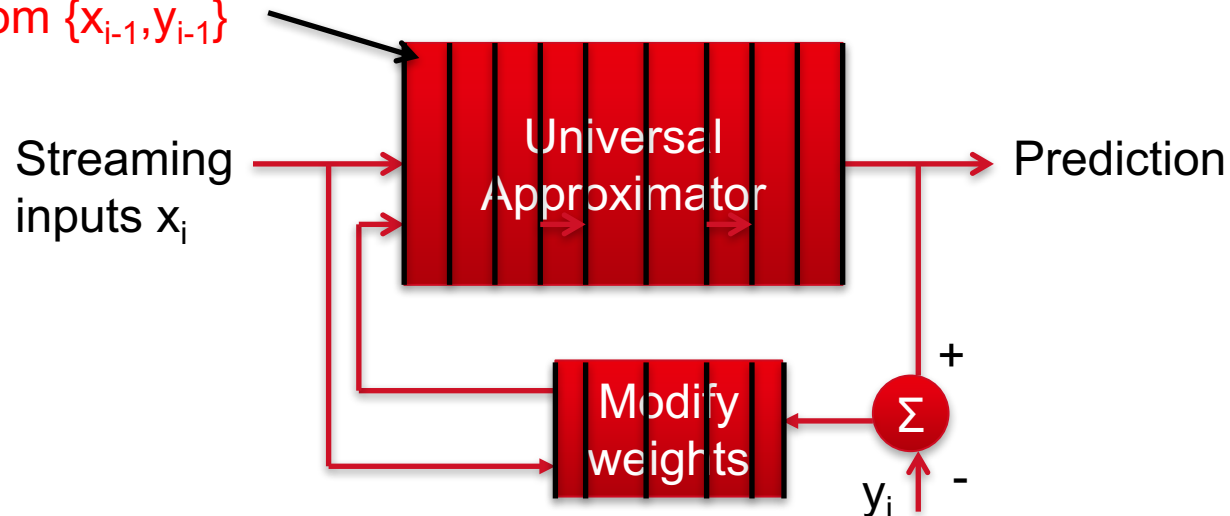
1/Latency



- › Exploration
- › **Parallelisation (pipelined KNLMS)**
- › Integration
- › Customisation

Dependency Problem

Cannot process x_i until we update weights from $\{x_{i-1}, y_{i-1}\}$



› Training is usually:

for (hyperparameters)

for (inputs)

learn_model()

› Alternative is to find L independent problems

- E.g. monitor L different things

› Our approach: run L independent problems (different parameters) in the pipeline

- Updates ready after L subproblems

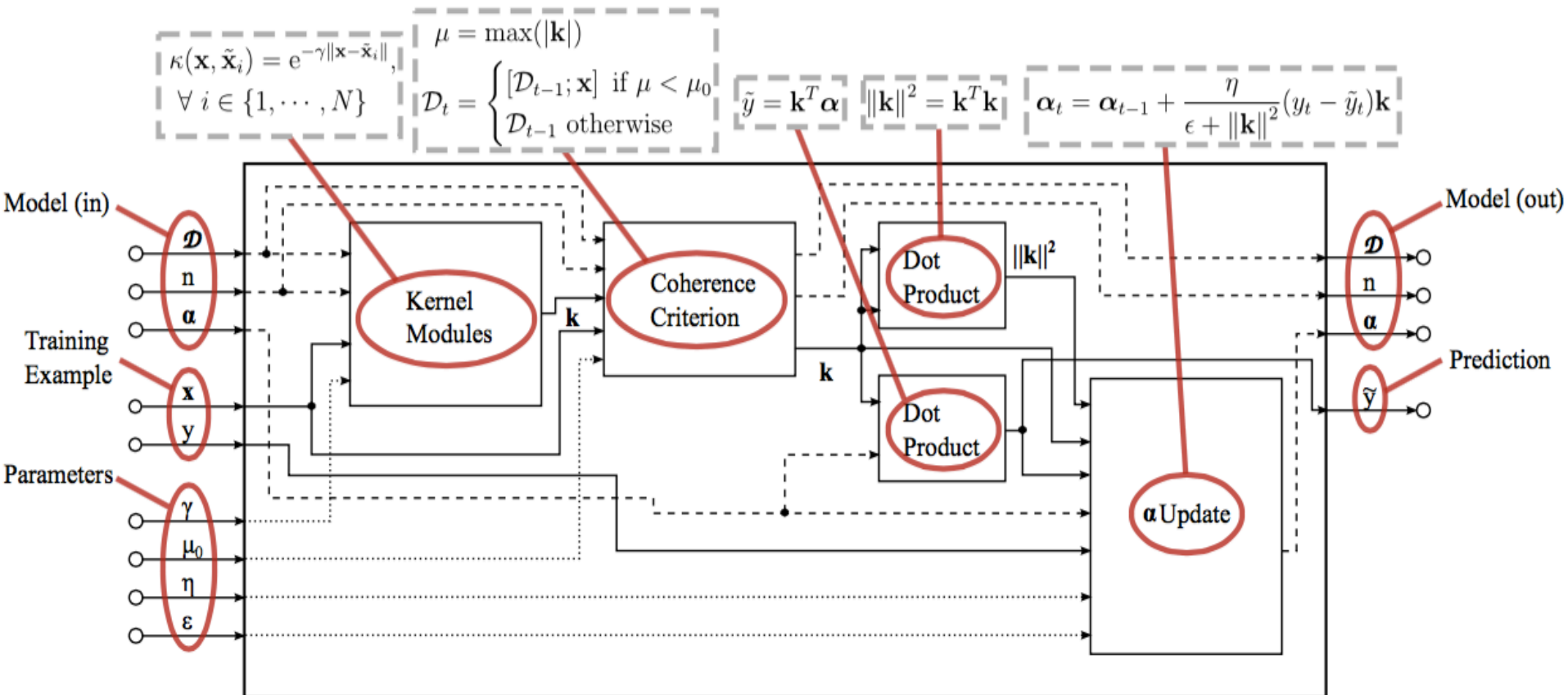
- Less data transfer

for (inputs)

for (hyperparameters)

learn_model()

• Similar approach for multiclass classification (train $C(C-1)/2$ binary classifiers)



Core with input vector $M=8$ and dictionary size $N=16$ (KNLMS)

| Implementation | Freq (MHz) | Time (ns) | Slowdown |
|------------------|------------|-----------|----------|
| System Pipelined | 250 | 4 | 1 |
| Non-pipelined | 250 | 800 | 200 |
| CPU (C) | 3,600 | 940 | 235 |

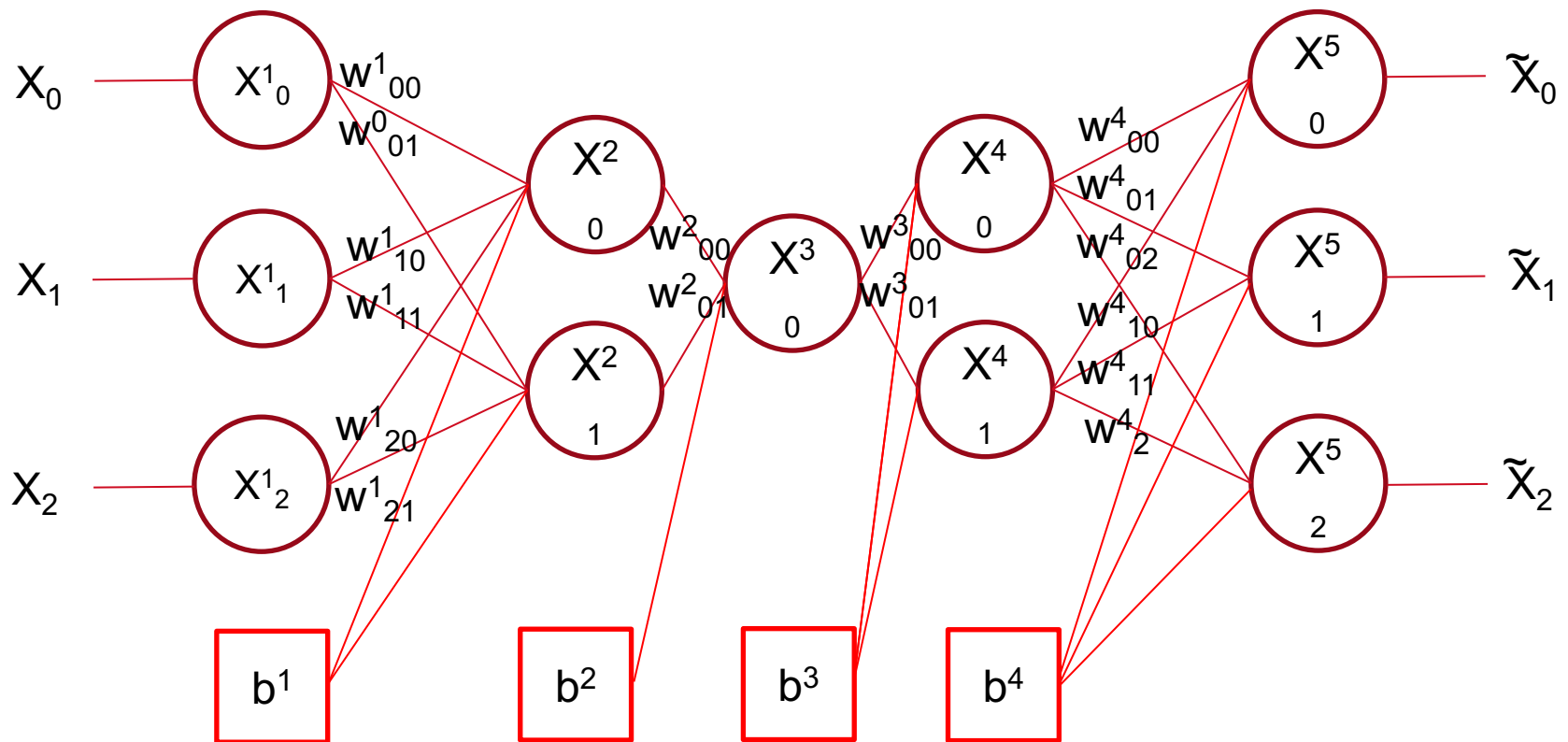
- › Can do online learning from 200 independent data streams at 70 Gbps (160 GFLOPS)

- › Exploration
- › Parallelisation
- › **Integration (radio frequency machine learning)**
- › Customisation

- › Processing radio frequency signals remains a challenge
 - high bandwidth and low latency difficult to achieve
- › Autoencoder to do anomaly detection

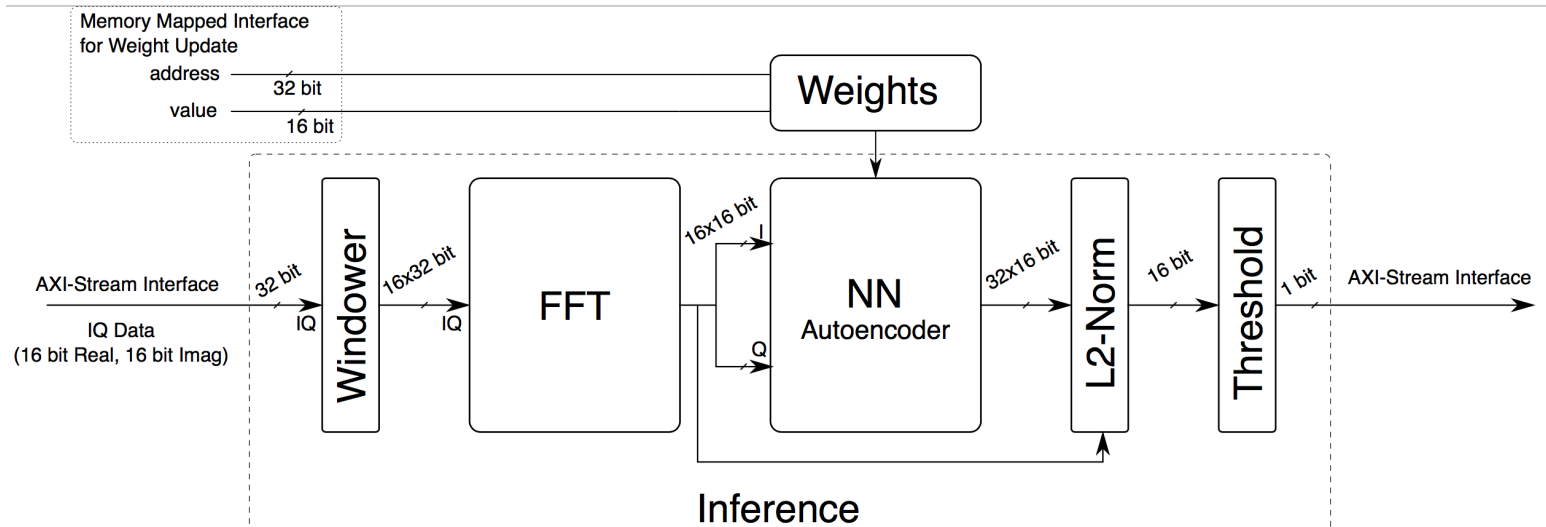


Train so $\tilde{x} \approx x$ (done in an unsupervised manner)

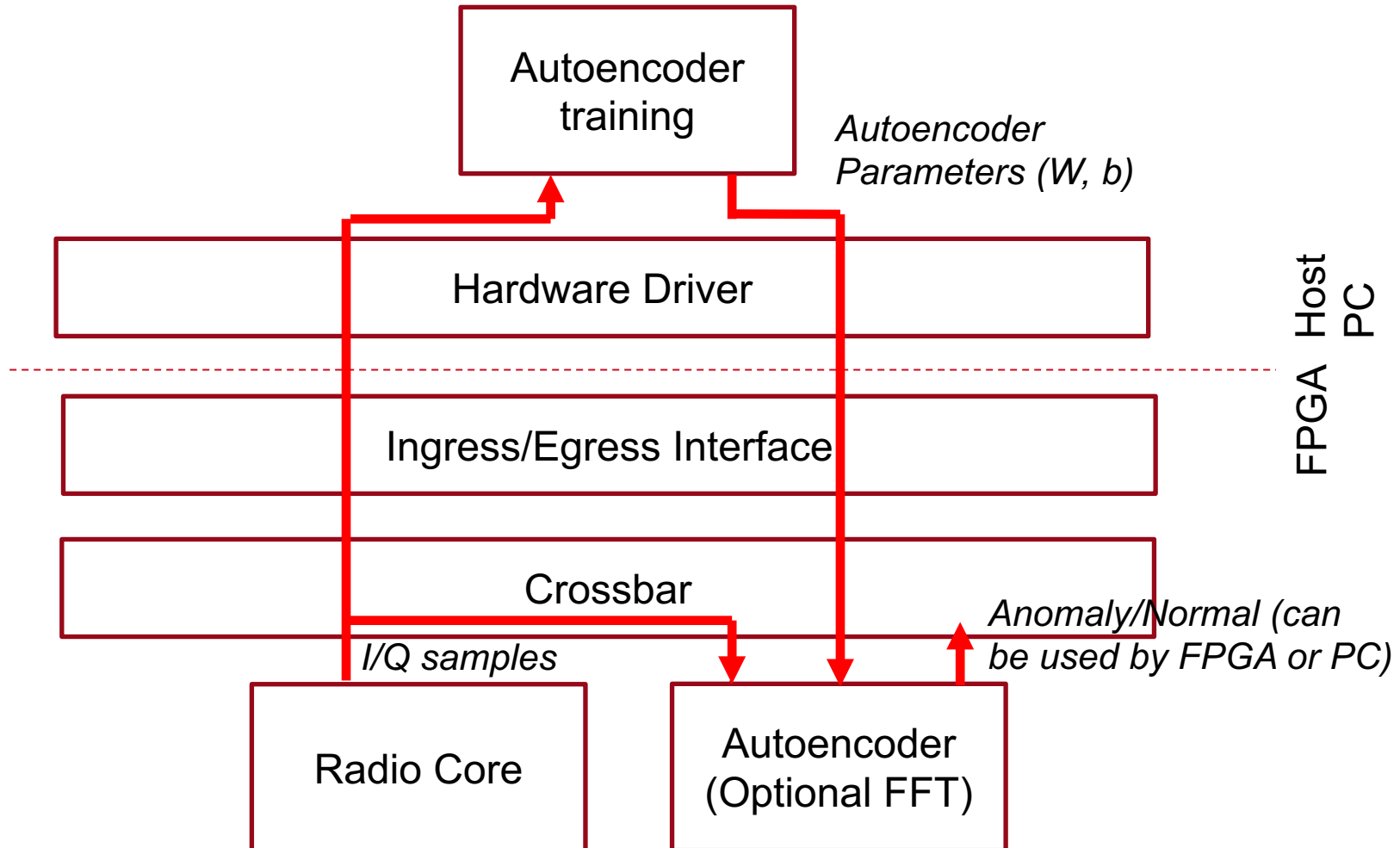


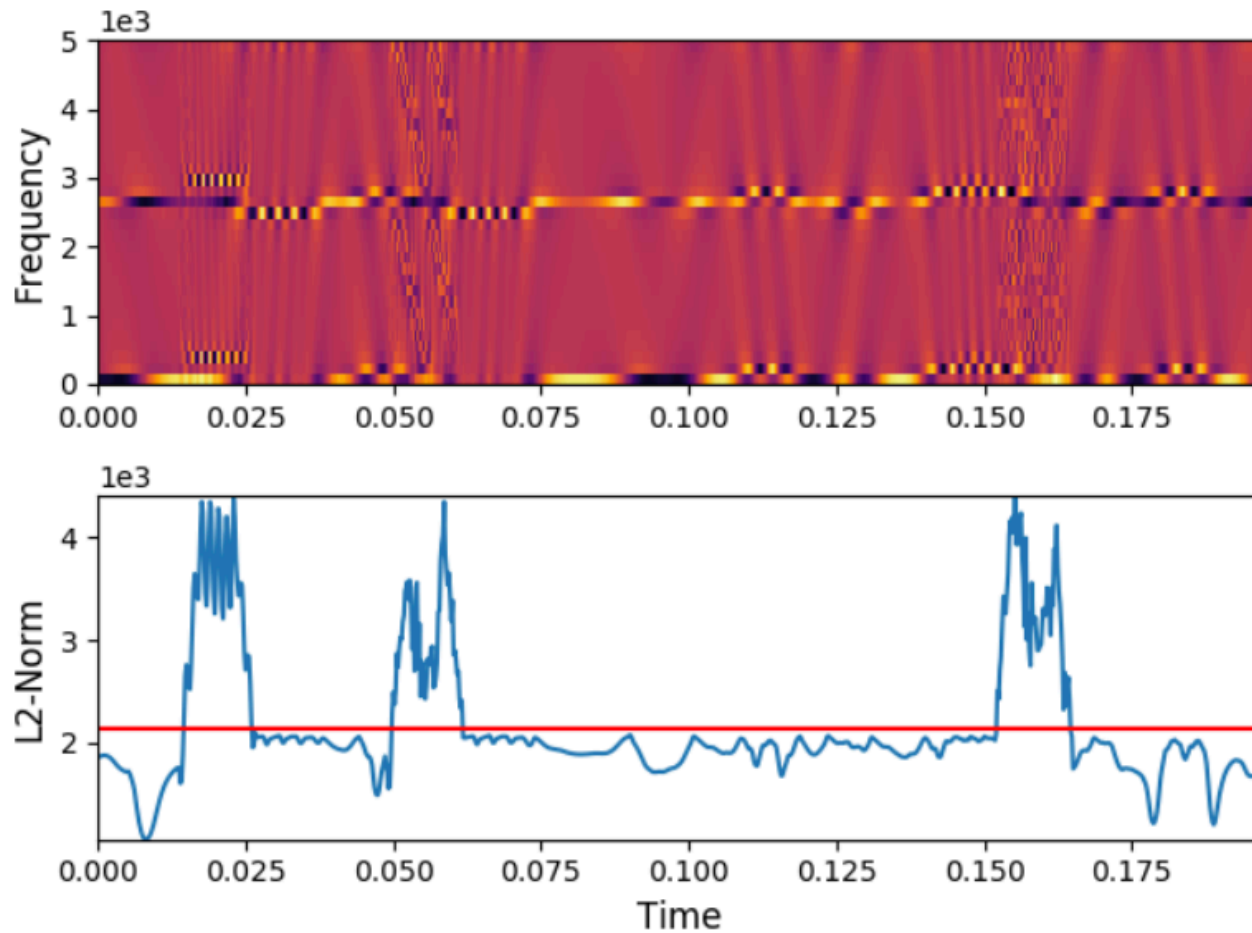
Autoencoder learns “normal” representation

- › Anomaly if distance between autoencoder output and input large
- › FPGA has sufficiently high performance to process each sample of waveform at 200 MHz!
 - This minimises latency and maximises throughput
 - Weights trained on uP and updated on FPGA without affecting inference



Implemented on Ettus X310 platform





Typical SDR latency >> 1 ms

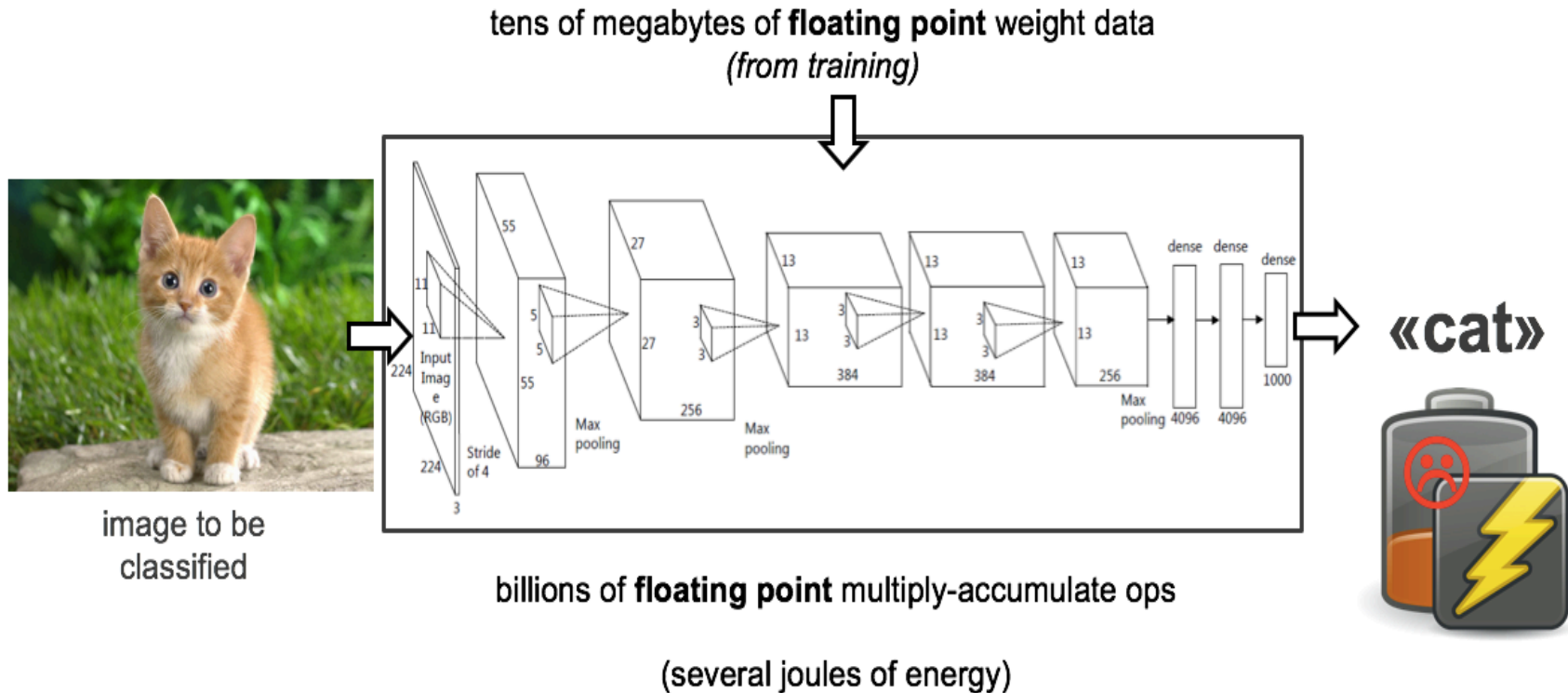
| Module | II | Latency (cycles) | BRAM | DSP | FF | LUT |
|--------------------|-----|---------------------|------|------|--------|-------|
| Windower | 1 | 0 | 0 | 0 | 1511 | 996 |
| FFT | 1 | 8 | 0 | 48 | 4698 | 2796 |
| NN | 1 | 17 | 4 | 1280 | 213436 | 13044 |
| L_2 -Norm | 1 | 4 | 0 | 32 | 1482 | 873 |
| Thres | 1 | 0 | 0 | 0 | 3 | 21 |
| Weight Update | 258 | 257 | 0 | 0 | 21955 | 4528 |
| Inference (FFT+NN) | 1 | 37 | 1068 | 1360 | 241522 | 45448 |
| Inference (NN) | 1 | 29 | 1068 | 1312 | 236824 | 42652 |
| Total | N/A | N/A | 1068 | 1360 | 263477 | 49976 |
| Total Util. | N/A | N/A | 67% | 88% | 51% | 19% |

| Operation | Throughput | Latency |
|-------------------|------------|---------|
| Inference(FFT+NN) | 5ns | 185ns |
| Inference(NN) | 5ns | 105ns |
| Weight Update | 1290ns | 1285ns |

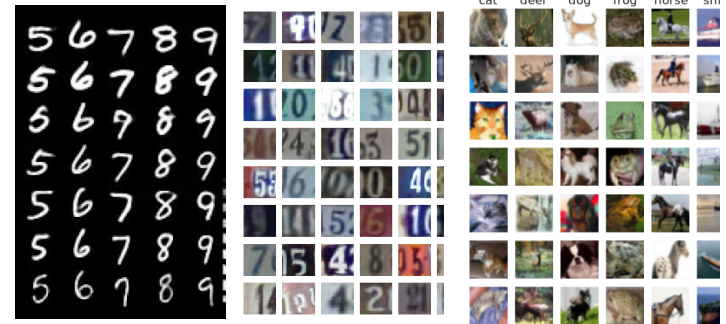
- › Exploration
- › Parallelisation
- › Integration
- › **Customisation (binarised neural networks)**

Inference with Convolutional Neural Networks

Slides from Yaman Umuroglu et. al., “FINN: A framework for fast, scalable binarized neural network inference,” FPGA’17



- › 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)
 - Competitive results on three smaller benchmarks
 - Open source training flow
 - Standard “deep learning” layers
 - Convolutions, max pooling, batch norm, fully connected...

| | 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)

Vivado HLS estimates on Xilinx UltraScale+ MPSoC ZU19EG

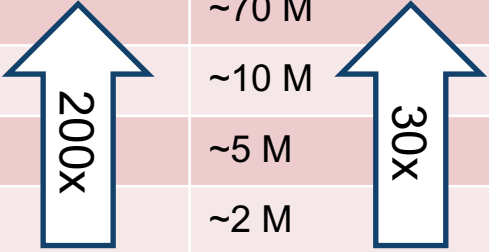
› *Much* smaller datapaths

- Multiply becomes XNOR, addition becomes popcount
- No DSPs needed, everything in LUTs
- Lower cost per op = more ops every cycle

› *Much* smaller weights

- Large networks can fit entirely into on-chip memory (OCM)
- More bandwidth, less energy compared to off-chip

| Precision | Peak TOPS | On-chip weights |
|-----------|-----------|-----------------|
| 1b | ~66 | ~70 M |
| 8b | ~4 | ~10 M |
| 16b | ~1 | ~5 M |
| 32b | ~0.3 | ~2 M |



› **fast** inference with **large BNNs**

| | 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

- › **Exploration (Online kernel methods)**
- › **Parallelisation**
- › **Integration**
- › **Customisation**

- › LSTM using HLS tutorial
 - › https://github.com/phwl/hls_lstm
- › Kernel methods code e.g. braiding
 - › <https://github.com/da-steve101/chisel-pipelined-olk>
- › FINN - can do trillions of binary operations per second
 - › <https://github.com/Xilinx/BNN-PYNQ>

› Exploration

- › Kernel methods optimised using different algorithms, mathematical techniques, computer architectures, arithmetic

› Parallelism

- › Increased by removing dependencies to ensure all stages do work every cycle

› Integration

- › In radio frequency, this allows latency to be reduced by 4 orders of magnitude

› Customisation

- › Increase parallelism by reducing precision
- › Keep weights on-chip to devote more hardware to arithmetic

› FPGAs can greatly assist with the implementation of ICEMI's theme of "measurement and intelligent sensing"

- › Learning & inference at 70 Gbps
- › Learning & inference with 100 ns latency
- › Image processing @ 12.3 Mfps
- › Multimodal measurements

› Radio frequency anomaly detector

- › We are using this to predict physical and media access layer protocols
- › Could also be used as a novel diagnostic instrument - monitor RF output of electronic equipment, detect anomalies



Thank you!



Philip Leong (philip.leong@sydney.edu.au)
<http://www.sydney.edu.au/people/philip.leong>