Low Precision Inference and Training for Deep Neural Networks

Philip Leong
Director, Computer Engineering Laboratory
http://phwl.org/talks

THE UNIVERSITY OF SYDNEY

CruxML
REAL-TIME COMPUTING & MACHINE LEARNING

THE UNIVERSITY OF SYDNEY

Computer Engineering Laboratory

- Focuses on how to use parallelism to solve demanding problems
  - Novel architectures, applications and design techniques using FPGAs
- Research: reconfigurable computing, radio frequency machine learning

Motivation

Tradeoff between performance and precision

- CPUs/GPUs designed to support datatypes of fixed wordlength
  - Double, float, long, short, char
- FPGA and ASICs can provide custom datapaths of arbitrary wordlength

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

200x

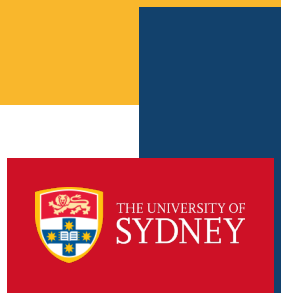
30x

Slide: Xilinx

- So how can we utilize low-precision for inference and training?

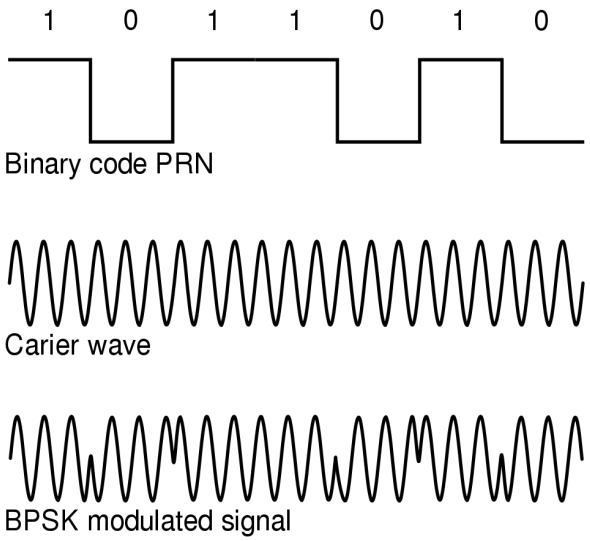
Inference

1



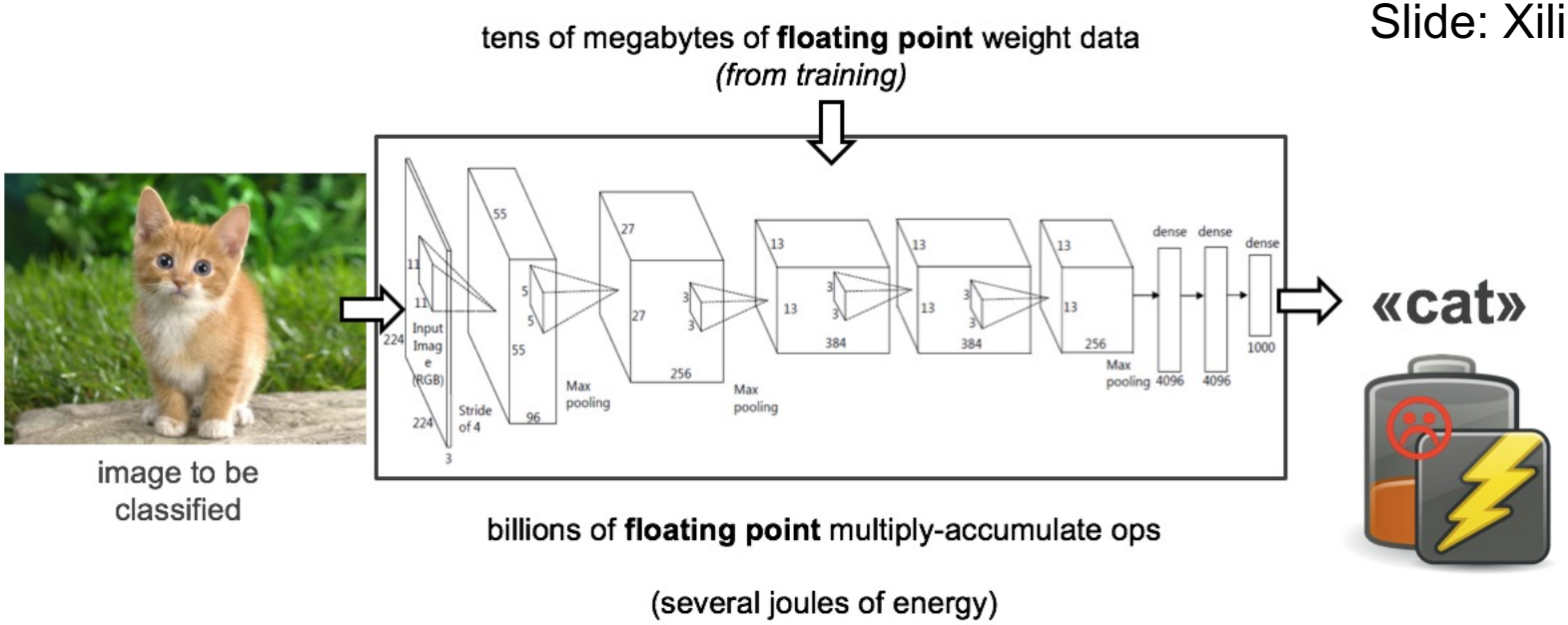
Background: Radio Frequency Machine Learning

- Radio data is high speed and low latency often required
- Monitor RF spectrum and determine the different modulations being used
- FPGAs offer possibility of integrating radio, signal processing and ML on the same chip
- Automatic modulation classification (AMC): detect modulation type from raw IQ samples



Background: Convolutional Neural Network

Slide: Xilinx



O'Shea et al, "Over-the-Air Deep Learning Based Radio Signal Classification"

- ResNet on the raw IQ data gives SOA results

Background: Binarized Neural Networks

FINN: A framework for fast, scalable binarized neural network inference," FPGA'17

- *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

200x

30x

Slide: Xilinx

But accuracy not good! How can we improve accuracy and performance?

AMC Implementation

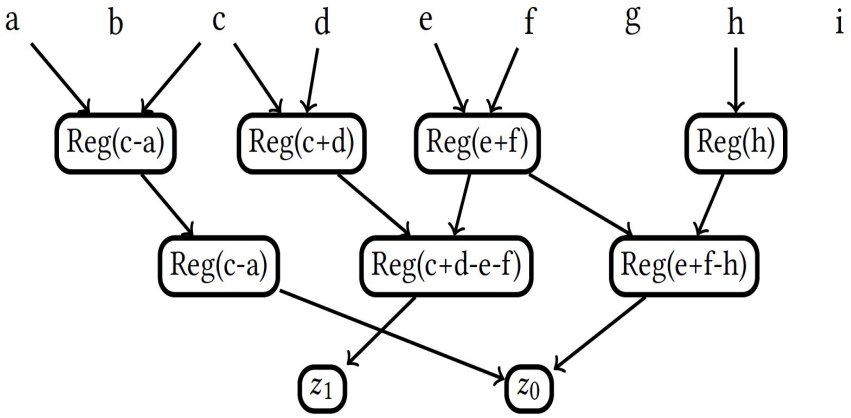
- Fully parallel implementation desirable for maximum performance but FPGA resources insufficient for contemporary CNN model
- Addressed by exploiting unstructured sparsity and the following techniques:
  1. Massively parallel ternary NN implemented as pruned adder trees
  2. Common subexpression merging
  3. 16-bit bit serial arithmetic to minimize accuracy loss with low area
  4. Sparsity control
- Apply this to AMC

Common Subexpression Elimination

- Weights are ternary
  - Reduces convolution to constructing adder tree
  - Subexpression merged to reduce implementation

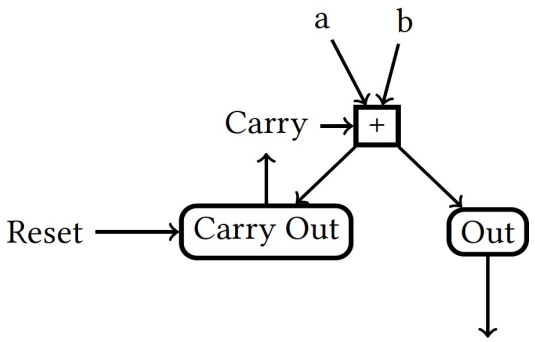
Computing $z_0 = c + e + f - (a + h)$ and $z_1 = c + d - e - f$

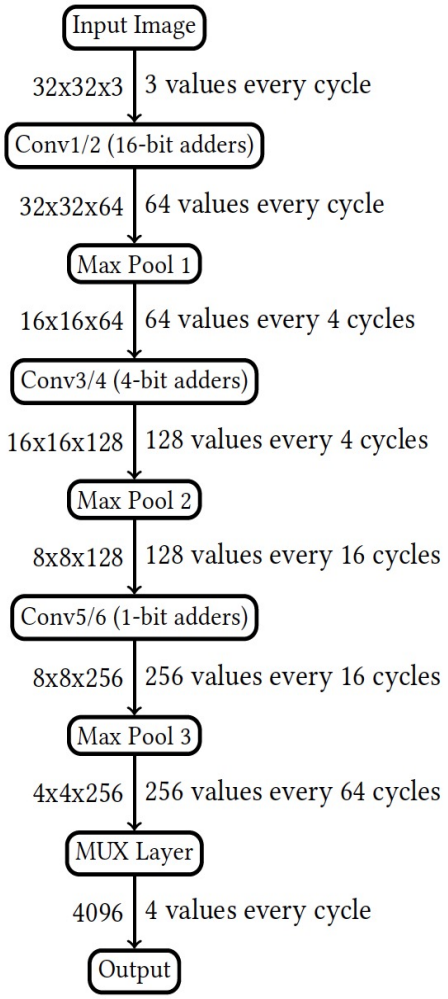
$$\begin{array}{ccc} a \times (-1) & b \times 0 & c \times 1 \\ d \times 0 & e \times 1 & f \times 1 \\ g \times 0 & h \times (-1) & i \times 0 \end{array}$$



Throughput matching with serial adders

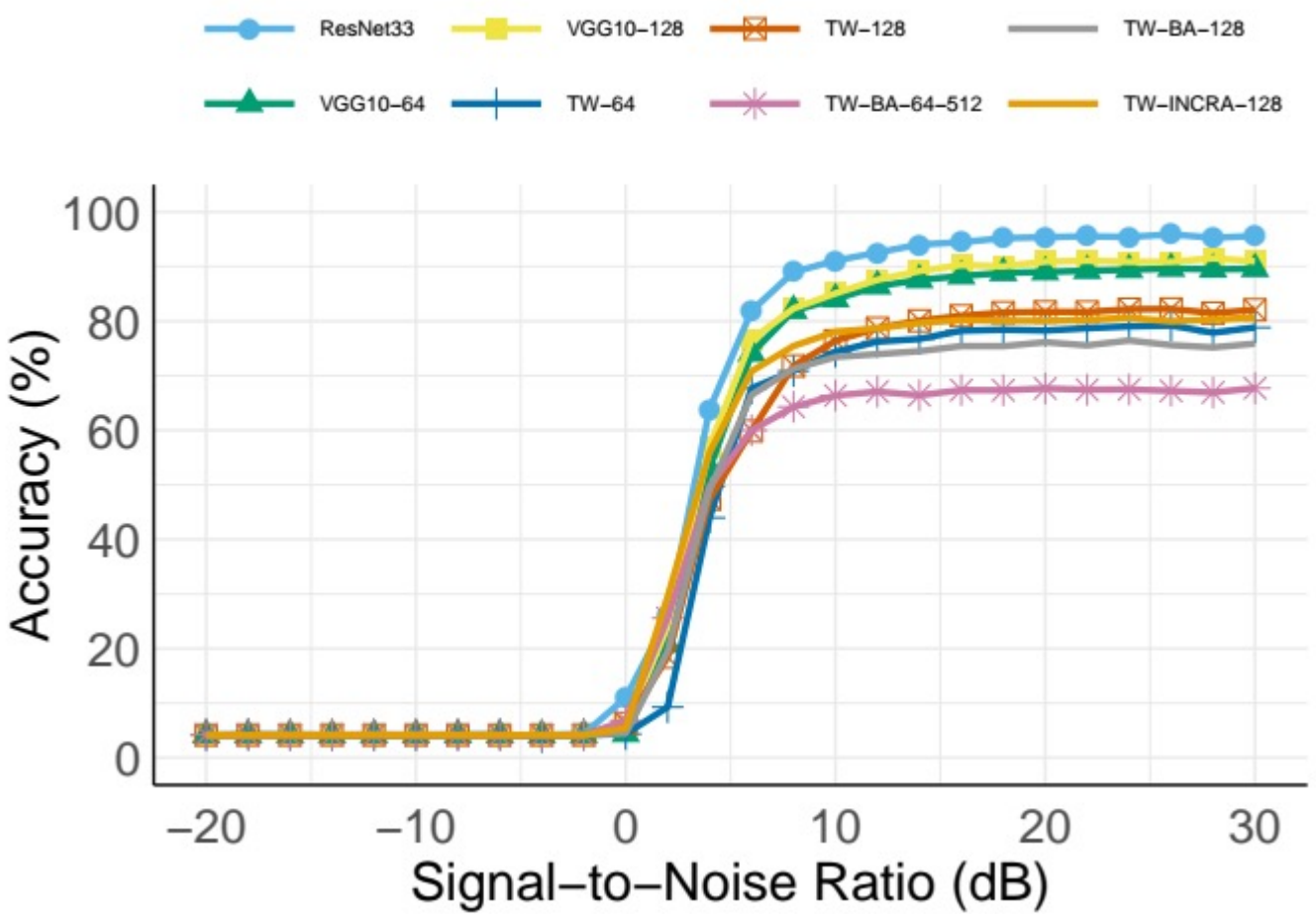
- Activations are 16-bit
  - Not all layers have same throughput
  - Use digit serial to make more compact
  - 4-bit digit serial has 1/4 area
  - 1-bit bit serial has 1/16 area





THE UNIVERSITY OF SYDNEY

Accuracy on RadioML 2018.01A dataset

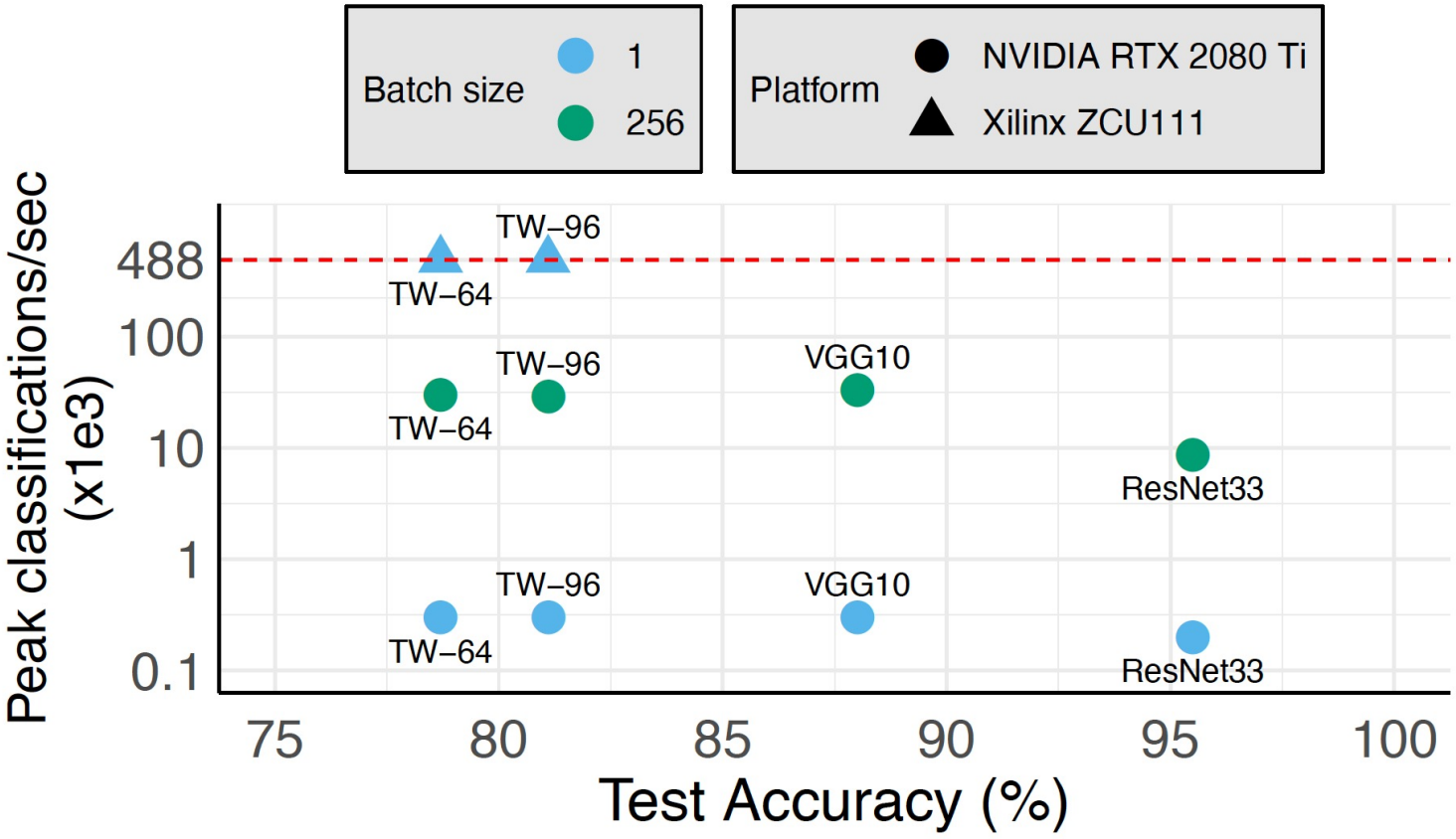


Incremental Precision

- Use incremental precision activations instead of 16 bit everywhere (to improve accuracy)
  - Use bit serial adders everywhere
  - Adjust precision to match the throughput
  - Same area as binary activations
  - Almost 5% accuracy gain over binary activations

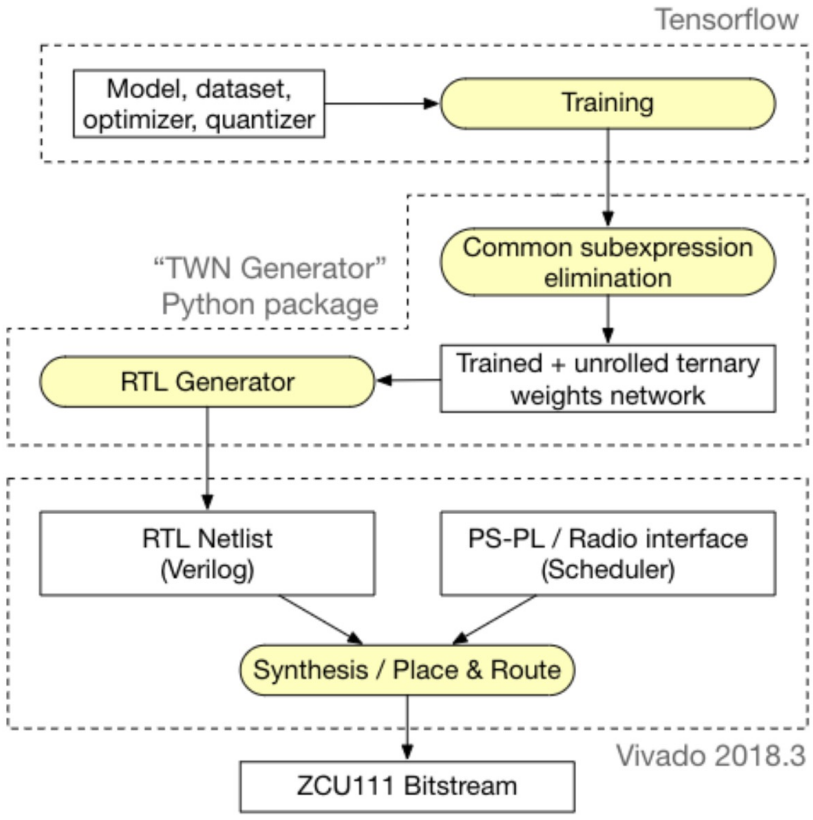
Model	TW-64	TW-96	TW-BA-128	TW-INCRA-128
CLBs	28k (53.5%)	47k (89.3%)	43k (80.7%)	42k (80.2%)
LUTs	124k (29.1%)	232k (54.7%)	234k (55.1%)	211k (49.6%)
FFs	217k (25.5%)	369k (43.4%)	333k (39.2%)	324k (38.1%)
BRAMs	524 (48.5%)	524 (48.5%)	523 (48.4%)	512.2 (48.3%)
DSPs	1496 (35%)	1207 (28.3%)	1408 (33.0%)	1407 (32.9%)
Accr	78.7	81.1	75.9	80.2

Accuracy


Batch size
1
256
Platform
NVIDIA RTX 2080 Ti
Xilinx ZCU111
Peak classifications/sec (x1e3)
488
100
10
1
0.1
TW–96
TW–64
TW–96
TW–64
VGG10
ResNet33
TW–96
TW–64
VGG10
ResNet33
75
80
85
90
95
100
Test Accuracy (%)

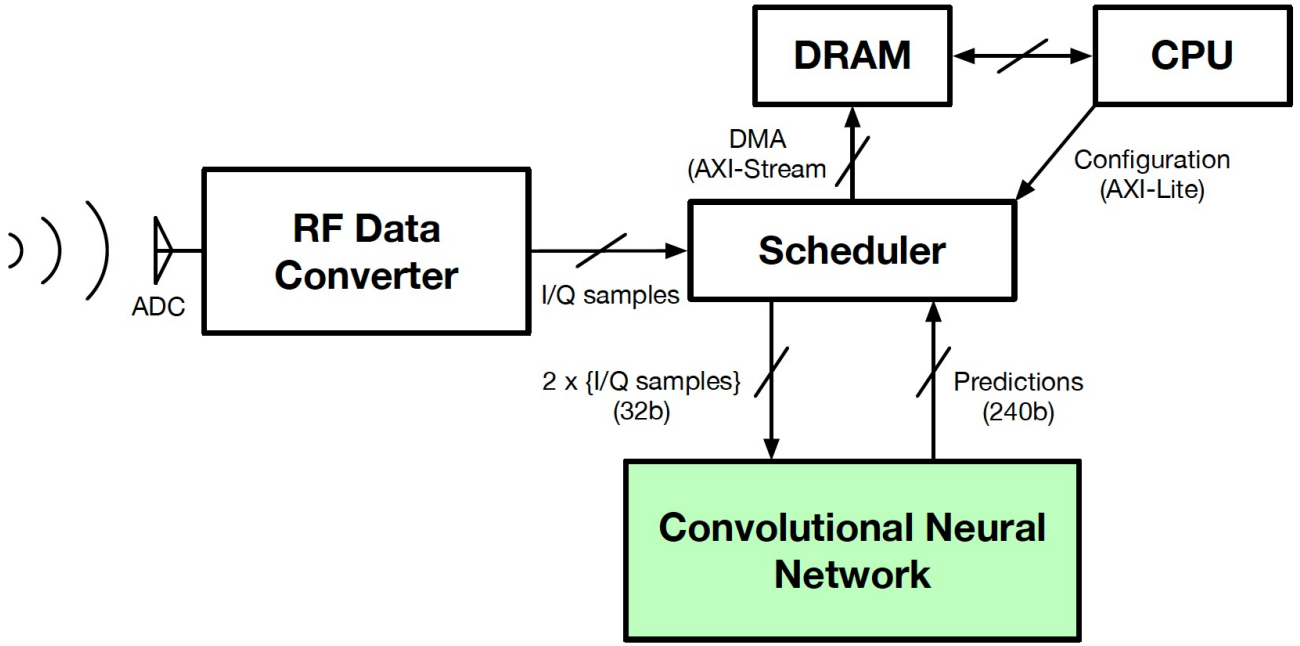
Implementation

- System implemented on ZCU111 RFSoC
- https://github.com/da-steve101/radio_modulation
- Open Source Verilog generator
- https://github.com/da-steve101/twn_generator

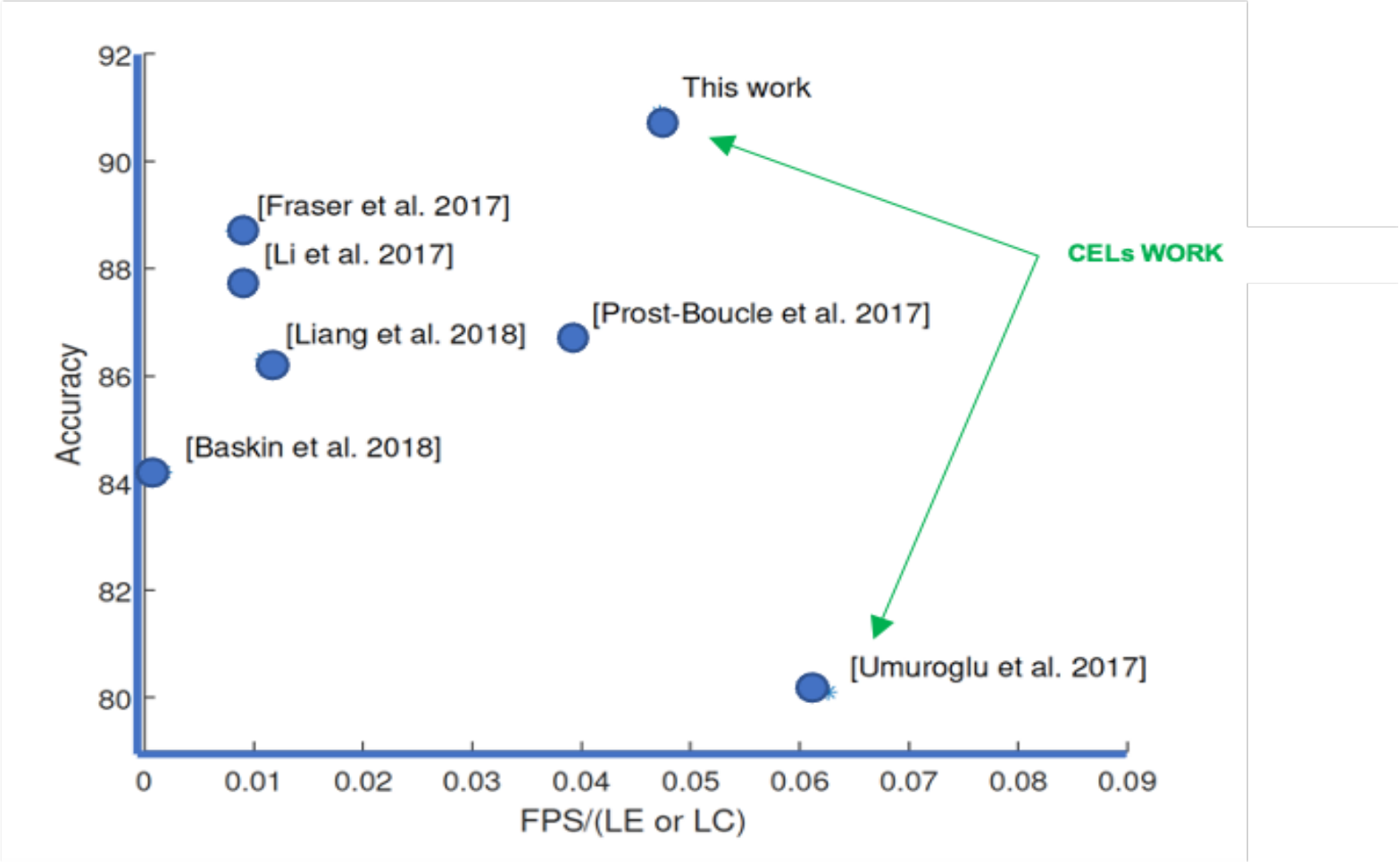


System Implementation

› Automatic Modulation classifier: 488K class/s, 8us latency for CNN



Accuracy vs FPS/LE on CIFAR10 (not AMC)



THE UNIVERSITY OF SYDNEY

Summary

- Presented an optimized network for AMC which
  - Applies common subexpression elimination and digit serial arithmetic to a fully unrolled ternary network
  - Integrates the entire design on a single chip for a low-latency batch size 1 implementation
- These serve to achieve a level of performance higher than previously reported
- Challenge of achieving state of the art accuracy remains

Training

2

THE UNIVERSITY OF SYDNEY

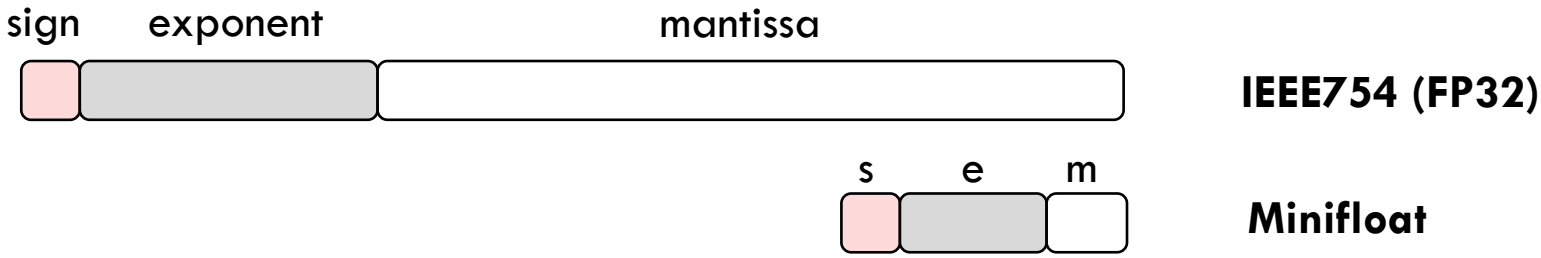
Motivation

- Deep learning has an efficiency problem!
  - E.g. Billions of parameters, 500+ GPUs

- **Specialized Number Representations**
  - Alternative to FP32/FP16
  - 4-8 bits for weights, activations and gradients
  - Cheaper and faster training systems
  - Datacenter to Edge?

Minifloat

- Narrow floating-point representation
  - Ours range between 4-8 bits
  - NaN/Infinity NOT supported

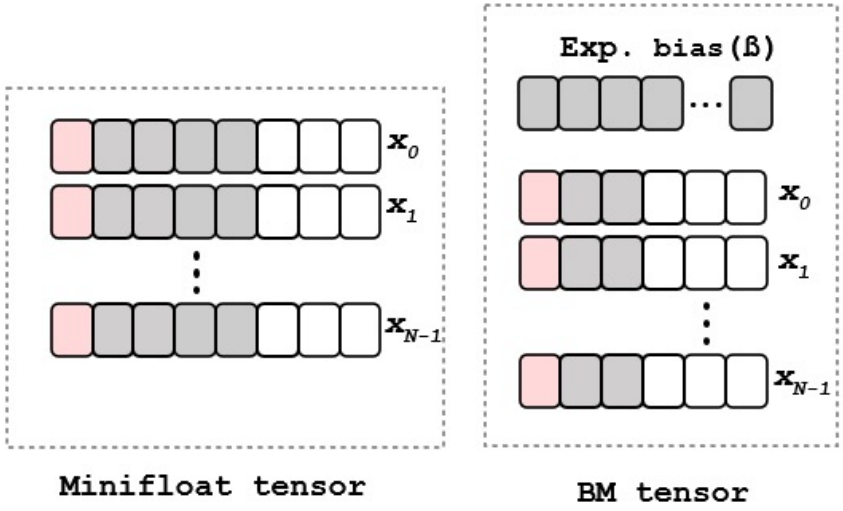


- Pros:
  - Memory (fewer bits)
  - Smaller hardware
- Cons:
  - Dynamic Range (exponent bits)

For training >= 8 bits

Block Minifloat

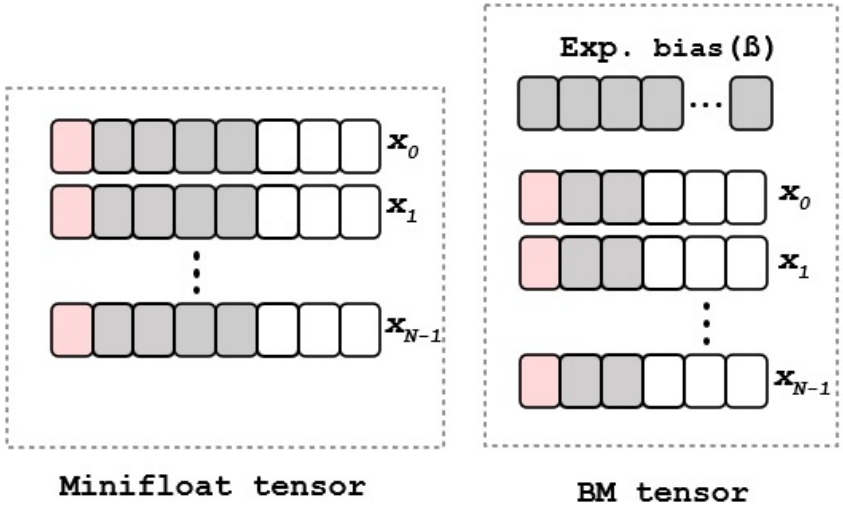
- Share exponent bias across **blocks** of N minifloat numbers

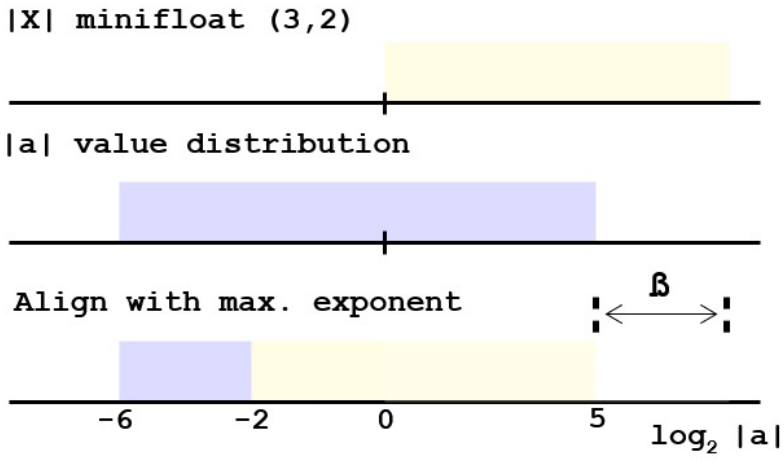


- Dynamic range (with fewer bits)
- Denser dot-products in hardware

Block Minifloat

- Share exponent bias across **blocks** of N minifloat numbers

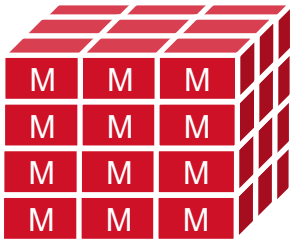




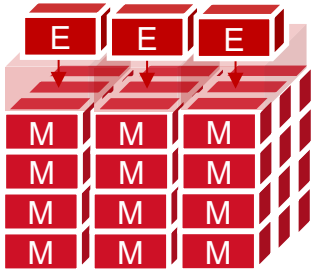
- Dynamic range (with fewer bits)
- Denser dot-products in hardware

- Align wtih **max** exponent
- Underflow is tolerated

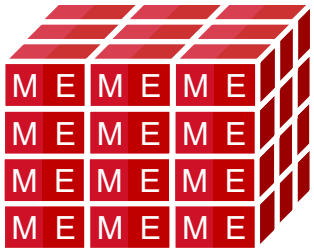
Block Minifloat



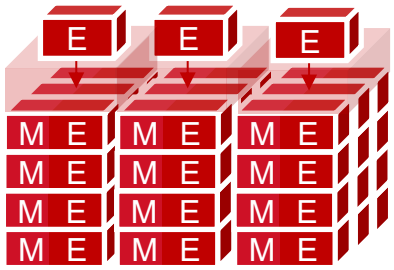
Fixed



BFP



Minifloat



Block Minifloat

- Block Minifloats – share exponent bias across **blocks** of N minifloat numbers

Fused Multiply-Add (FMA) with Kulisch Accumulation

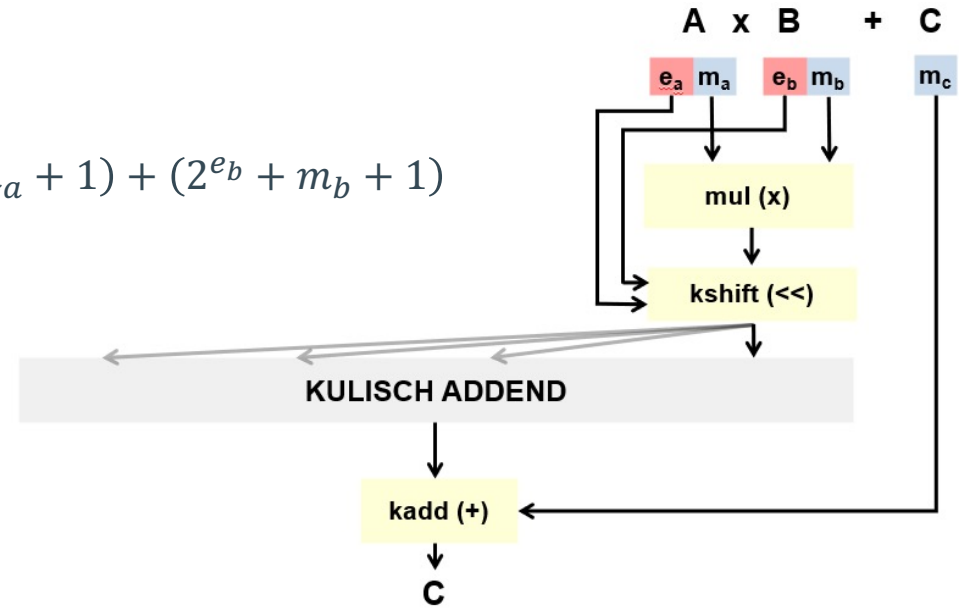
- **Kulisch Accumulator:** Fixed point accumulator wide enough to compute error-free sum of floating-point products

- Integer-like hardware complexity for **exponent <=4 bits**

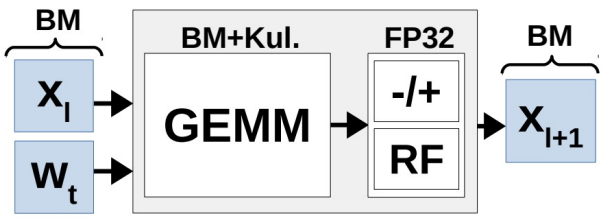
Area Cost:

$$kadd = 1 + (2^{e_a} + m_a + 1) + (2^{e_b} + m_b + 1)$$

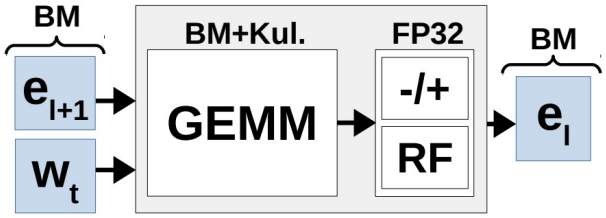
$$kshift = 2^{e_a} + 2^{e_b}$$



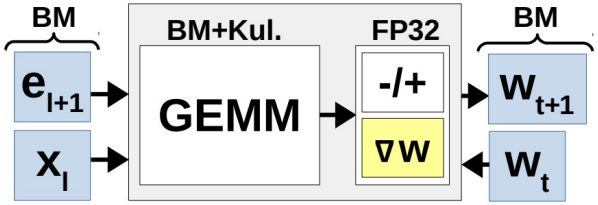
End-to-end Training with BM



(a) Fwd activation



(b) Bwd activation grad.



(c) Bwd weight grad. and update

1x floating point operation every N MACs

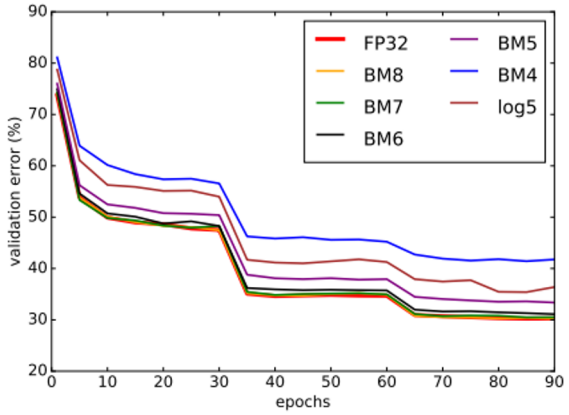
Implementation Details

- Three techniques to reduce data loss:
  - Gradual underflow, Block Design, Hybrid Formats
- Simulate specialized BM hardware on GPU (with FP32)
  - Apply Block Minifloat to all weights, acts, grads
- Our Spectrum of Block Minifloats

BM8 (ours)	(2,5)/(4,3)
BM7 (ours)	(2,4)/(4,2)
BM6 (ours)	(2,3)/(3,2)
BM5 (ours)	(2,2)/(3,1)
BM5-log (ours)	(4,0)/(4,0)
BM4 (ours)	(2,1)/(3,0)
BM4-log (ours)	(3,0)/(3,0)

Experiments and Results

- **Training experiments:**
  - Datasets: (ImageNet, VOC, PTB, IWSLT)
  - Models: (ResNet, LSTM, TF base, SSD-Lite, EfficientNet)

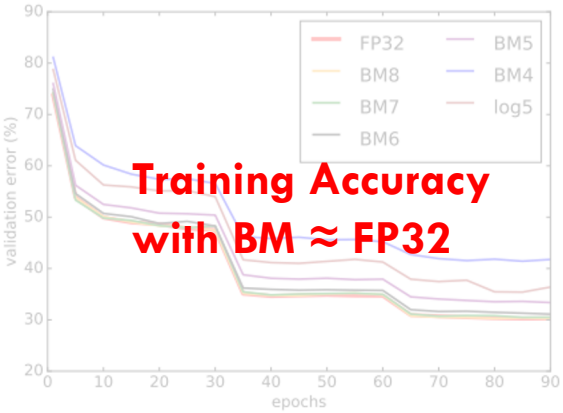


- **RTL synthesis:**
  - Fused multiply-add (FMA)
  - 4x4 systolic matrix mutlipliers

Component	Area (μm^2)	Power (μW)
FP32	4782	10051
FP8 (w/ FP16 add)	829	1429
INT8 (w/ INT32 add)	417	1269
BM8	391	1141
BM6	**200**	**624**
INT8 (4x4 systolic)	7005	20253
FP8 (4x4 systolic)	18201	56202
BM8 (4x4 systolic)	**6976**	**18765**

Experiments and Results

- **Training experiments:**
  - Datasets: (ImageNet, VOC, PTB, IWSLT)
  - Models: (ResNet, LSTM, TF base, SSD-Lite, EfficientNet)

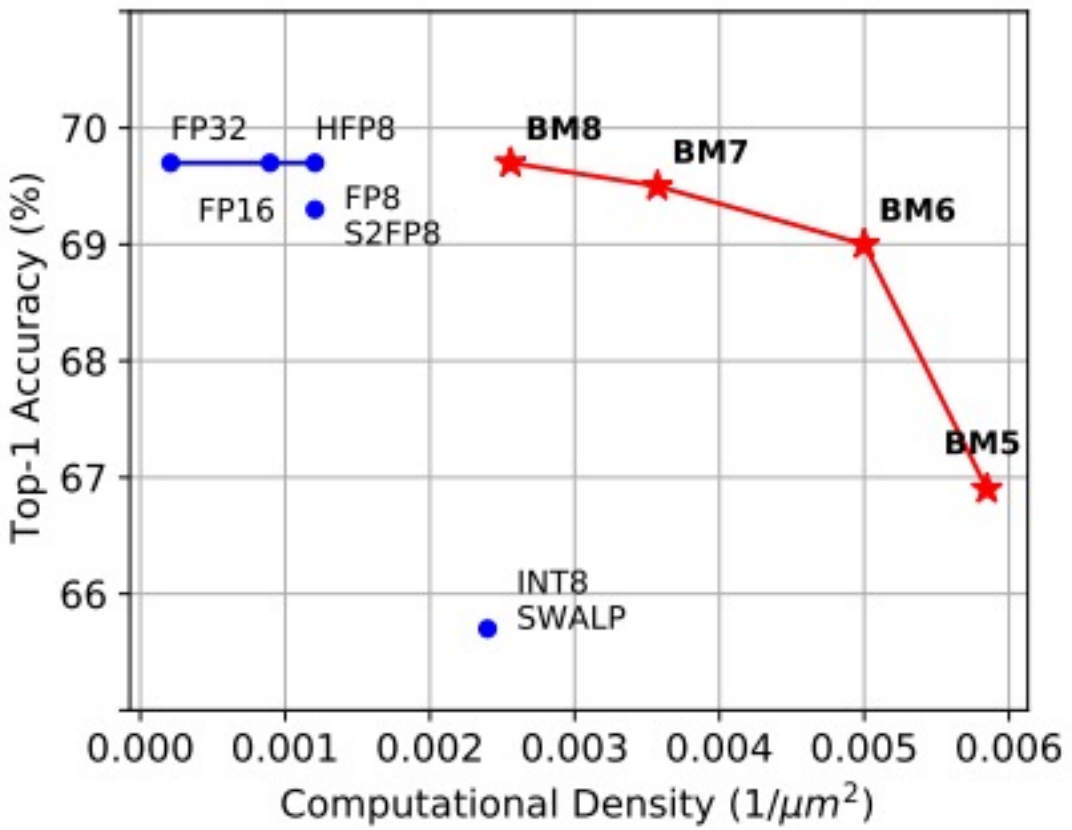


- **RTL synthesis:**
  - Fused multiply-add (FMA)
  - 4x4 systolic matrix mutlipliers

Component	Area (μm^2)	Power (μW)
FP32	4782	10051
FP8 (w/ FP16 add)	829	1429
INT8 (w/ INT32 add)	417	1269
BM8	301	1141
BM6	**200**	**624**
INT8 (4x4 systolic)	7005	20253
FP8 (4x4 systolic)	18201	56202
BM8 (4x4 systolic)	**6976**	**18765**

BM units are:
- **Smaller**
- **Consume less Power**

Imagenet



Model: ResNet-18

Dataset: ImageNet

THE UNIVERSITY OF SYDNEY

Summary

- Block Minifloat sustains high training accuracy with fewer bits and narrow exponent encodings (e <= 4 bits)
- Faster Training is possible:
  - Fewer bits – increases performance in memory-bound
  - Narrow exponents – yield denser arithmetic units in compute-bound
- This work may be particularly advantageous in moving training into Edge devices
- github.com/sfox14/block_minifloat

THE UNIVERSITY OF SYDNEY

Conclusion

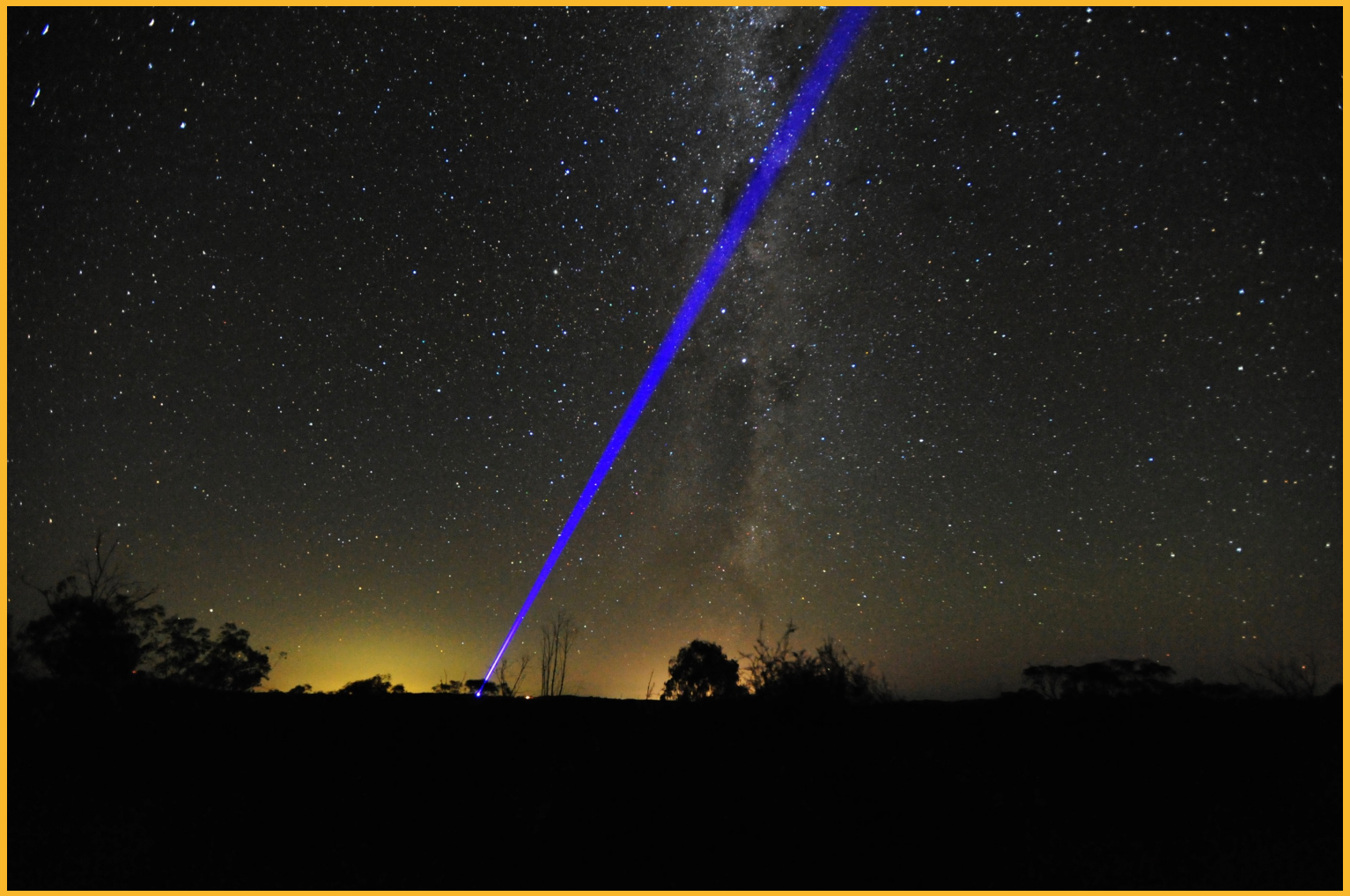
- It is well known that DNN hardware performance can be significantly improved through precision optimisations
- We have demonstrated
  - Feasibility of fully parallel, ternary single chip DNN implementations using adder trees, throughput matching and incremental activations
  - It is feasible to train DNNs with 8-bit precision using block minifloats
- More research on exploring tradeoffs between precision and accuracy is needed to make this technology practical for mainstream implementations

THE UNIVERSITY OF SYDNEY

References

[1] Stephen Tridgell, David Boland, Philip HW Leong, Ryan Kastner, Alireza Khodamoradi, and Siddhartha. Real-time automatic modulation classification using RFSoC. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2020, New Orleans, LA, USA, May 18-22, 2020*, 82–89. IEEE, 2020. URL: amc_raw20.pdf, doi:10.1109 / IPDPSW50202.2020.00021.

[2] Sean Fox, Seyedramin Rasoulinezhad, Julian Faraone, and David Boland Philip H.W. Leong. A block minifloat representation for training deep neural networks. In *Proc. of The International Conference on Learning Representations (ICLR)*. 2021. URL: bm_iclr21.pdf.



THE UNIVERSITY OF SYDNEY

CruxML
REAL-TIME COMPUTING & MACHINE LEARNING

Philip Leong
philip.leong@sydney.edu.au