Hardware-aware pruning of real-time neural networks

Benjamin Ramhorst and George A. Constantinides
Imperial College London

Vladimir Lončar
MIT

Outline

Background

Hardware-aware pruning

Results

Conclusions

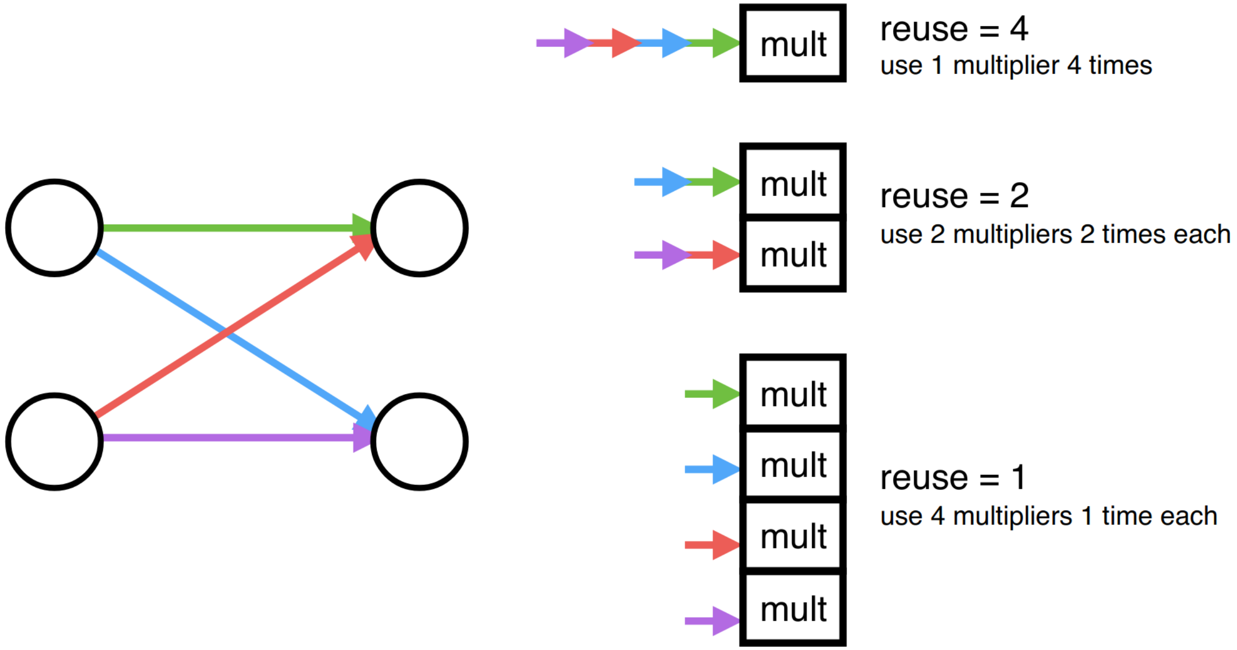
hls4ml

Full on-chip design and high parallelism – **high resource utilisation**

Key variable - **reuse factor (RF)**

Previous compression studies:

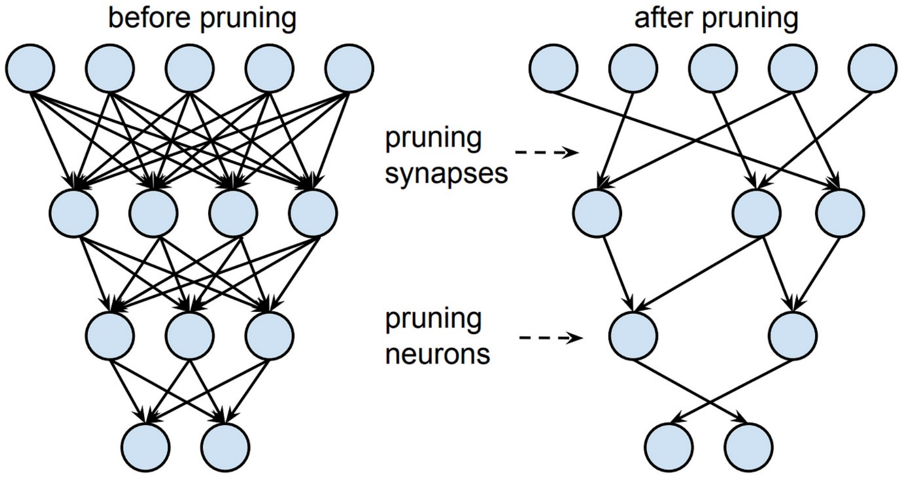
- Quantization [1]
- Unstructured pruning [2, 3]



Pruning

Pruning – sparsifying the network by setting ***less important*** weights to zero:

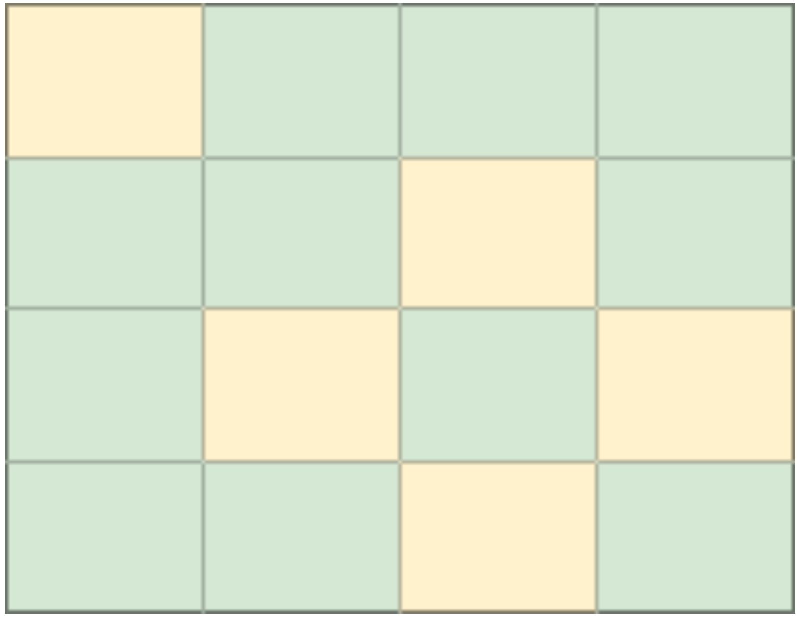
- Typically implemented **iteratively**: zero out some weights – fine-tune model with some of the weights set to zero – increase sparsity and repeat
- To help pruning, add l_1 **regularisation**
- LeCun *et al.* (1989) [4], Han *et al. (2015)* [5]



Unstructured pruning

Unstructured pruning – little hardware improvements – requires additional co-design:

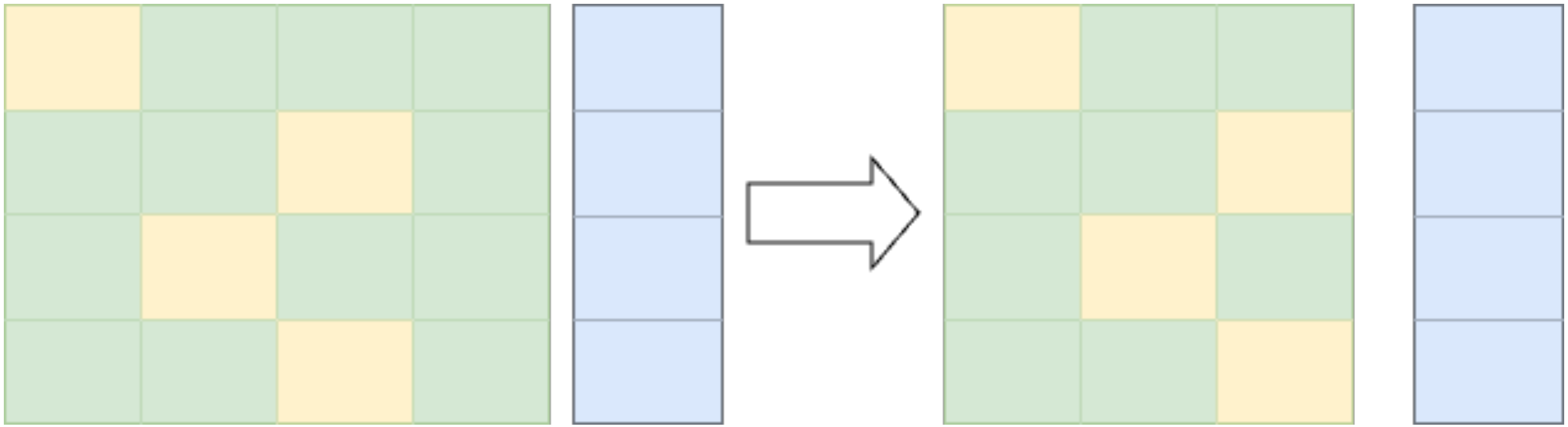
- Compressed sparse row (CSR) / compressed sparse column (CSC) require three attributes for every non-zero element



Structured pruning

Structured pruning – significant hardware improvements, but greater impact on accuracy:

- Safely ignore last column in matrix-vector multiplication and set last element in output to zero



TensorFlow Model Optimization

Powerful tool for optimizing Keras & TensorFlow models

Iteratively removes low-magnitude weights

Supports:

- Unstructured pruning for all layers
- Block pruning (m, n) for 2-dimensional matrices
- Structural pruning (m, n) applied to the last dimension of tensor
- Latency pruning for XNNPack, only applicable to 1x1 Conv2D layers

TensorFlow Model Optimization

Drawbacks of TensorFlow Model Optimization:

- No support for **structured pruning in Conv1D / Conv2D layers**
- No support for **automatically removing zero structures** from the pruned network – little hardware improvements
- No support for gradient-based weight ranking
- The **sparsity** of each layer needs to chosen **manually** OR **equal layer-wise sparsity**

Outline

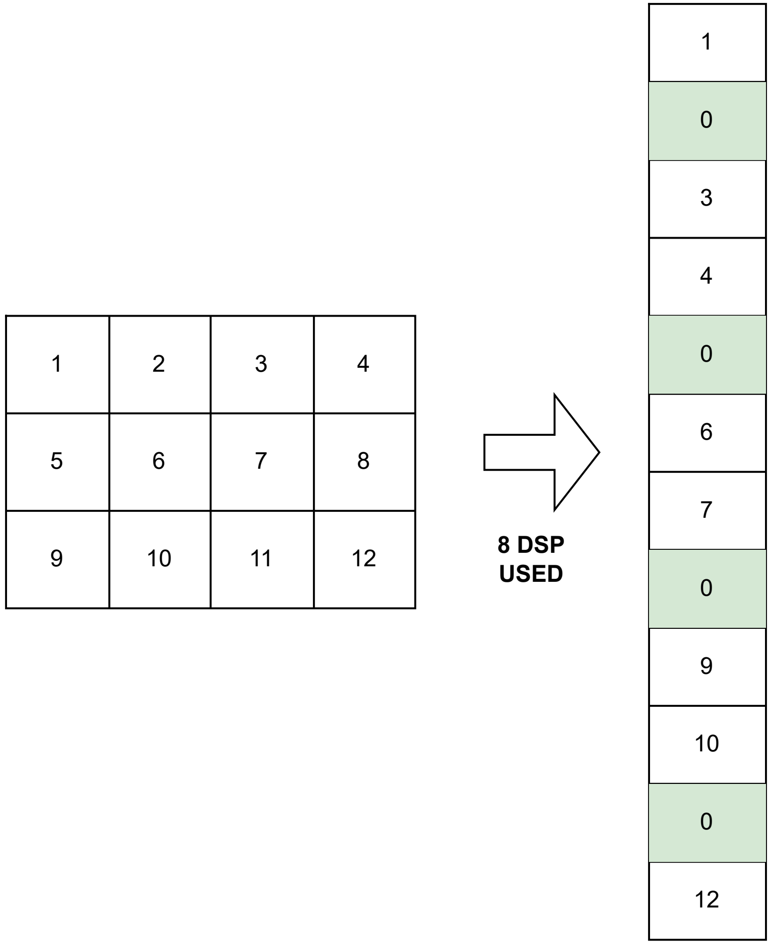
Background

Hardware-aware pruning

Results

Conclusions

Hardware-aware pruning



RF = 1 - design is "fully unrolled"

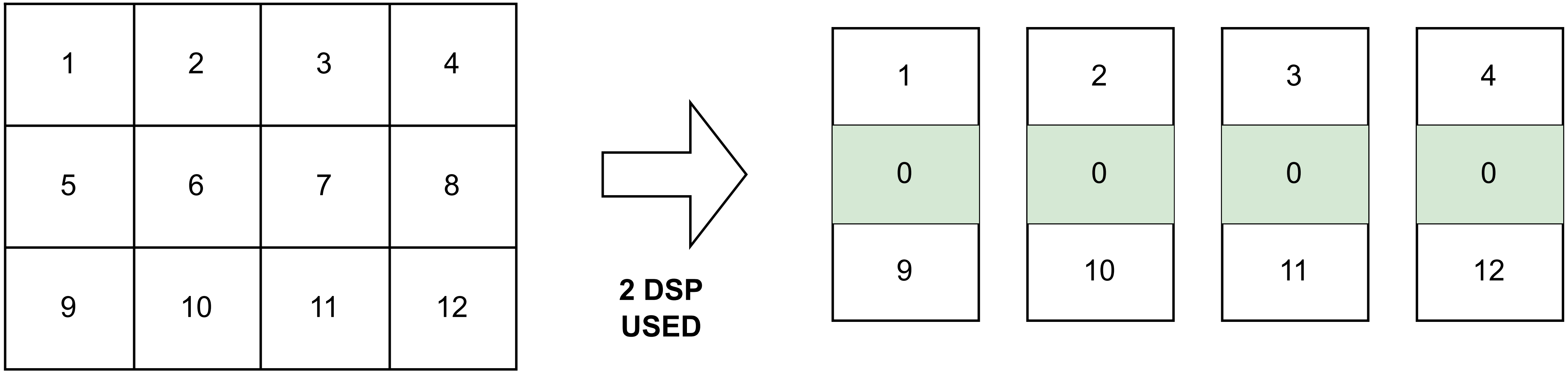
One weight = One DSP

HLS compiler optimizes any multiplications by zero

Fully unrolled designs **do not scale**

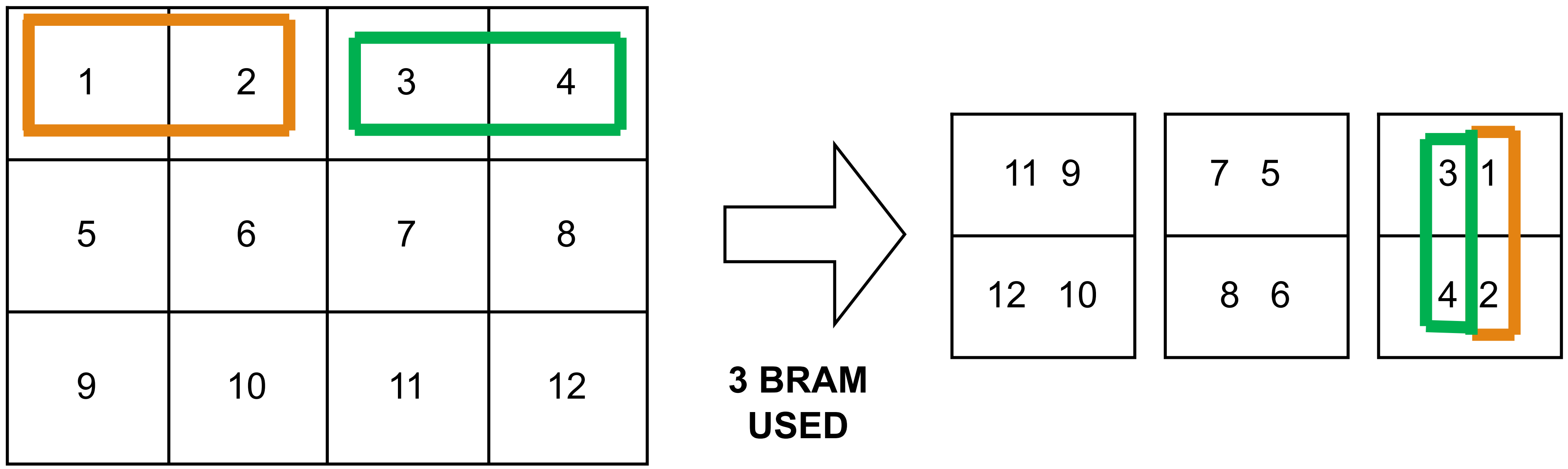
Hardware-aware pruning

Extension for RF > 1 - prune all the weights processed by the same DSP

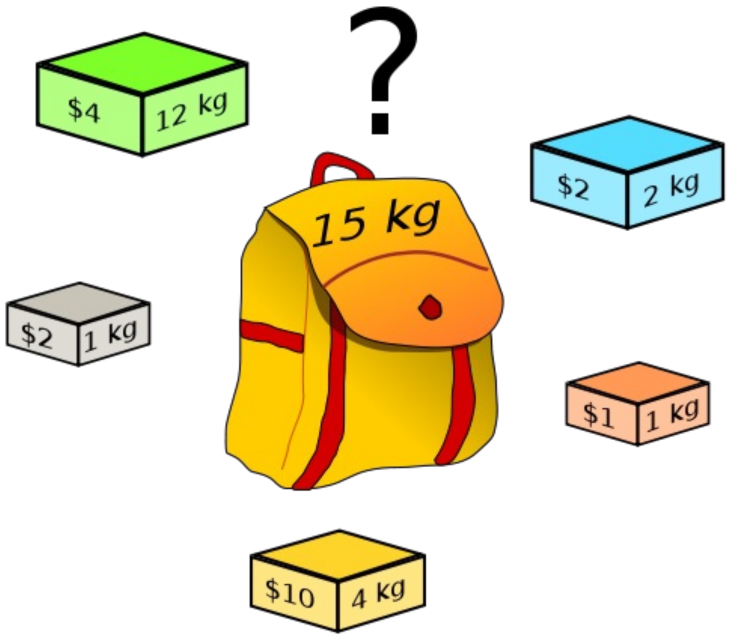


Pruning for BRAM optimisation

Group “consecutive” DSP blocks to remove one block of RAM



Knapsack problem



Given a set of n items, each with value v_i and weight w_i, what is the subset of weights that maximises value, while keeping the total weight under the capacity of the knapsack.

$$\max_{x} \boldsymbol{v}^T \boldsymbol{x}$$

$$\text{s.t. } \boldsymbol{w}^T \boldsymbol{x} \leq c$$

$$x_i \in \{0, 1\}$$

Image source: https://en.wikipedia.org/wiki/Knapsack_problem#/media/File:Knapsack.svg

Pruning algorithm

1. Identify hardware-aware tensors and add custom regularisation loss

2. Solve knapsack problem, with capacity set to $s\%$ of initial resources:

    - Selects what groups to keep and remove

3. Retrain remaining weights

4. Update sparsity $s\%$ and repeat steps 3 & 4

Optimising Vivado DSP

```python
# Optimize model
optimized_model = optimize_model(
    baseline_model, model_attributes, VivadoDSPEstimator, scheduler,
    X_train, y_train, X_val, y_val, batch_size, epochs, optimizer, loss_fn, metric, increasing, rtol
)
```

Outline

Background

Hardware-aware pruning

Results

Conclusions

Jet classification

Well-studied particle physics benchmark – Duarte *et al.* [2]

Consider 16 particle features (multiplicity, momentum etc.) and classify into **“interesting”** collisions W boson, Z boson, t quark or **“background”** collisions quark q or gluon g.

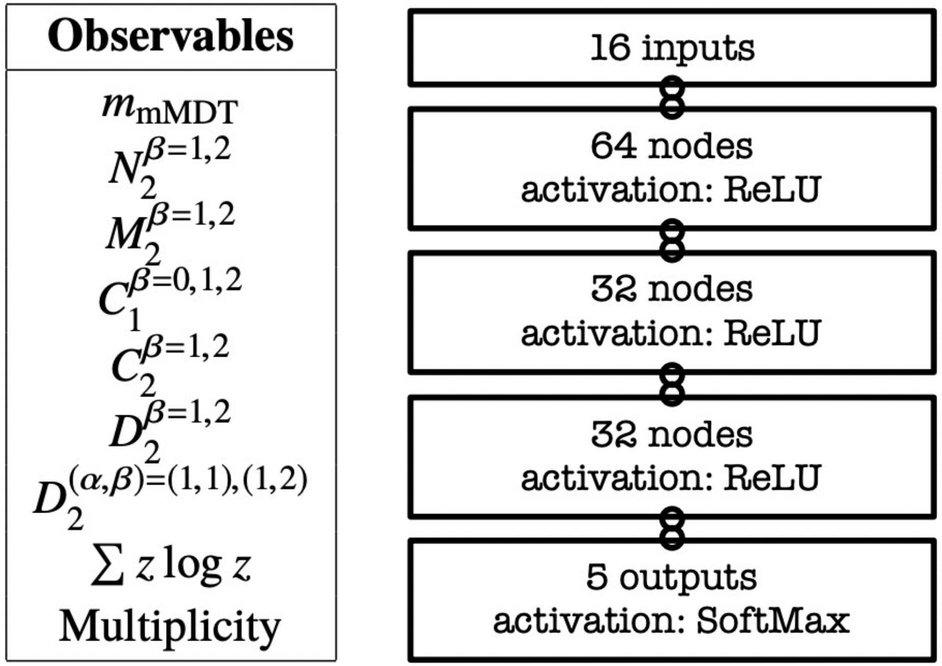


Image source: https://arxiv.org/pdf/1804.06913.pdf

Jet classification

RF	Model	Quantised accuracy [%]	Latency [ns]	LUT	FF	BRAM (reduction)	DSP (reduction)
2	BM	**76.39**	168	42,103	25,790	951	2,133
	BP-DSP	76.29	**105**	**5,504**	**3,036**	246 (3.9x)	**175 (12.2x)**
	BP-MO	76.23	**105**	9,971	3,682	**182 (5.2x)**	217 (9.8x)
4	BM	**76.39**	210	25,274	21,583	478	1,069
	BP-DSP	75.84	**161**	**6,484**	4,232	138 (3.5x)	**90 (11.9x)**
	BP-MO	75.83	**161**	6,835	**3,736**	**111 (4.3x)**	92 (11.6x)
8	BM	**76.39**	315	20,949	19,613	241	537
	BP-DSP	75.96	**252**	**9,632**	**5,488**	89 (2.7x)	**68 (7.9x)**
	BP-MO	75.76	259	10,368	5,841	**70 (3.4x)**	83 (6.5x)
16	BM	**76.39**	539	19,141	19,598	124	271
	BP-DSP	76.06	**392**	**6,693**	**5,322**	54 (2.3x)	**47 (5.8x)**
	BP-MO	75.90	413	10,701	7,630	**53 (2.3x)**	71 (3.9x)

Effects of pruning on jet classification post P&R with 7ns clock period

BM = Baseline

BP-DSP = DSP-optimised model

BP-MO = BRAM- & DSP optimised

SVHN classification

Image source: http://ufldl.stanford.edu/housenumbers/

SVHN classification

RF	Model	Quantised accuracy [%]	Latency [µs]	LUT	FF	BRAM (reduction)	DSP (reduction)
3	BM	90.80	57.03	101,111	65,43[illegible]	2,140	4,683
	BP-DSP	**92.36**	**43.58**	**59,279**	**46,5[illegible]4**	**1,550 (1.4x)**	**1,215 (3.9x)**
9	BM	90.80	90.81	55,130	**48,[illegible]66**	820	1,713
	BP-DSP	**91.06**	**84.08**	**47,854**	48,4[illegible]1	**574 (1.4x)**	**471 (3.6x)**
27	BM	90.80	212.25	50,292	**47,45[illegible]**	**252**	628
	BP-DSP	**91.88**	**205.53**	**47,658**	50,585	290	**285 (2.2x)**

Effects of pruning on SVHN classification post P&R with 8ns clock period

Fashion MNIST classification

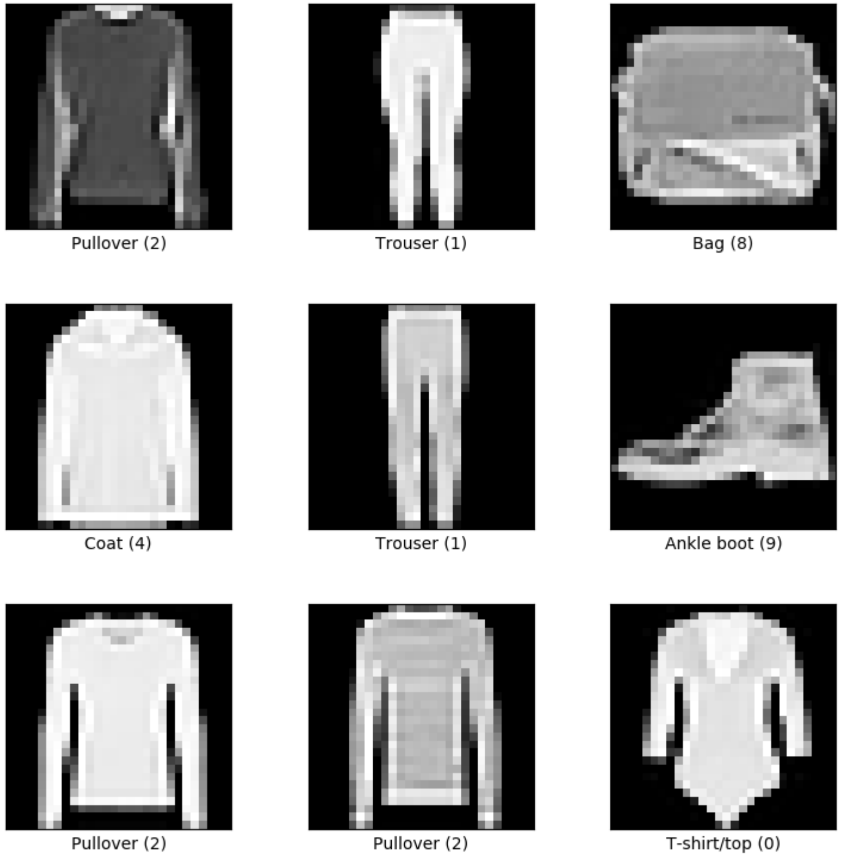


Image source: https://www.tensorflow.org/datasets/catalog/fashion_mnist

Fashion MNIST classification

Clock [ns]	Model	Accuracy [%]	Latency [µs]	LUT	FF	BRAM (reduction)	DSP (reduction)
10	BM	89.28	7.95	**88,034**	**54,650**	982	4,175
	BP-MO	**89.30**	**7.93**	90,290	64,660	**788 (1.2x)**	**881 (4.7x)**
12	BM	89.28	9.52	86,403	**52,295**	982	4,175
	BP-MO	**89.30**	**9.50**	**85,845**	63,092	**466 (2.1x)**	**881 (4.7x)**

Effects of heterogonous pruning on Fashion MNIST classification post P&R

Outline

Background

Hardware-aware pruning

Results

Conclusions

Conclusions

A novel, **hardware-aware pruning method**, derived from the underlying mapping to hardware and modelled using linear programming

Extensions to hls4ml, now fully supporting **quantisation-aware training with QKeras**, **hardware-aware pruning** and **real-time inference**

Between **55%** and **92%** reductions in **DSP** and up to **81%** in **BRAM** utilisation

Future:

Extensions to other platforms and layers

Integration with mixed pruning and quantisation methods

Links & More

Source code:

- https://github.com/fastmachinelearning/hls4ml/pull/768
- https://github.com/fastmachinelearning/hls4ml/pull/809

Branch & Docs:

- https://github.com/fastmachinelearning/hls4ml/tree/hardware-aware-pruning

Questions?

References

[0] Parts of this presentation were adopted from an earlier presentation given to the FastML community: “hls4ml Optimization API”. April 21st 2023.

[1] C. N. Coelho, A. Kuusela *et al.*, “Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors,” Nature Machine Intelligence, vol. 3, no. 8, p. 675–686, 2021

[2] J. Duarte, S. Han *et al.,* “Fast inference of deep neural networks in FPGAs for particle physics,” Journal of Instrumentation, vol. 13, no. 07, p. P07027, jul 2018. [Online]. Available: https://dx.doi.org/10.1088/1748-0221/13/07/P07027

[3] T. Aarrestad *et al.*, “Fast convolutional neural networks on FPGAs with hls4ml,” Machine Learning: Science and Technology, vol. 2, no. 4,p. 045015, jul 2021. [Online]. Available: https://dx.doi.org/10.1088/2632-2153/ac0ea1

[5] Y. Lecun, J. Denker, and S. Solla, “Optimal brain damage,” vol. 2, 01 1989, pp. 598–605.

[6] S. Han, J. Pool et al., “Learning both Weights and Connections for Efficient Neural Network,” in NIPS, 2015

[7] M. Shen et al. HALP: Hardware-Aware Latency Pruning. 2021