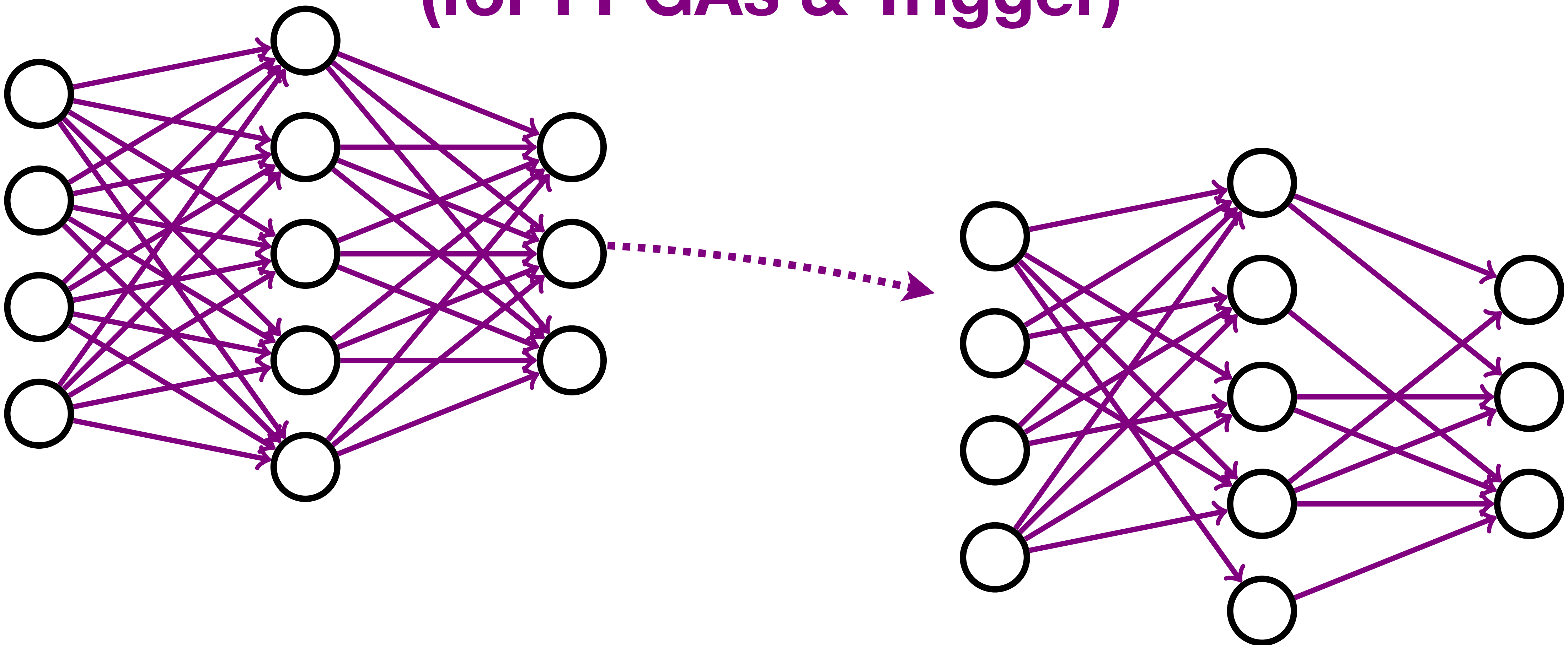
Compression of Deep Neural Networks (for FPGAs & Trigger)



Javier Duarte
Fermilab

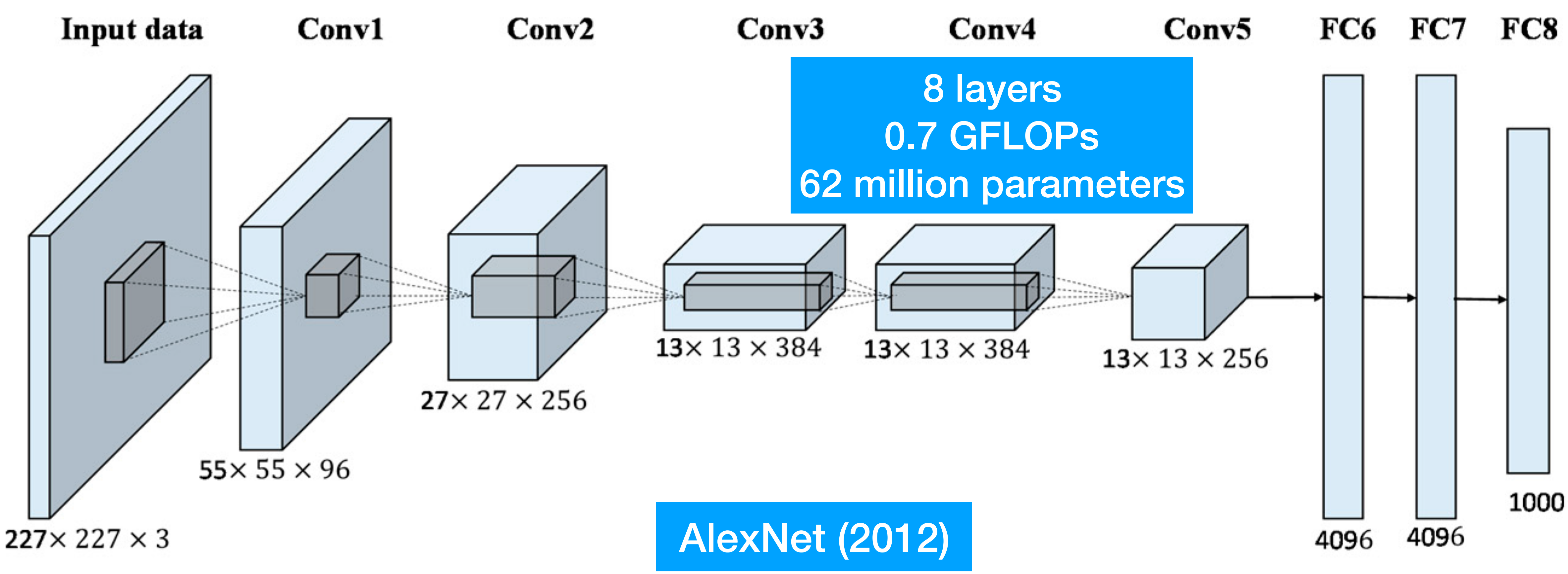
Reconstruction, Trigger, and Machine Learning for the HL-LHC
MIT
4/27/2018

Outline

- Introduction and Motivation
  - Why compress neural networks?
- Compression of neural networks
  - Example: iterative retraining with regularization
  - Other techniques
- Examples of Compressed CNNs
  - SqueezeNet
  - Energy-Aware Pruning
  - Ternary/Binary Nets
- Summary and Outlook

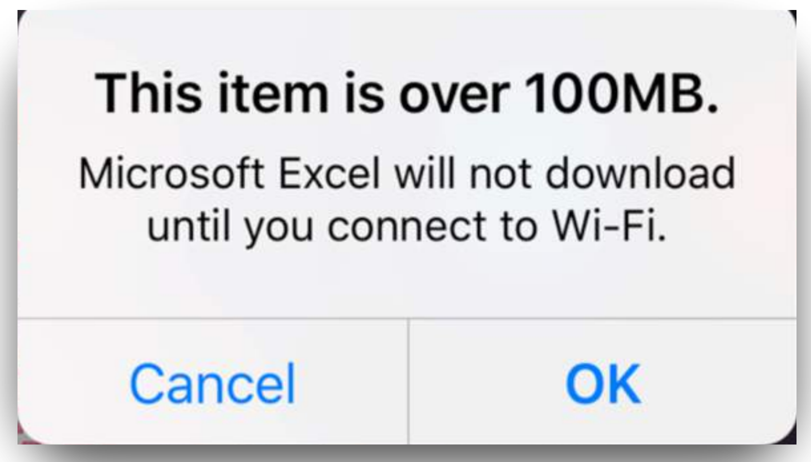
Neural Network Overparametrization

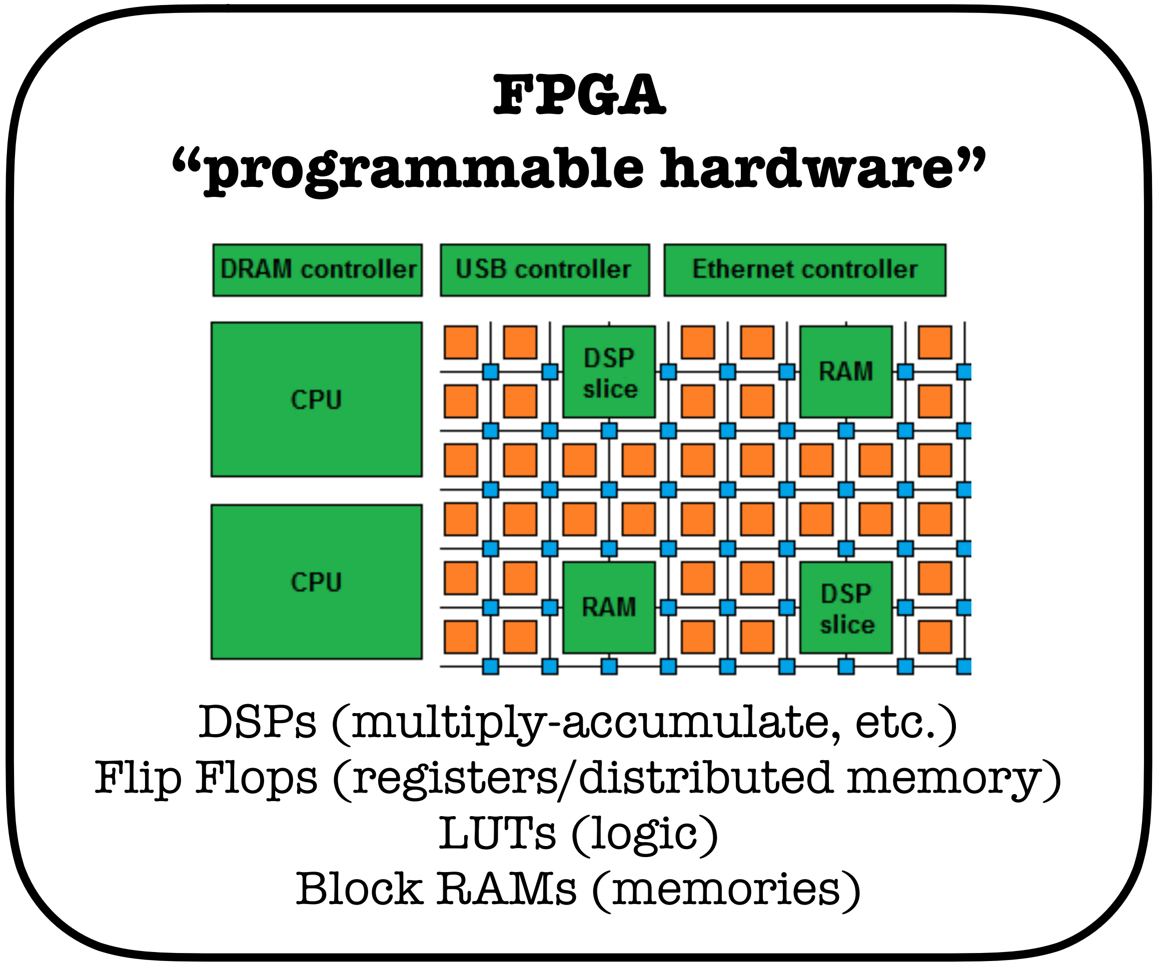
- Neural Networks are generally **overparametrized**
- You can control overfitting (dropout, regularization, large training samples, …) but in the end you have a model with **many redundant weights**
- For applications with **limited memory**, **resources**, or **power** want to minimize network **size**, **complexity**, and **memory references**



Why compress?

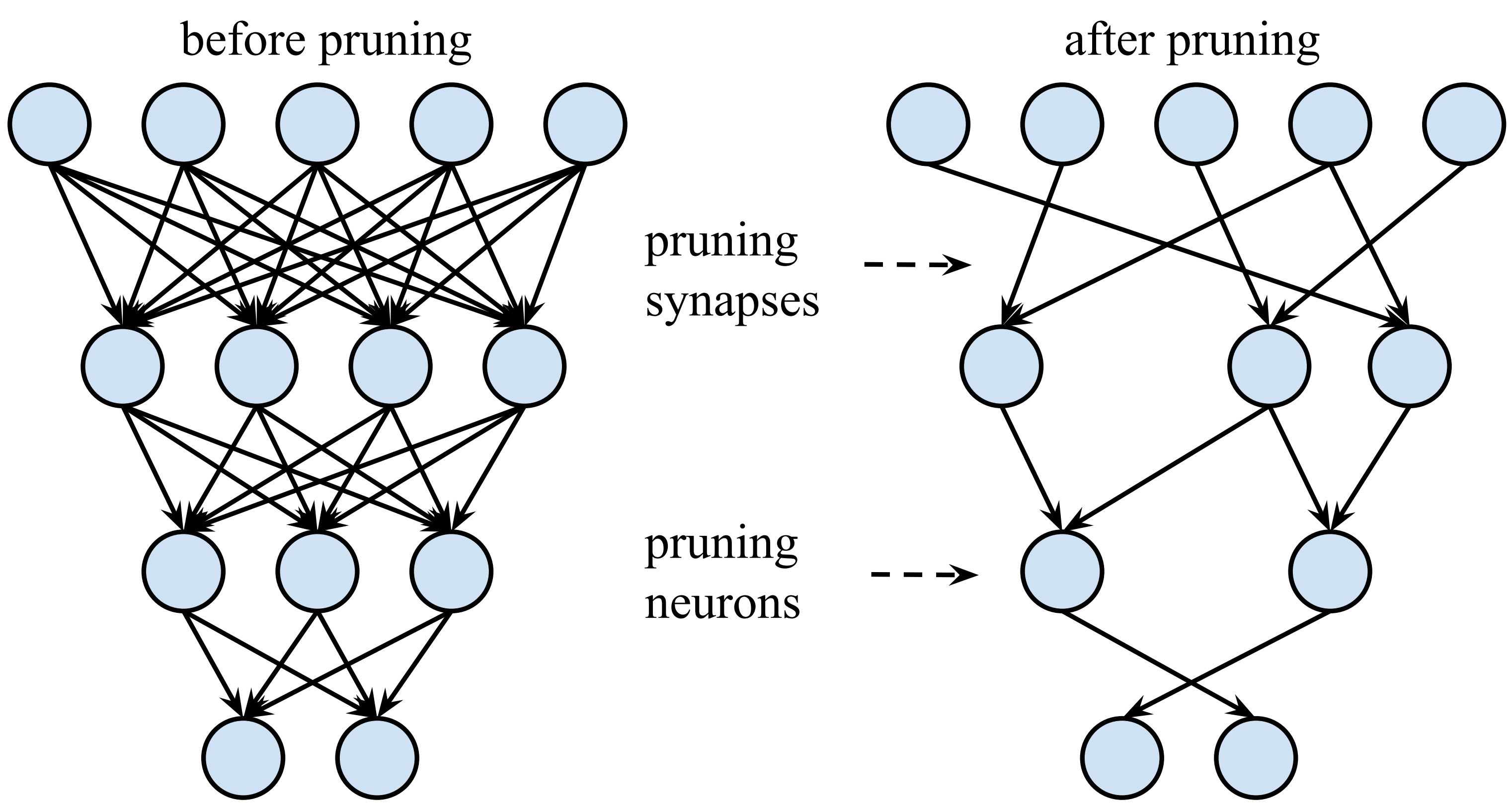
- If you can substitute matrix multiplication for sparse matrix multiplication, you can speed up computations especially on highly parallelized architectures like **FPGAs** (skip unnecessary computations)
- Reducing size and energy consumption is better for mobile applications





Efficient Neural Networks

- Compression/Pruning
  - Removing redundant synapses and neurons
- Quantization
  - Restrict the weights, biases, and activations to certain quantized values
  - Fixed point, integers, ternary, binary, etc.



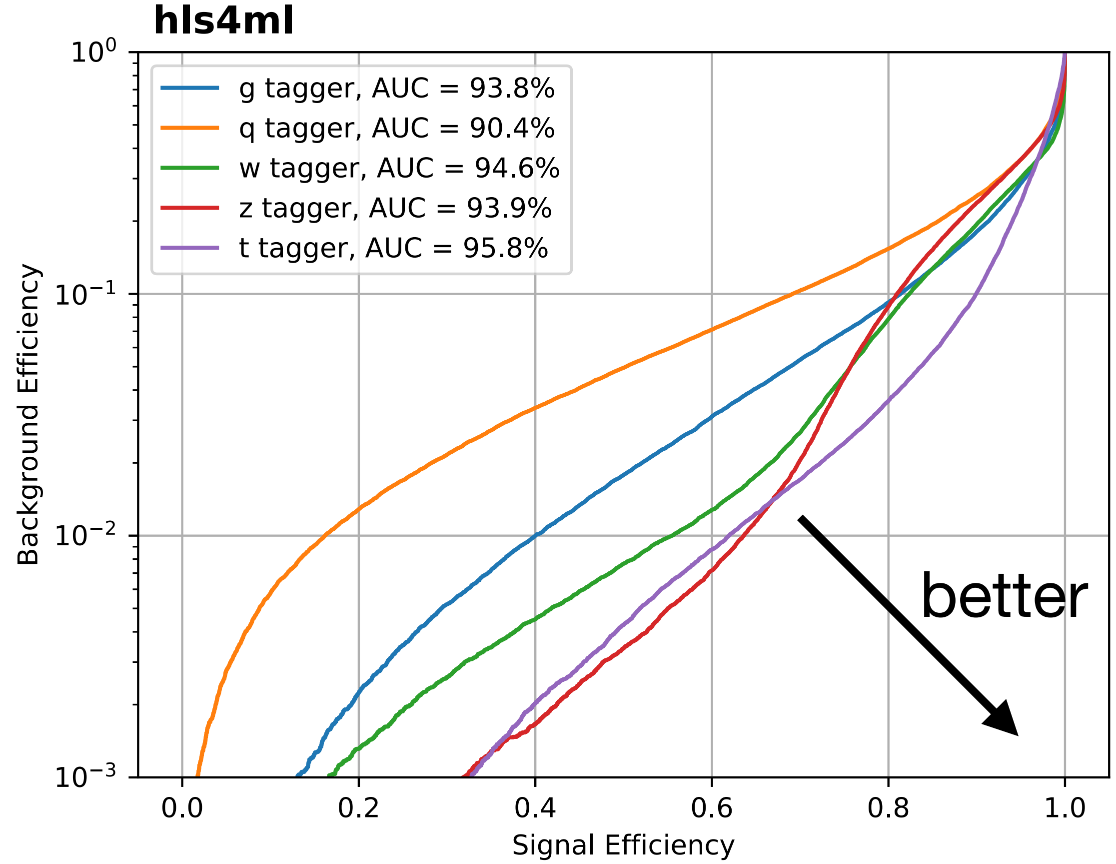
For further reading: [arXiv:1510.00149](https://arxiv.org/abs/1510.00149)

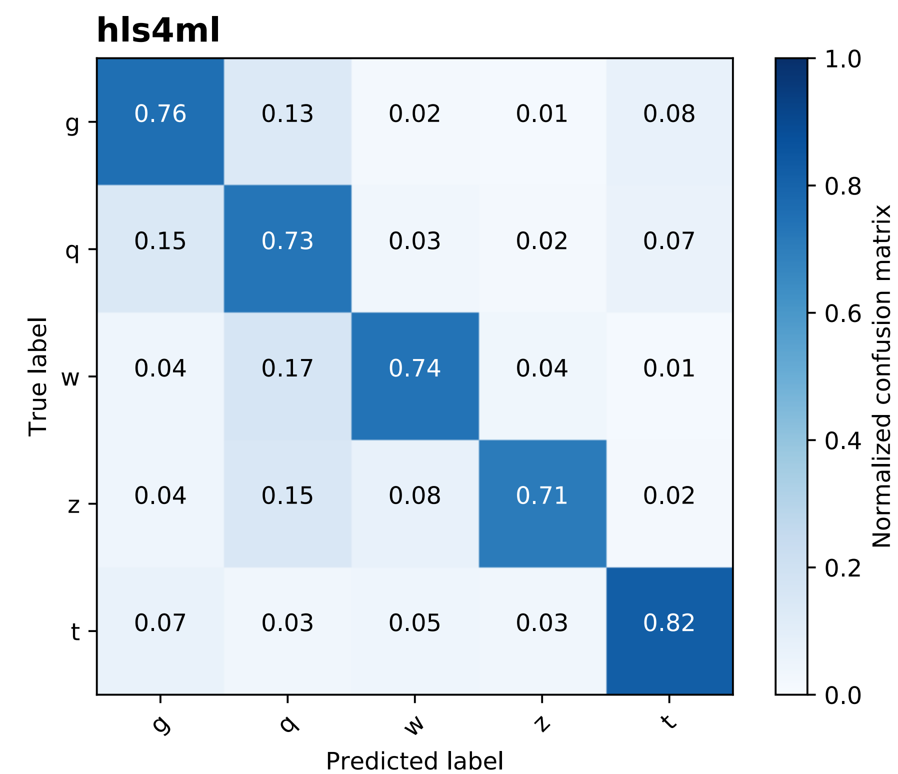
Simple Example: Jet Substructure

- 5 output multi-classifier
  - Does a jet originate from a quark, gluon, W/Z boson, top quark?
- Fully connected network
- 16 expert inputs
  - jet mass, multiplicity, ECFs

16 inputs
↓
64 nodes, ReLU
↓
32 nodes, ReLU
↓
32 nodes, ReLU
↓
5 outputs, softmax

auc = area under ROC curve

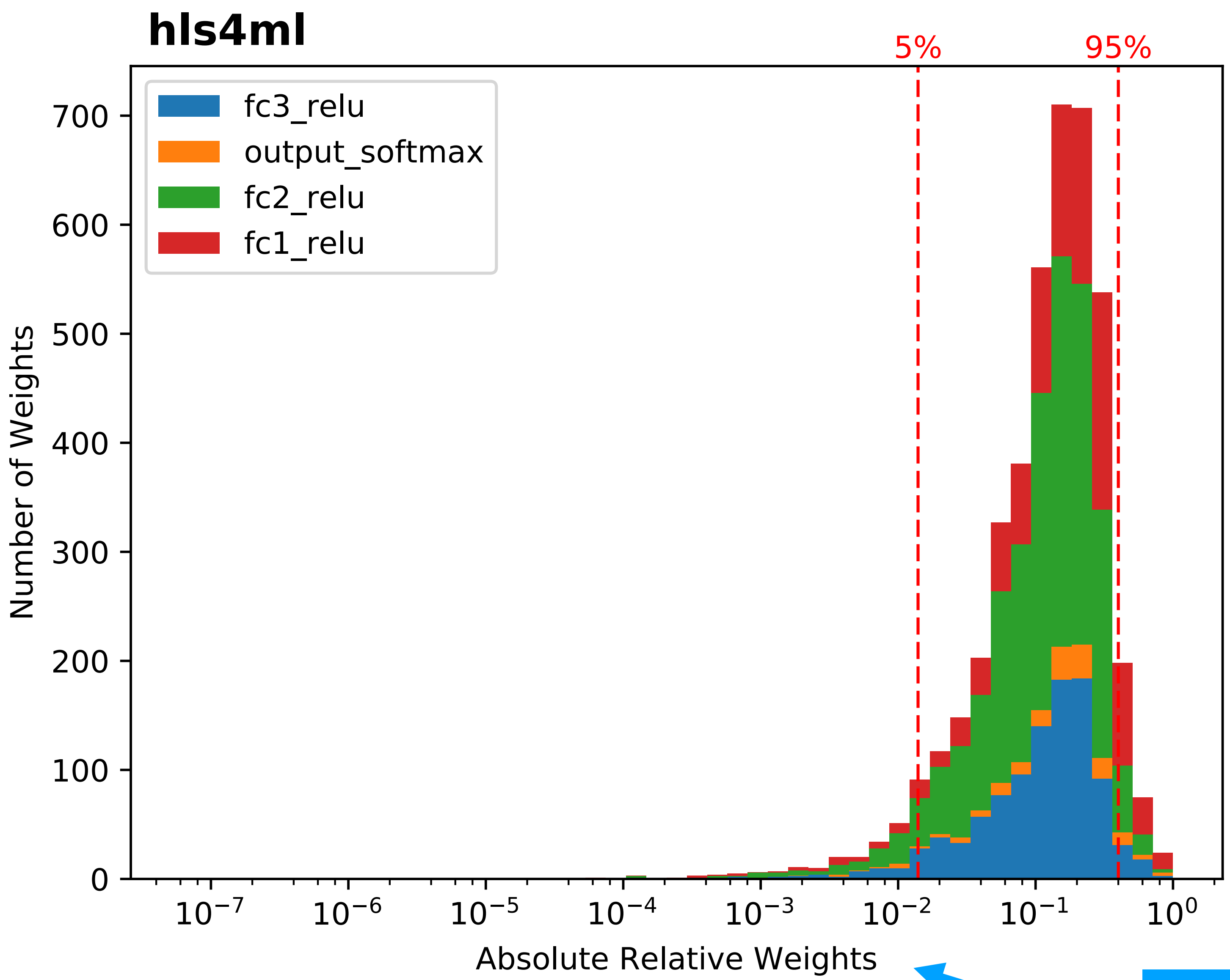


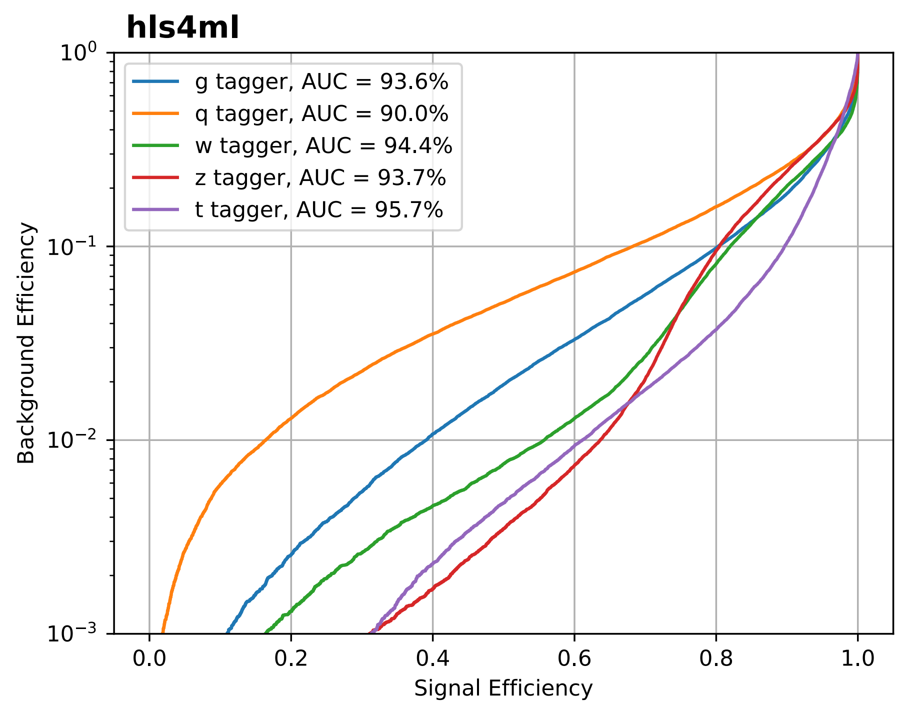


Distribution of Weights

https://github.com/hls-fpga-machine-learning/keras-training

- Distribution of weights after training
- Not obvious which weights to prune





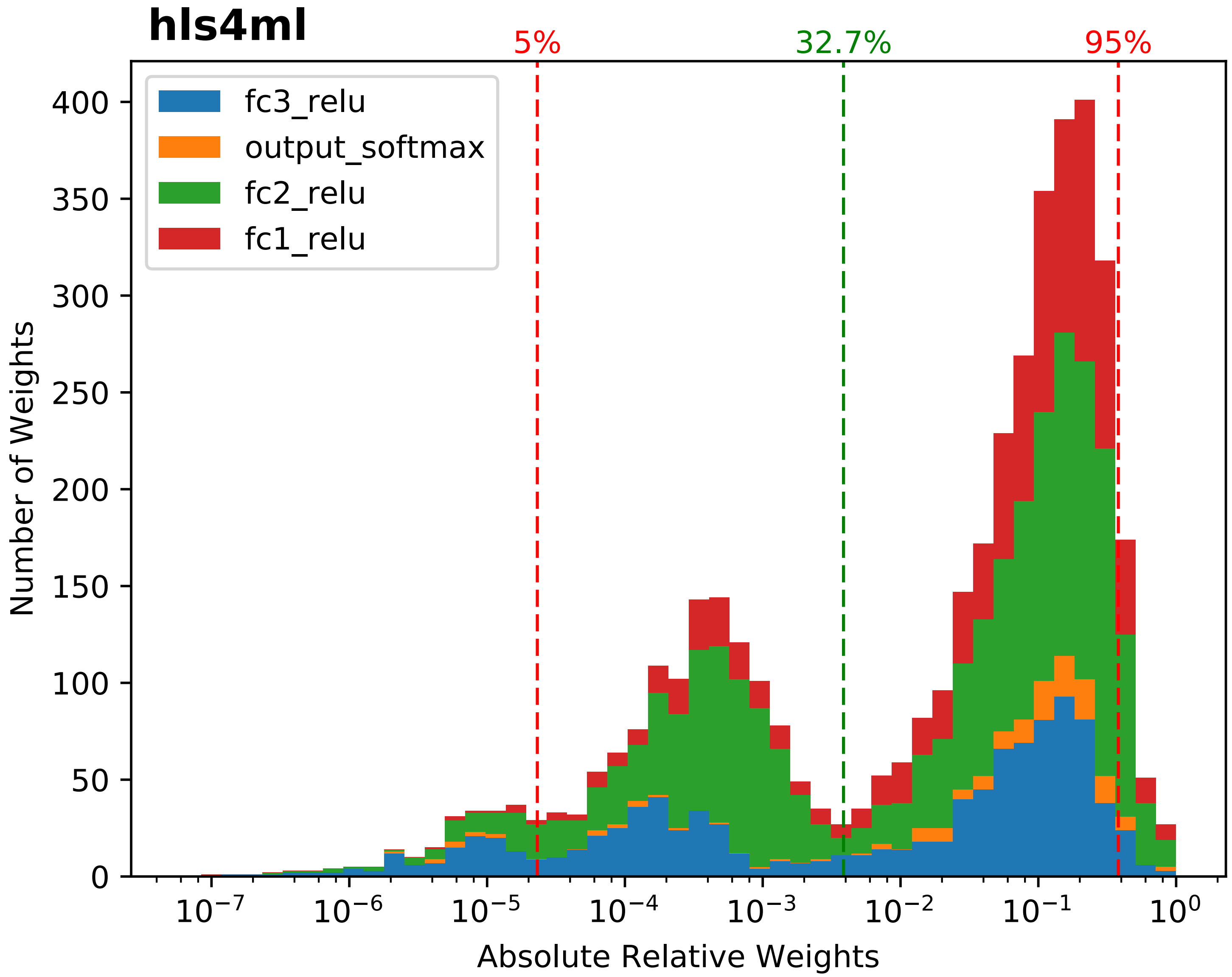
Normalized to the max in each layer

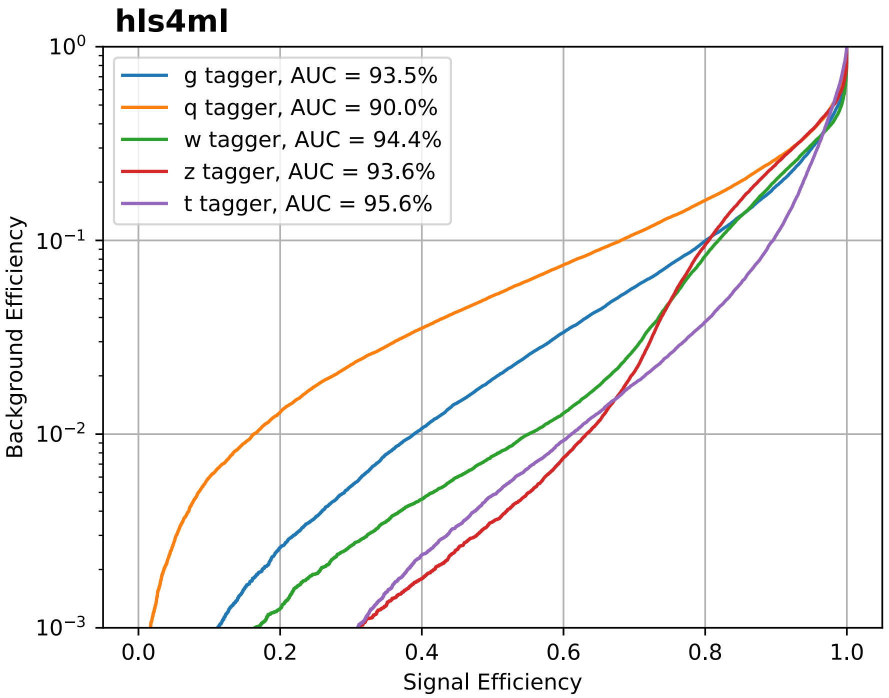
Weights after Regularization

- Distribution of weights after training with L_1 regularization, lambda = 10^{-4}
- Two populations of weights

$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1$$

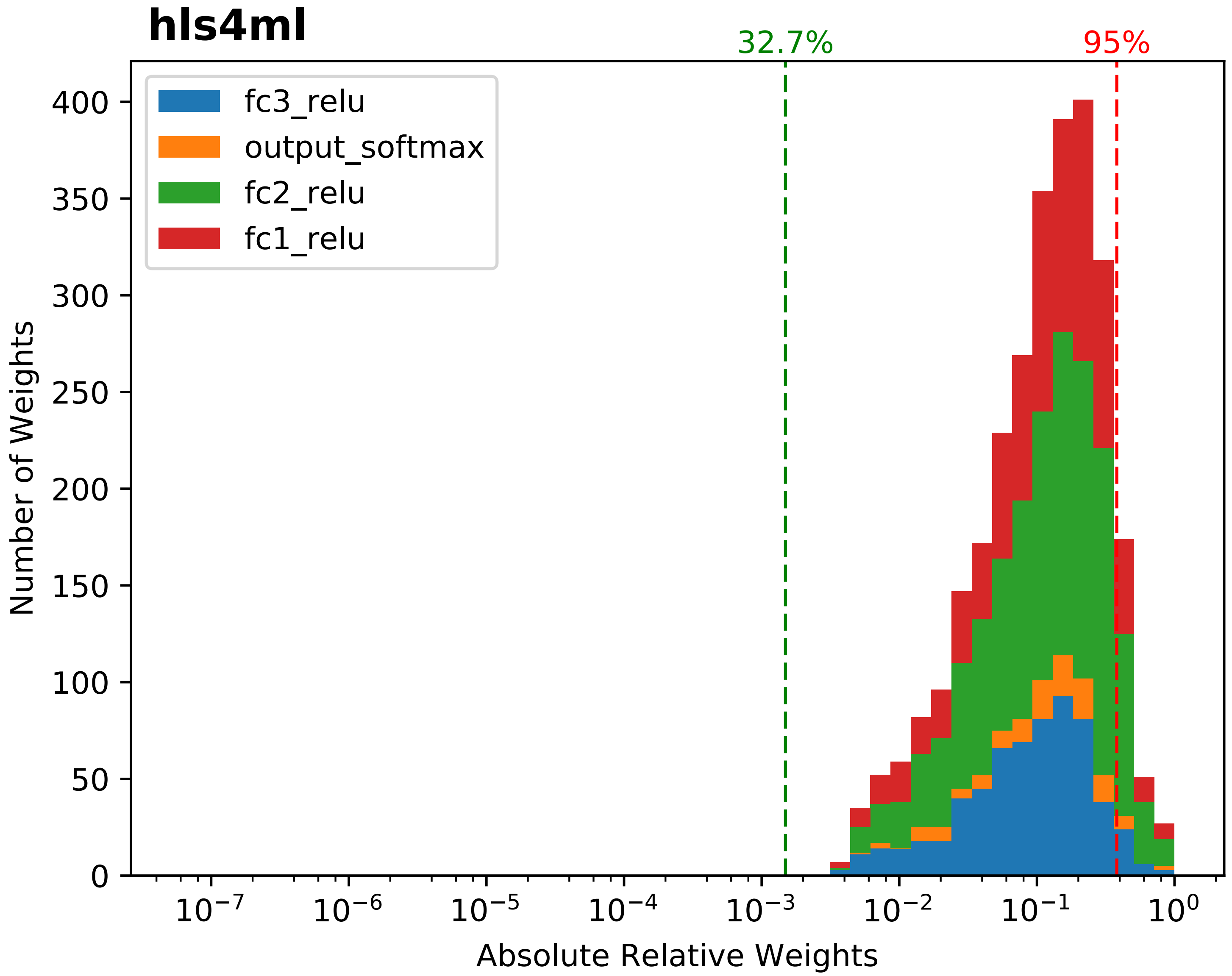
$$\|w\|_1 = \sum_i |w_i|$$

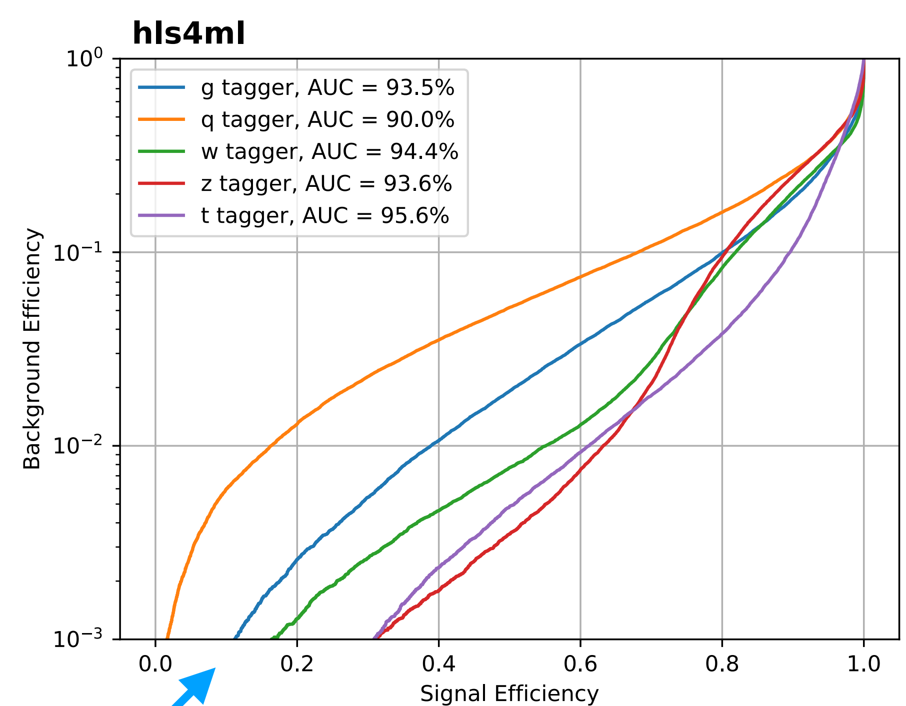




Weights after Pruning

- Prune the bottom population
- No effect on output classifier (even before retraining!)

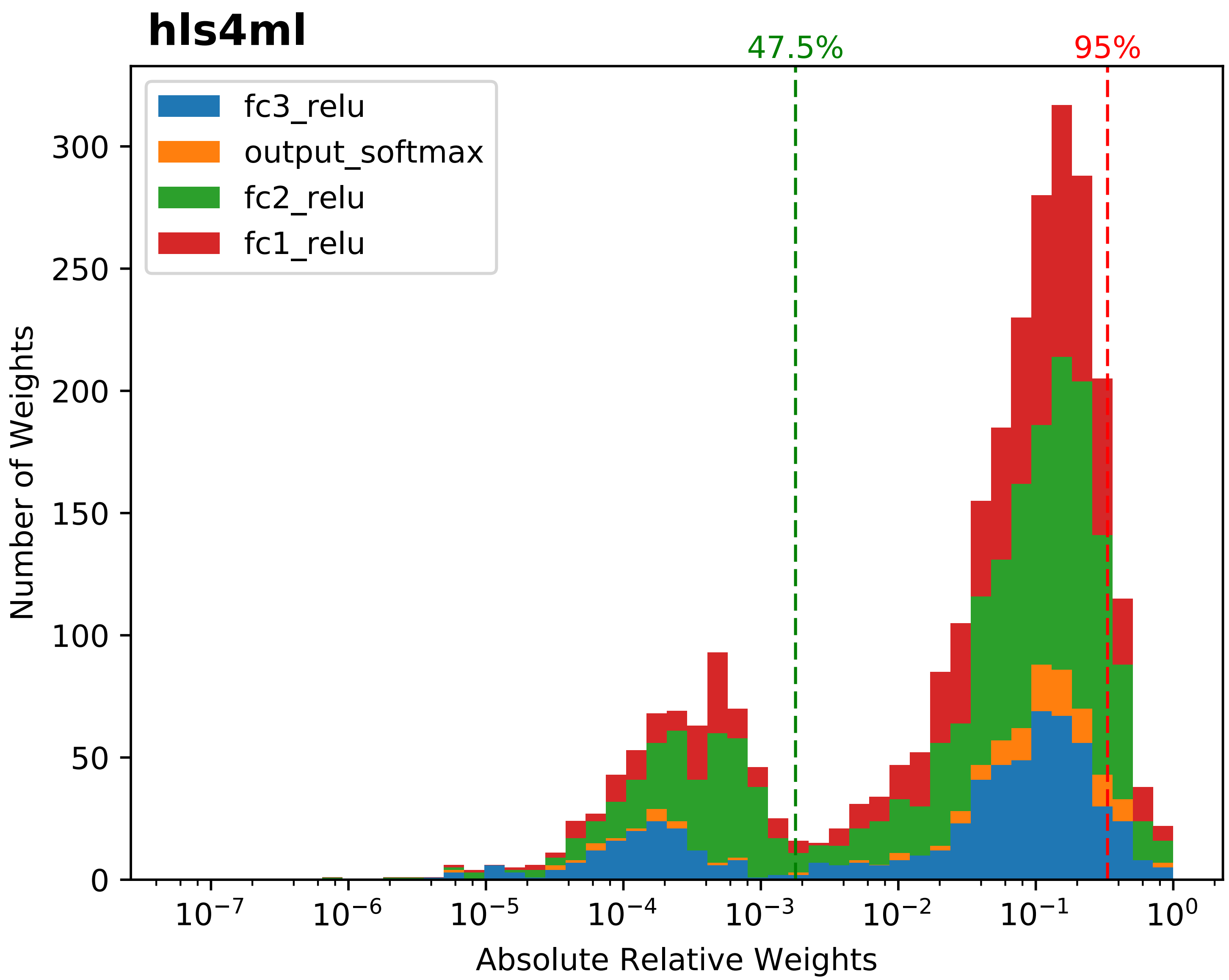




No effect

Retraining with Constraints

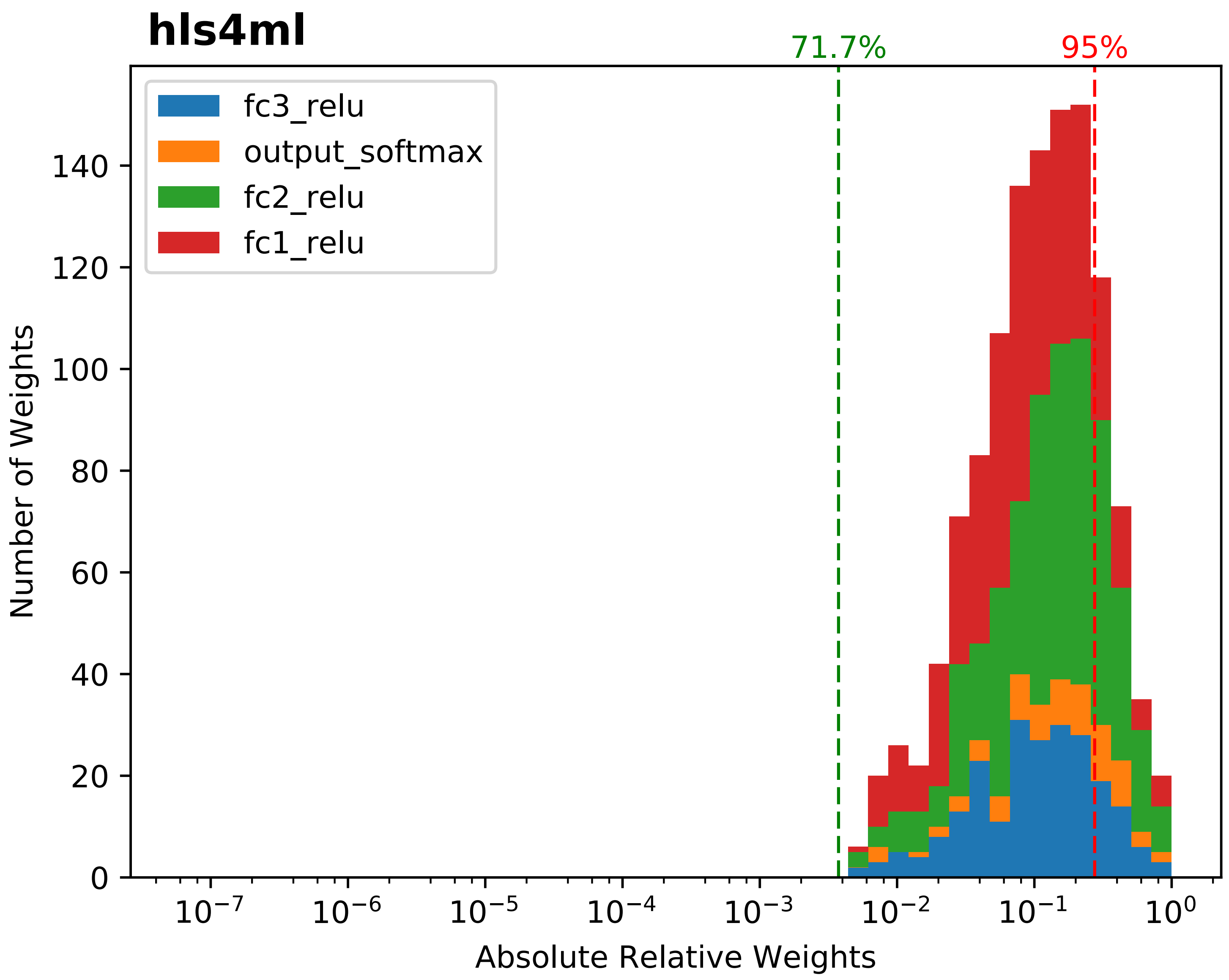
- Retrain with [kernel constraint]() to keep the pruned weights fixed to zero
- Keep L_1 regularization (to find additional weights to prune)

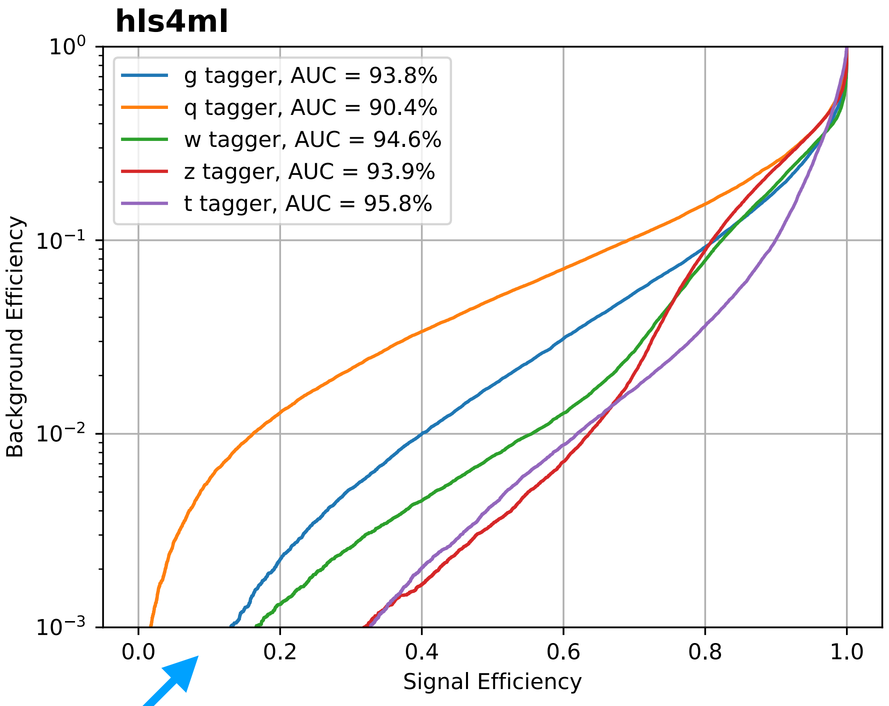



```python
class ZeroSomeWeights(Constraint):
    """ZeroSomeWeights weight constraint.

    Constrains certain weights incident to each hidden unit
    to be zero.

    # Arguments
        binary_tensor: binary tensor of 0 or 1s corresponding to which weights to zero.
    """

    def __init__(self, binary_tensor=None):
        self.binary_tensor = binary_tensor

    def __call__(self, w):
        if self.binary_tensor is not None:
            w = multiply([w,  K.variable(value=self.binary_tensor)])
        return w

    def get_config(self):
        return {'binary_tensor': self.binary_tensor}
```

Weights after Iterative Pruning & Retraining

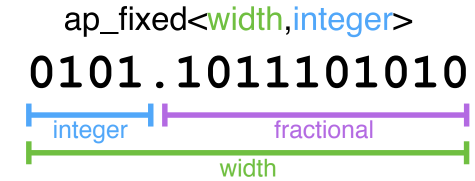
- After 7 iterations, pruned away 72% of the weights
- No effect on output classifier

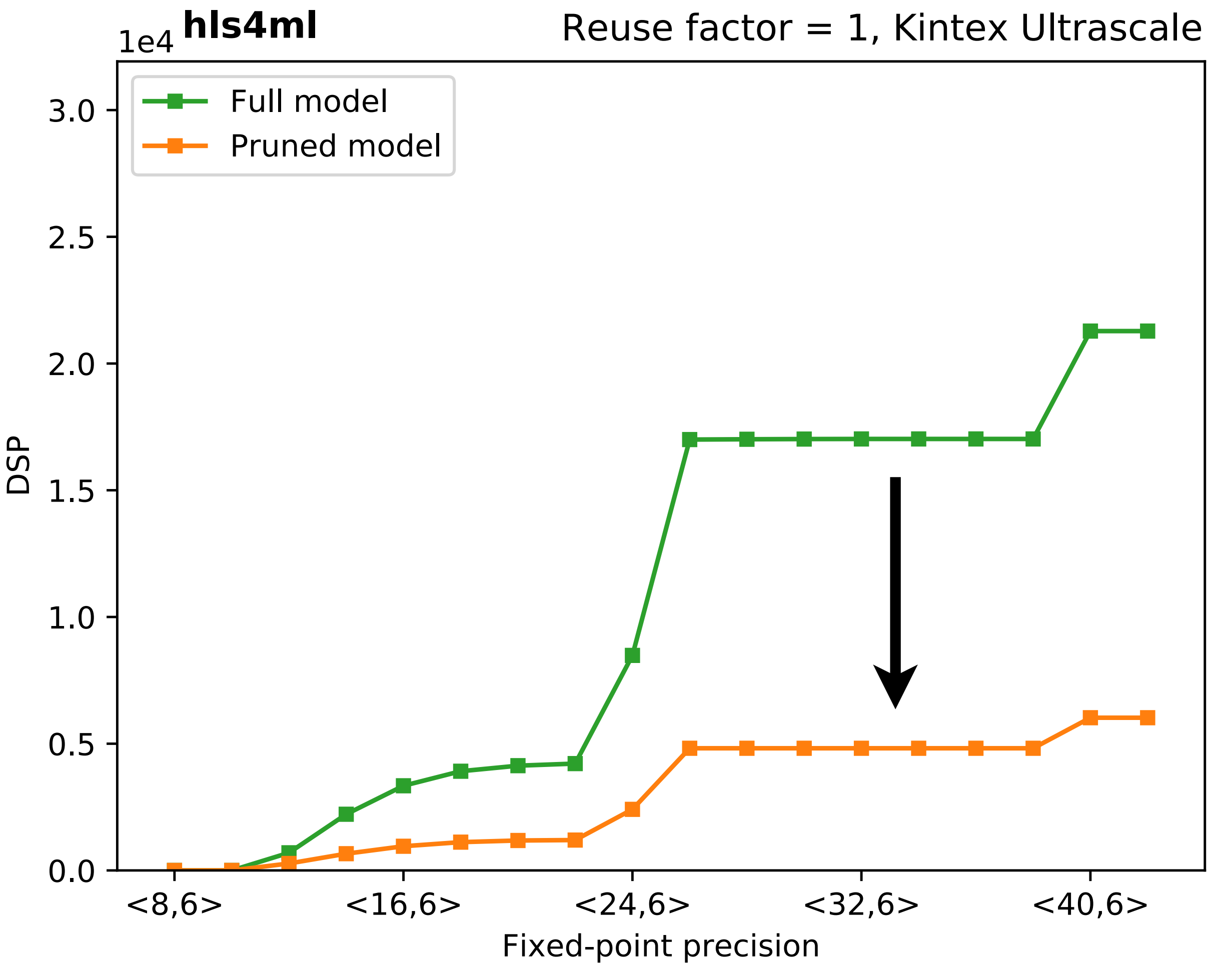


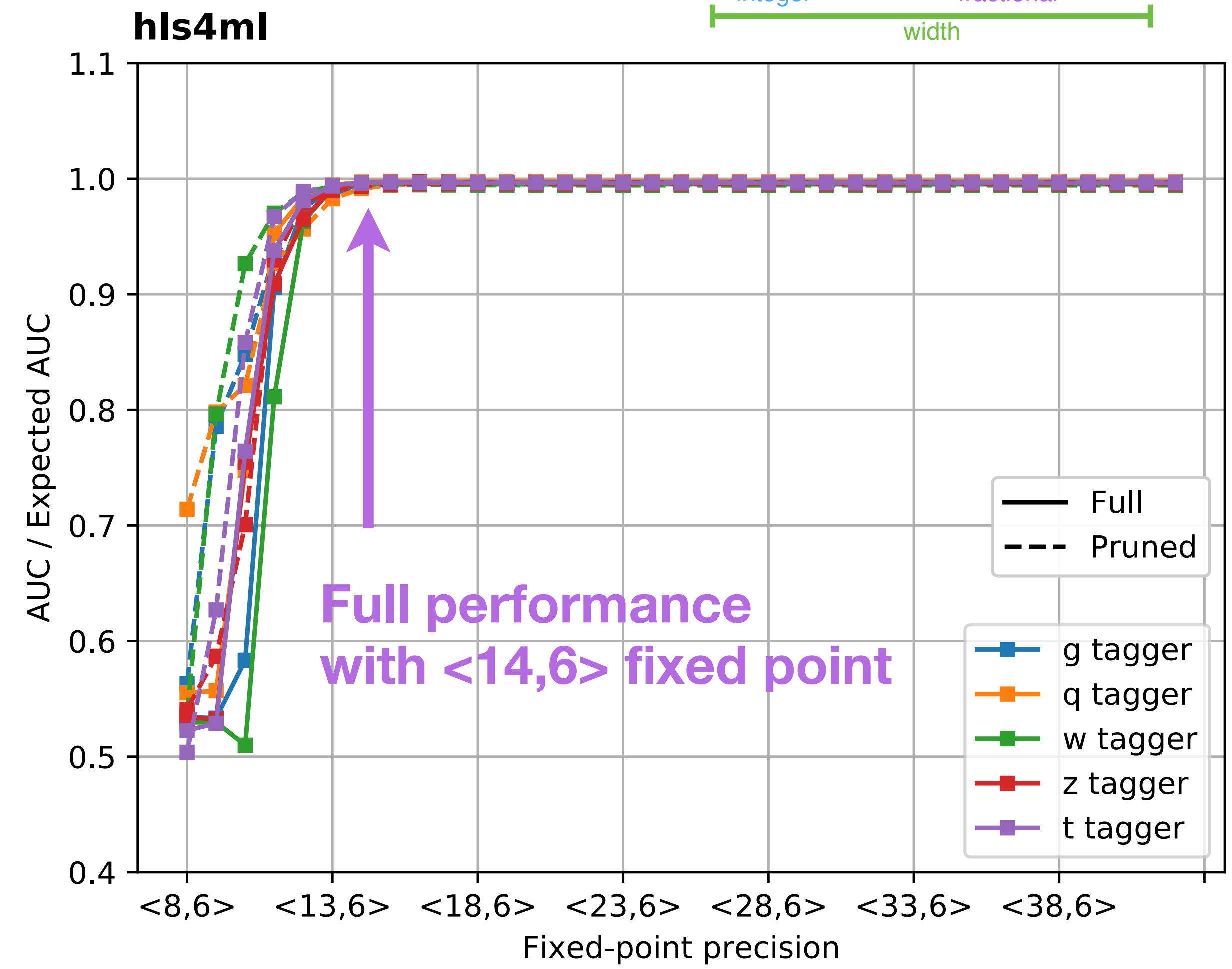


No effect

Effect of Compression for FPGA Inference





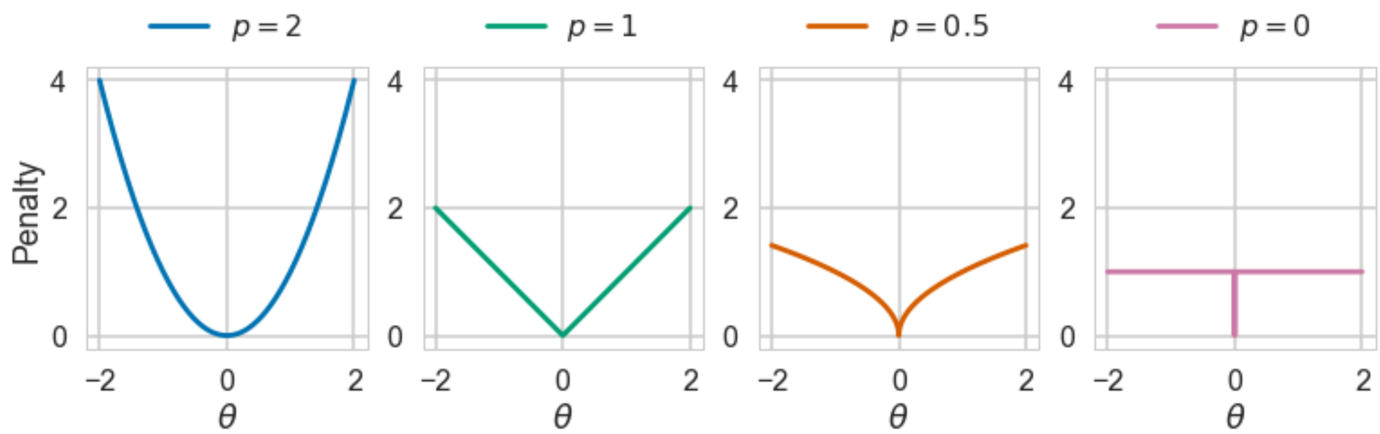


- Big reduction in DSP usage with pruned model!
- Note: we didn't retrain using quantized weights (should get us down to ~8 bits instead of ~14 bits)

Other Compression Schemes

Louizos et al. 2017
arXiv:1712.01312

$$\|w\|_p = \left(\sum_i |w_i|^p\right)^{1/p}$$



- Train with L_p ($0 \leq p < 1$) regularization to promote sparsity (though difficult to optimize)
  - as $p \to 0$, $L_p \to L_0$
- "Optimal brain damage": use second derivatives of loss function to rank **parameter saliency** (rather than using parameter magnitude)

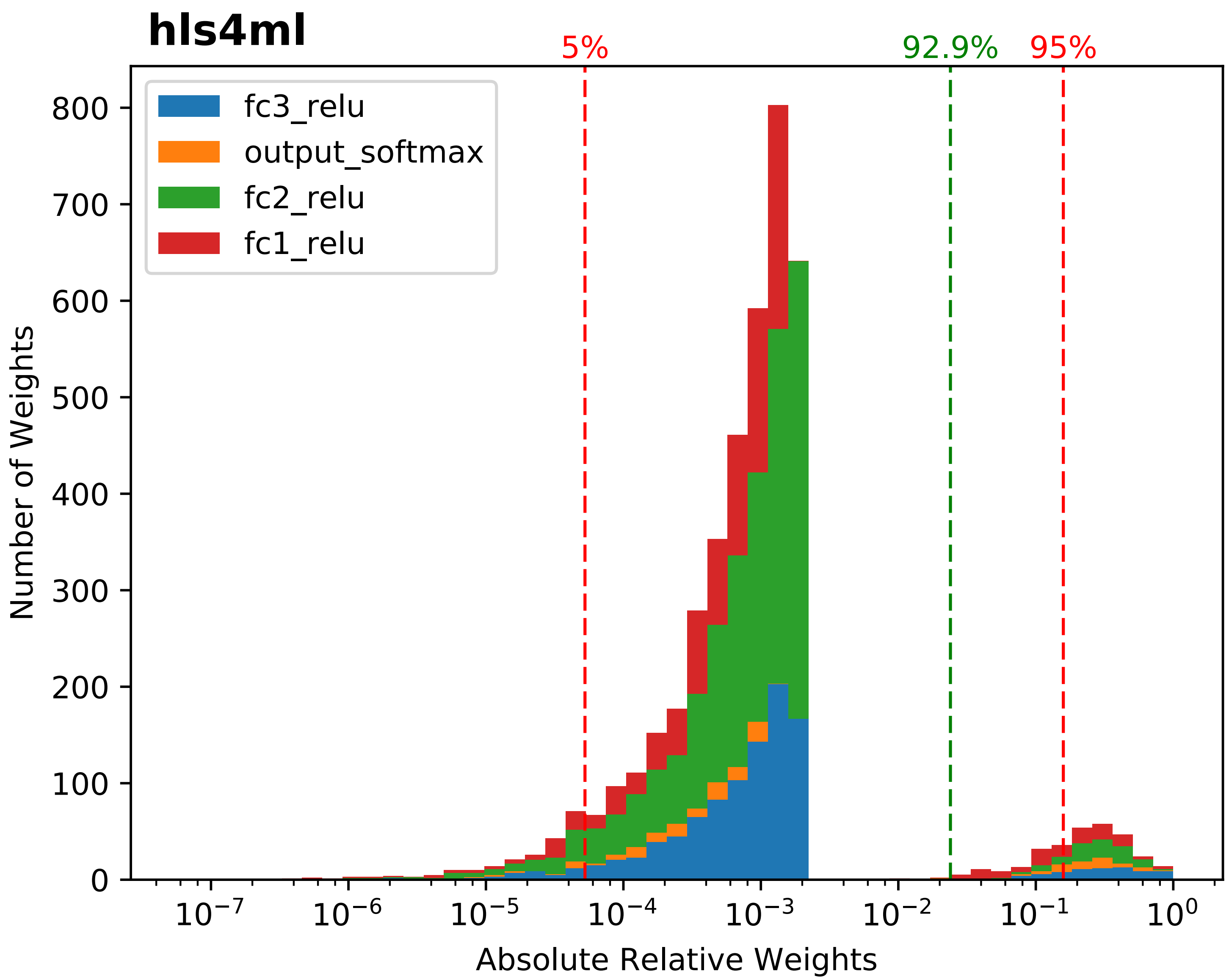
LeCun et al. 1989
NIPS 250

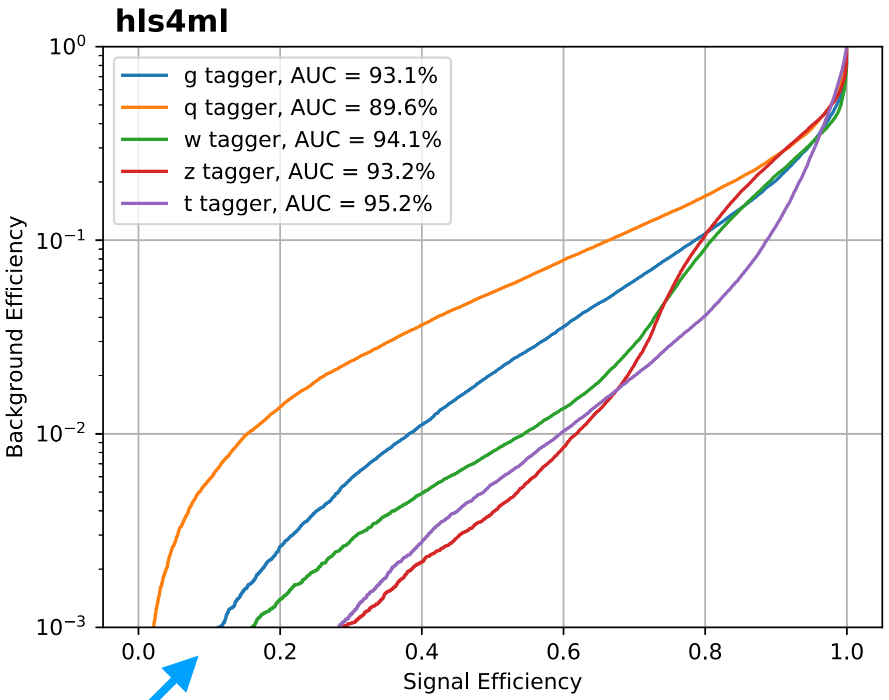
- Deep Compression
  - Trained quantization
  - Weight-sharing using k-means clustering to identify weights to share
  - Huffman coding (optimal prefix)

Han et al. 2015
arXiv:1510.00149

Aggressive L_p Regularization

- Train with L_p, $p = 1/10$, $\lambda = 10^{-3}$ can prune away 93% of weights
- Small effect on output classifier



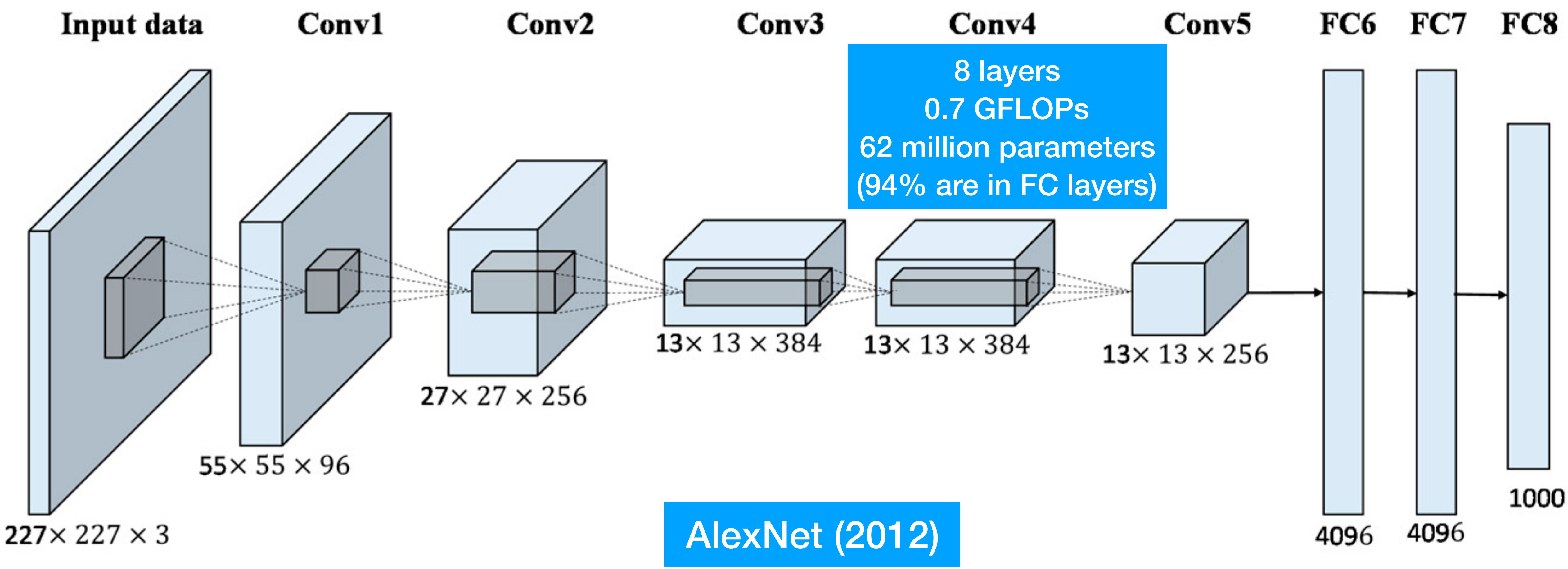


Small effect

Big Convolutional Neural Networks

- Main task is computer vision/image recognition
- Control the number of parameters by baking in assumptions like locality and translation invariance to **share weights** within a layer

Krizhevsky, et al.
[NIPS 4824]

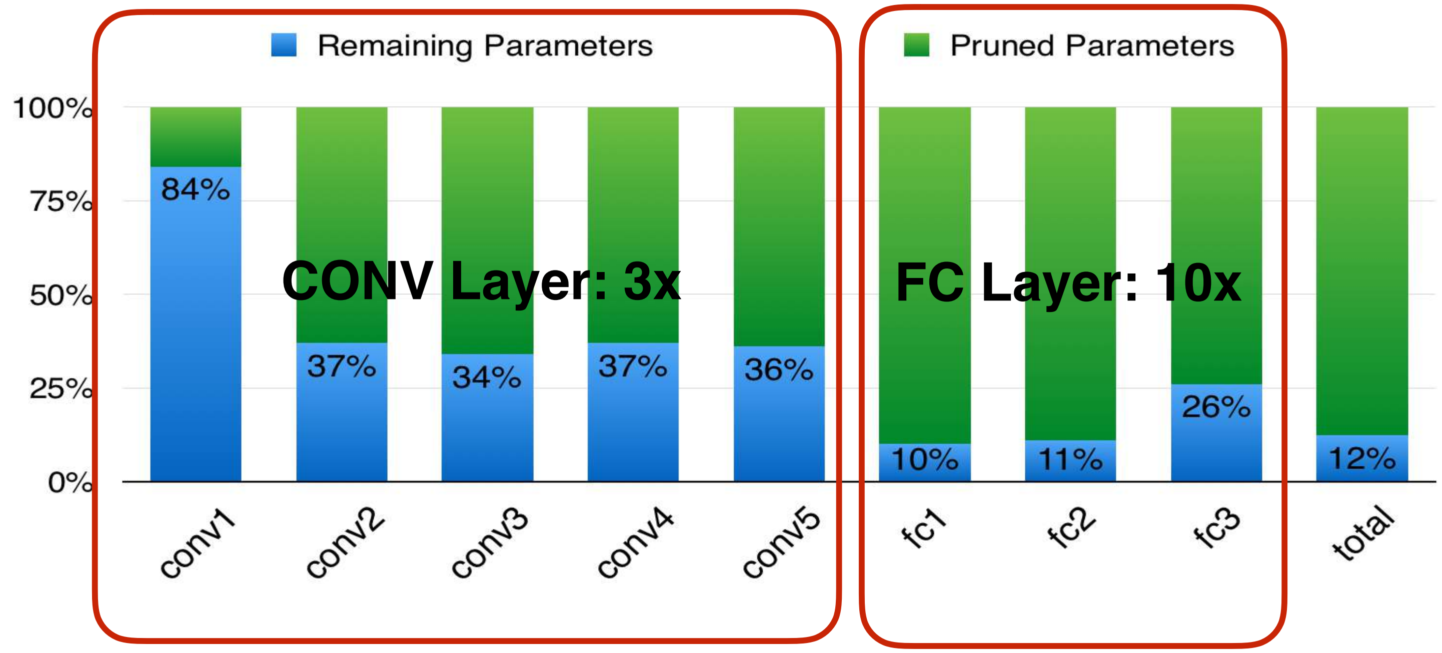


Pruned AlexNet

Han et al. 2015
arXiv:1510.00149

- Using “Deep Compression” can prune AlexNet by factor of 35x (Han et al. 2015)

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%



SqueezeNet

Han et al. 2016
arXiv:1602.07360

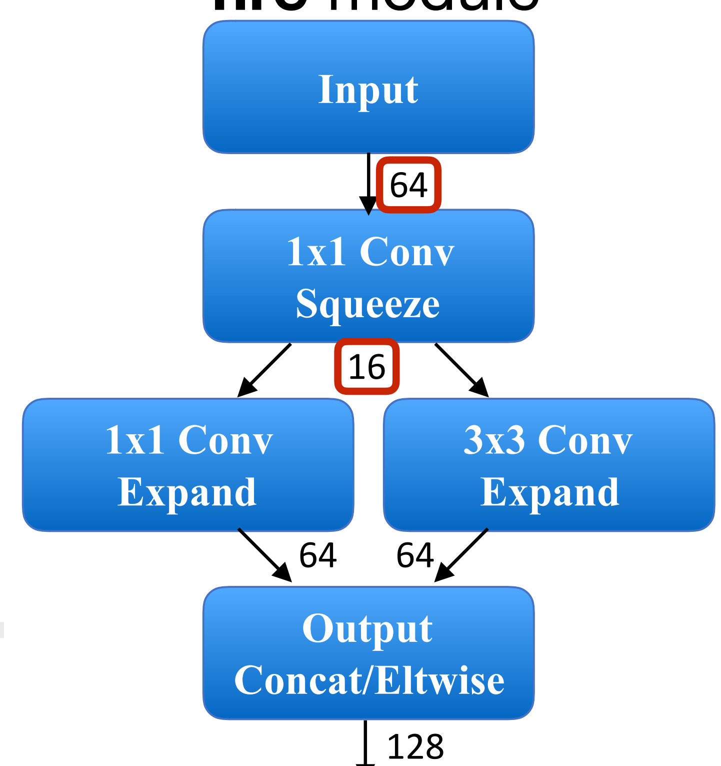
- 6-bit SqueezeNet smaller than 32-bit AlexNet by a factor of 510 and achieves the same accuracy (Han et al. 2016)
- Not just a compressed AlexNet, but re-thinking of architecture

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

Strategies:

1. **Replace 3x3 filters with 1x1 filters (9x fewer parameters)**
2. Decrease the number of input channels to 3x3 filters using squeeze layers
3. Downsample late in the network so that convolution layers have large activation maps

microarchitecture:
fire module

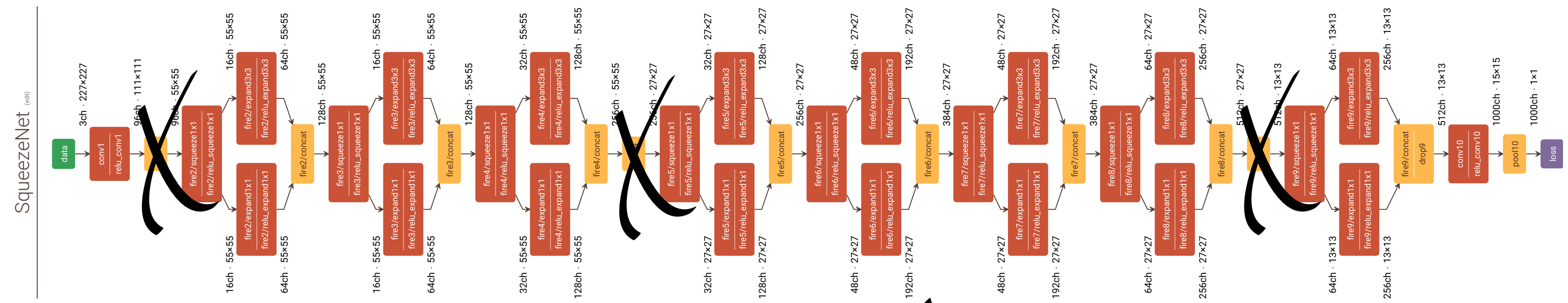


SqueezeNet on FPGA

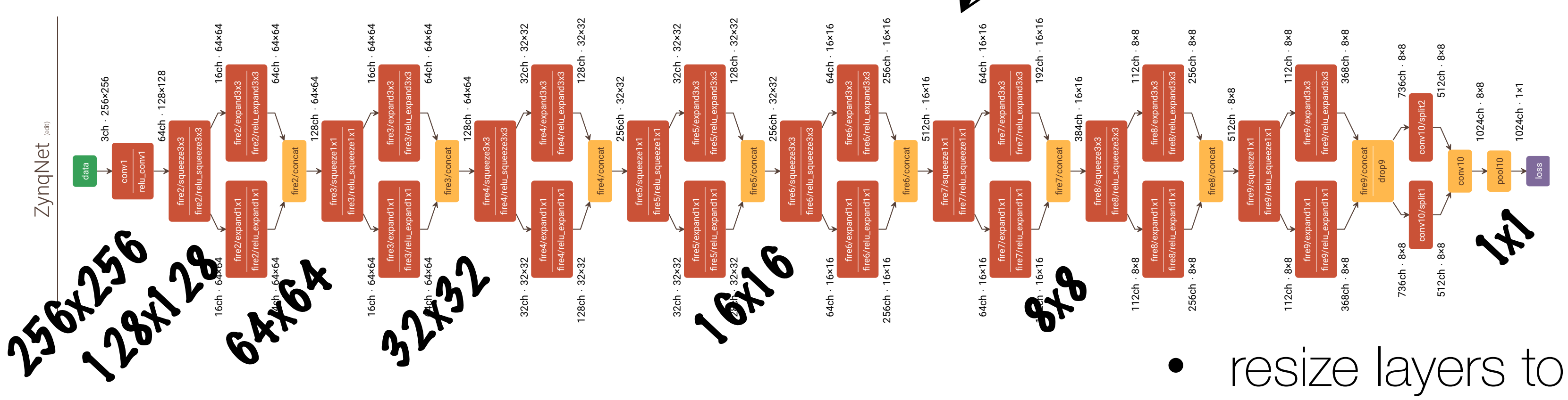
- Fits on one FPGA with on board memory (Gschwend 2016)

Additional optimization for FPGA

Gschwend 2016
[ZynqNet]



- remove Pooling



- resize layers to 2^N

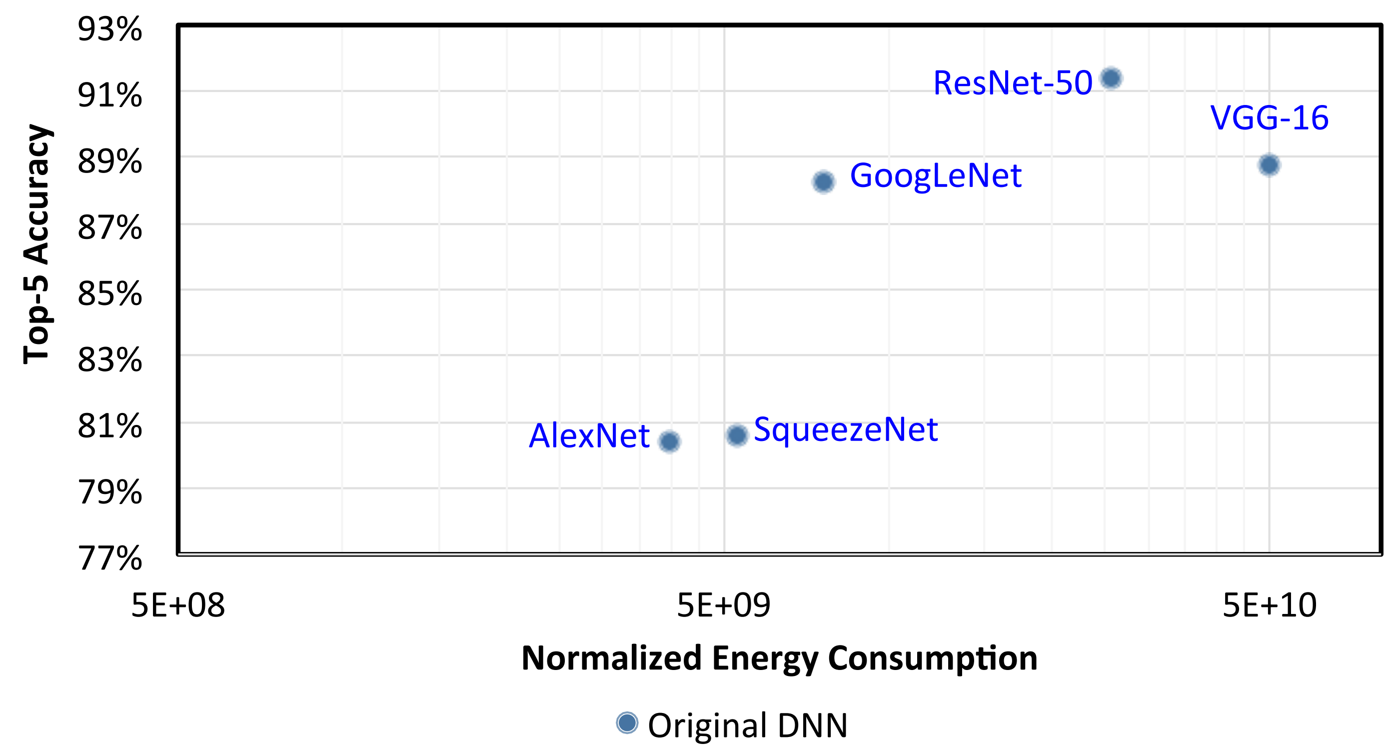
FPGA resources (Kintex 7)

resource	Block RAM	DSP Slices	FF	LUT
used	996	739	137k	154k
available	1090	900	437k	218k
utilization	91 %	82 %	31 %	70 %

Energy-Aware Pruning

Yang et al. 2017
arXiv:1611.05128

- Key insights:
  - Less operations do not necessarily mean less energy consumption
  - CONV layers dominate the overall energy consumption
  - Prune while directly optimizing for energy consumption

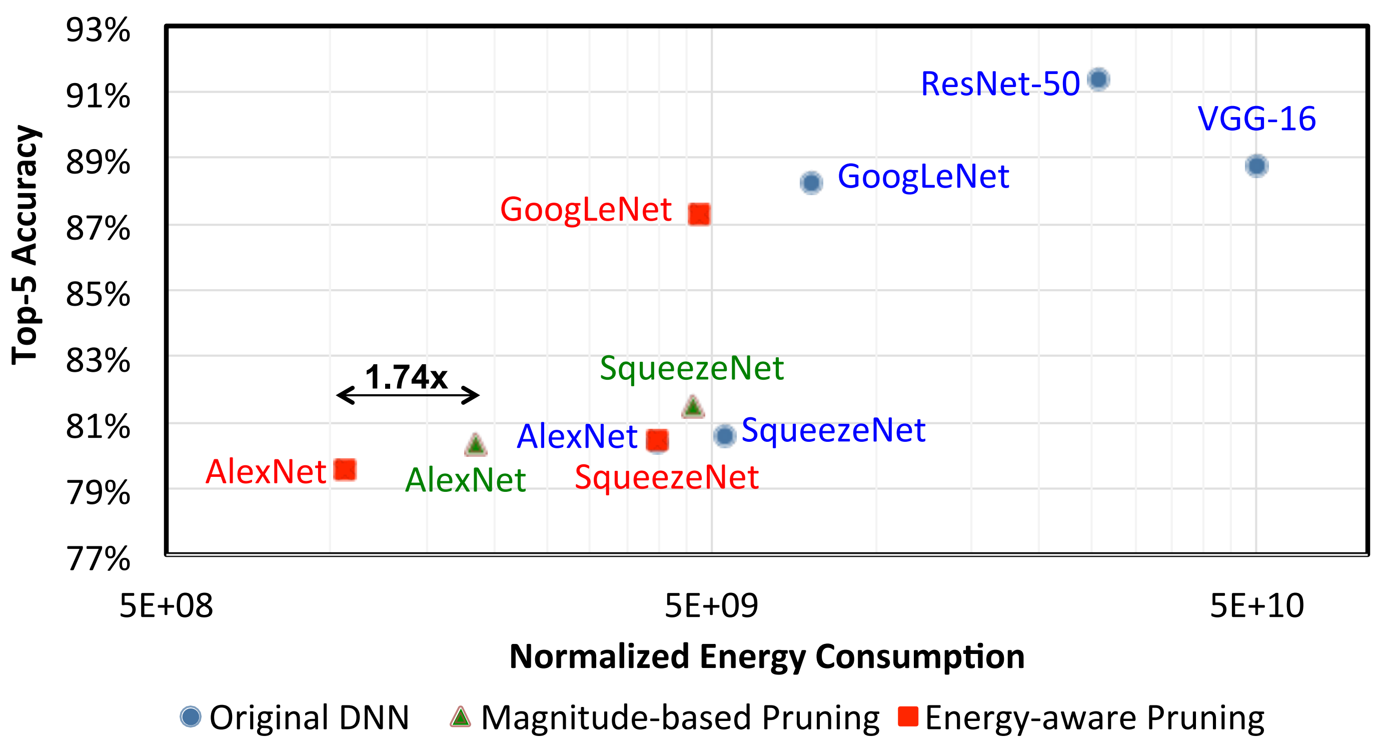


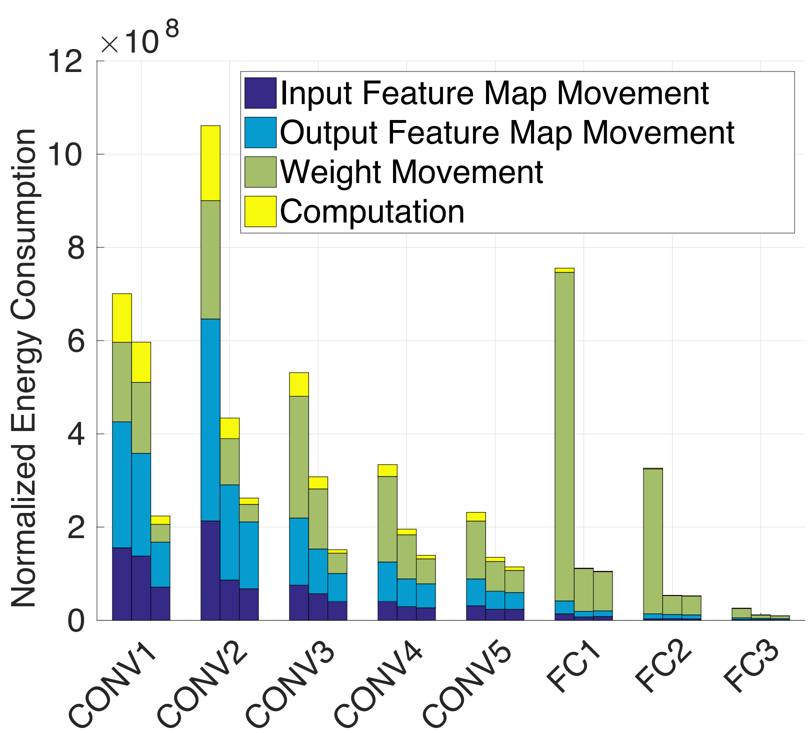
How much energy does your NN consume?
https://energyestimation.mit.edu/

Energy-Aware Pruning

Yang et al. 2017
[arXiv:1611.05128](https://arxiv.org/abs/1611.05128)

- Key insights:
  - Less operations do not necessarily mean less energy consumption
  - CONV layers dominate the overall energy consumption
  - Prune while directly optimizing for energy consumption



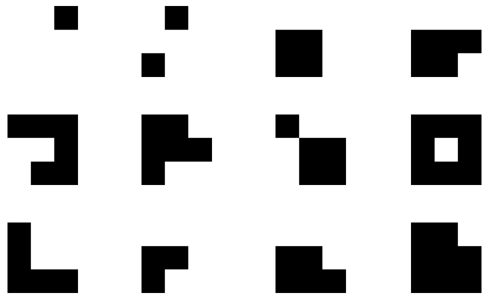


Left-to-right: AlexNet, pruned AlexNet, energy-aware pruned AlexNet

Binary/Ternary Networks

- Ultimate of quantization/compression
  - BinaryConnect, BinaryNet: weights (+1, -1)
  - Binary Weight Nets: weights (+w, -w),
  - Ternary Weight Nets: weights (+w, 0, -w)
  - Trained Ternary Quantization: (+w_1, 0, -w_2)

Binary weight filters



Sze et al. (Survey)
arXiv:1703.09039

Binary weight filters

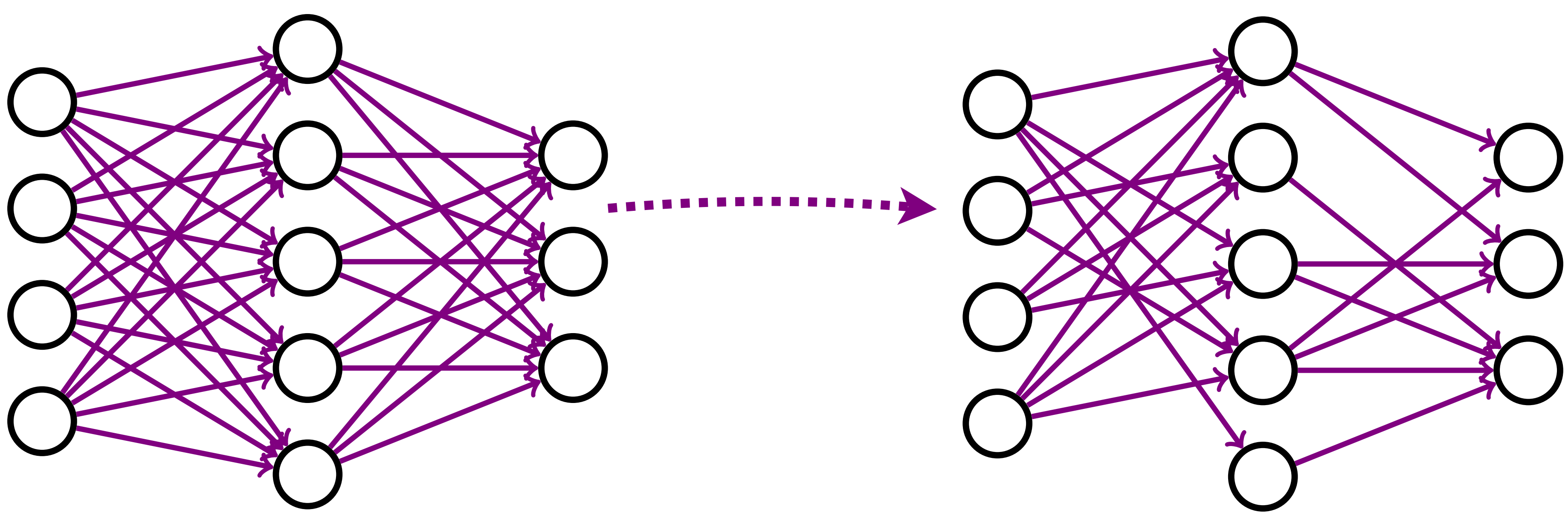
Reduce Precision Method		bitwidth		Accuracy loss vs. 32-bit float (%)
		Weights	Activations	
Dynamic Fixed Point	w/o fine-tuning [121]	8	10	0.4
	w/ fine-tuning [122]	8	8	0.6
Reduce Weight	BinaryConnect [127]	1	32 (float)	19.2
	Binary Weight Network (BWN) [129]	1*	32 (float)	0.8
	Ternary Weight Networks (TWN) [131]	2*	32 (float)	3.7
	Trained Ternary Quantization (TTQ) [132]	2*	32 (float)	0.6
Reduce Weight and Activation	XNOR-Net [129]	1*	1*	11
	Binarized Neural Networks (BNN) [128]	1	1	29.8
	DoReFa-Net [120]	1*	2*	7.63
	Quantized Neural Networks (QNN) [119]	1	2*	6.5
	HWGQ-Net [130]	1*	2*	5.2
Non-linear Quantization	LogNet [135]	5 (conv), 4 (fc)	4	3.2
	Incremental Network Quantization (INQ) [136]	5	32 (float)	-0.2
	Deep Compression [118]	8 (conv), 4 (fc)	16	0
		4 (conv), 2 (fc)	16	2.6

TABLE III
METHODS TO REDUCE NUMERICAL PRECISION FOR ALEXNET. ACCURACY MEASURED FOR TOP-5 ERROR ON IMAGENET. *NOT APPLIED TO FIRST AND/OR LAST LAYERS

https://github.com/MatthieuCourbariaux/BinaryNet
https://github.com/BertMoons/QuantizedNeuralNetworks-Keras-Tensorflow
https://github.com/DingKe/nn_playground/tree/master/ternarynet

Summary and Outlook

- Network compression (pruning and quantization) is an important aspect of efficiently computing ML algorithms
  - Especially important for LHC trigger applications on FPGAs
- Many different techniques / implementations
  - Implementations are currently scattered across random GitHub repositories
  - Should become a standard “tool” in our ML toolkit



Backup