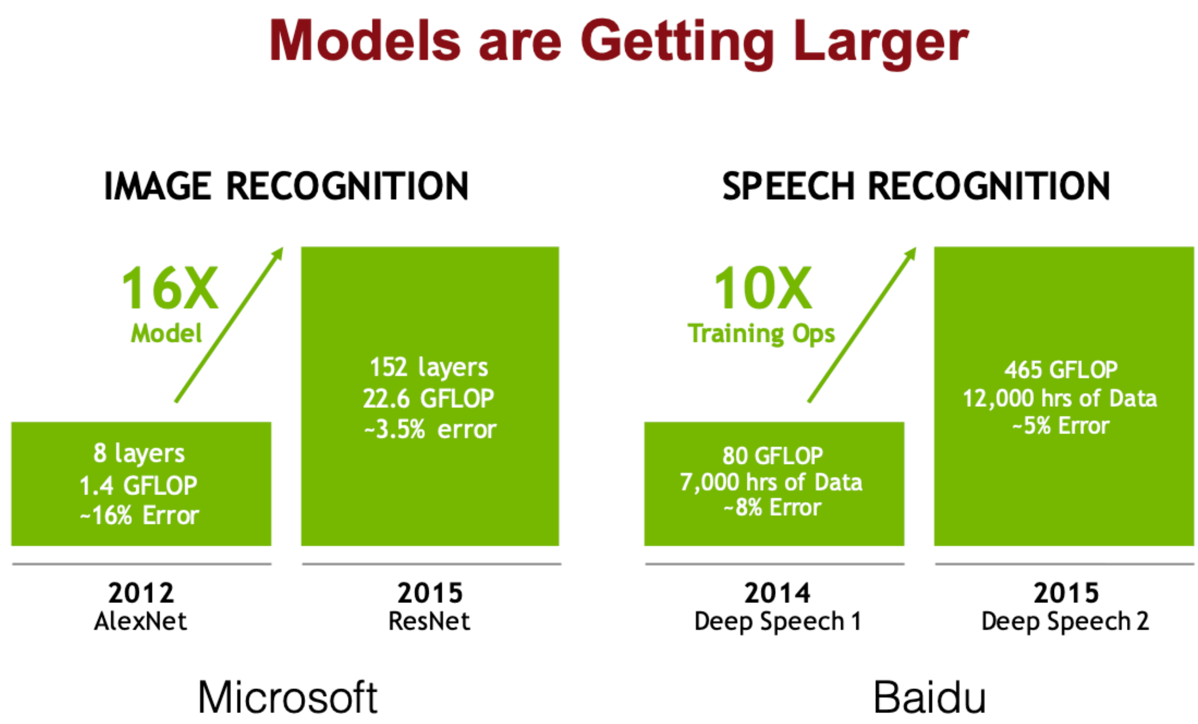
Lecture 9 - Compression (High-perf hardware)



Larger data sets and models lead to better accuracy but also increase computation time. Therefore progress in deep neural networks is limited by how fast the networks can be computed.

Likewise the application of convnets to low latency inference problems, such as pedestrian detection in self driving car video imagery, is limited by how fast a small set of images, possibly a single image, can be classified.

Acceleration

- Run a network faster (Performance, inf/s)
- Run a network more efficiently
  - Energy (inf/J)
  - Cost (inf/s$)

- Inference
  - Just running the network forward

- Training
  - Running the network forward
  - Back-propagation of gradient
  - Update of parameters

Key operations are Matrix Vector multiplications (dense data), sparse during inference.

Why GPUs? SIMD What about other approaches? We will cover 1) Reduced Precision Arithmetic (the goal is to increase the amount of data we can perform operations over), 2) Compression (reduce operations we have to perform), and 3) Better algorithms (Low-Rank approximation).

1. Reducing precision

Reducing precision

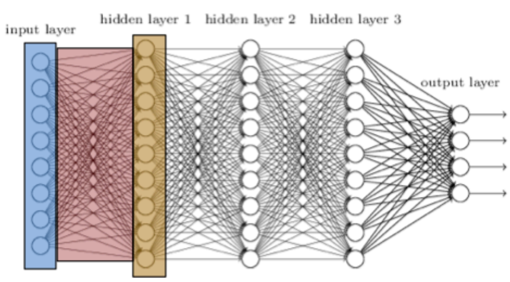
Reduces storage

Reduces energy

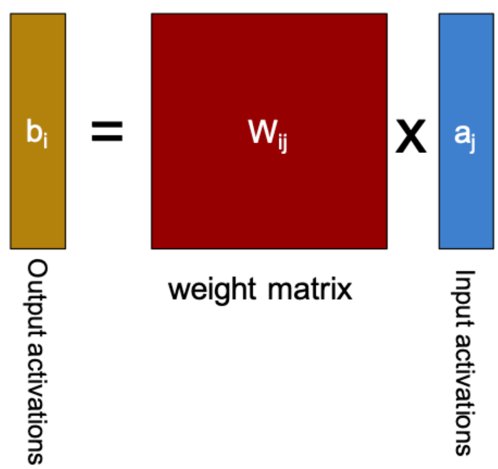
Improves performance

Has little effect on accuracy – to a point

DNN, key operation is dense M x V



$$b_i = f\left(\sum_j w_{ij}\, a_i\right)$$



How much accuracy do we need in the computations:

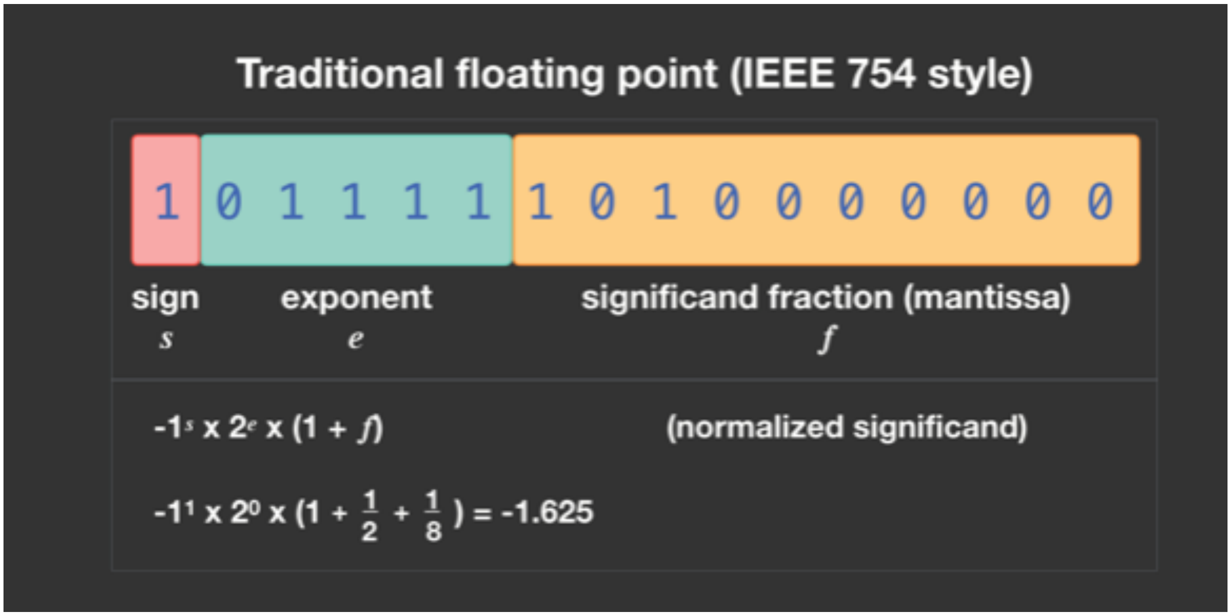
$$b_i = f\left(\sum_j w_{ij}\, a_i\right)$$

$$w_{ij} = w_{ij} + \alpha a_i g_j$$

Number Representation

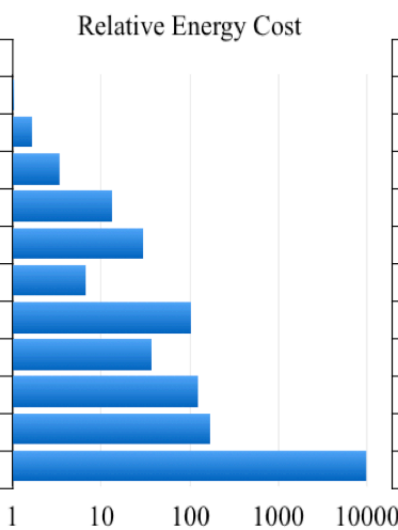
	S	E	M	Range	Accuracy
FP32	1	8	23	10^{-38} - 10^{38}	.000006%
FP16	1	5	10	6×10^{-5} - 6×10^{4}	.05%
Int32	1		31	0 – 2×10^{9}	½
Int16	1		15	0 – 6×10^{4}	½
Int8	1		7	0 – 127	½

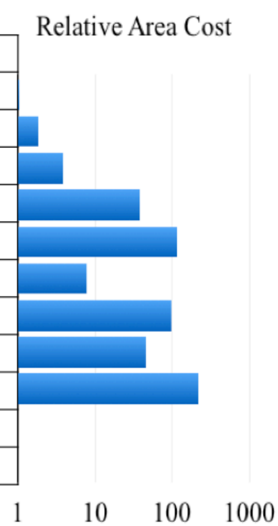
FP32 = single precision
FP16 = half-precision



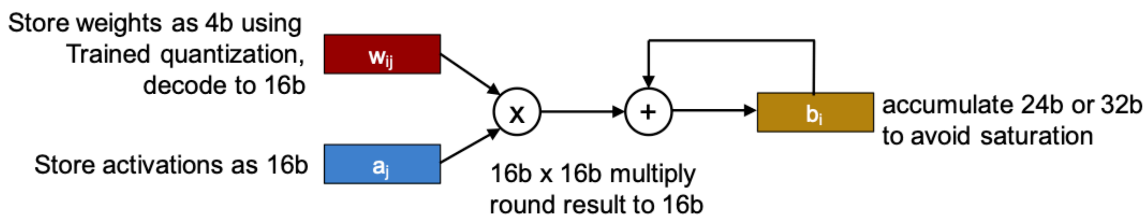
Cost of Operations

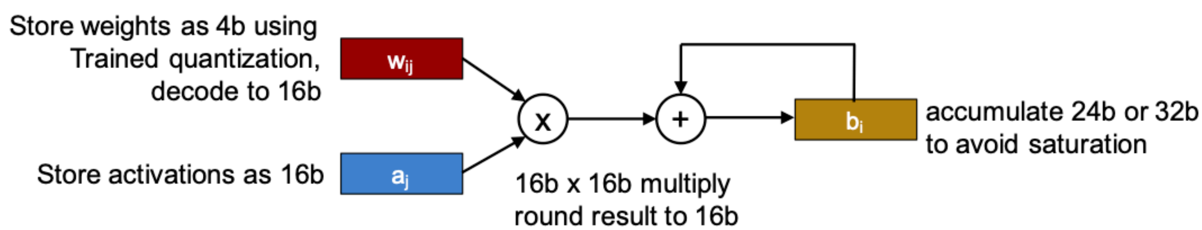
Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





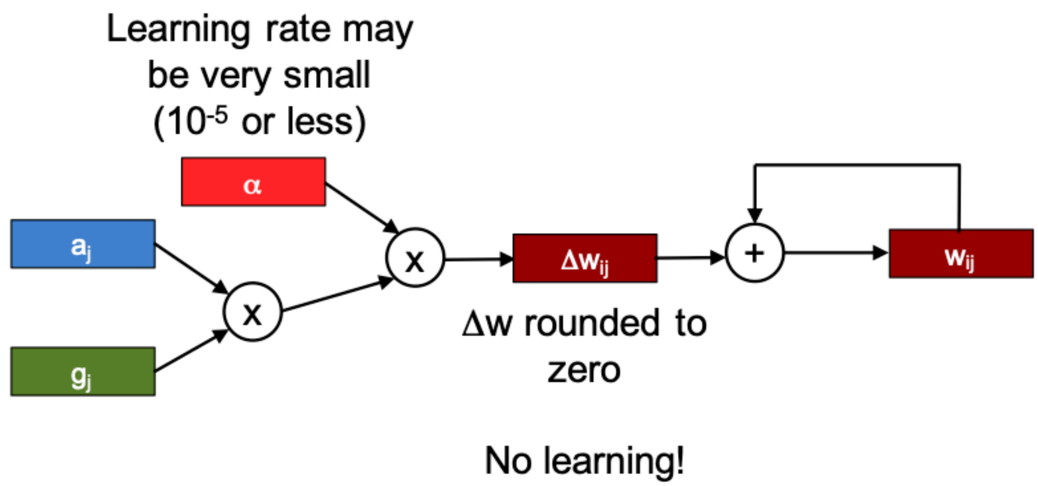
Mixed Precision



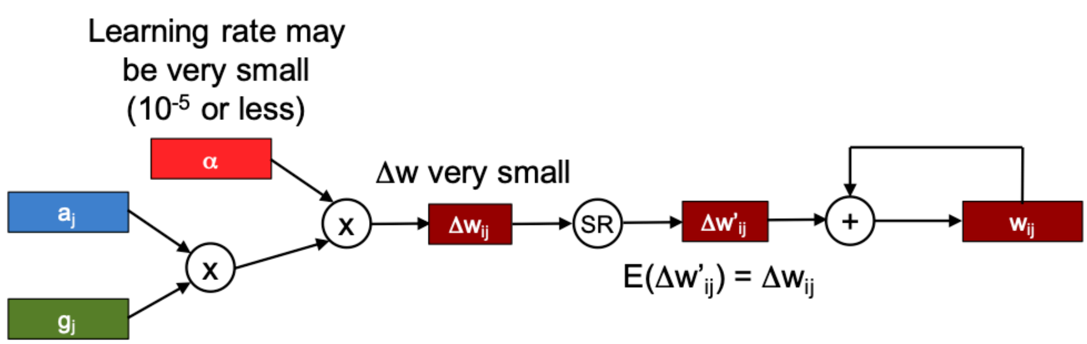


Batch normalization important to ‘center’ dynamic range

Weight Update



Stochastic Rounding



Stochastic rounding -> let's say we add 0.3 to 0 100 times if we round 0.3 we will get zero. If we round it 70% of the time to 0 and 30% to 1 then we get E[Sum]=30

$$\text{Round}(x) = \begin{cases} \lfloor x \rfloor & \text{with probability } 1 - (x - \lfloor x \rfloor) \\ \lfloor x \rfloor + 1 & \text{with probability } x - \lfloor x \rfloor \end{cases}$$

Examples

3.5 has a 50% chance to round to 3, and a 50% chance to round to 4

2.4 has a 60% chance to round to 2, and a 40% chance to round to 3

1.6 has a 40% chance to round to 1, and a 60% chance to round to 2

-2.1 has a 90% chance to round to -2, and a 10% chance to round to -3

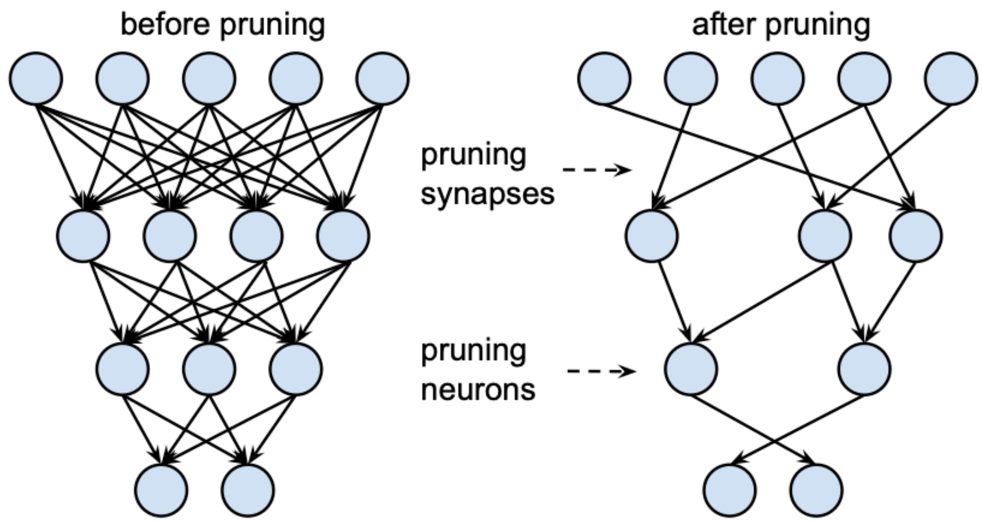
-4.7 has a 30% chance to round to -4, and a 70% chance to round to -5

Summary of Reduced Precision

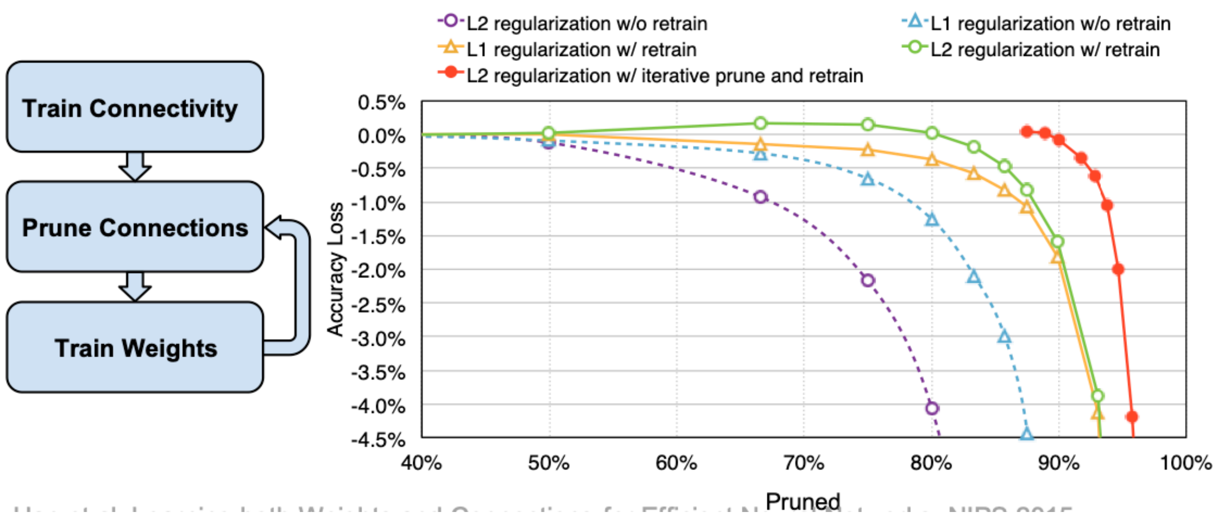
- Can save memory capacity, memory bandwidth, memory power, and arithmetic power by using smaller numbers
- FP16 works with little effort
  - 2x gain in memory, 4x in multiply power
- With care, one can use
  - 8b for convolutions
  - 4b for fully-connected layers
- Batch normalization – important to ‘center’ ranges
- Stochastic rounding – important to retain small increments

2. Pruning

Pruning

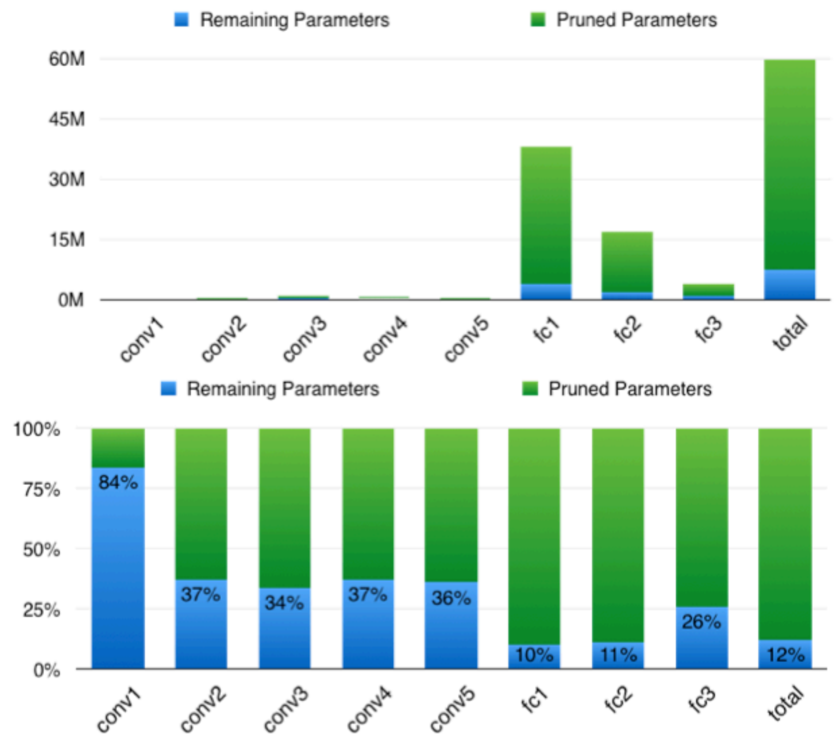


Retrain to Recover Accuracy

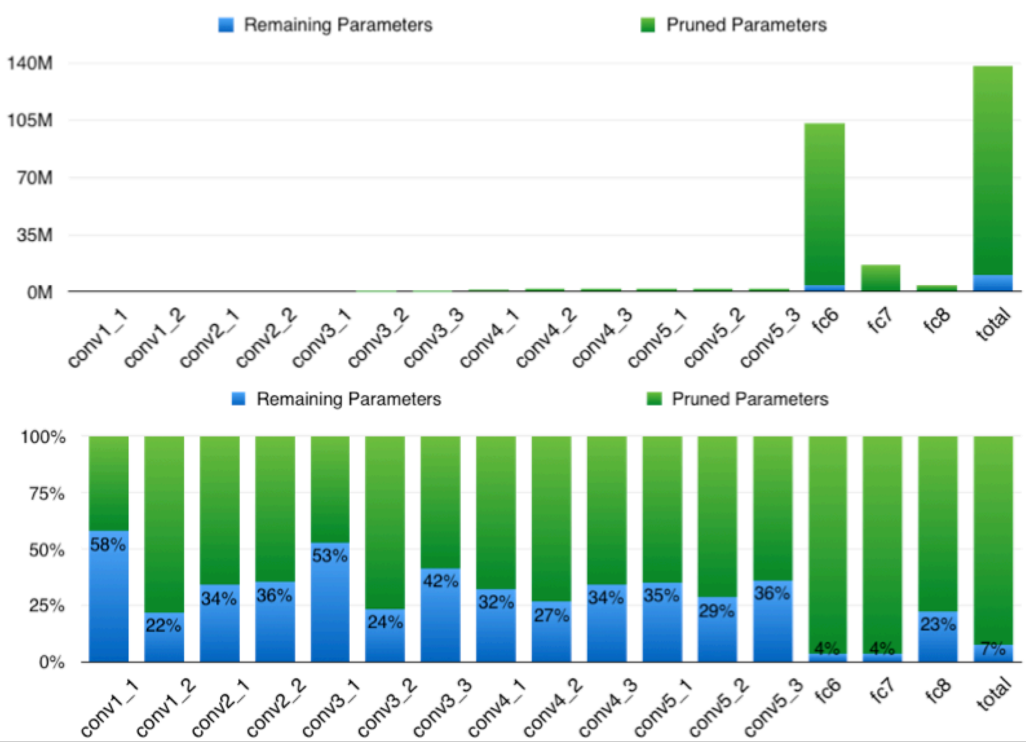


Han et al. Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015

Pruning of AlexNet



Pruning of VGG-16



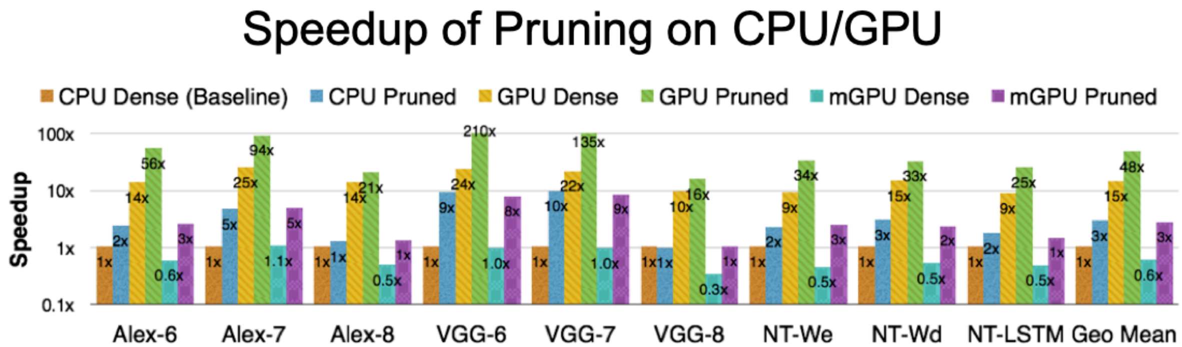
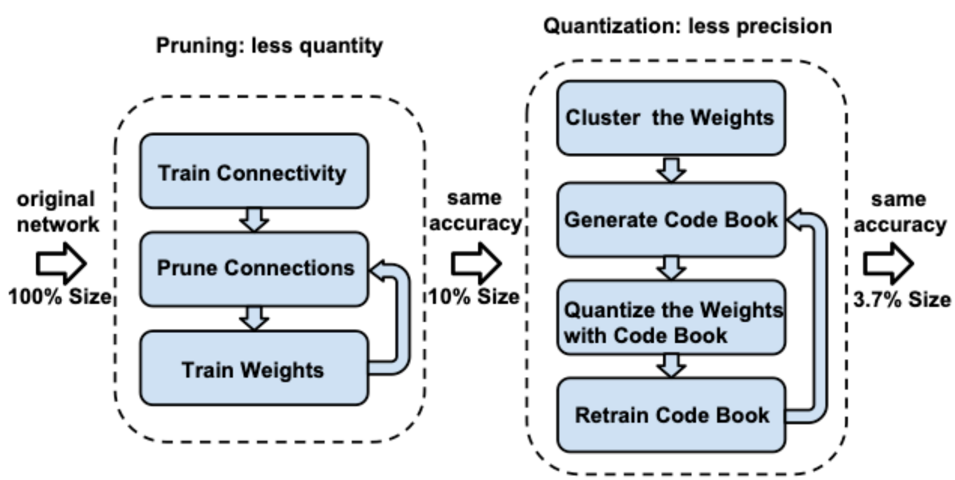


Figure 9: Compared with the original network, pruned network layer achieved 3× speedup on CPU, 3.5× on GPU and 4.2× on mobile GPU on average. Batch size = 1 targeting real time processing. Performance number normalized to CPU.

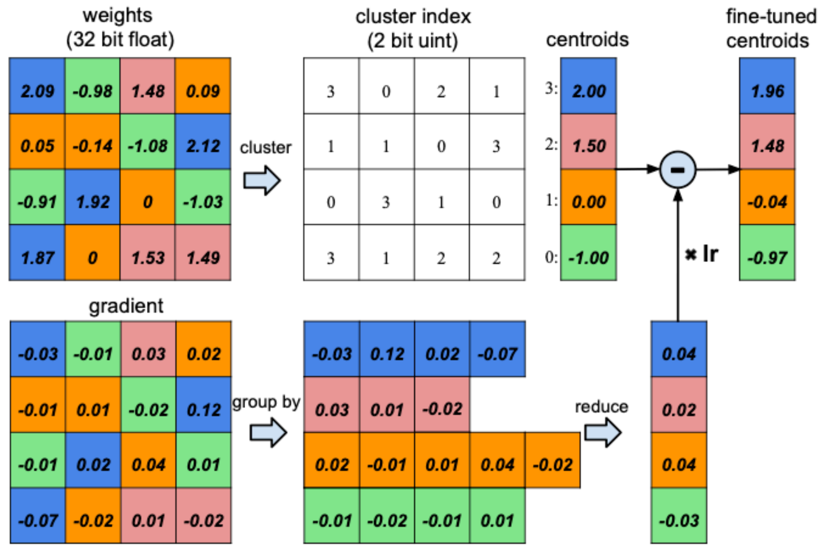
Intel Core i7 5930K: MKL CBLAS GEMV, MKL SPBLAS CSRMV
NVIDIA GeForce GTX Titan X: cuBLAS GEMV, cuSPARSE CSRMV
NVIDIA Tegra K1: cuBLAS GEMV, cuSPARSE CSRMV

3. Reduce weight storage for remaining weights

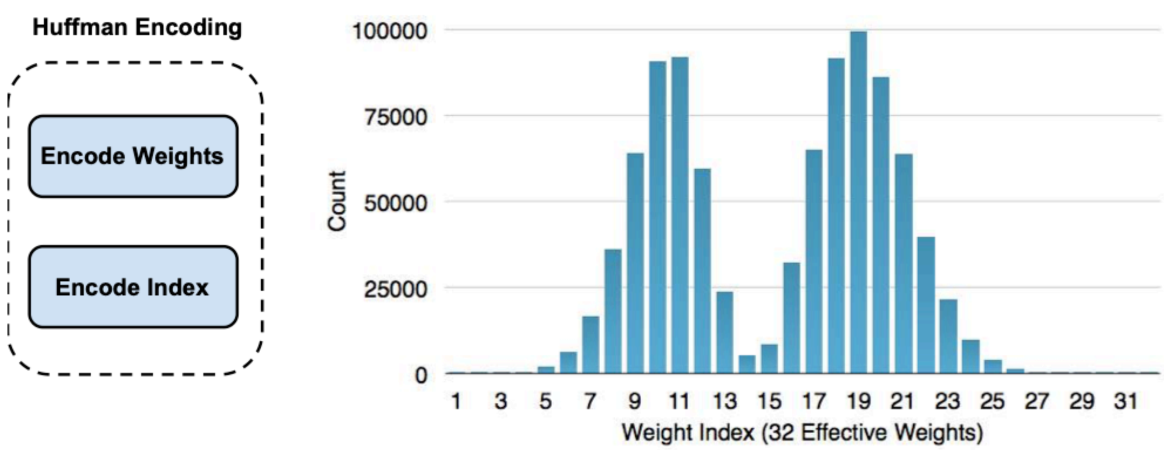
Trained Quantization
(Weight Sharing)



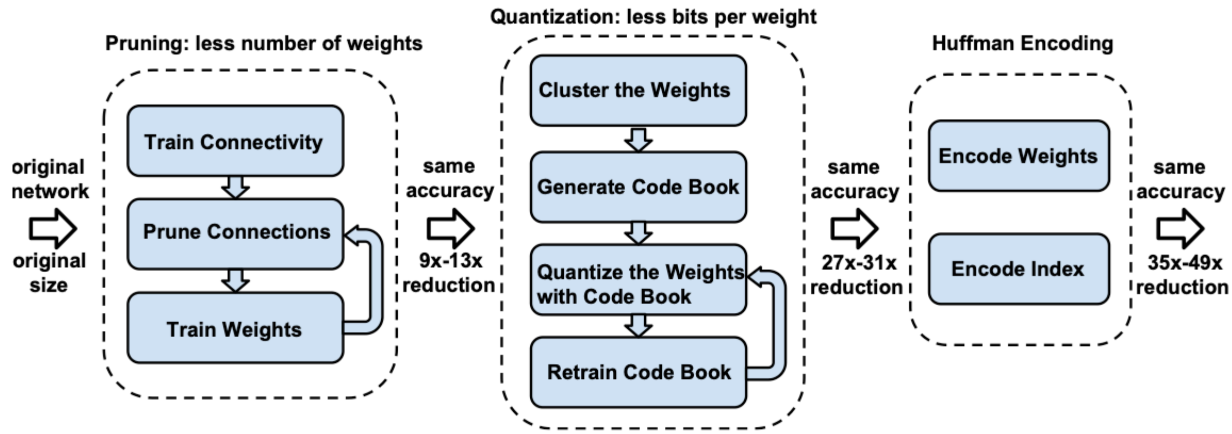
Weight Sharing via K-Means



Huffman Coding

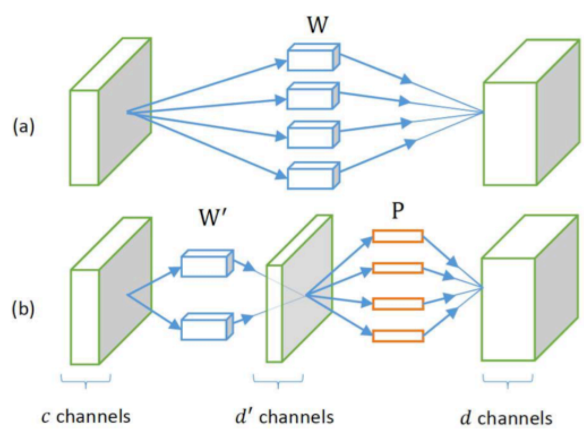


- In-frequent weights: use more bits to represent
- Frequent weights: use less bits to represent



5. Low Rank Approximations

- Layer responses lie in a low-rank subspace
- Decompose a convolutional layer with d filters with filter size $k \times k \times c$ to
  - A layer with d' filters ($k \times k \times c$)
  - A layer with d filter ($1 \times 1 \times d'$)



Low Rank Approximation for FC

Build a mapping from row / column indices of matrix $\boldsymbol{W} = [W(x, y)]$ to vectors $\boldsymbol{i}$ and $\boldsymbol{j}$: $x \leftrightarrow \boldsymbol{i} = (i_1, \dots, i_d)$ and $y \leftrightarrow \boldsymbol{j} = (j_1, \dots, j_d)$.

TT-format for matrix $\boldsymbol{W}$:

$$\boldsymbol{W}(i_1, \dots, i_d; j_1, \dots, j_d) = \boldsymbol{W}(x(\boldsymbol{i}), y(\boldsymbol{j})) = \underbrace{\boldsymbol{G}_1[i_1, j_1]}_{1 \times r} \underbrace{\boldsymbol{G}_2[i_2, j_2]}_{r \times r} \dots \underbrace{\boldsymbol{G}_d[i_d, j_d]}_{r \times 1}$$

Type	1 im. time (ms)	100 im. time (ms)
CPU fully-connected layer	16.1	97.2
CPU TT-layer	1.2	94.7
GPU fully-connected layer	2.7	33
GPU TT-layer	1.9	12.9