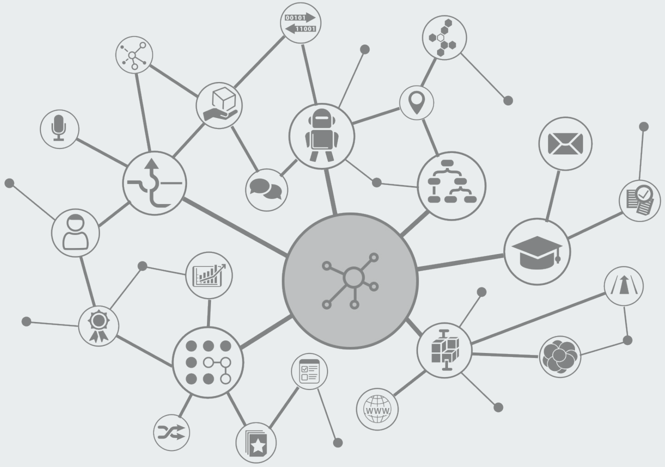
Deep Learning

Time and Memory Efficiency



Part 1 : JAEYOUNG CHUN
Part 2 : ADIR MORGAN

APRIL 29, 2018
WEIZMANN INSTITUTE OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

Agenda

PART 1

1. Motivation
2. Classical Algorithms / Build Blocks
3. Hardware

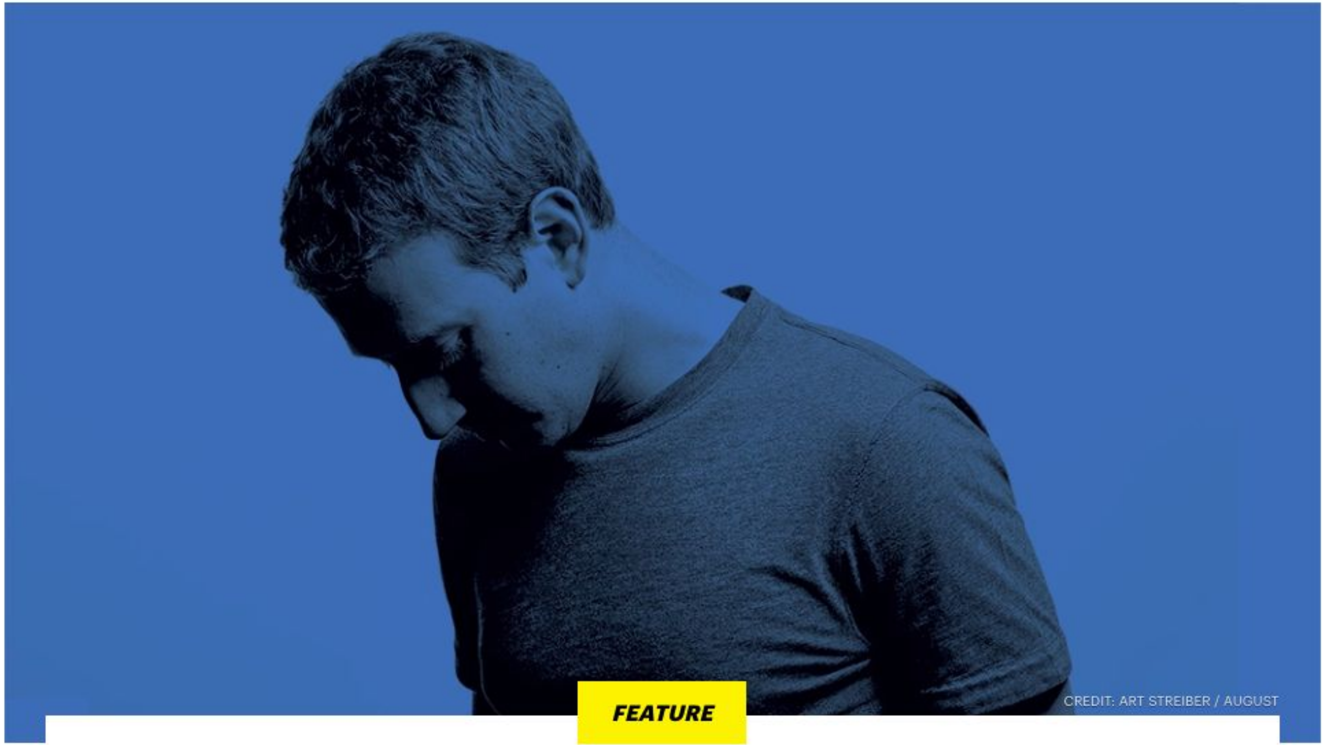
PART 2

4. Recent Works

Motivation

Why Seeking Efficiency?

VARIETY FILM TV MUSIC TECH GAMING THEATER REAL ESTATE AWARDS VIDEO V500



CREDIT: ART STREIBER / AUGUST

FEATURE

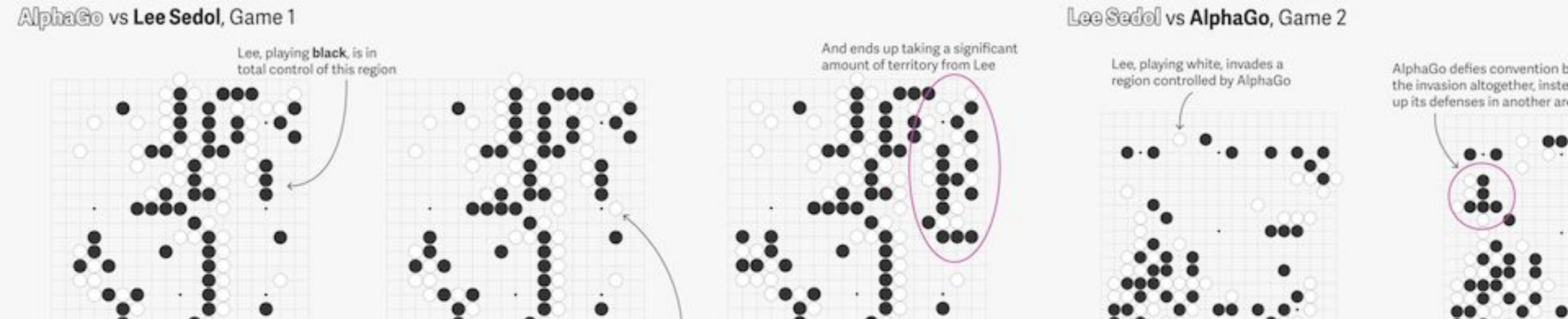
Facebook Under Fire: How Privacy Crisis Could Change Big Data Forever

AlphaGo

1,920 CPUs

280 GPUs

$3,000 Electricity Bill per game

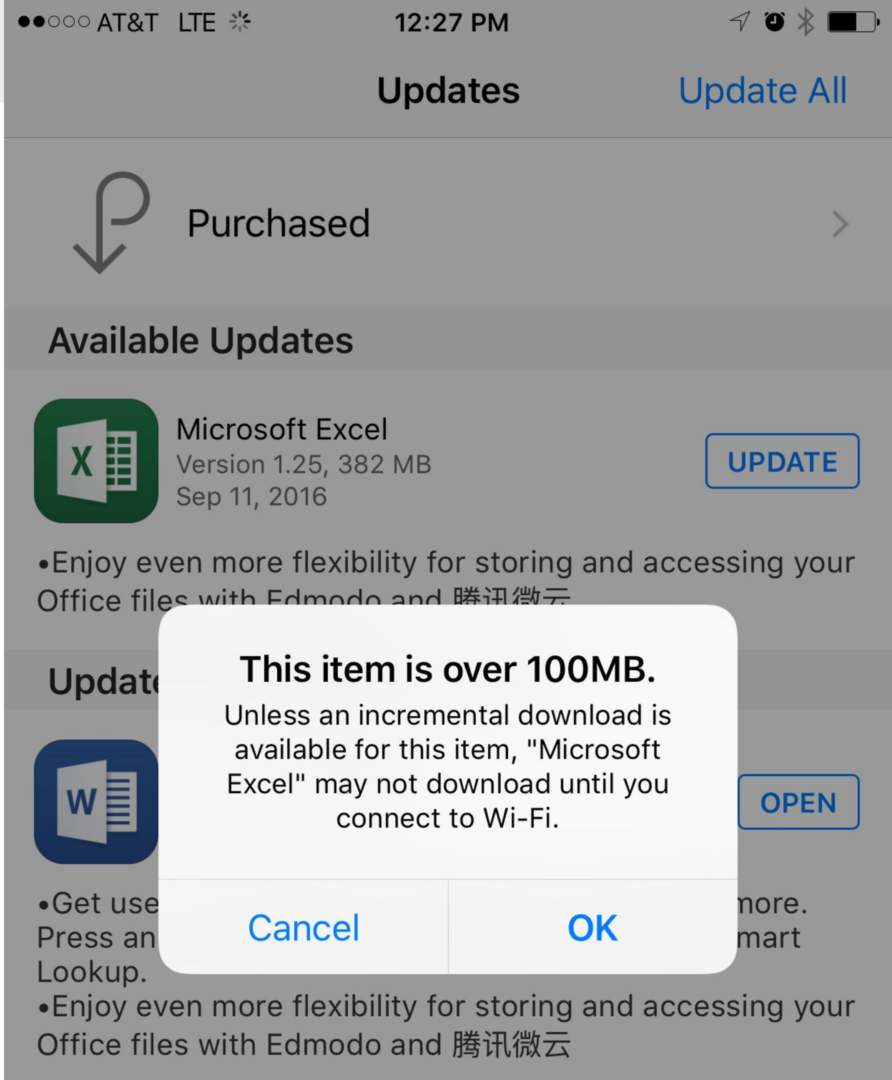


AppStore Download Restriction

To download an app over 100MB onto your mobile device,

you must connect to WiFi.

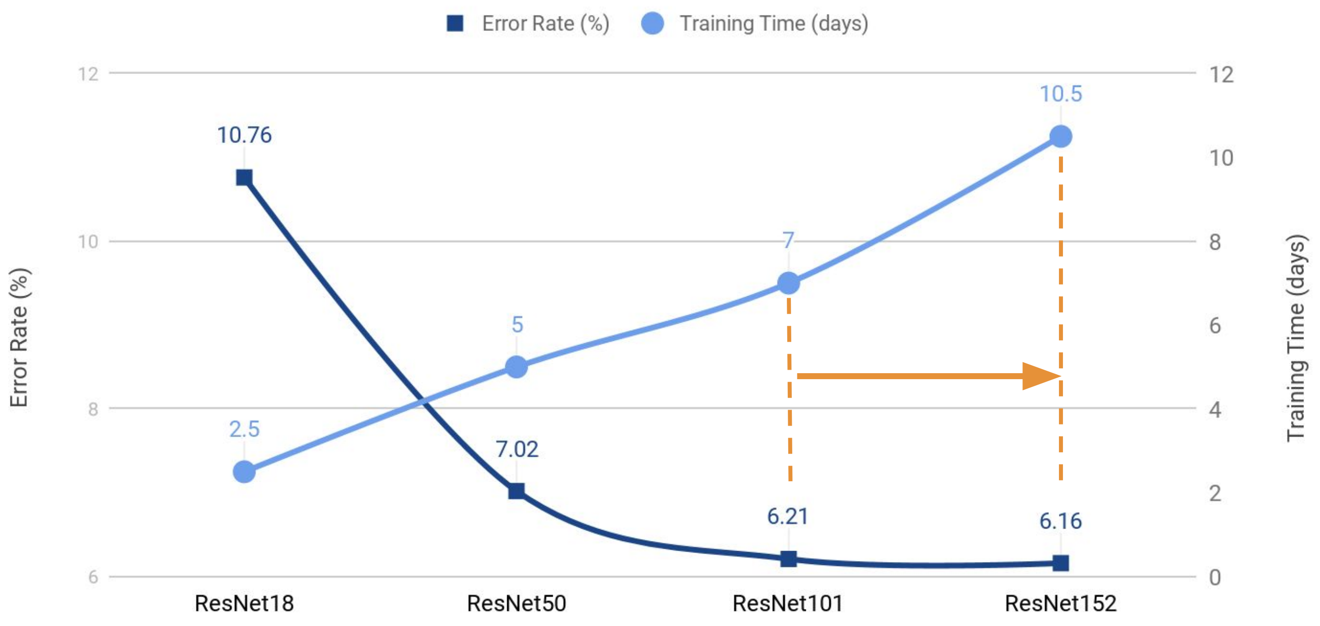
Putting this in perspective,
VGG-16 has 130 million parameters (520MB).



Model Size

2012 AlexNet		2015 ResNet
8 LAYERS	16x →	152 LAYERS
1.4 GFLOPS		22.6 GFLOPS
~16% Top-5 Error Rate		~3.5% Top-5 Error Rate

Speed of Training

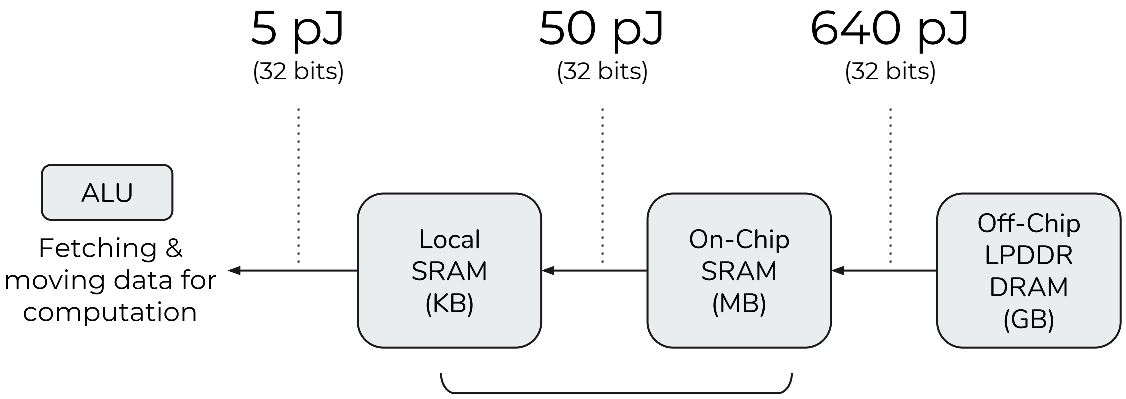


.05↓ ERR %

3.5↑ DAYS

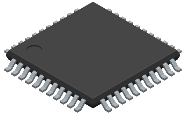
Energy Consumption

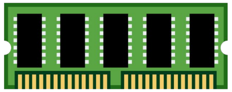
Cost of data movement is much more huge.
When compared, arithmetic ops is more like a noise.



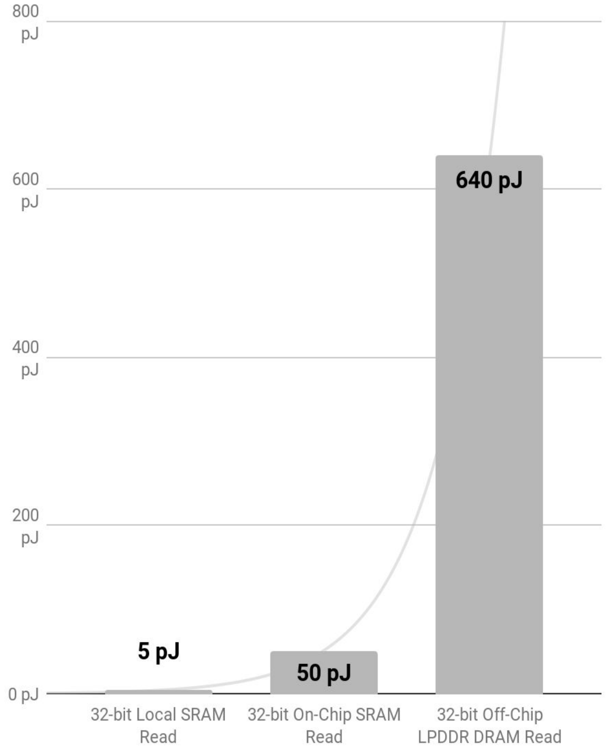
3.7 pJ
(32-bit FP Multiplication)

c.f.) 1.1 pJ for 16-bit FP Mult
0.2 pJ for 8-bit Mult

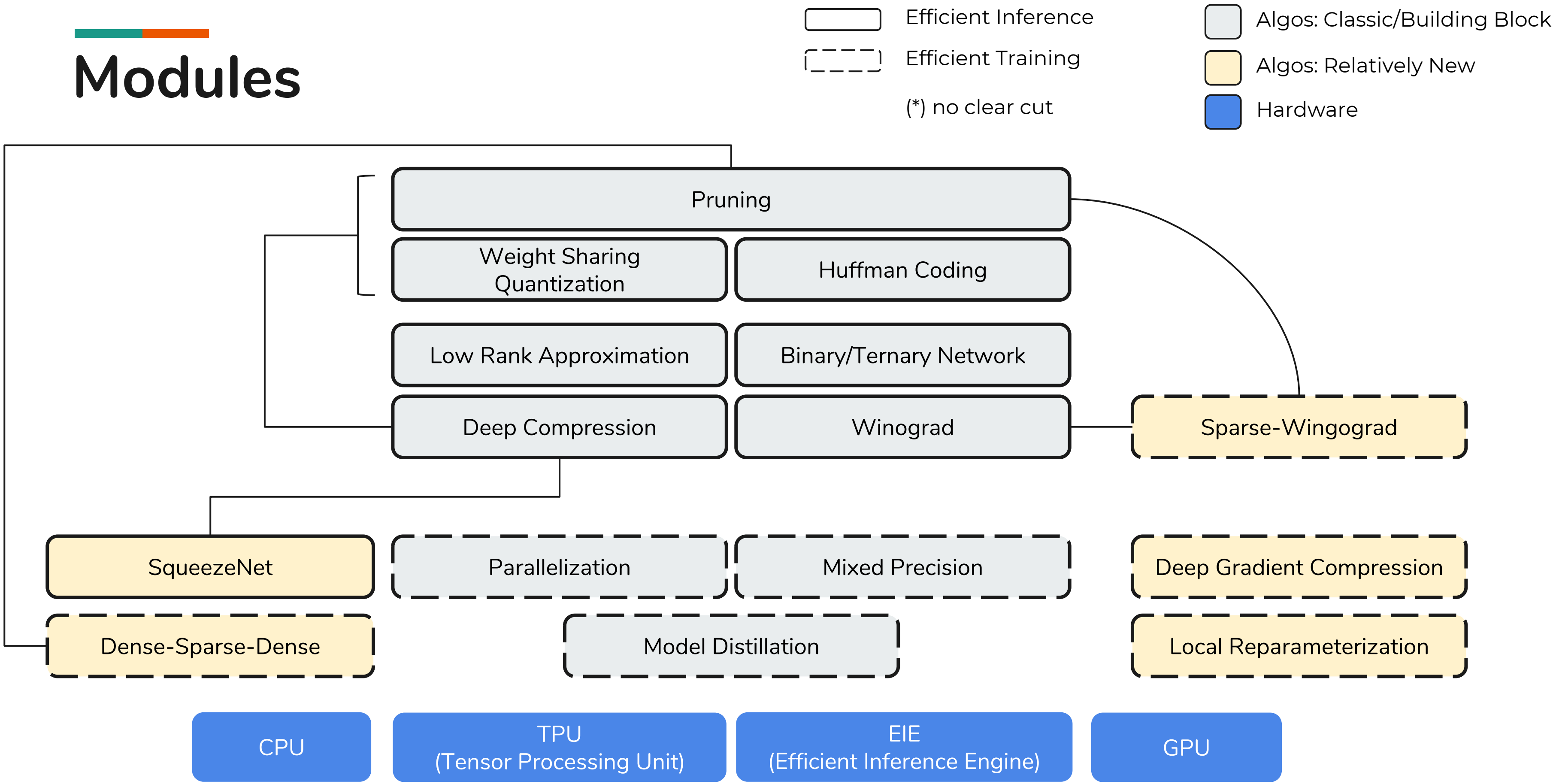




LPDDR



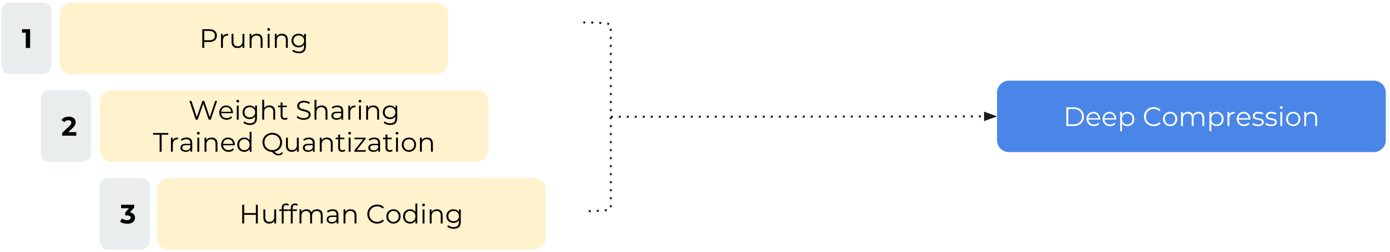
Modules



Jaeyoung Chun

Something to Keep in Mind...

- Losing any accuracy?
- Multiple methods interfering each other?



Understanding the underlying concepts & getting insights, and intuitions $>$ Implementation and mathematical details

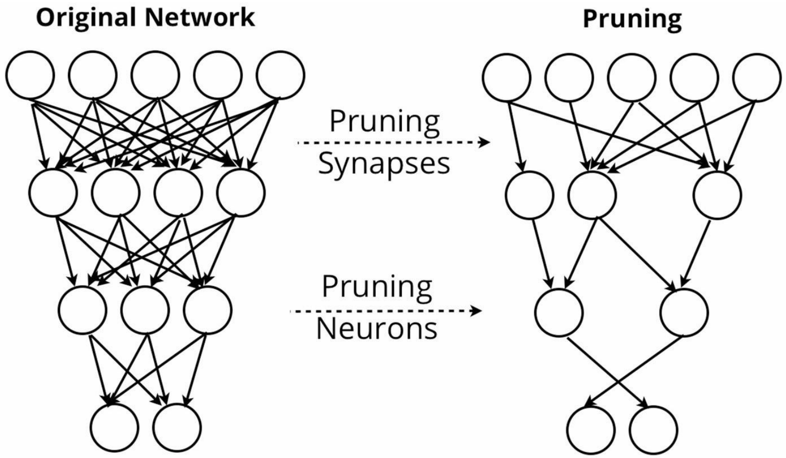
Algorithms for Efficient Inference

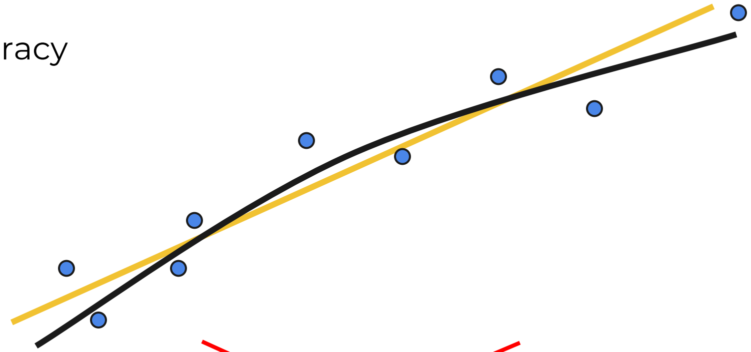
Jaeyoung Chun

Lecun et al. NIPS’89, Han et al. NIPS’15

Pruning

Less number of parameters with almost no loss of accuracy





$$\cancel{-0.01x^2} + x + 1$$

not just to reduce the network complexity,
but also to avoid overfitting

Pruning

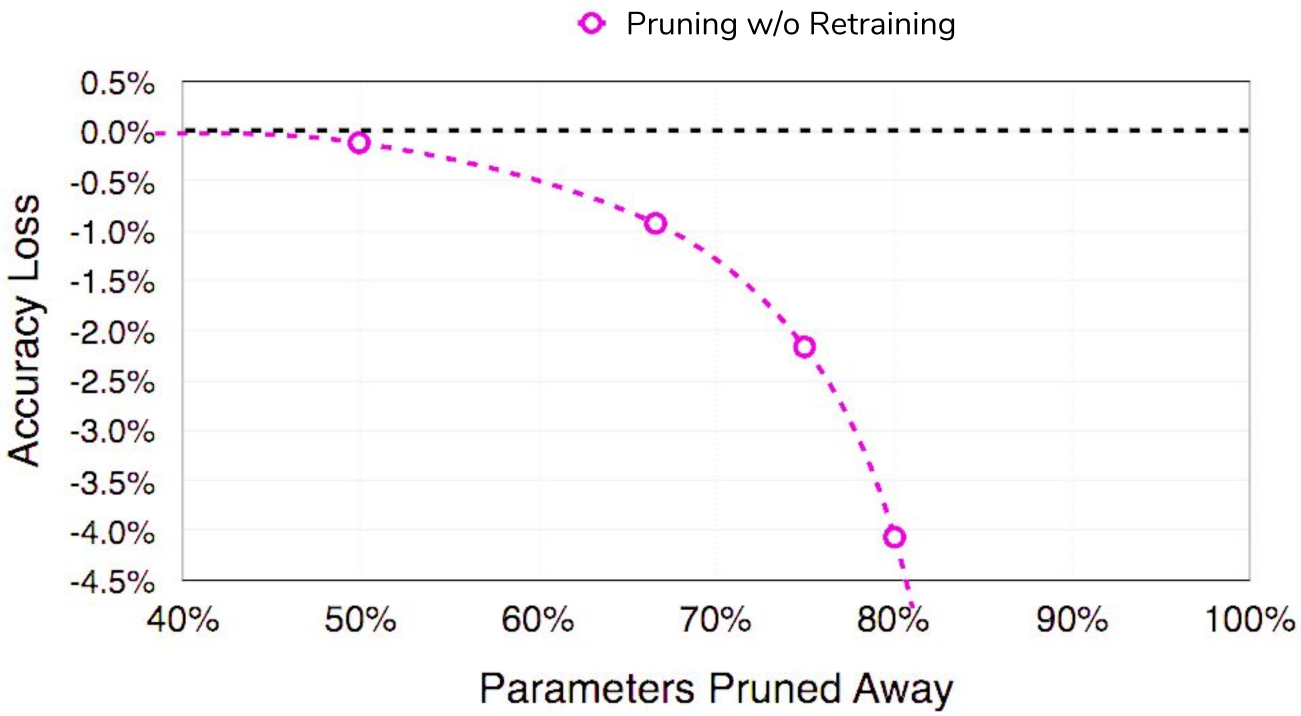
Train Connectivity → **Prune Connections** → **Retrain Weights**

1. Learn the connectivity via normal network training, as you would normally do.

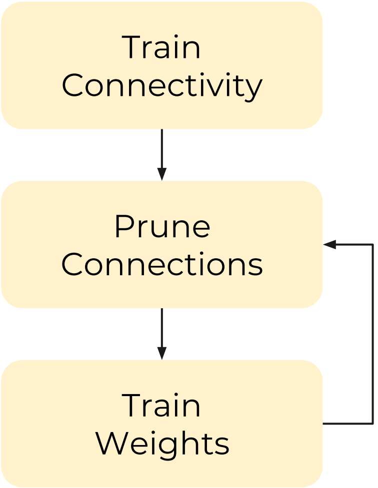
2. Prune the small-weight connections from the network.
(below a certain threshold)

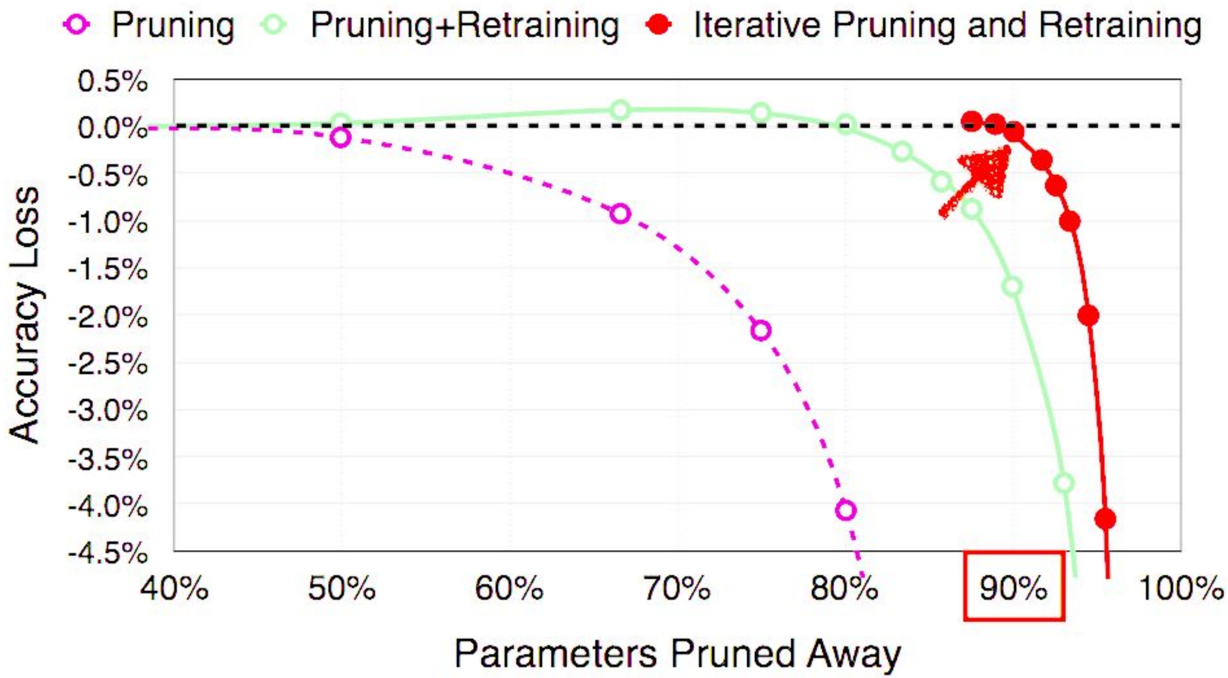
3. Retrain the network to learn the final weights for the remaining sparse connections.

Pruning

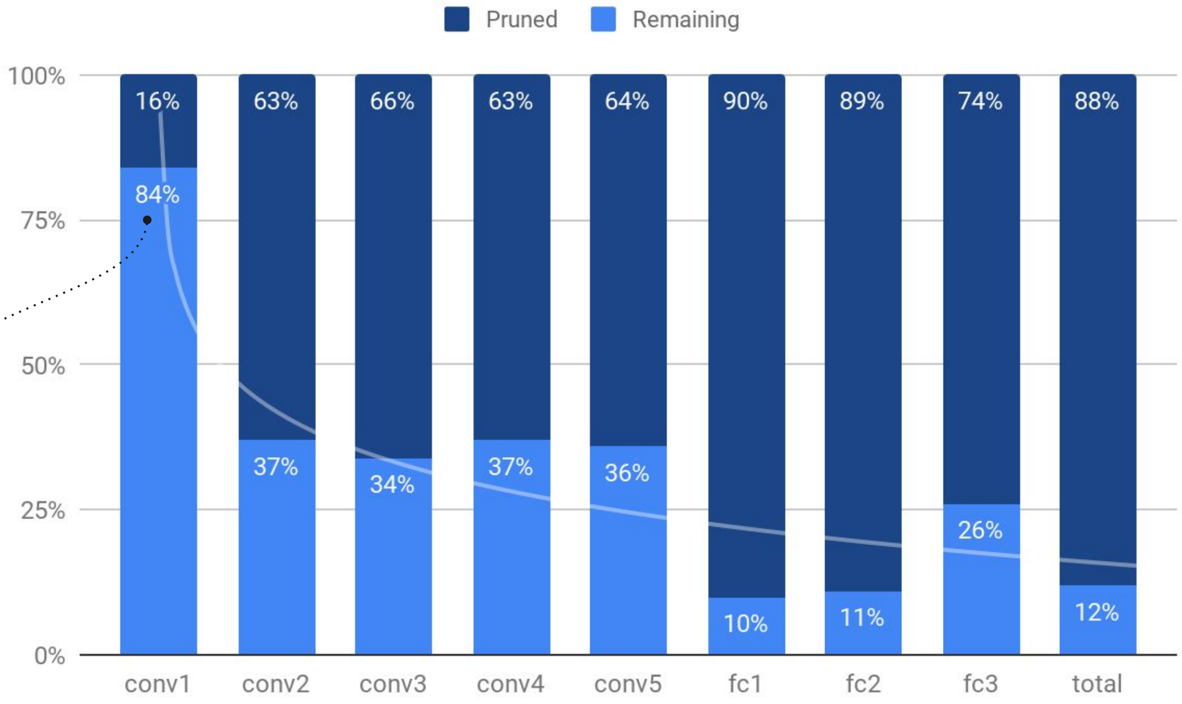

Pruning w/o Retraining
0.5%
0.0%
-0.5%
-1.0%
-1.5%
-2.0%
-2.5%
-3.0%
-3.5%
-4.0%
-4.5%
Accuracy Loss
40%
50%
60%
70%
80%
90%
100%
Parameters Pruned Away

Pruning


Train Connectivity
Prune Connections
Train Weights


Pruning
Pruning+Retraining
Iterative Pruning and Retraining
0.5%
0.0%
-0.5%
-1.0%
-1.5%
-2.0%
-2.5%
-3.0%
-3.5%
-4.0%
-4.5%
Accuracy Loss
40%
50%
60%
70%
80%
90%
100%
Parameters Pruned Away

Pruning: AlexNet



Pretty much you have to look at all the pixels of the image

Pruning + Image Captioning

ORIGINAL

a basketball player in a white uniform is a playing with a ball.

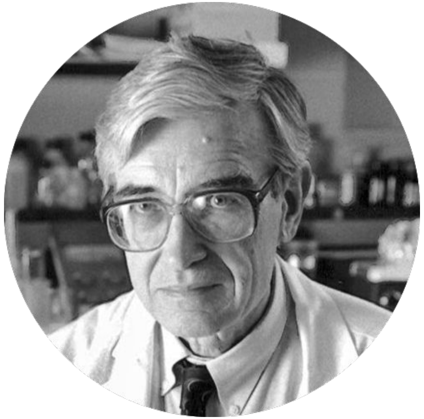
PRUNED 90%

a basketball player in a white uniform is a playing with a basketball.

If our brain loses 90% of neurons, can we still describe this image with this high accuracy?

Christopher A Walsh. Peter Huttenlocher (1931-2013). Nature, 502(7470):172–172, 2013.

Pruning in Human?



Peter Huttenlocher
(1931–2013)

500 trillion synapses — Adolescent

1,000 trillion synapses — 1 Year Old

50 trillion synapses — New Born

Adult

Weight Sharing

Before | After

Weight	Edge (Connection)	Weight	Edge (Connection)
2.09		**2.00**	
2.12		**2.00**	
1.92		**2.00**	
1.87		**2.00**	

Less number of bits per parameter
Too accurate leads to overfitting anyway

Effective Weight
or Shared Weight

Weight Sharing

Weights are not shared across layers. The shared weights approximate the original network because the method determines weight sharing after a network is fully trained.

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

4x4=16 numbers

K-Means Clustering (with K=4)

Color (Cluster)	Original Weights	Effective Weight (Centroid)
Purple	[2.09, 2.12, 1.92, 1.87]	2.00
Green	[1.48, 1.53, 1.49]	1.50
Orange	[0.09, 0.05, -0.14, 0, 0]	0.00
Yellow	[-0.98, -1.08, -0.91, -1.03]	-1.00

Only 4 numbers

Weight Sharing

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Color	Effective Weight
Purple	2.00
Green	1.50
Orange	0.00
Yellow	-1.00

2.00	-1.00	1.50	0.00
0.00	0.00	-1.00	2.00
-1.00	2.00	0.00	-1.00
2.00	0.00	1.50	1.50

Also called “Codebook”

Weight Sharing / Quantization

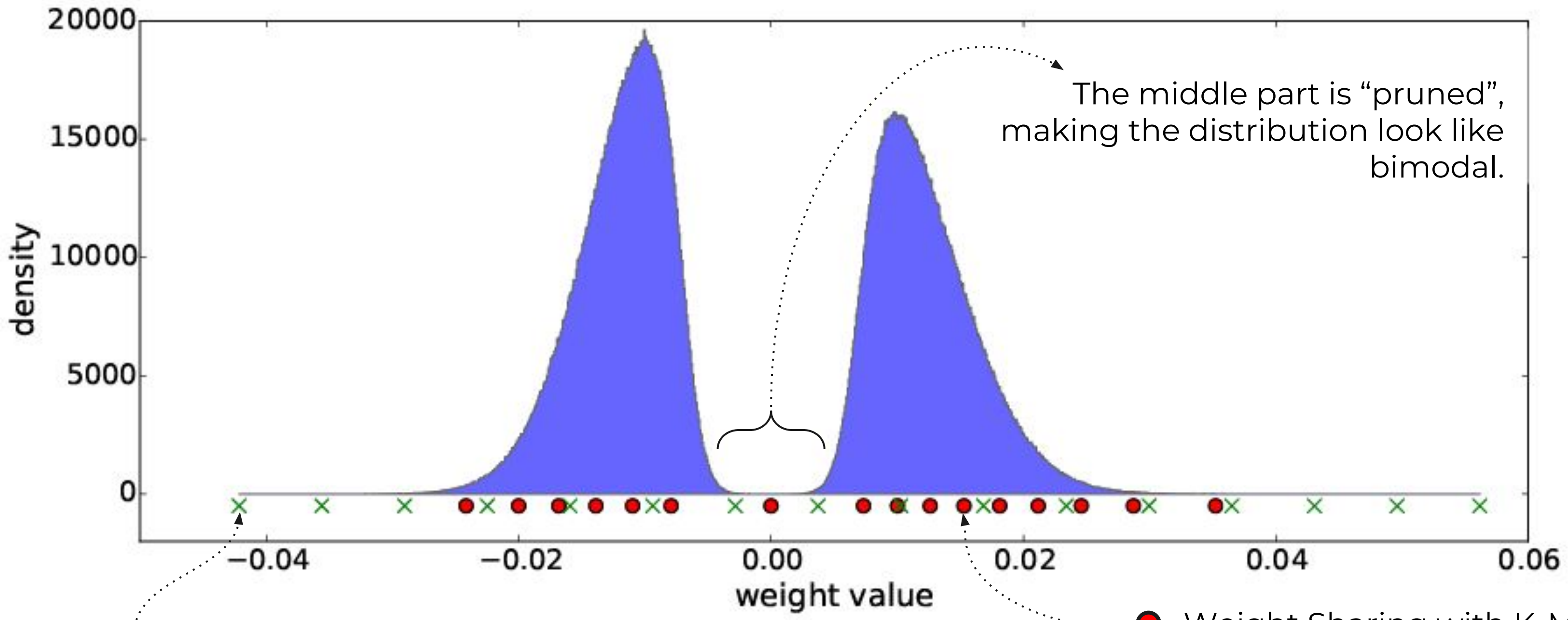
Color	Effective Weight	Index (Int)	Index (Binary)
Purple	2.00	0	00
Green	1.50	1	01
Orange	0.00	2	10
Yellow	-1.00	3	11

Lookup

00	11	01	10
10	10	11	00
11	00	10	11
00	10	01	01

No more 32-bit FP and only 2 bits

Trained Quantization: Weight Distribution

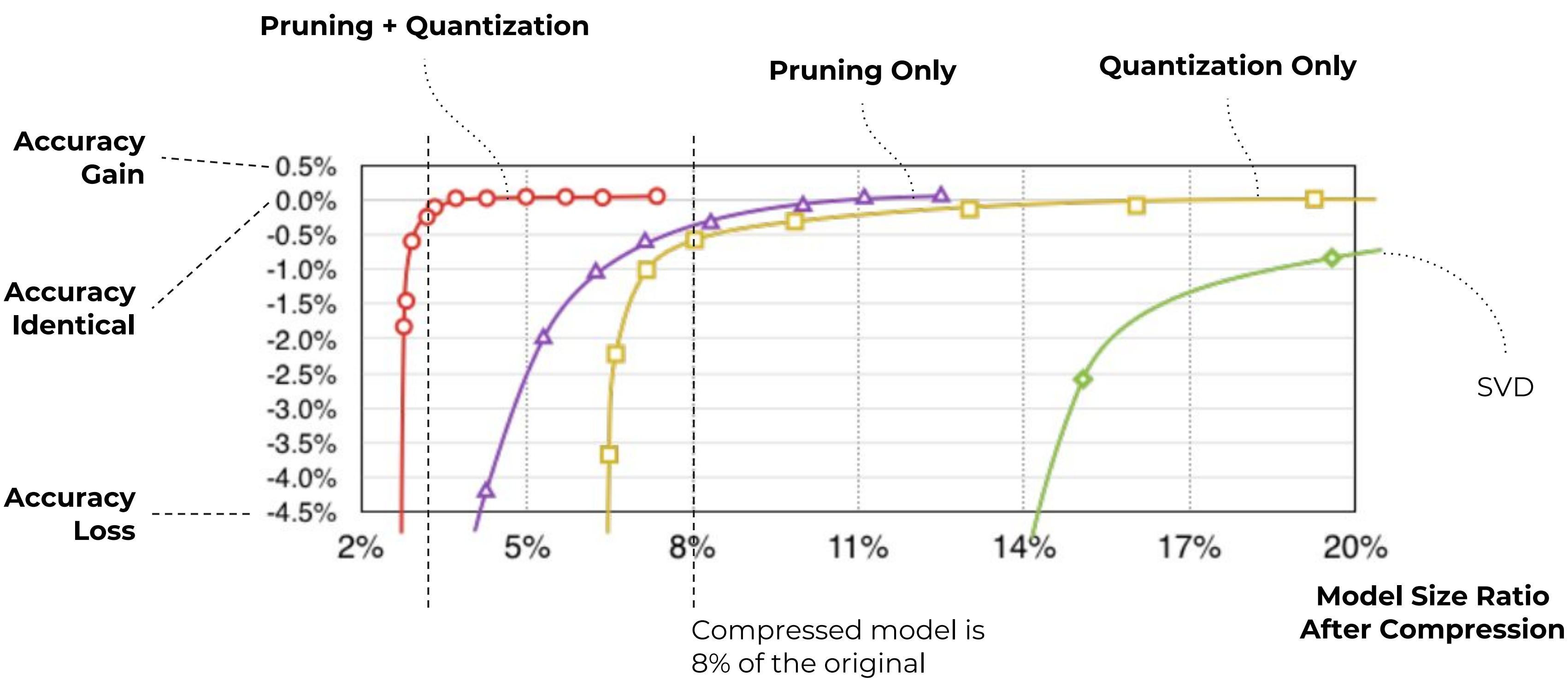


The middle part is "pruned", making the distribution look like bimodal.

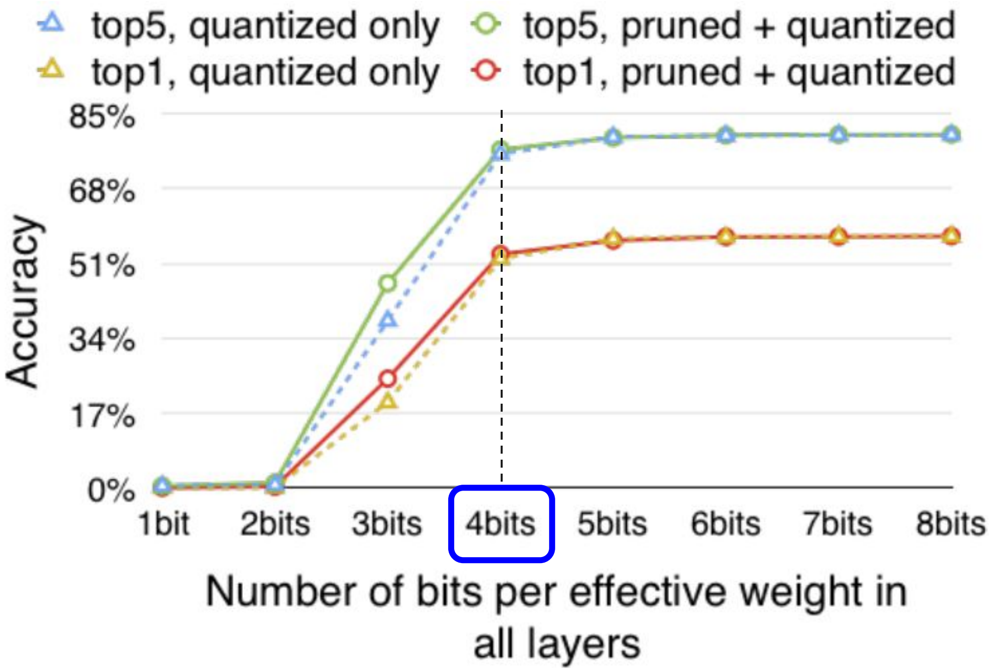
x Naive quantization with 4-bit integer
Sampling the space uniformly which doesn't capture the distribution well.

● Weight Sharing with K-Means better represents the distribution (esp. bimodal part)

Pruning and/or Quantization: Accuracy



Pruning and/or Quantization: Accuracy



Huffman Coding

00	11	00	01	10	00
00	00	00	00	00	00
10	10	00	11	00	00
11	00	00	10	11	00
00	10	00	01	01	00
00	00	00	00	00	00

6x6=36 numbers

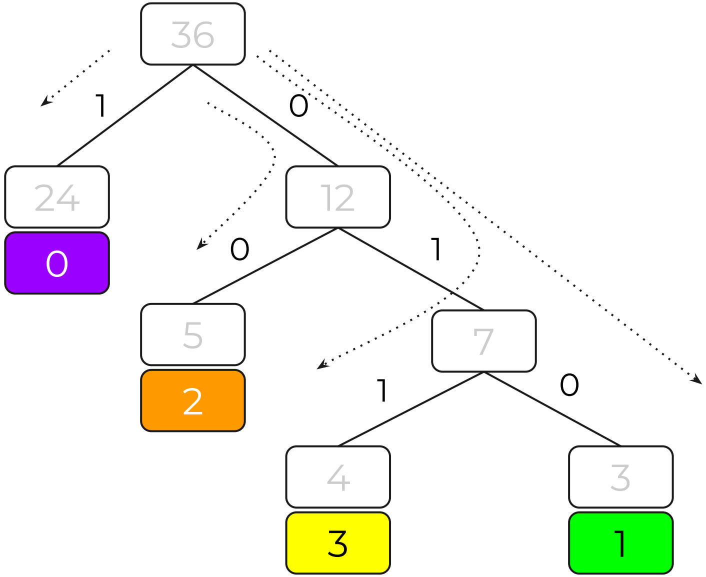
2 bits * (24 + 5 + 4 + 3) = 72 bits

Color	Effective Weight	Index (Int)	Index (Binary)	Count	%
Purple	2.00	0	00	24	66.67
Orange	0.00	2	10	5	13.89
Yellow	-1.00	3	11	4	11.11
Green	1.50	1	01	3	8.33

Frequent weights → use **less** bits to represent
Infrequent weights → use **more** bits to represent

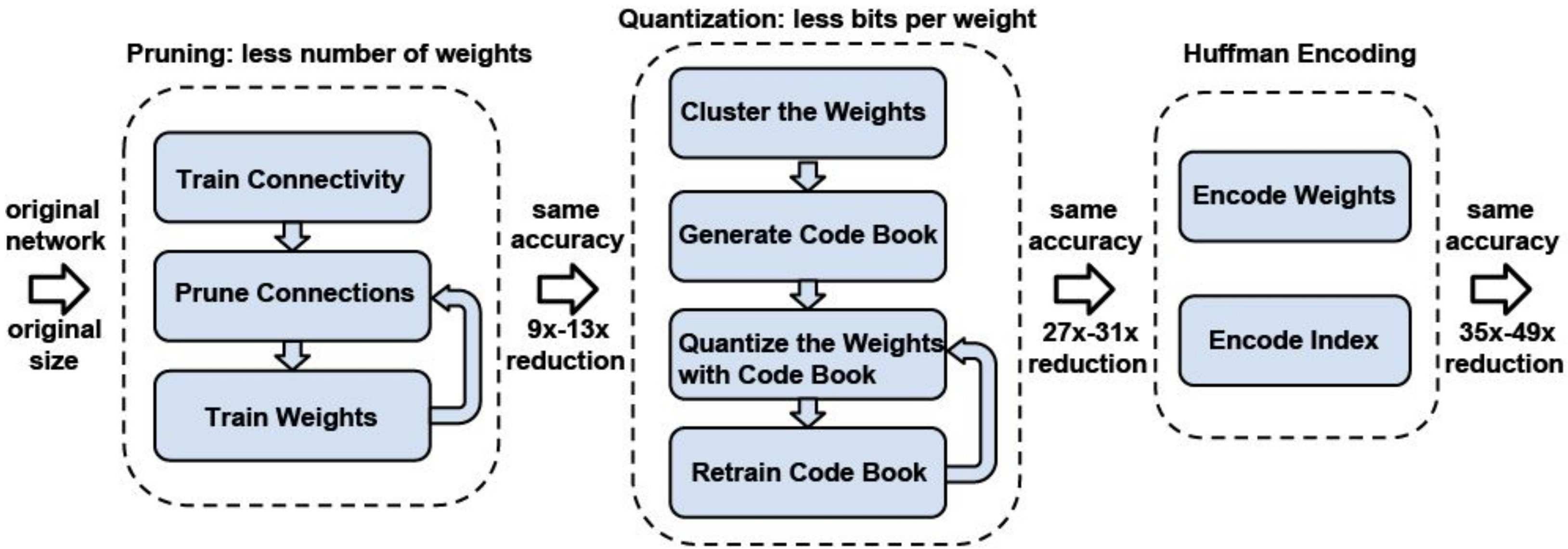
Huffman Coding

Color	Effective Weight	Index (Int)	Huffman Code	Count	%
Purple	2.00	0	1	24	66.67
Orange	0.00	2	00	5	13.89
Yellow	-1.00	3	011	4	11.11
Green	1.50	1	010	3	8.33



(1 bit * 24) + (2 bits * 5) + (3 bits * 4) + (3 bits * 3) = 55 bits

Deep Compression

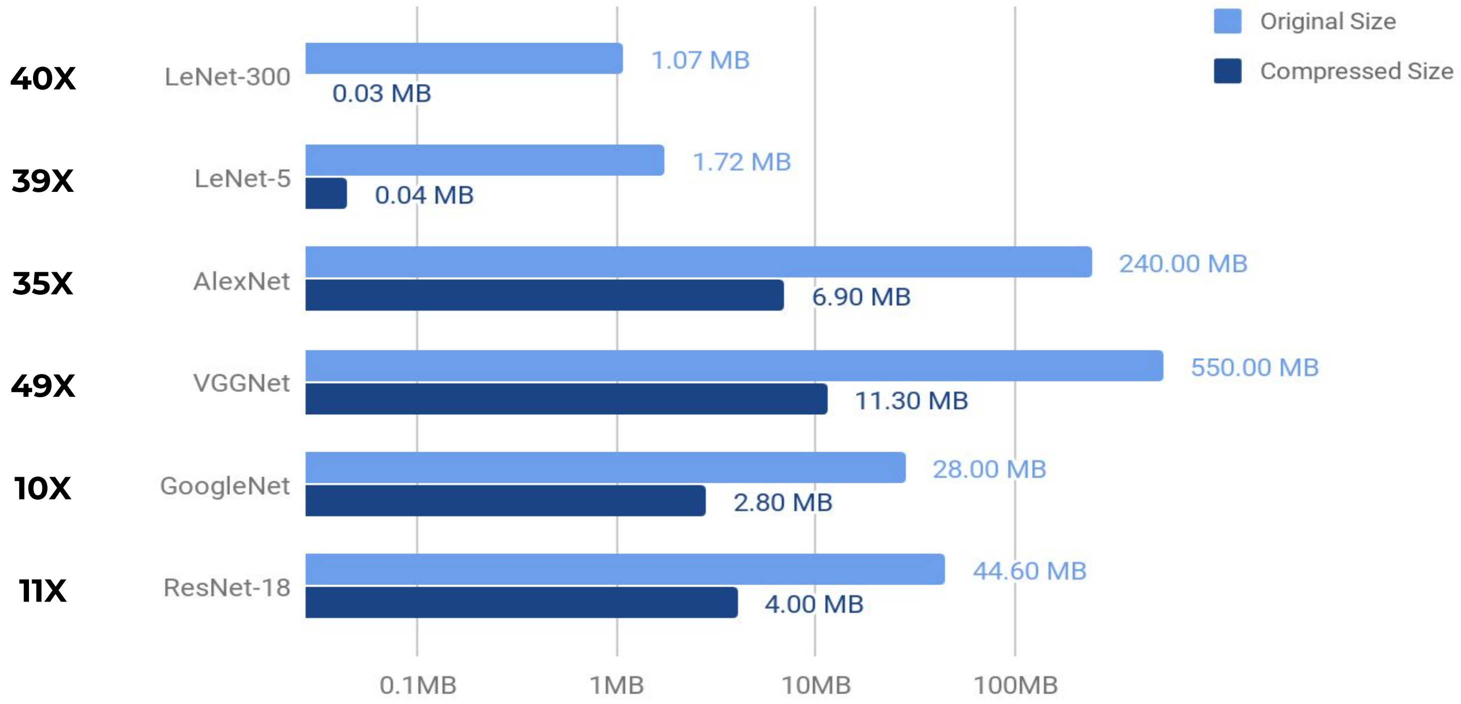


Stage 1

Stage 2

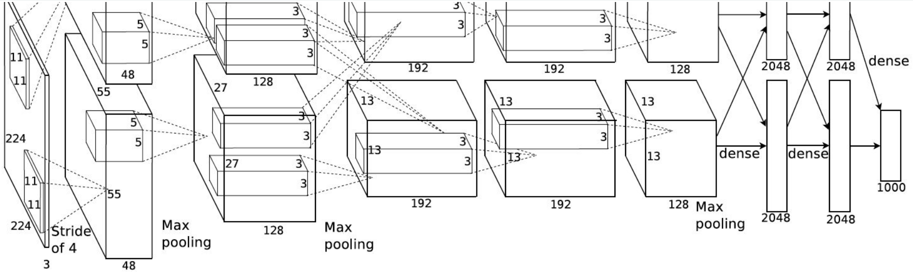
Stage 3

Compression Ratio (w/o accuracy loss)



Mark Horowitz “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014

Deep Compression



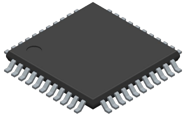
Large DNNs such as AlexNet, VGGNet are **fully fit** into **on-chip SRAM**.

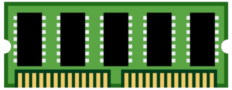
5 pJ
(32 bits)

50 pJ
(32 bits)

640 pJ
(32 bits)

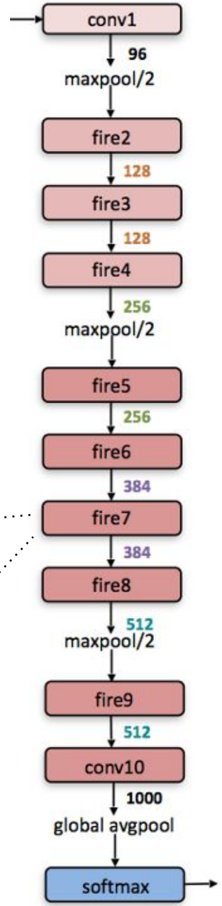
ALU ← Fetching & moving data for computation ← Local SRAM (KB) ← On-Chip SRAM (MB) ← Off-Chip LPDDR DRAM (GB)





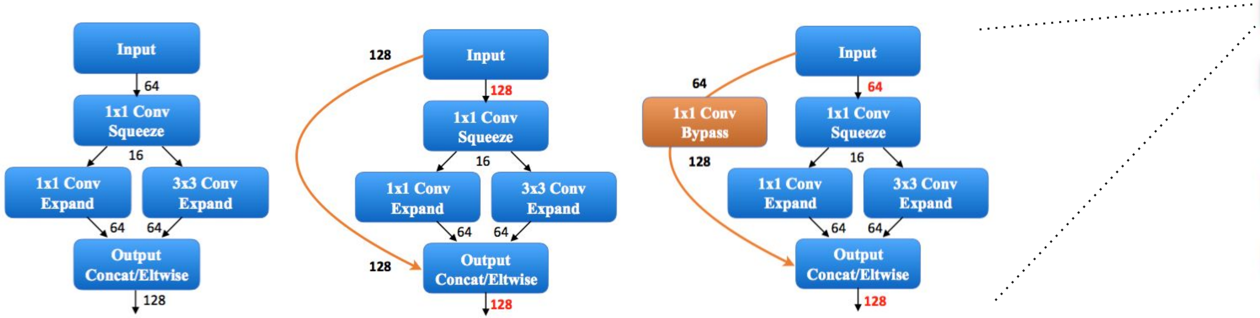
LPDDR

SqueezeNet
+ Deep Compression

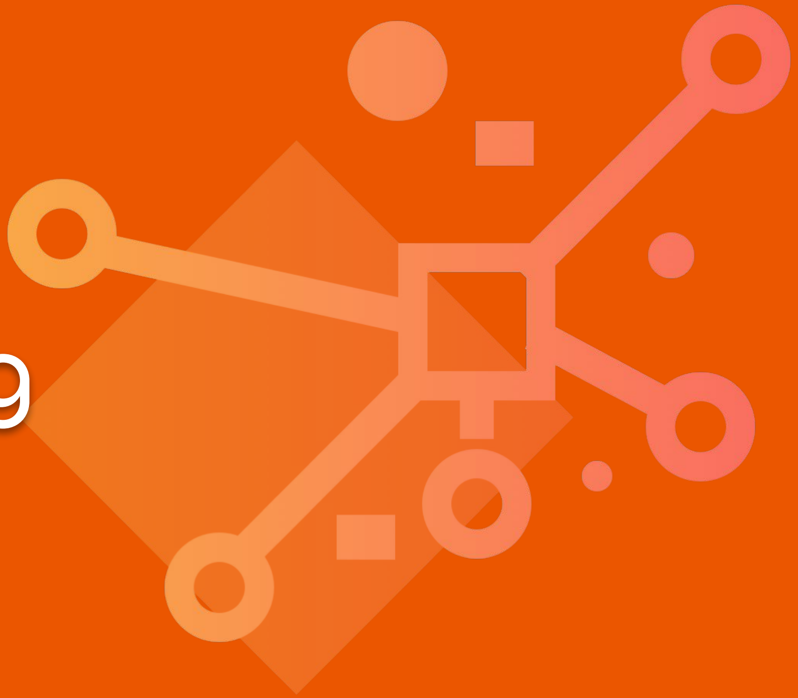


“Dog”. ehh?

Can we apply Deep Compression to
already compact model such as SqueezeNet?

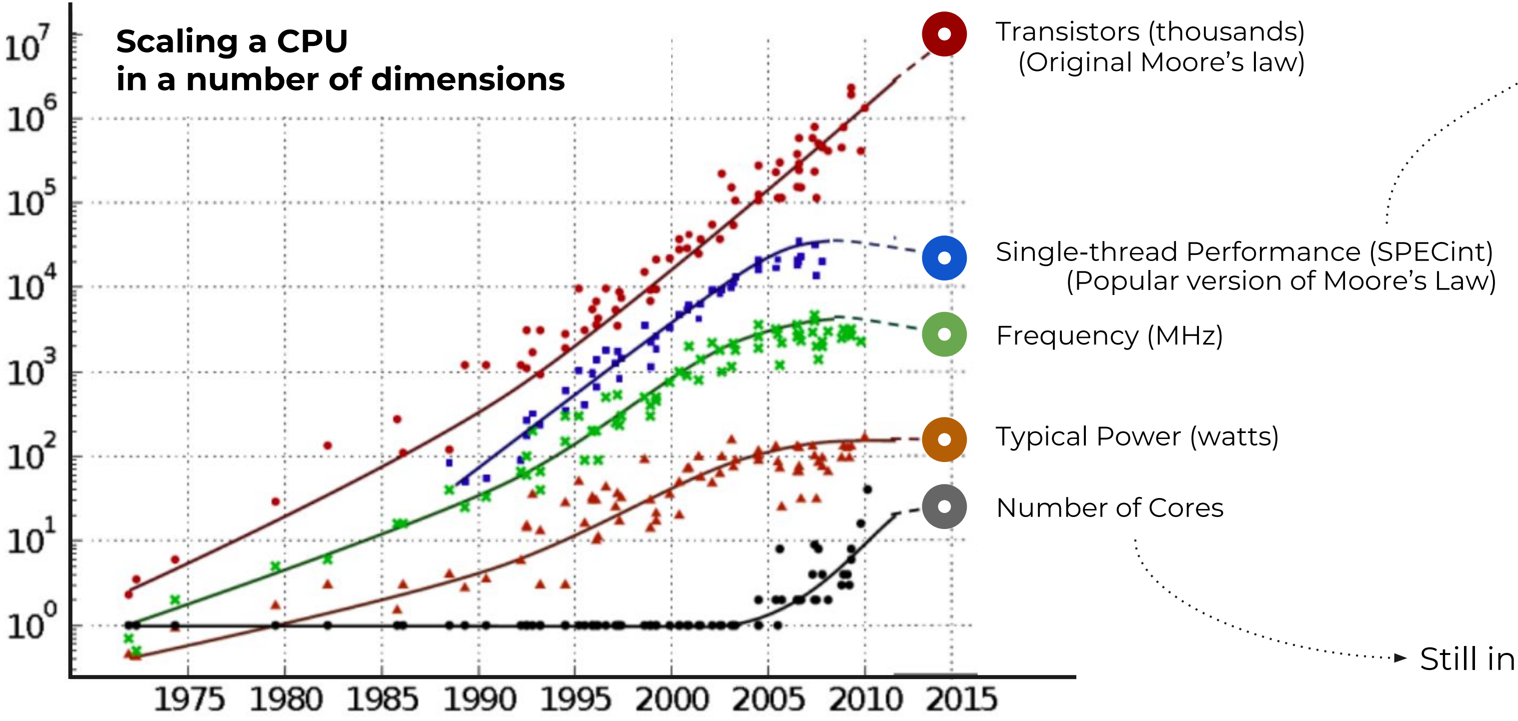


Algorithms for Efficient Training



Moore’s Law

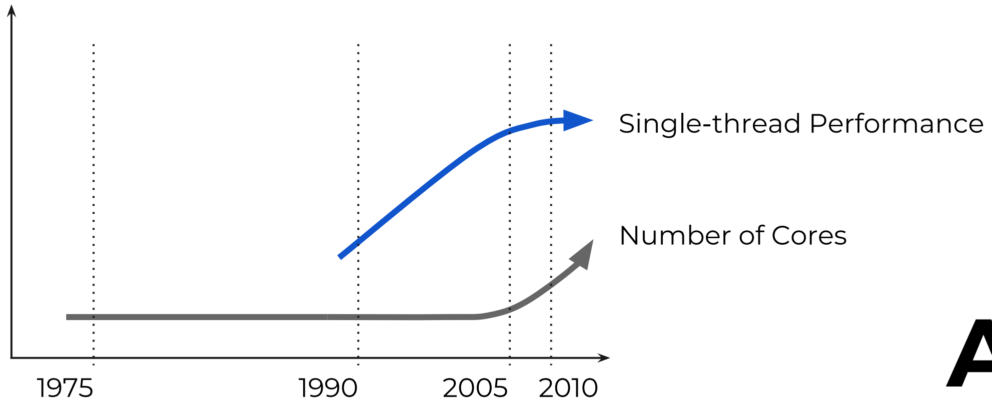
A benchmark spec for
CPU integer processing power



Still increasing

Without Moore's Law (popular version)

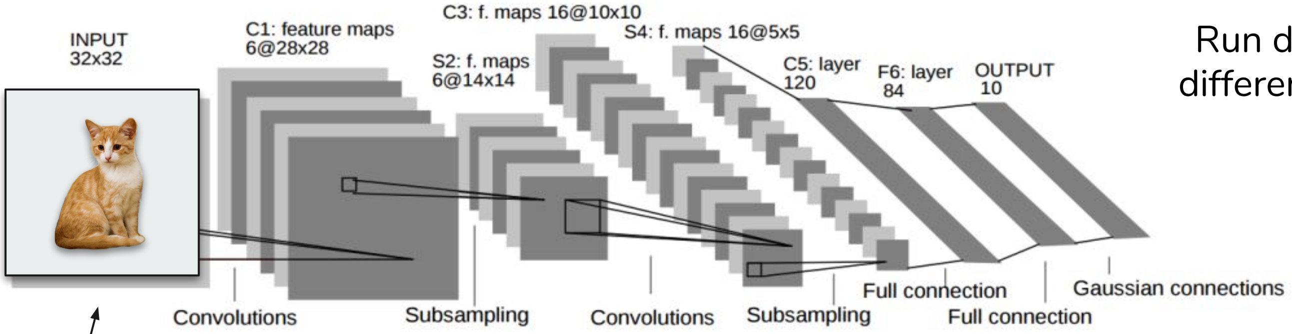
Q: How are we going to continue to scale the performance we need to build a better DNN?



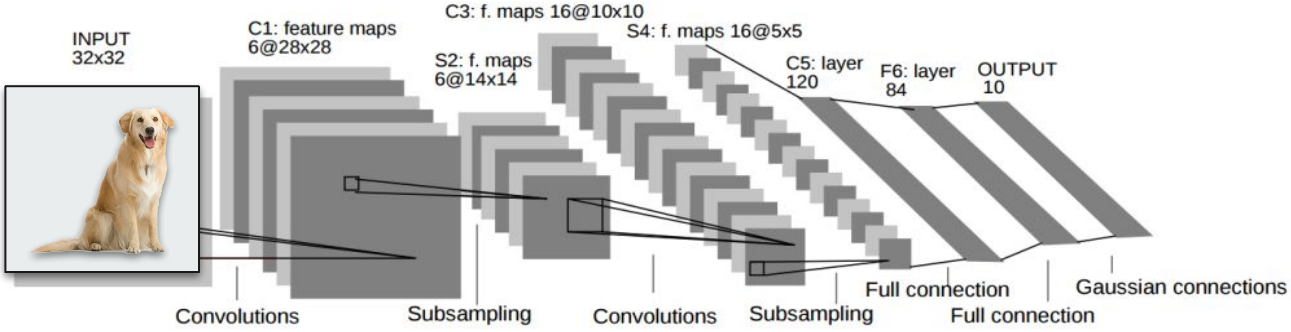
A: Use multiple processors in parallel

Data Parallelism

Run different images on
different GPUs in parallel

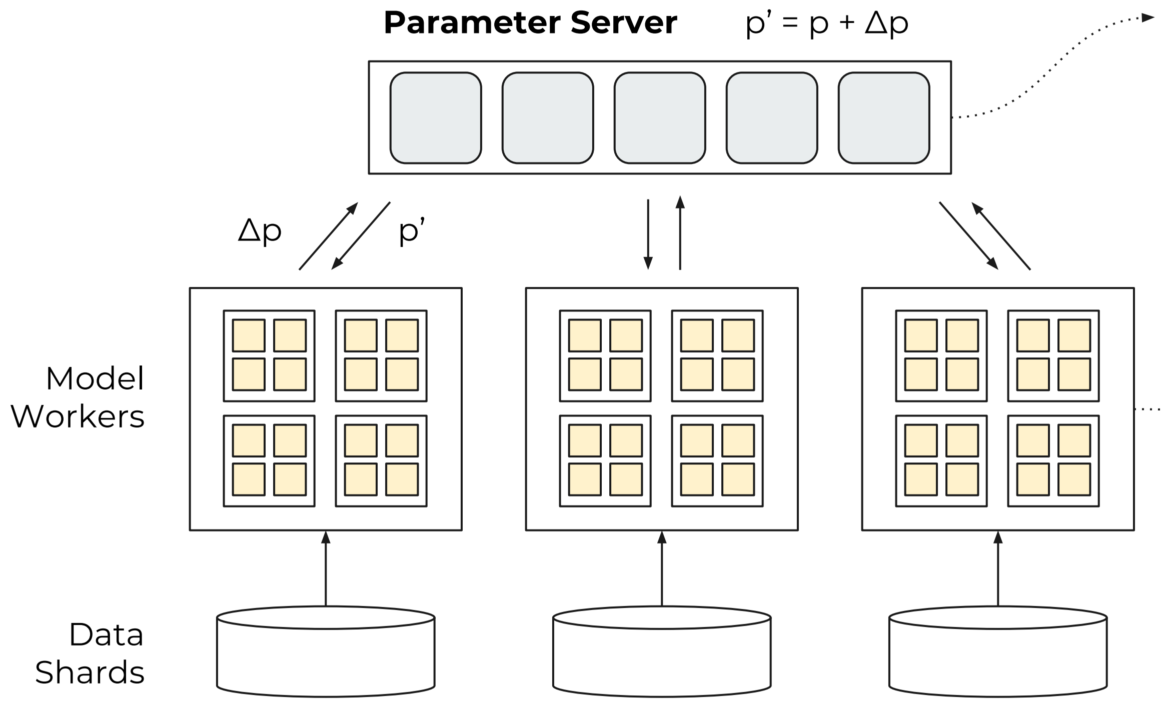


CIFAR-10



Weight updates must
be coordinated
between workers.

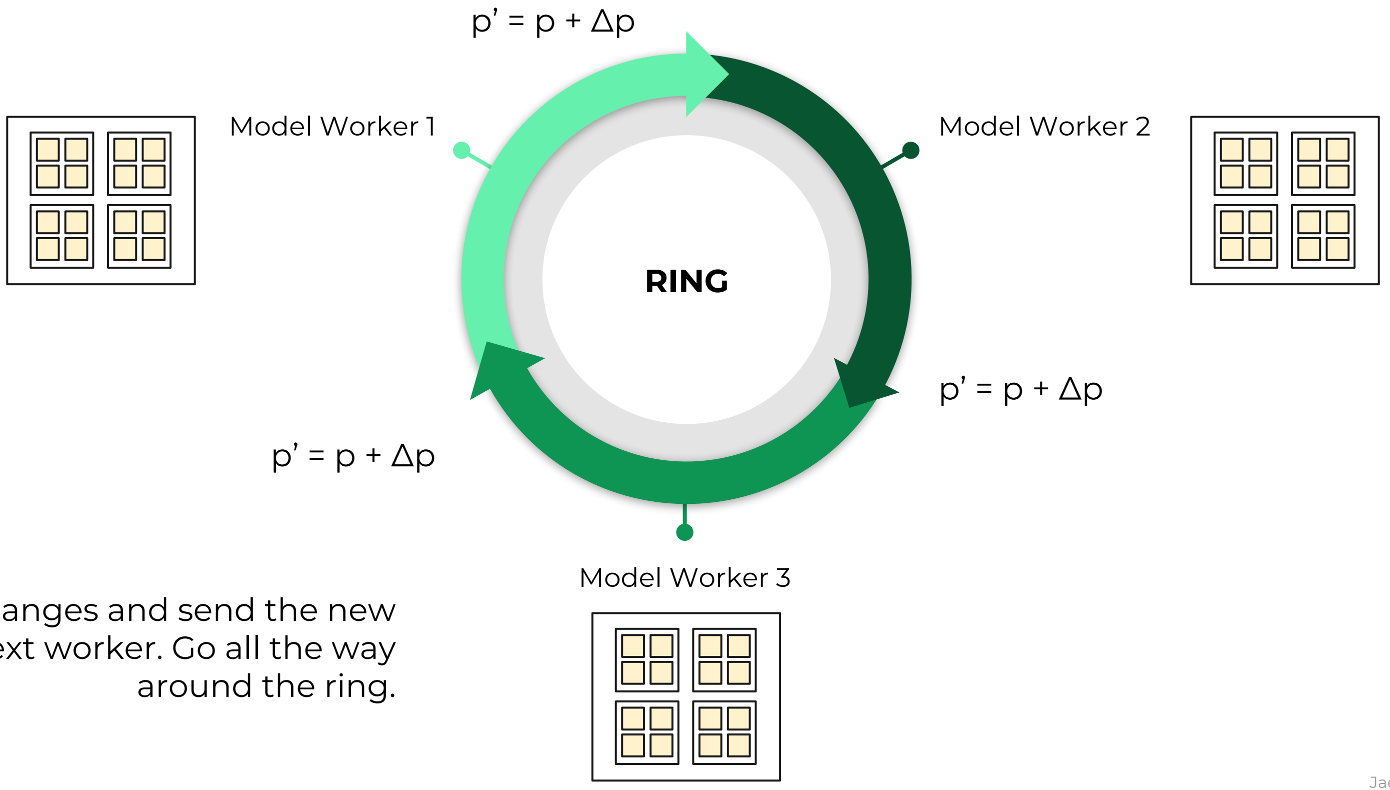
Data Parallelism



Aggregate everyone’s Δp’s, add to the weights, calculate the new weights, and sends them back to everybody.

Each GPU works on some portion of the dataset and sends their Δp (changes in weights) to Parameter Server

Data Parallelism



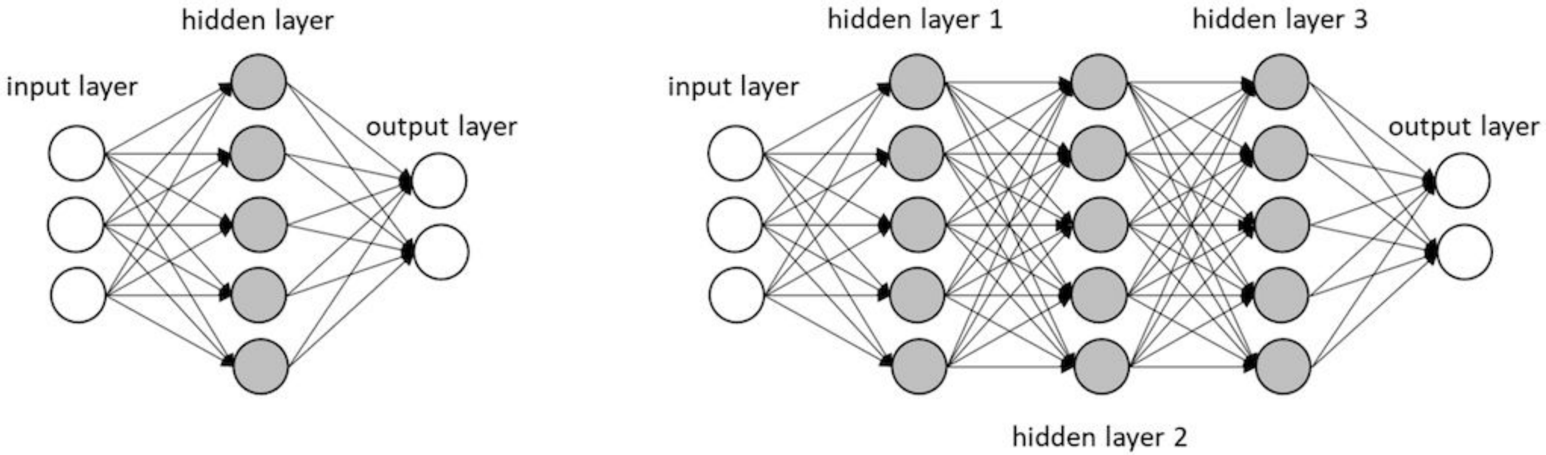
Add my weight changes and send the new weight to the next worker. Go all the way around the ring.

Hyper-Parameter Parallelism

Try many alternative networks in parallel

- Different number of layers
- Different size of convolutional kernels
- Different number of neurons per layer
- . . .

→ Search in the parameter space

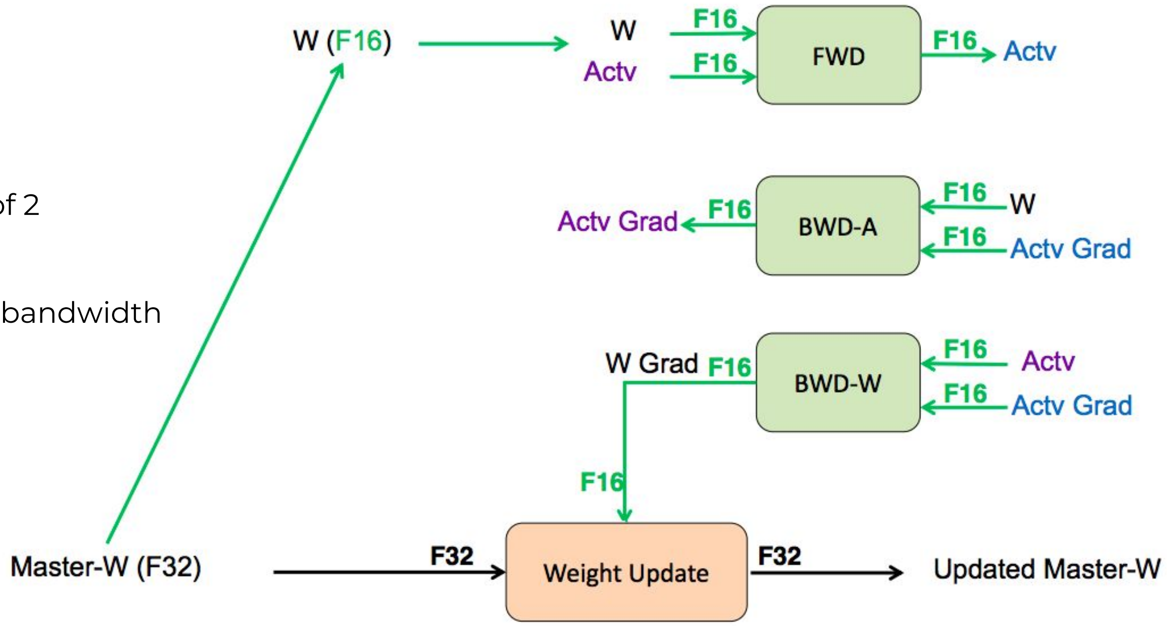


Mixed Precision Training

More precision than required → reduce precision

Save by factor of 2

- Storage
- Memory bandwidth



Mixed Precision Training: Comparison

AlexNet

Mode	Top1 accuracy, %	Top5 accuracy, %
Fp32	58.62	81.25
Mixed precision training	58.12	80.71

Inception V3

Mode	Top1 accuracy, %	Top5 accuracy, %
Fp32	71.75	90.52
Mixed precision training	71.17	90.10

ResNet-50

Mode	Top1 accuracy, %	Top5 accuracy, %
Fp32	73.85	91.44
Mixed precision training	73.6	91.11

Hardware

Quick Overview of Hardware Side, But Why?

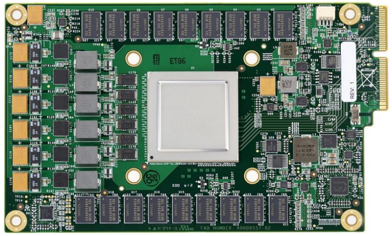
01 We write algorithms and software that runs on hardware.

02 Some of the algorithms we reviewed are actually used in the hardware design.

“An in-depth look at Google’s first Tensor Processing Unit (TPU)”
https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

Google TPU

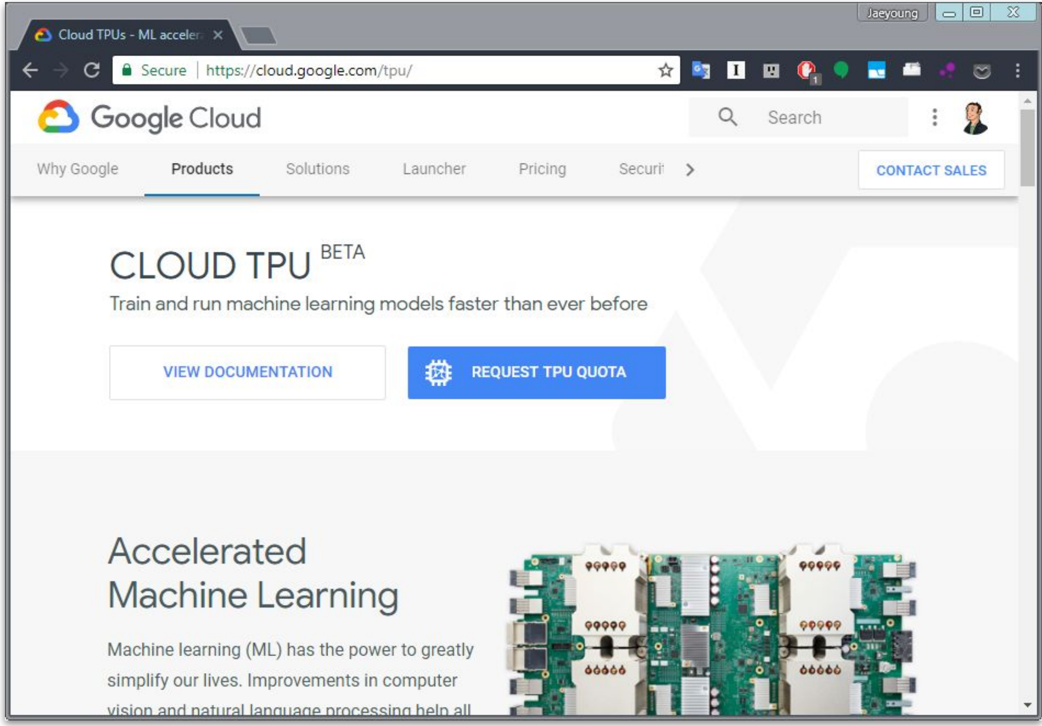
- Tensor Processing Unit
- Compared to GPU & CPU
    - 15x to 30x faster
    - 30x to 80x better energy efficiency
- Only internal use
    - e.g. AlphaGo, Street View, Photos, ...



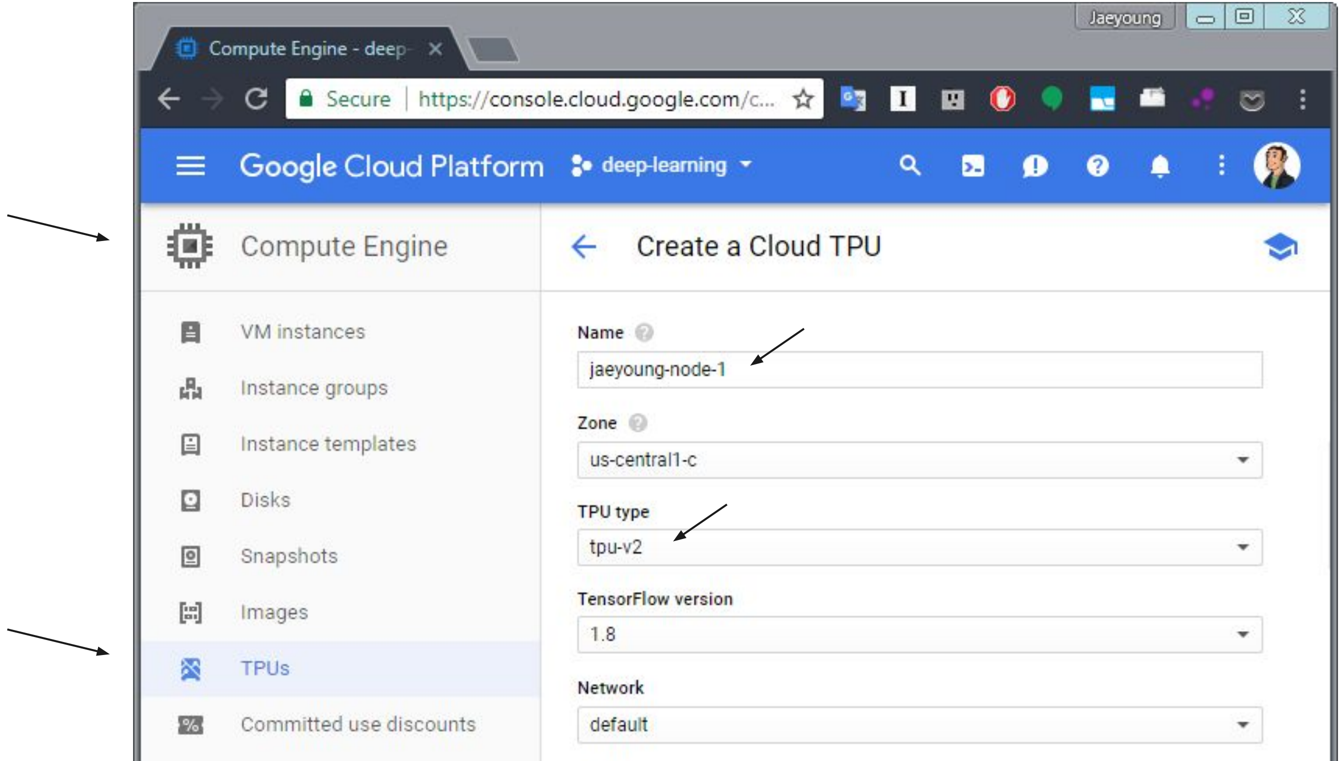
Can be inserted into a SATA hard disk slot for easy/fast deployment to existing server infrastructure

Google Cloud TPU (2nd gen)

- Mid Feb 2018
  Cloud TPU^{BETA} announced.
- Supports for inference as well as training



Google Cloud TPU (2nd gen)



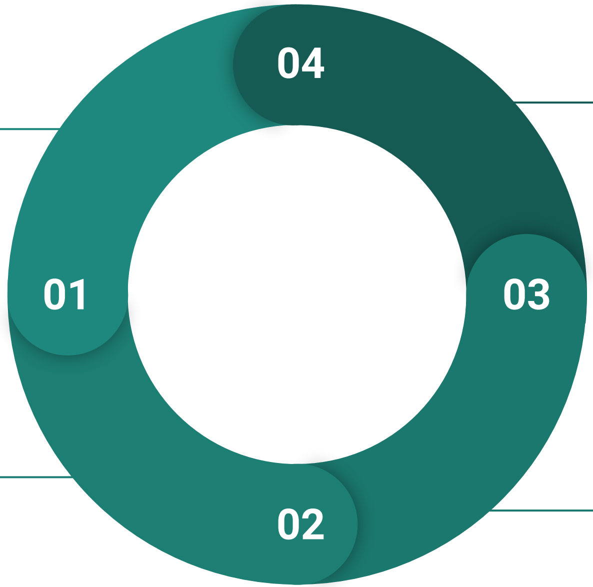
The Secret of Google TPU

Quantization

Optimization technique that uses an 8-bit integer to approximate an arbitrary value between a preset minimum and a maximum value

CISC Instruction Set

High-level instructions specifically designed for neural network inference.



Matrix Multiply Unit

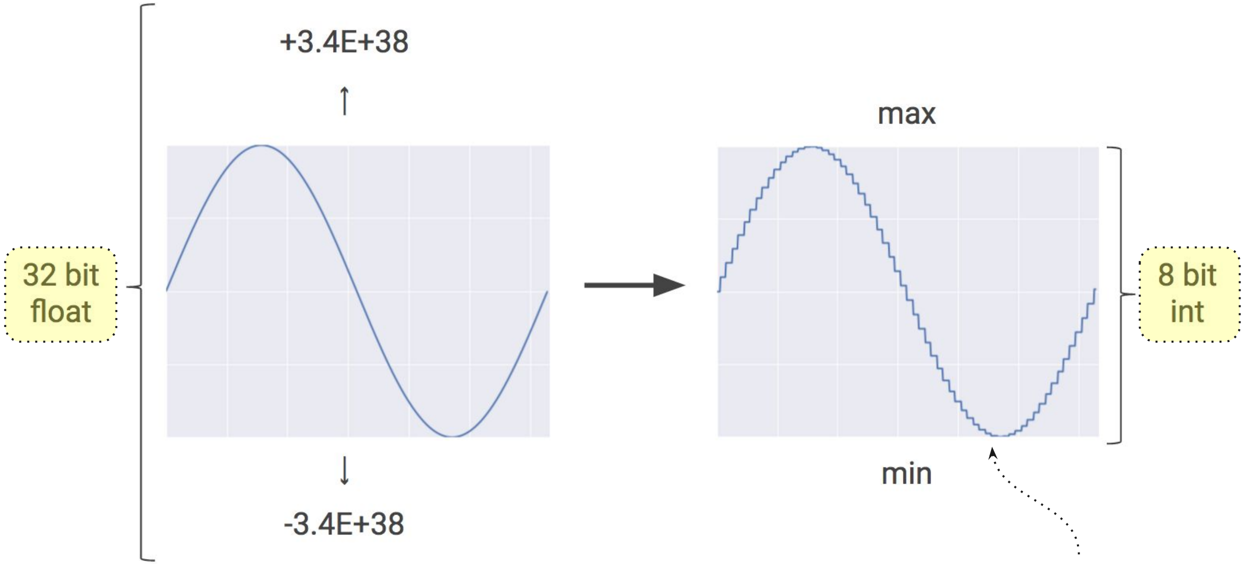
Processes hundreds of thousands of operations (= matrix operation) in a single clock cycle.

Minimal Design

Optimized for neural network inference only. In the TPU, the control logic is minimal and takes under 2% of the die.

Google TPU: Quantization

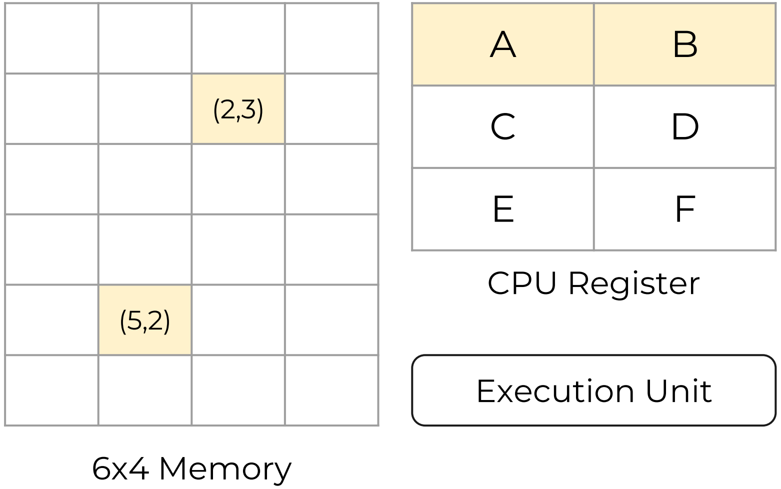
We have already seen the power of Quantization when discussing Deep Compression.



Not smooth, but with careful design,
we can still prevent accuracy loss.

Jouppi et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit” (2017)
“RISC vs CISC”, https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

Google TPU: CISC Instruction Set



Reduced vs. Complex Instruction Set Computer

RISC	CISC
LOAD A, 2:3 **LOAD** B, 5:2 **PROD** A, B **STORE** 2:3, A	**MULT** 2:3, 5:2
Low-level simple instructions that are commonly used e.g. load, store, add, multiply	**High-level instructions** that perform complex operations e.g. compute multiply-and-add many times

Google TPU: CISC Instruction Set

TPU Instruction	Function
Read_Host_Memory	Read data from memory
Read_Weights	Read weights from memory
MatrixMultiply / Convolve	Multiply or convolve with the data and weights, accumulate the results
Activate	Apply activation functions
Write_Host_Memory	Write result to memory

High-level instructions specifically designed for neural network inference

Jouppi et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit” (2017)

Google TPU: Matrix Multiplier Unit

Unified Buffer for Local Activations (96K x 256 x 8b = 24 MiB)

Matrix Multiply Unit (256 x 256 x 8b = 64K)

DRAM Port DDR3

Host Interface

Accumulators (4K x 256 x 32b = 4 Mib)

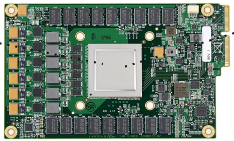
Control

Activation Pipeline

PCIe Interface

Misc I/O

DRAM Port DDR3



65,536

(multiply-and-add operations per cycle)

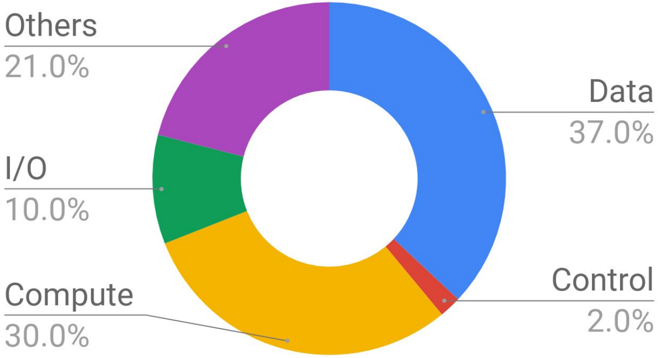
700

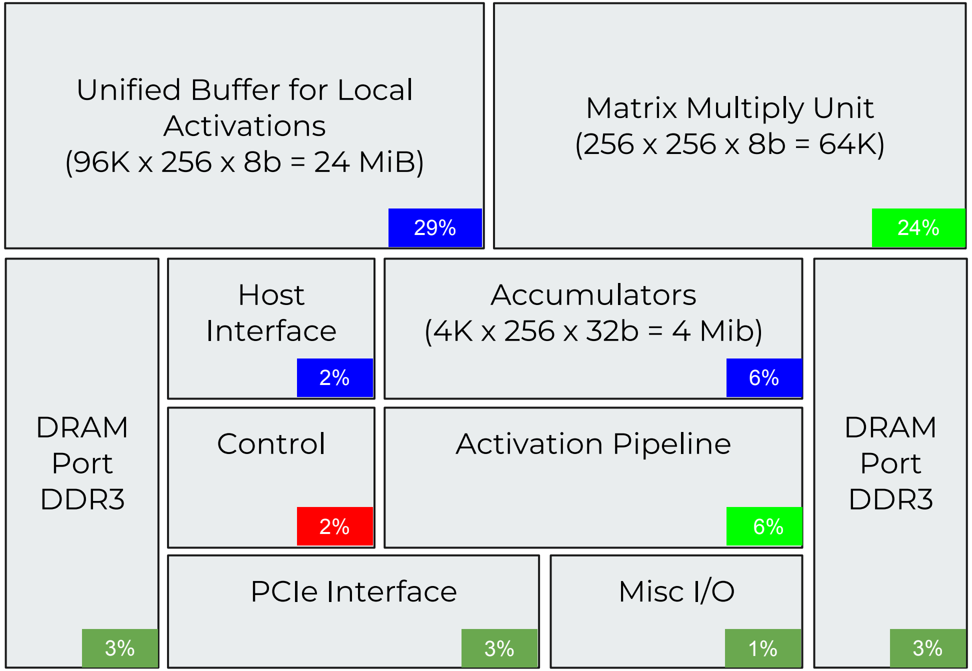
(TPU clock in MHz)

$46 \mathrm{x} 10^{12}$

(65,536 × 700M operations per sec)

Google TPU: Minimal Design





EIE (Efficient Inference Engine)

10x
less computation

Sparse Weight

1

W (weight) x A (activation)
If W = 0 → 0 x A = 0

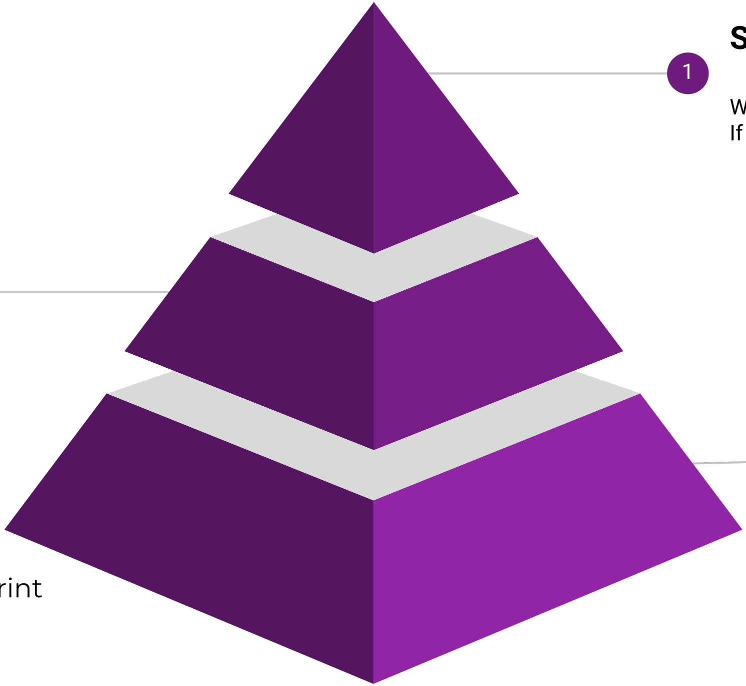
5x
less memory footprint

Weight Sharing

With K-means clustering,
e.g. the blues (2.09, 2.12., 1.92, 1.87) are
treated as 2.0 instead
(i.e. effective weights)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

2



8x less memory footprint

Sparse Activation

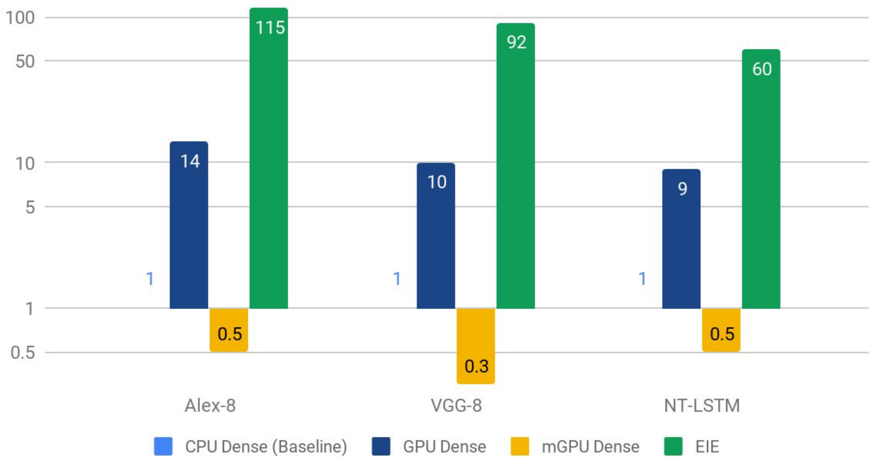
3

W (weight) x A (activation)
If A = 0 → W x 0 = 0

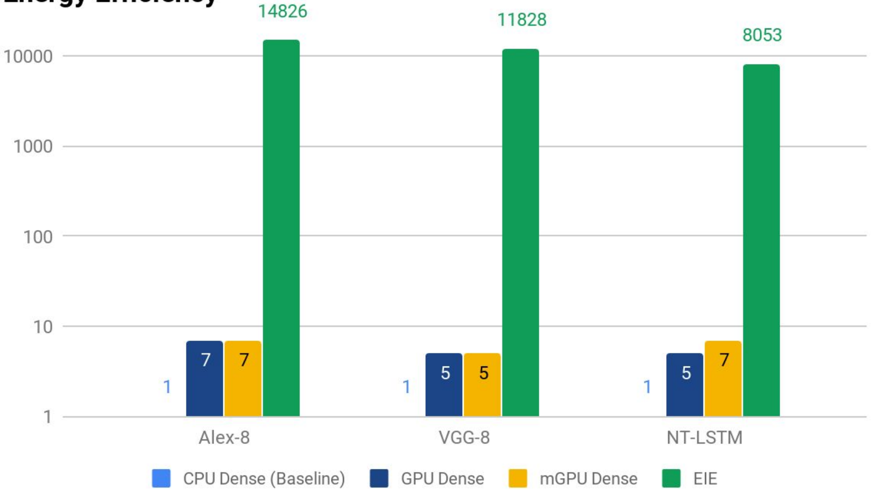
3x less computation

EIE: Speedup and Energy Efficiency

Speedup



Energy Efficiency



Break