

# lec15

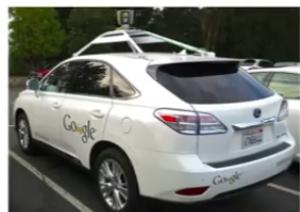
Creation Date: 17/01/2020 22:00

Last Modified Date: 22/01/2020 12:42

## Lec 15: Efficient Methods and Hardware for Deep Learning

### Deep Learning is Changing Our Lives

Self-Driving Car



Machine Translation



AlphaGo



Smart Robots



Stanford University

ford  
University

3

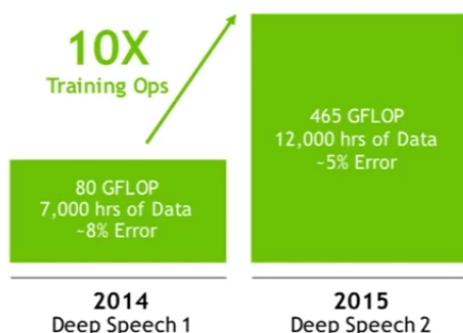
### Models are Getting Larger

IMAGE RECOGNITION



Microsoft

SPEECH RECOGNITION



Baidu

Dally, NIPS'2016 workshop on Efficient Methods for Deep Neural Networks

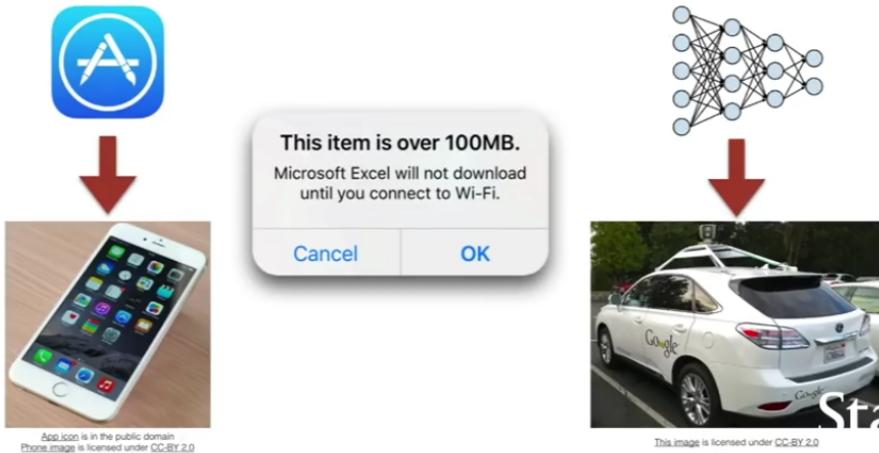
Stanford University

ford  
University

4

## The first Challenge: Model Size

Hard to distribute large models through over-the-air update



Stanford University

5

## The Second Challenge: Speed

	Error rate	Training time
ResNet18:	10.76%	2.5 days
ResNet50:	7.02%	5 days
ResNet101:	6.21%	1 week
ResNet152:	6.16%	1.5 weeks

Such long training time limits ML researcher's productivity

Training time benchmarked with fb.resnet.torch using four M40 GPUs

Stanford University

6

## The Third Challenge: Energy Efficiency



This image is in the public domain

AlphaGo: 1920 CPUs and 280 GPUs,  
**\$3000 electric bill** per game



This image is in the public domain



Phone image is licensed under CC BY 2.0



on mobile: **drains battery**  
on data-center: **increases TCO**



This image is licensed under CC BY 2.0

Stanford  
University

Stanford University

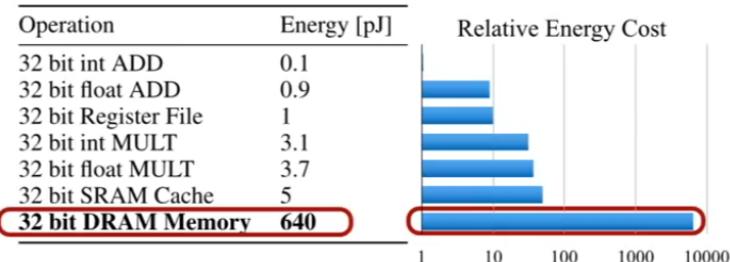
7

Challenges are:

- Model Size
- Speed
- Energy Efficiency

## Where is the Energy Consumed?

**larger model => more memory reference => more energy**



Battery images are in the public domain  
Image 1, Image 2, Image 3, Image 4



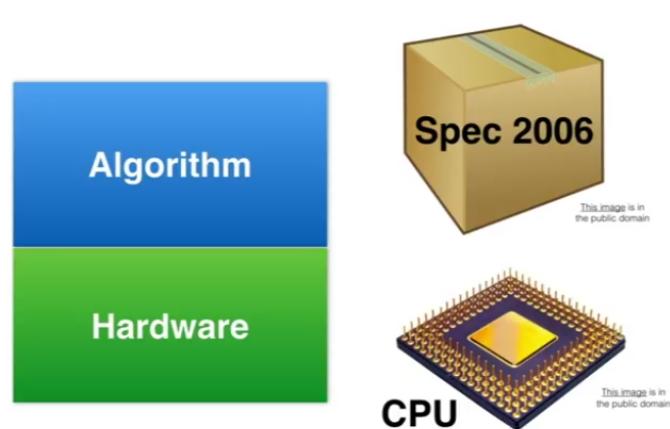
ford  
University

Stanford University

10

Before:

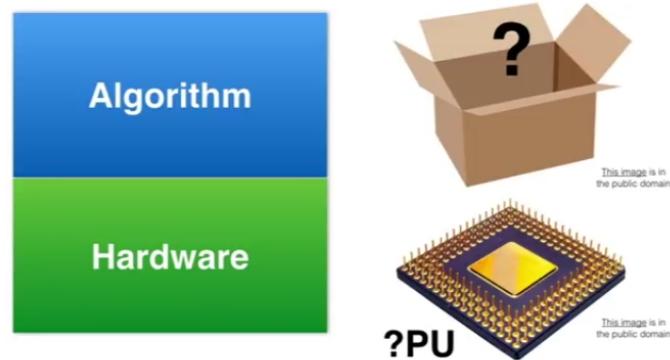
## Application as a Black Box



ford  
University  
Stanford University 12

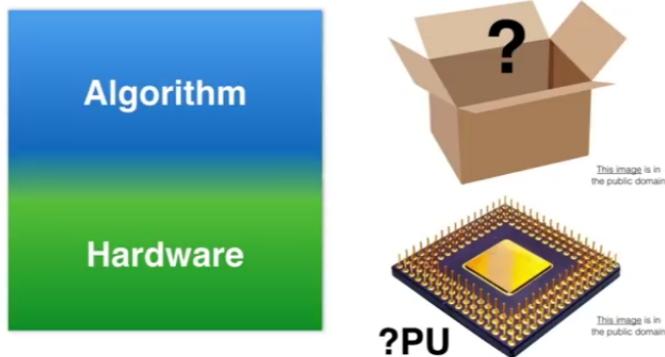
What we should do now:

## Open the Box before Hardware Design



ford  
University  
Stanford University 13

# Open the Box before Hardware Design



Breaks the boundary between algorithm and hardware

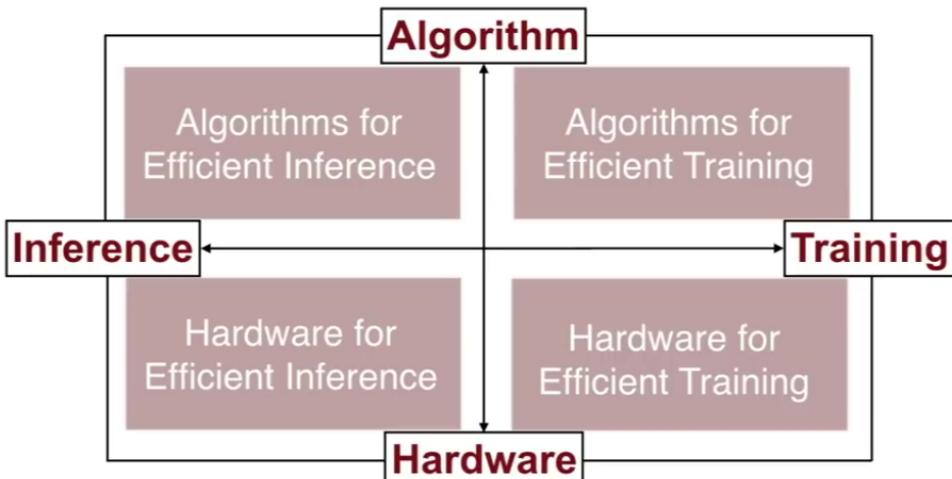
ford  
University

Stanford University

13

- Breaks the boundary between algorithm and hardware.

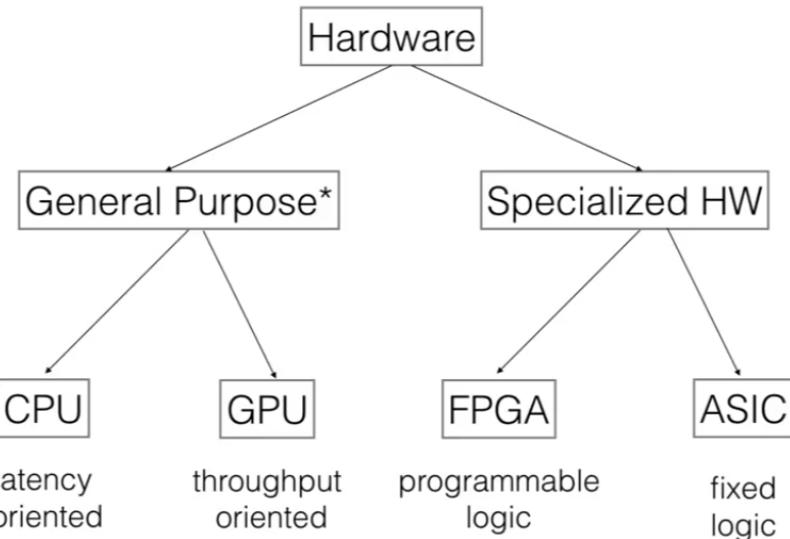
## Agenda



Stanford University

ford  
University

# Hardware 101: the Family



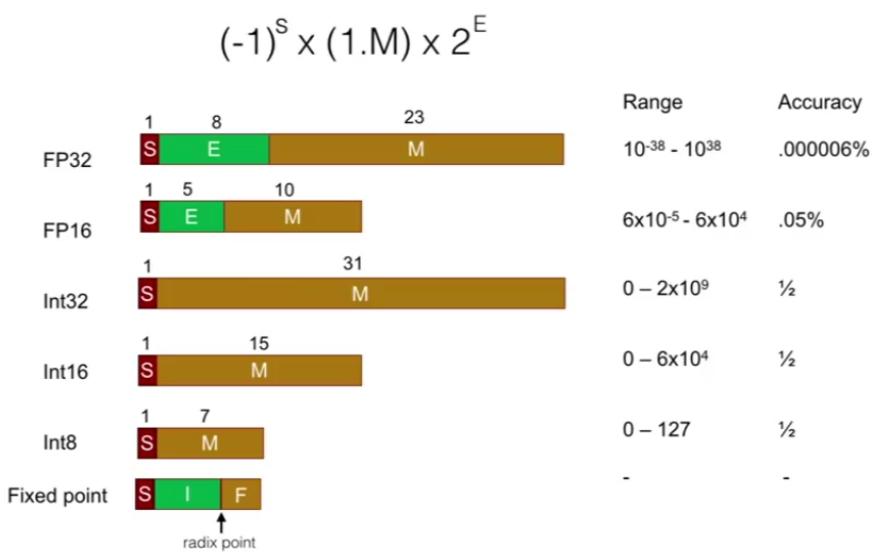
\* including GPGPU

Stanford University

ord  
University

- Specialized HW is tuned for a specific kind of application or domain of applications.
- CPU is latency oriented because it is single threaded (like a big elephant)
- GPU has many small weak cores (like an army of ants)
- FPGA(Field Programmable Gate Array) is programmable and the logic can be changed. Less efficient but can we used for prototyping.
- ASIC(Application Specific Integrated Circuit) has a fixed logic for a certain application.
- Google's TPU is a kind of ASICs

# Hardware 101: Number Representation



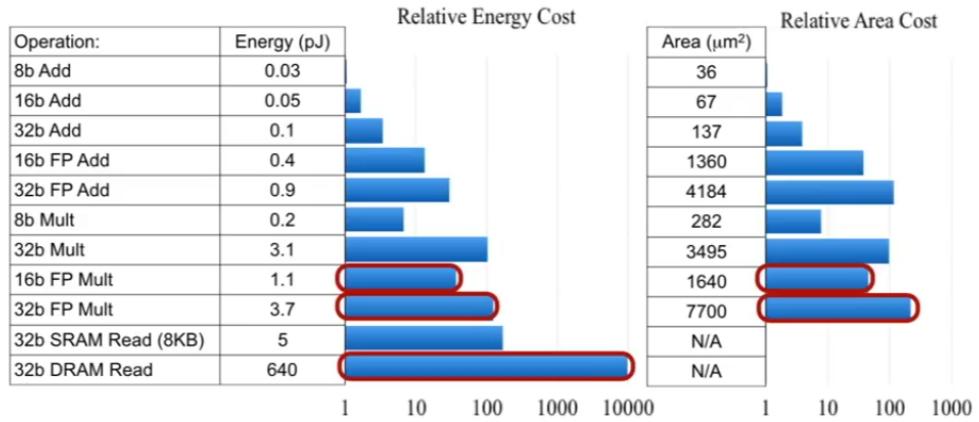
Dally, High Performance Hardware for Machine Learning, NIPS'2015

Stanford University

ord  
University

- S-Sign Bit, M-Mantissa Bits, E-Exponent Bits

## Hardware 101: Number Representation



Energy numbers are from Mark Horowitz "Computing's Energy Problem (and what we can do about it)", ISSCC 2014  
 Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.

ford  
University  
Stanford University

## Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

Stanford University  
Stanford  
University

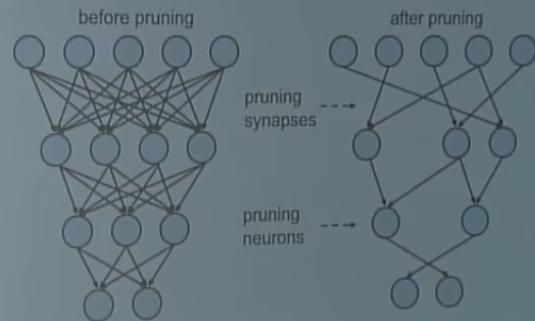
**Pruning:**

## Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

Stanford University  
**Stanford**  
University

### Pruning Neural Networks



[Lecun et al. NIPS'89]

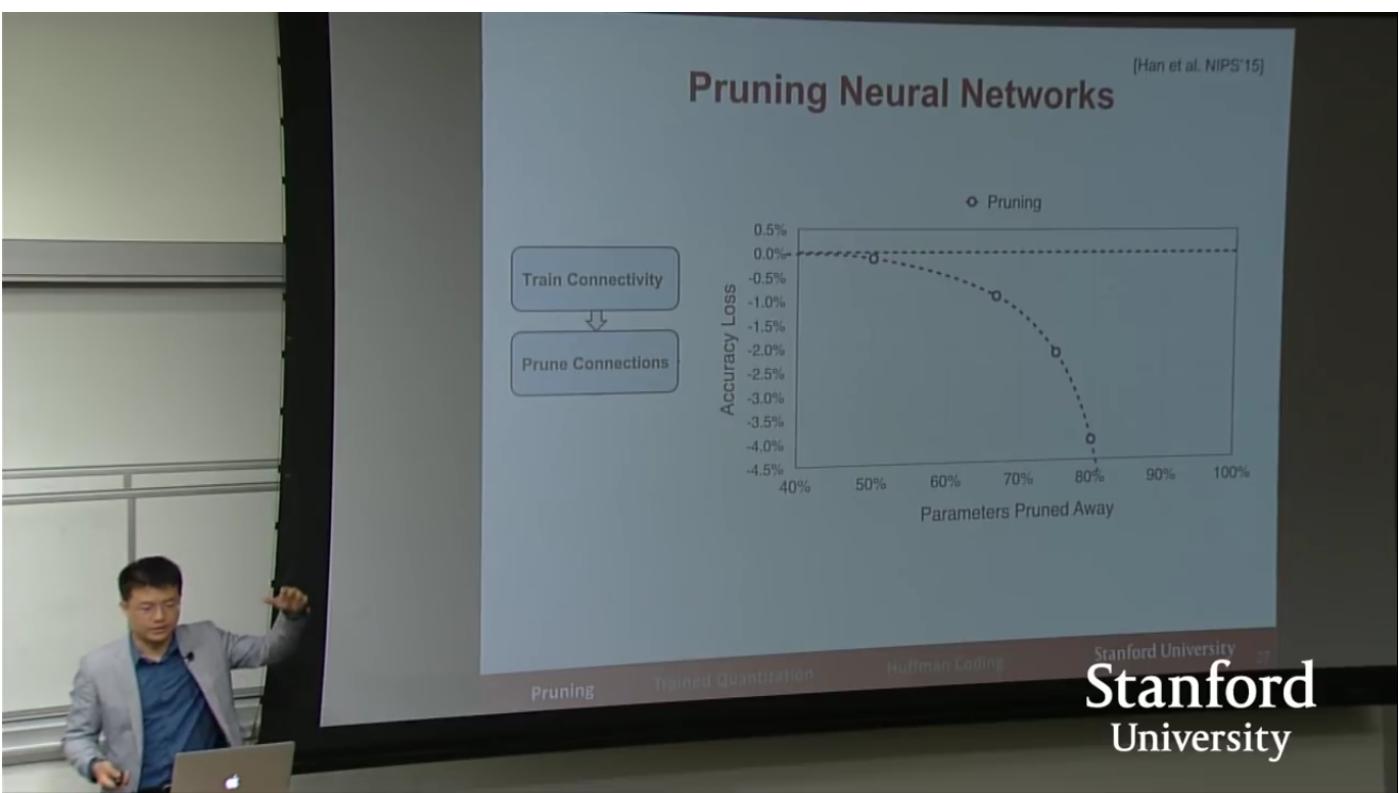
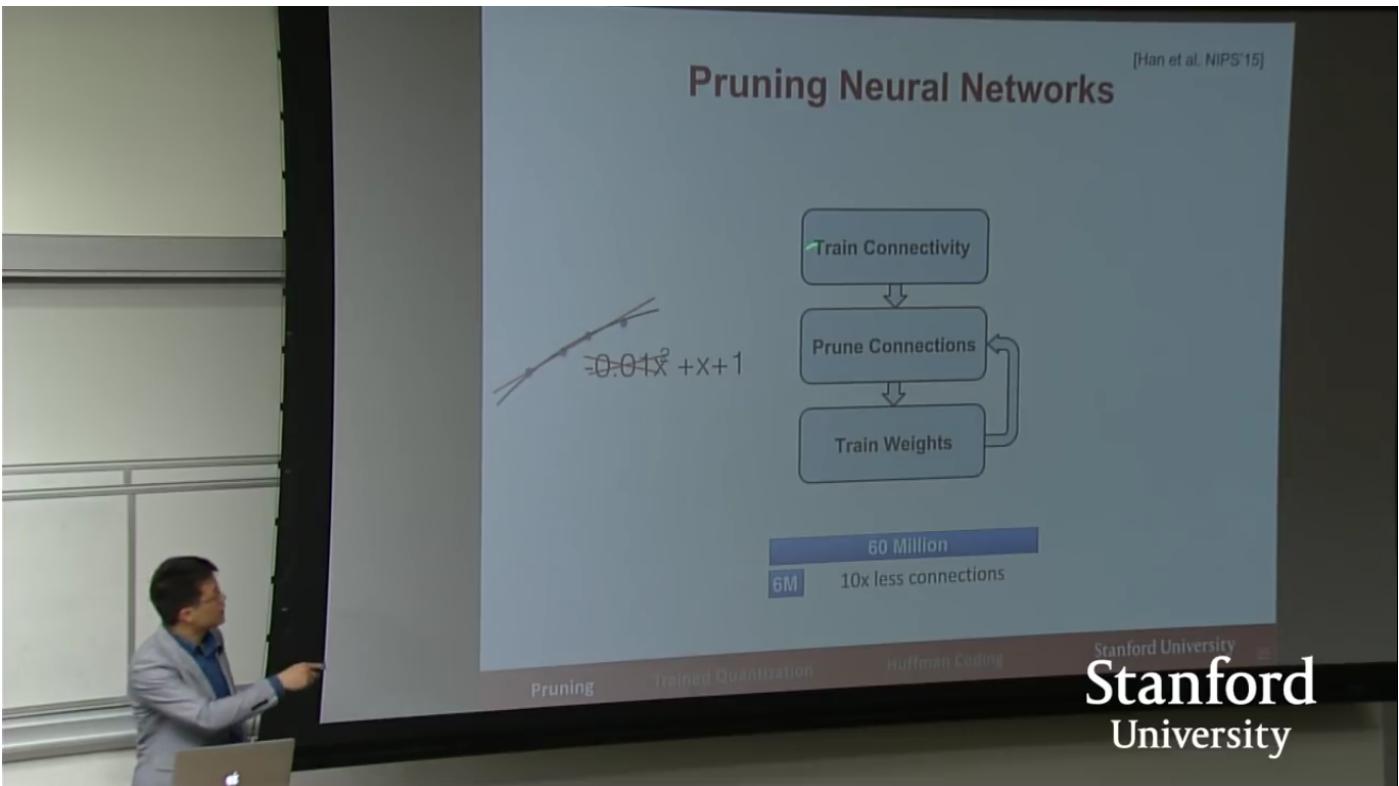
[Han et al. NIPS'15]

Pruning

Trained Quantization

Huffman Coding

Stanford University  
**Stanford**  
University



**Retrain to Recover Accuracy**

[Han et al. NIPS'15]

Train Connectivity  
↓  
Prune Connections  
↓  
Train Weights

Accuracy Loss

Parameters Pruned Away

Pruning      Pruning+Retraining

Parameters Pruned Away (%)	Pruning Accuracy Loss (%)	Pruning+Retraining Accuracy Loss (%)
40%	-0.2	-0.2
50%	-0.3	-0.3
60%	-0.7	-0.7
70%	-1.2	-1.2
80%	-3.8	-3.8
85%	-4.2	-3.5
90%	-4.5	-2.5
95%	-4.8	-1.5
100%	-5.0	-0.5

Stanford University

- Pruning even though reduces Accuracy until a certain threshold, **Pruning and Re-Training can help in regaining the Accuracy of the model.**

**Iteratively Retrain to Recover Accuracy**

[Han et al. NIPS'15]

Train Connectivity  
↓  
Prune Connections  
↓  
Train Weights

Accuracy Loss

Parameters Pruned Away

Pruning      Pruning+Retraining      Iterative Pruning and Retraining

Parameters Pruned Away (%)	Pruning Accuracy Loss (%)	Pruning+Retraining Accuracy Loss (%)	Iterative Pruning and Retraining Accuracy Loss (%)
40%	-0.2	-0.2	-0.2
50%	-0.3	-0.3	-0.3
60%	-0.7	-0.7	-0.7
70%	-1.2	-1.2	-1.2
80%	-3.8	-3.8	-3.8
85%	-4.2	-3.5	-3.5
90%	-4.5	-2.5	-2.5
95%	-4.8	-1.5	-1.5
100%	-5.0	-0.5	-0.5

Stanford University

- We can notice that even at a Pruning Rate of 90% the accuracy of the model is retrained due to iterative pruning and retraining.

[Han et al. NIPS'15]

## Pruning RNN and LSTM

\*Karpathy et al. 'Deep Visual-Semantic Alignments for Generating Image Descriptions', 2015.

Figure reprinted with permission from ICLR 2016, reproduced for educational purposes.

Pruned Weights (%)	no retrain (BLEU 3)	pretrained (BLEU 3)	retrain (BLEU 3)
50	~22.5	~24	~22.5
60	~20.5	~24	~24
70	~15.5	~24	~24
80	~8.5	~24	~24
90	~7.5	~24	~24
95	~7.5	~24	~20.5

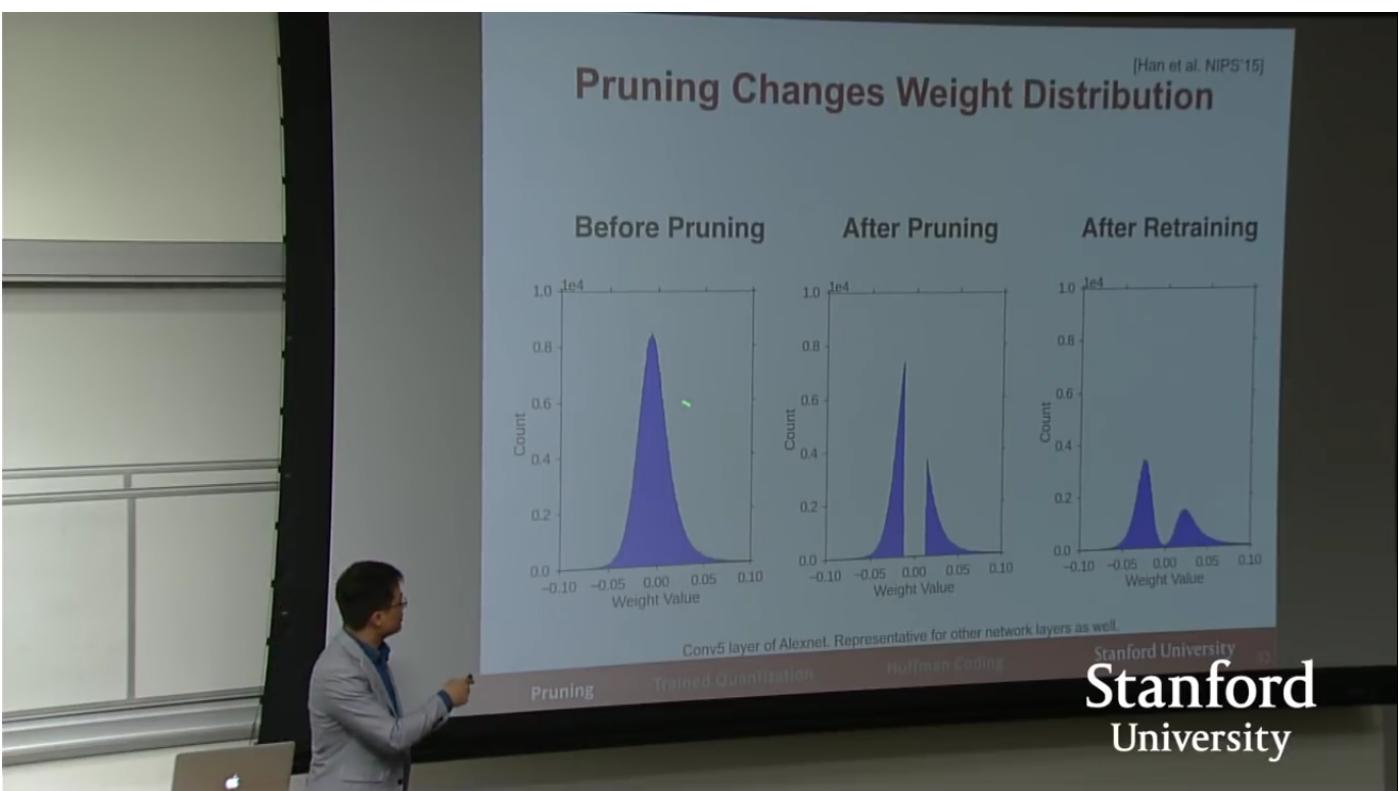
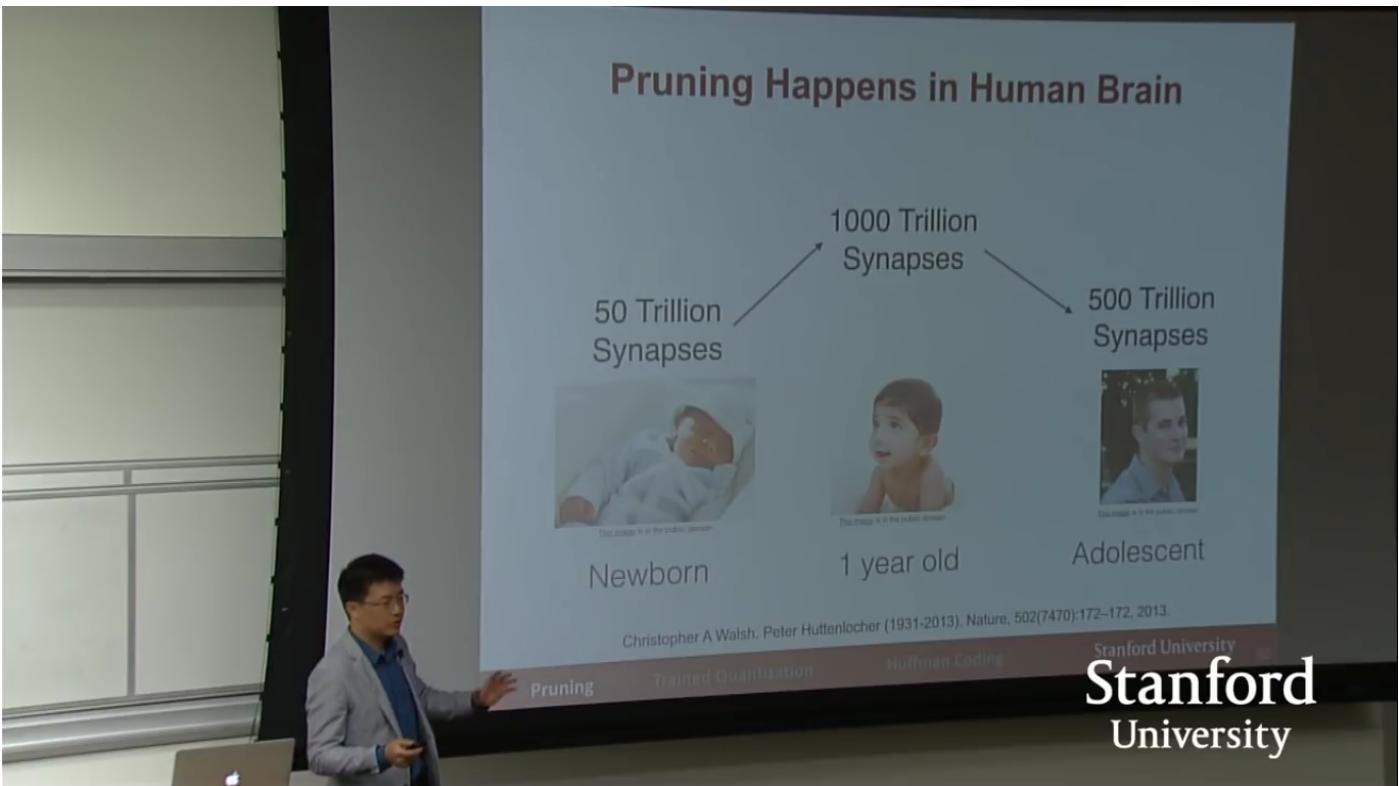
Pruning      Trained Quantization      Huffman Coding      Stanford University

[Han et al. NIPS'15]

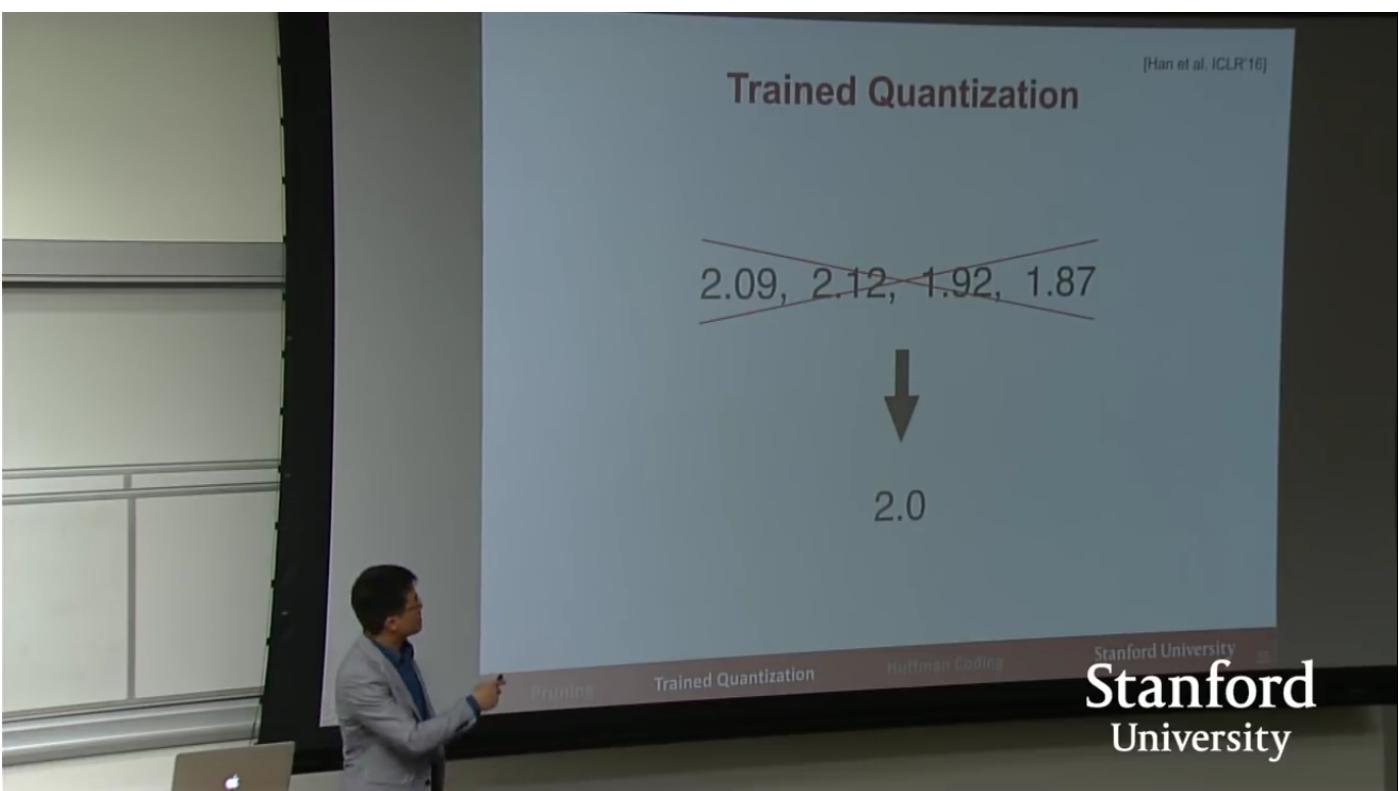
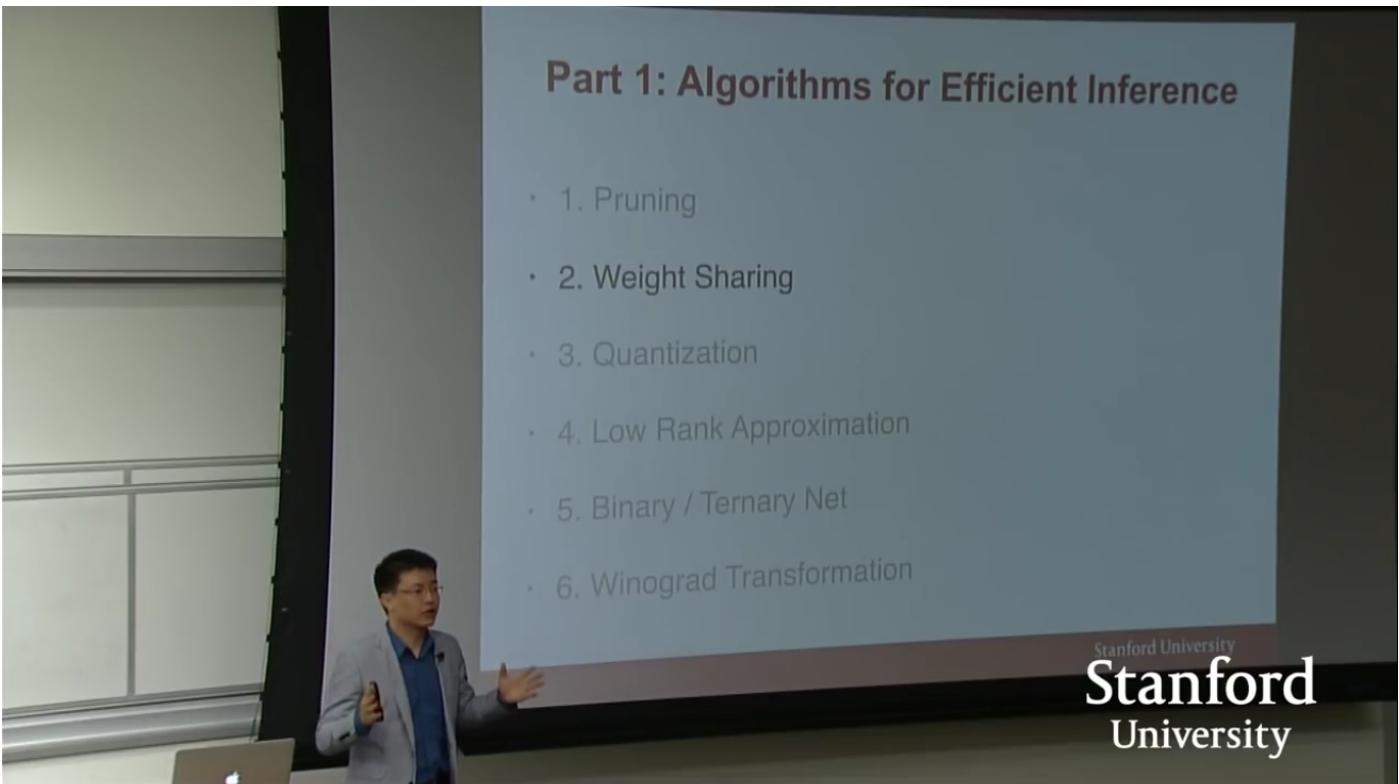
## Pruning RNN and LSTM

- Original: a basketball player in a white uniform is playing with a ball
- Pruned 90%: a basketball player in a white uniform is playing with a basketball
- Original : a brown dog is running through a grassy field
- Pruned 90%: a brown dog is running through a grassy area
- Original : a man is riding a surfboard on a wave
- Pruned 90%: a man in a wetsuit is riding a wave on a beach
- Original : a soccer player in red is running in the field
- Pruned 95%: a man in a red shirt and black and white black shirt is running through a field

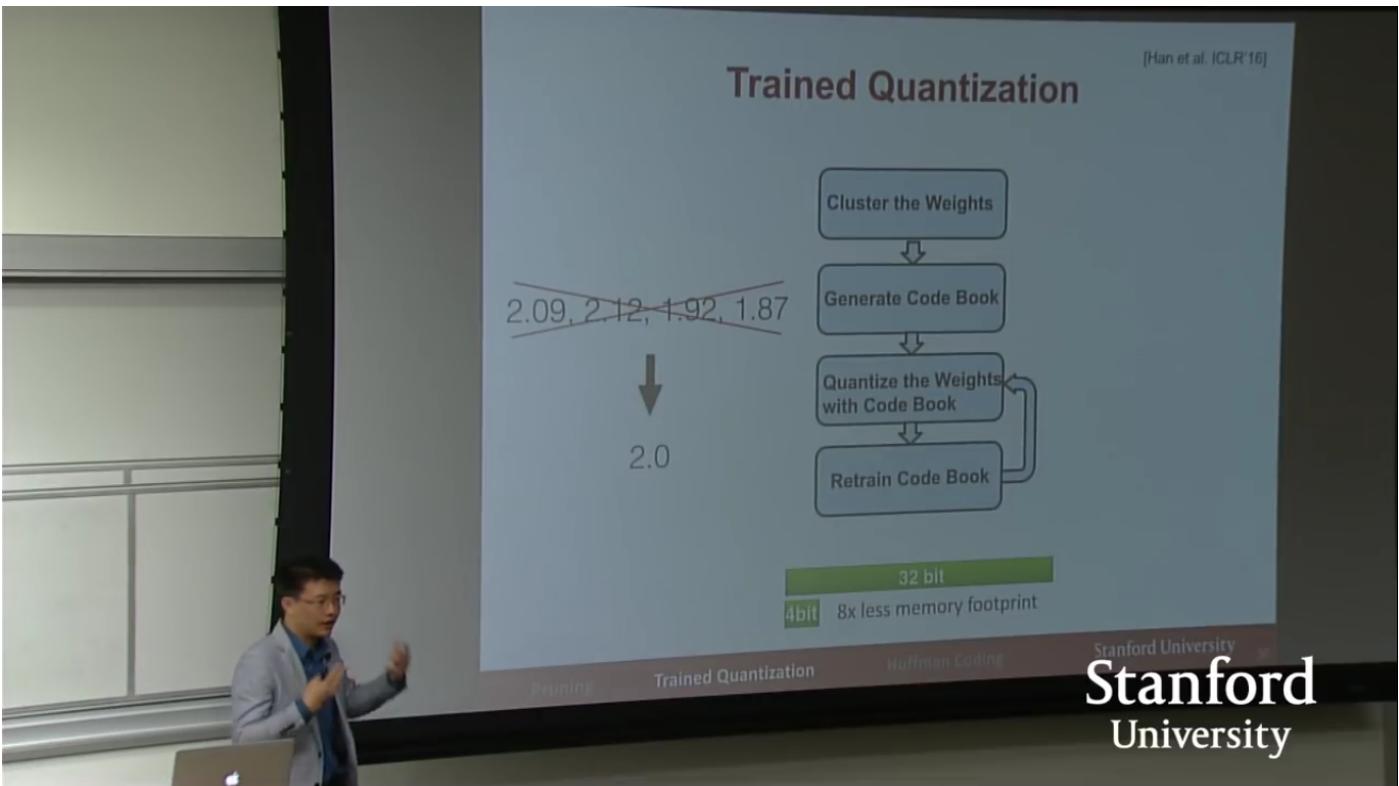
Pruning      Trained Quantization      Huffman Coding      Stanford University



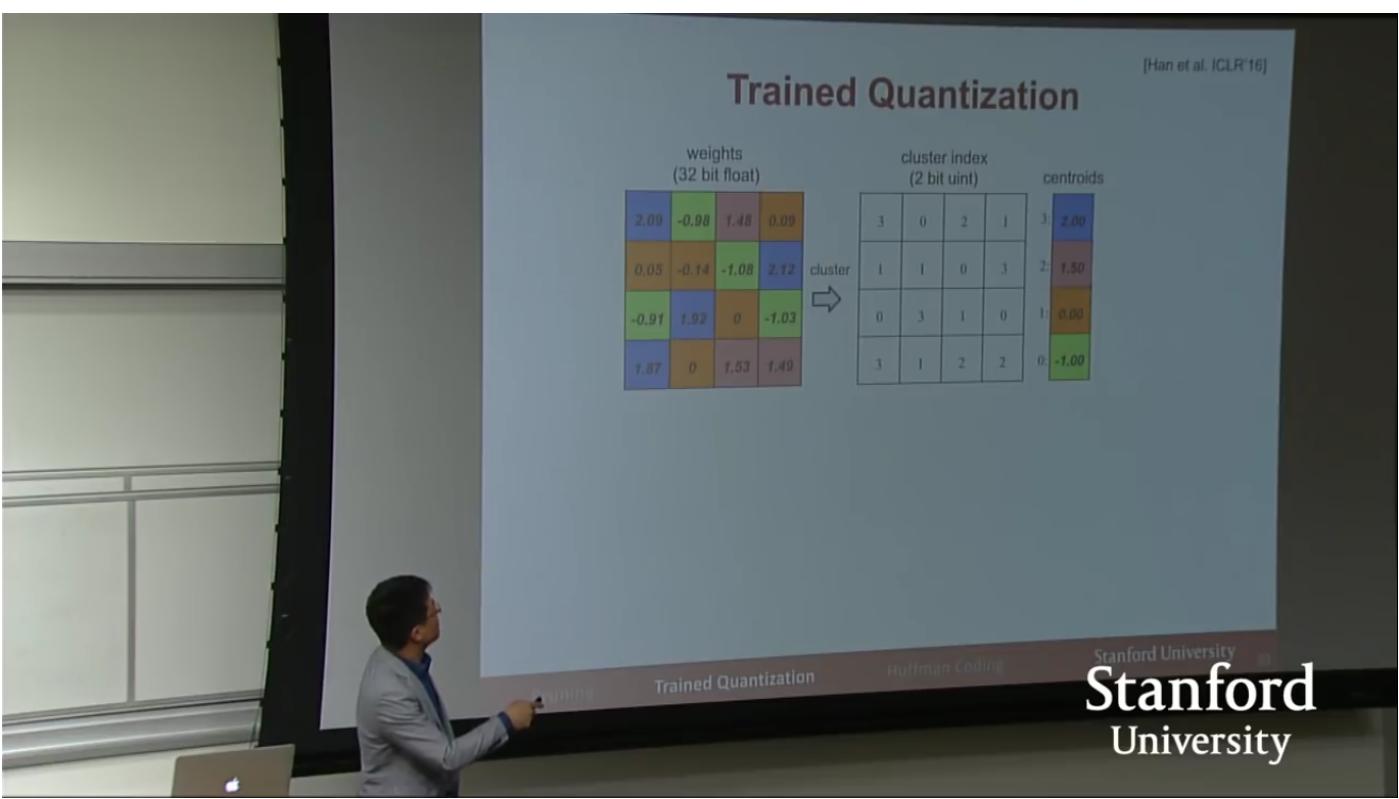
**Weight Sharing:**



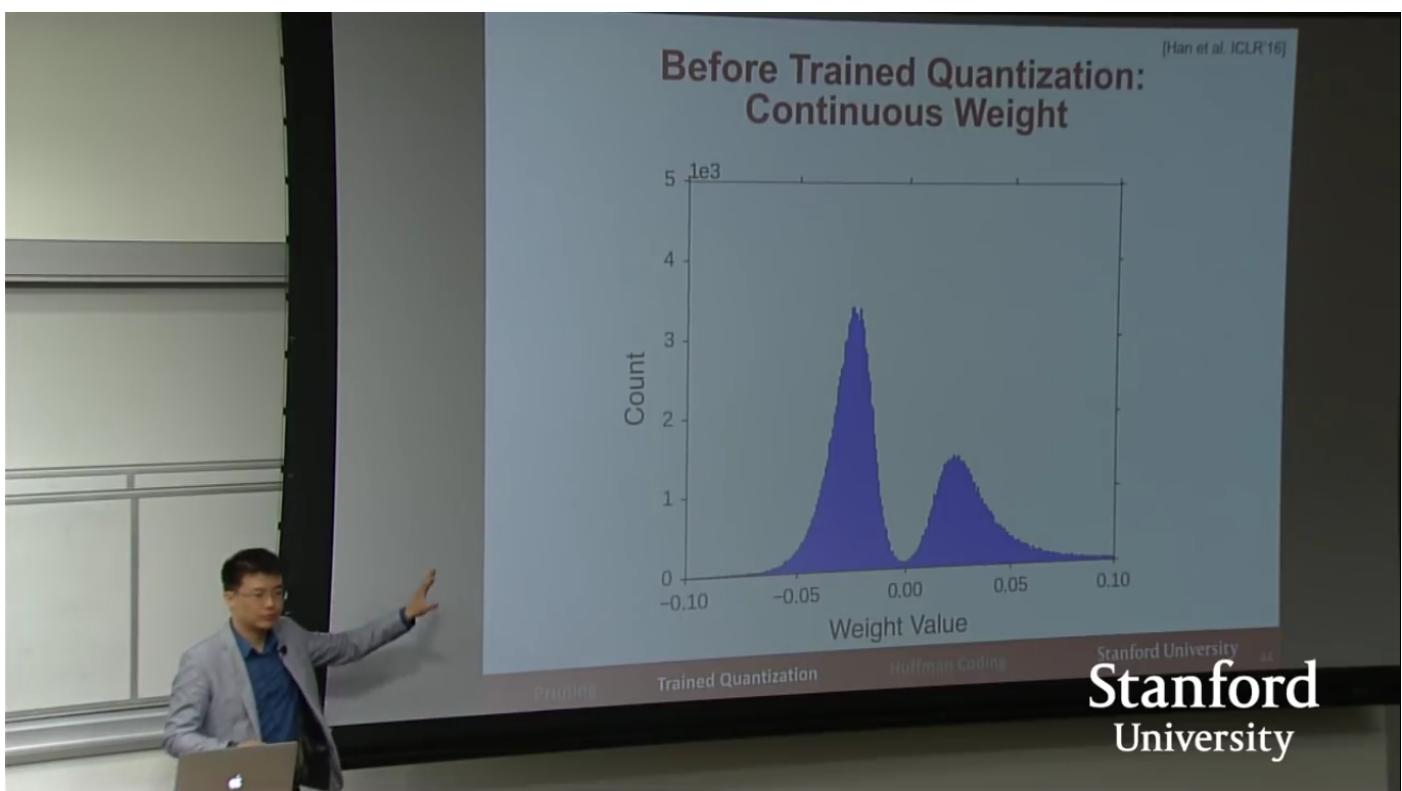
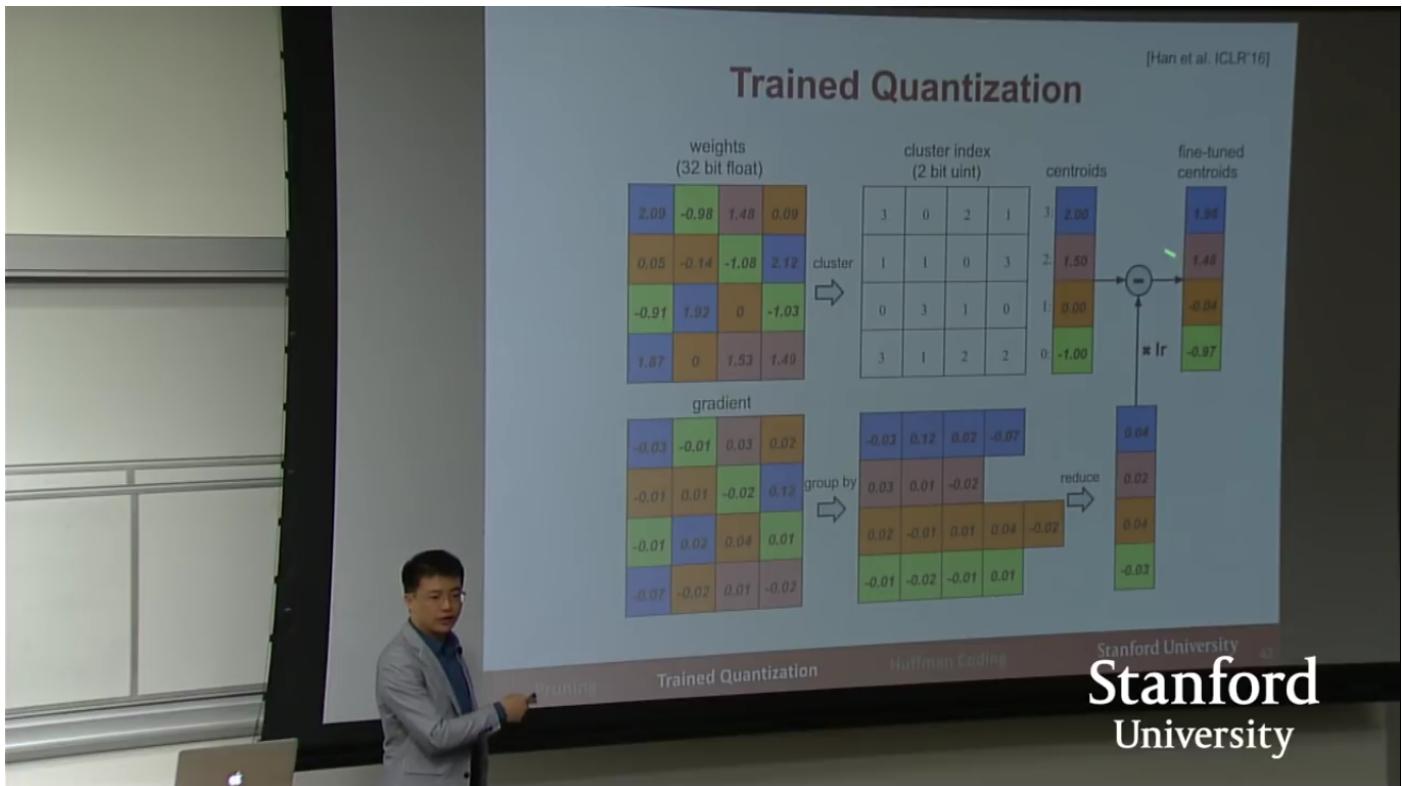
- Not all weights need to have the exact value. We can use a single number to represent them. Too accurate a number just leads to overfitting.
- So we can use a centroid value to represent them instead of using the full precision weight.
-

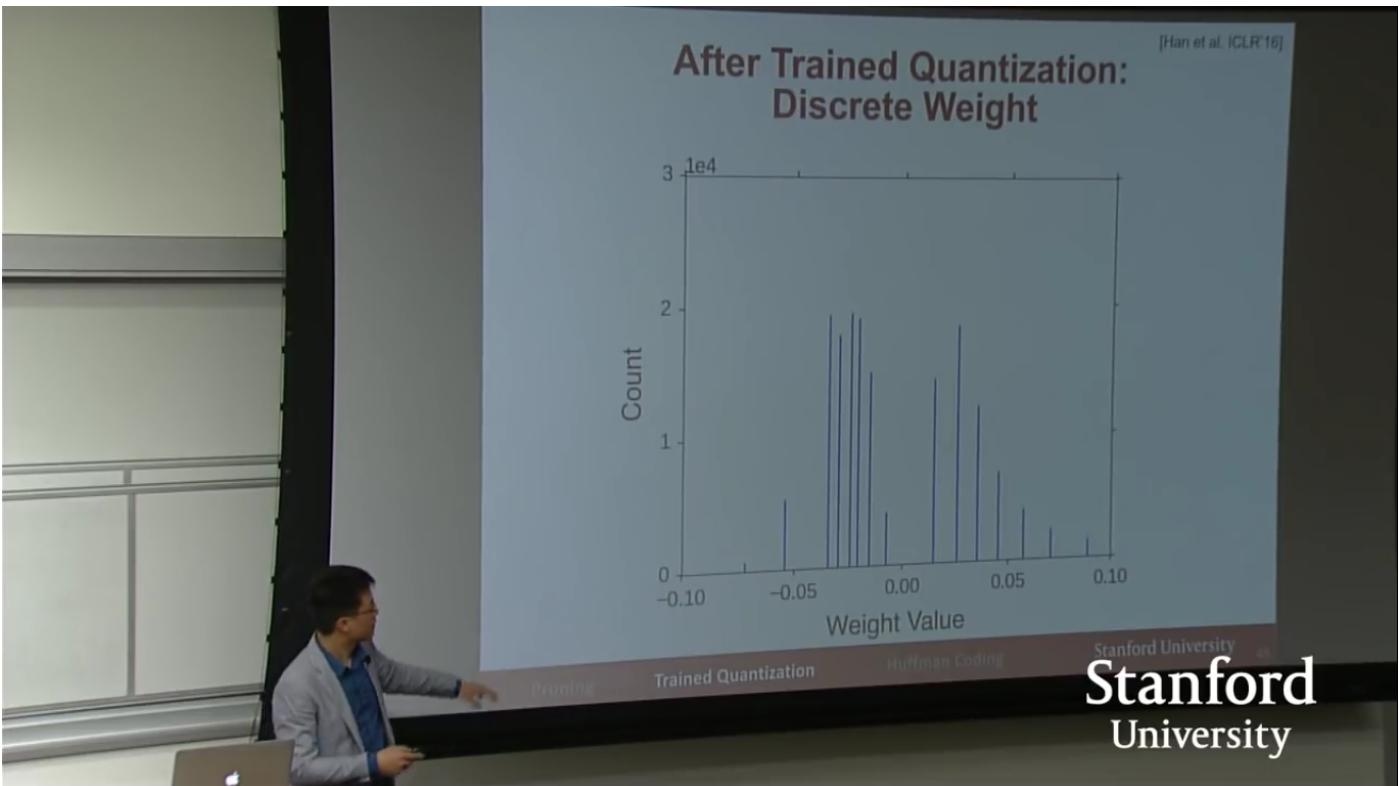


- Due to pruning, we will have less number of parameters, and due to less number of parameters we will need less number of bits to represent those parameters.

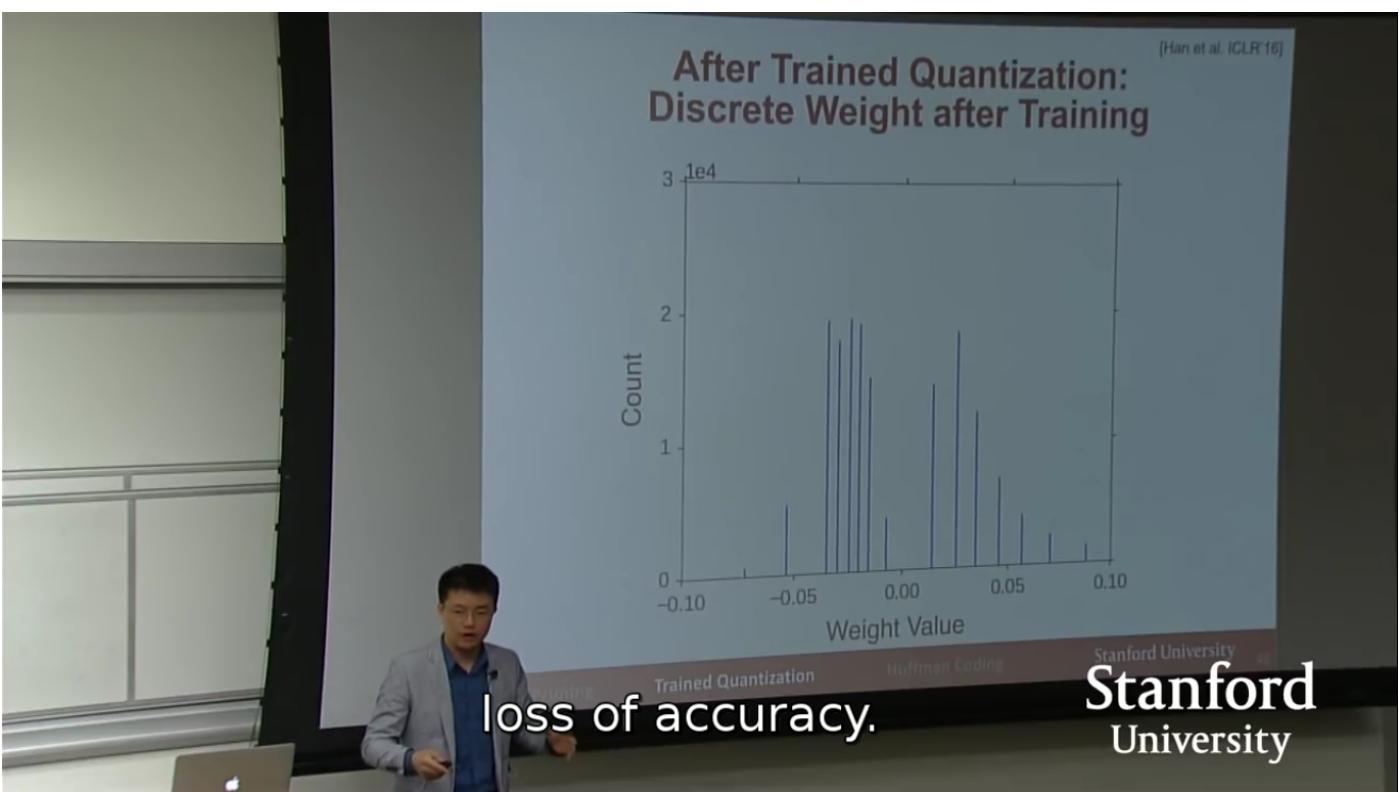


- Instead of using the 32 bit float, we are now using only 2 bit to represent the centroid values.
- But how do we train such shared weighted networks?

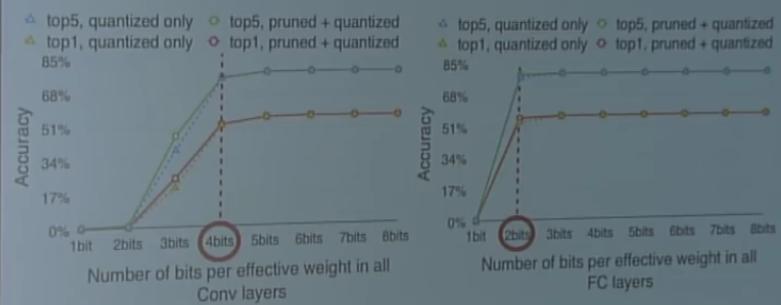




- Before we had Continuous weights and now we have Discrete weights.



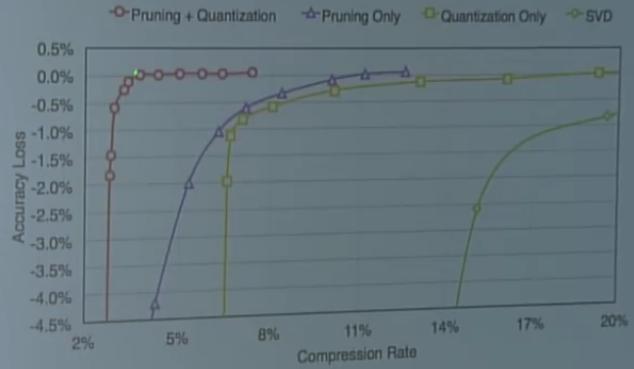
## How Many Bits do We Need?



A man in a grey suit is standing at a podium, pointing at the screen with a remote control.

Stanford University

## Pruning + Trained Quantization Work Together



A man in a grey suit is standing at a podium, pointing at the screen with a remote control.

Stanford University

- Compression Rate(Lower the Better). We can notice that when the model is made 3% of its original size, the Accuracy Loss is not degrading.

[Han et al. ICLR'16]

## Huffman Coding

Huffman Encoding

- Encode Weights
- Encode Index

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

Pruning Trained Quantization Huffman Coding Stanford University

Stanford University

[Han et al. ICLR'16]

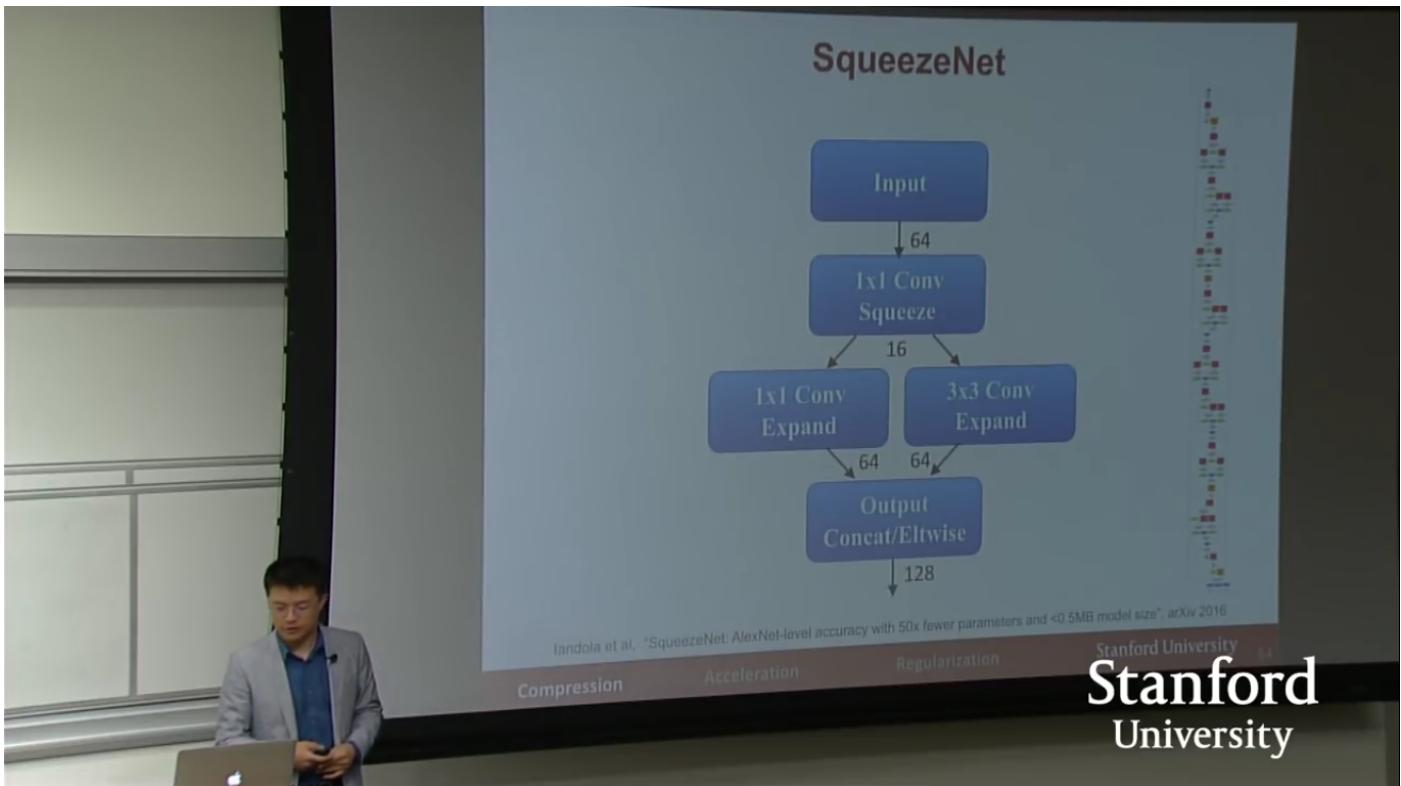
## Results: Compression Ratio

Network	Original Size	Compressed Size	Compression Ratio	Original Accuracy	Compressed Accuracy
LeNet-300	1070KB	→ 27KB	40x	98.36%	→ 98.42%
LeNet-5	1720KB	→ 44KB	39x	99.20%	→ 99.26%
AlexNet	240MB	→ 6.9MB	35x	80.27%	→ 80.30%
VGGNet	550MB	→ 11.3MB	49x	88.68%	→ 89.09%
GoogleNet	28MB	→ 2.8MB	10x	88.90%	→ 88.92%
ResNet-18	44.6MB	→ 4.0MB	11x	89.24%	→ 89.28%

Can we make compact models to begin with?

Compression Acceleration Regularization Stanford University

Stanford University



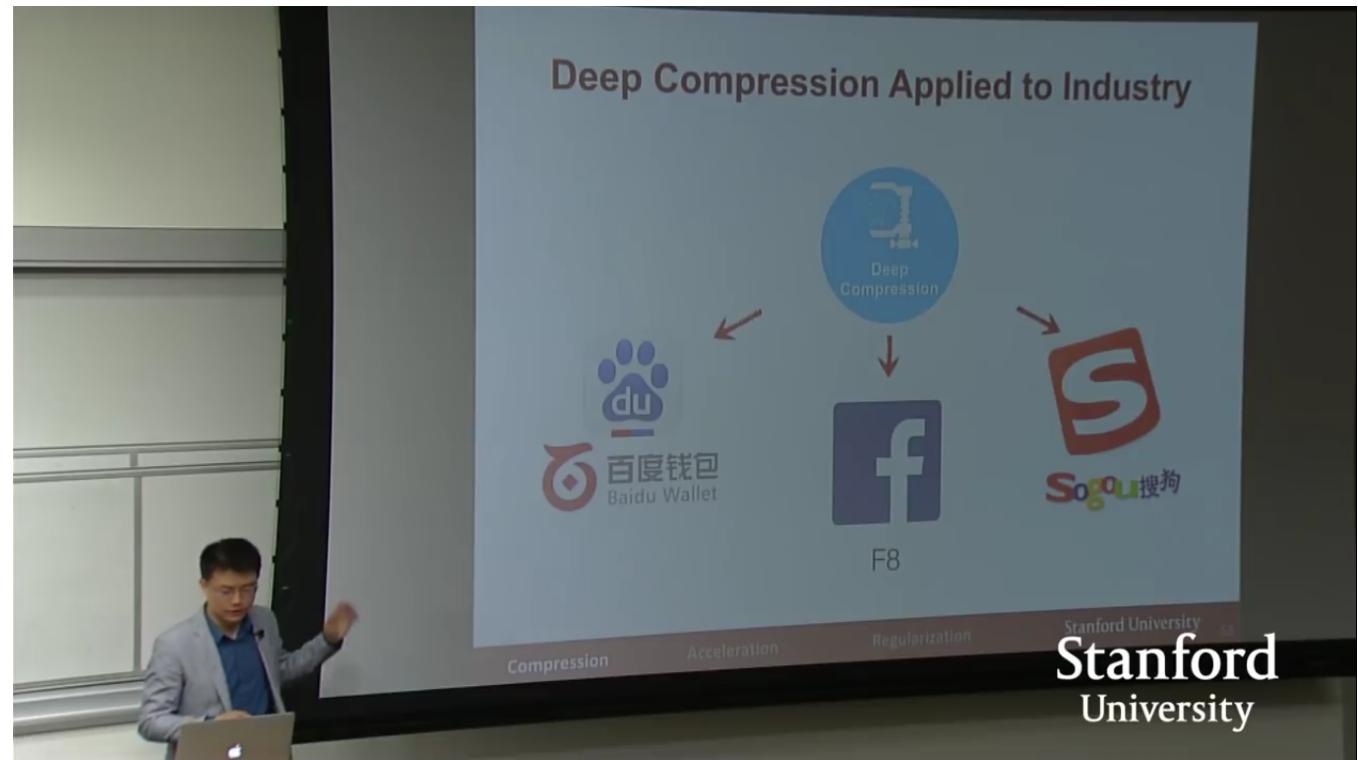
**Compressing SqueezeNet**

Network	Approach	Size	Ratio	Top-1 Accuracy	Top-5 Accuracy
AlexNet	-	240MB	1x	57.2%	80.3%
AlexNet	SVD	48MB	5x	56.0%	79.4%
AlexNet	Deep Compression	6.9MB	35x	57.2%	80.3%
SqueezeNet	-	4.8MB	50x	57.5%	80.3%
SqueezeNet	Deep Compression	0.47MB	510x	57.5%	80.3%

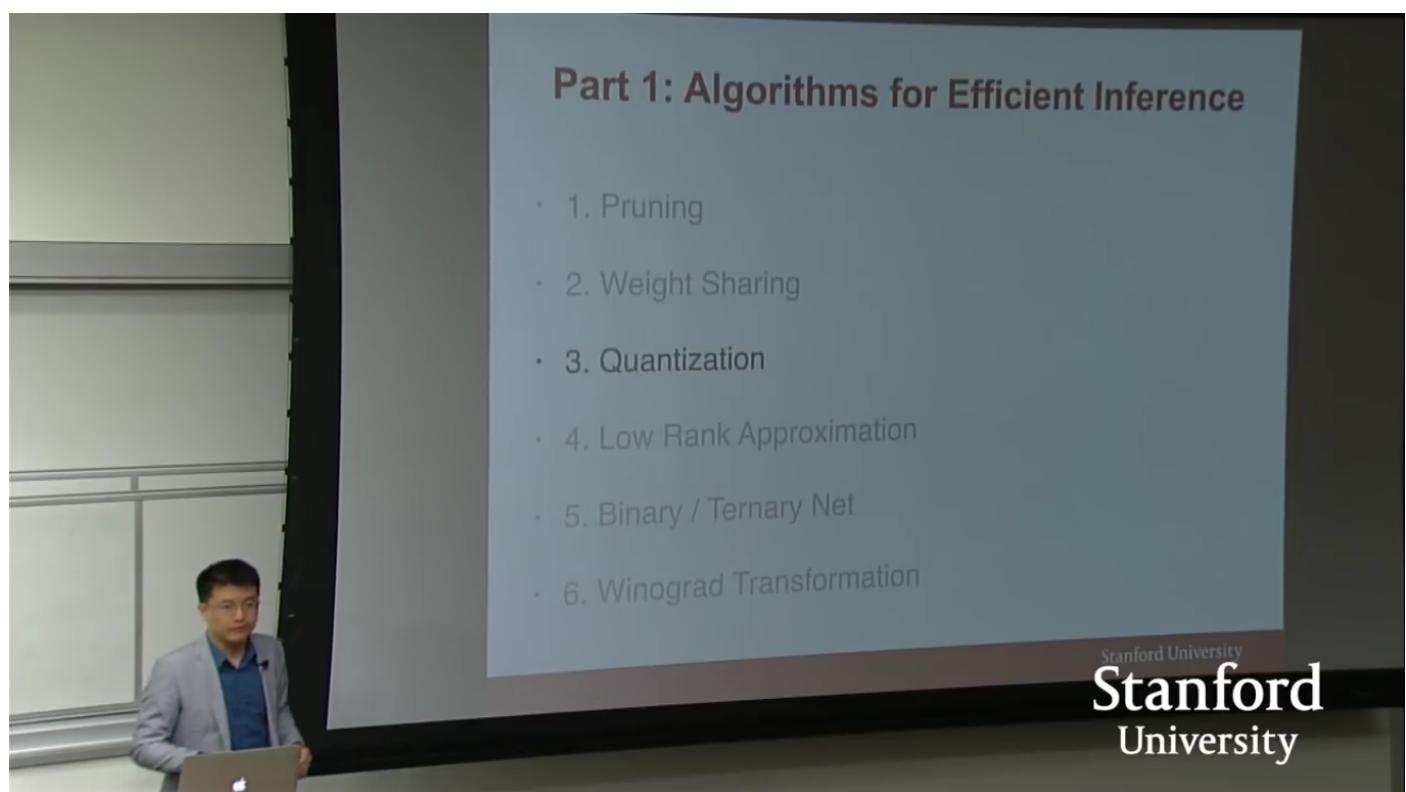
Iandola et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", arXiv 2016

Compression      Acceleration      Regularization      Stanford University

- SqueezeNet is already a small sized model and now after compression it is reducing to 0.47MB and still has the same Top-1 Accuracy as AlexNet. Due to this very small size it can even be placed in cache of SRAMs.



## Quantization:



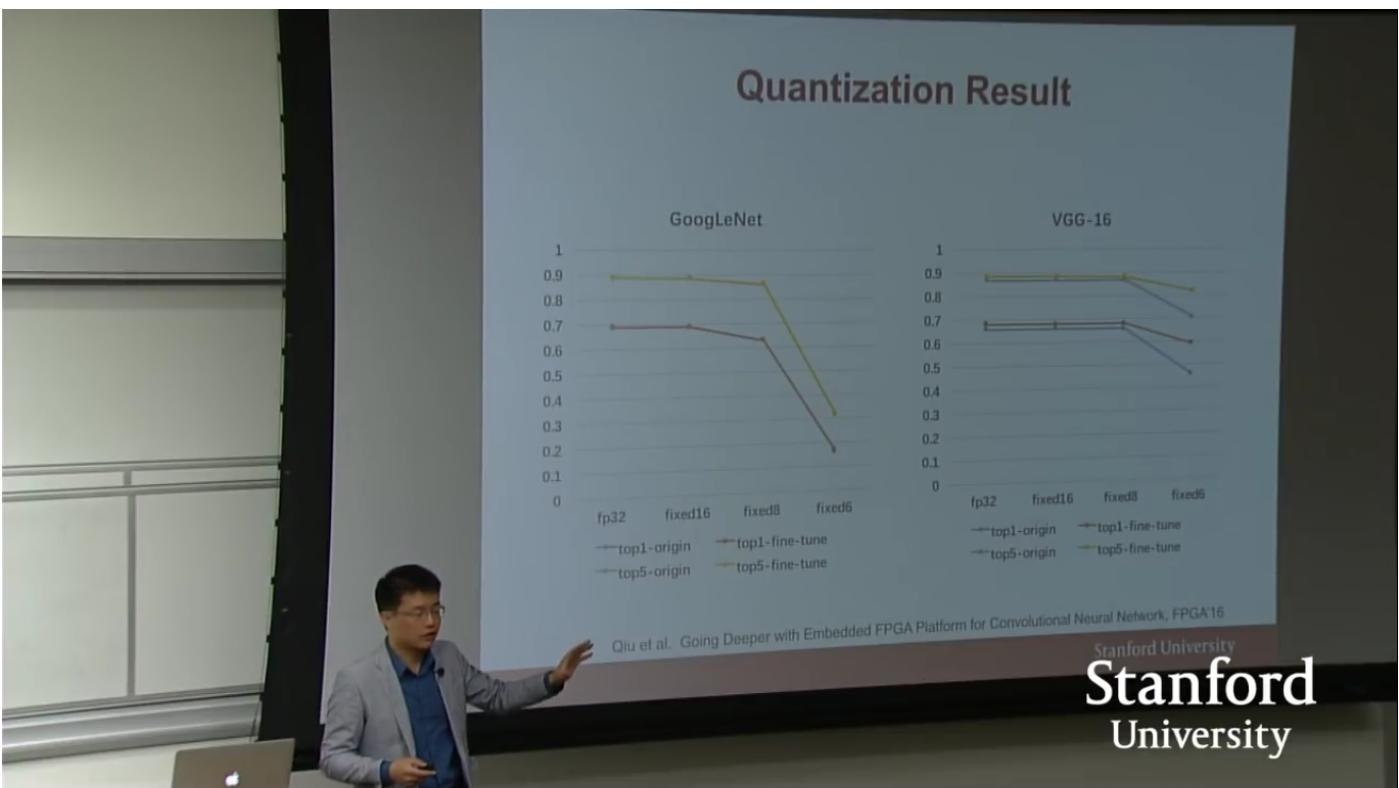
- TPUs widely use Quantization.
- In TPUs just 8 bits during inference.

**Quantizing the Weight and Activation**

- Train with float
- Quantizing the weight and activation:
  - Gather the statistics for weight and activation
  - Choose proper radix point position
- Fine-tune in float format
- Convert to fixed-point format

Qiu et al., Going Deeper with Embedded FPGA Platform for Convolutional Neural Network, FPGA'16

Stanford University



**Low Rank Approximation:**

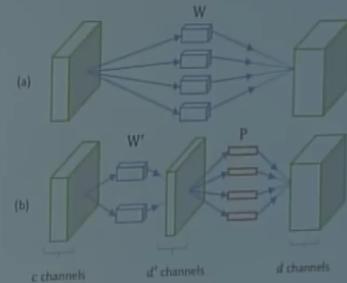
## Part 1: Algorithms for Efficient Inference

- 1. Pruning
- 2. Weight Sharing
- 3. Quantization
- 4. Low Rank Approximation
- 5. Binary / Ternary Net
- 6. Winograd Transformation

Stanford University  
**Stanford**  
University

## Low Rank Approximation for Conv

- 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 ( $1 \times 1 \times d'$ )



Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

Stanford University  
**Stanford**  
University

- We are breaking a CONV layer into 2 CONV layers

## Low Rank Approximation for Conv

speedup	rank sel.	Conv1	Conv2	Conv3	Conv4	Conv5	Conv6	Conv7	err. ↑ %
2×	no	32	110	199	219	219	219	219	1.18
2×	yes	32	83	182	211	239	237	253	0.93
2.4×	no	32	96	174	191	191	191	191	1.77
2.4×	yes	32	74	162	187	207	205	219	1.35
3×	no	32	77	139	153	153	153	153	2.56
3×	yes	32	62	138	149	166	162	167	2.34
4×	no	32	57	104	115	115	115	115	4.32
4×	yes	32	50	112	114	122	117	119	4.20
5×	no	32	46	83	92	92	92	90	6.53
5×	yes	32	41	94	93	98	92	90	6.47

Zhang et al Efficient and Accurate Approximations of Nonlinear Convolutional Networks CVPR'15

Stanford University

## Low Rank Approximation for FC

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

TT-format for matrix  $W$ :

$$W(i_1, \dots, i_d; j_1, \dots, j_d) = W(x(i), y(j)) = \underbrace{G_1[i_1, j_1]}_{1 \times r} \underbrace{G_2[i_2, j_2]}_{r \times r} \dots \underbrace{G_d[i_d, j_d]}_{r \times 1}$$

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

Novikov et al Tensorizing Neural Networks, NIPS'15

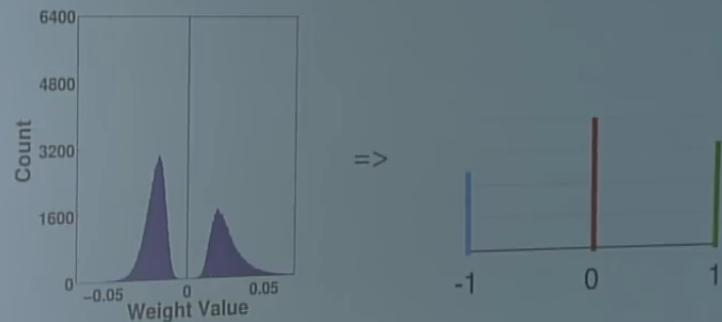
Stanford University

- We are using Tensor Tree to break down one fully connected layer into a tree of fully connected layers.

####Binary and Ternary Net:

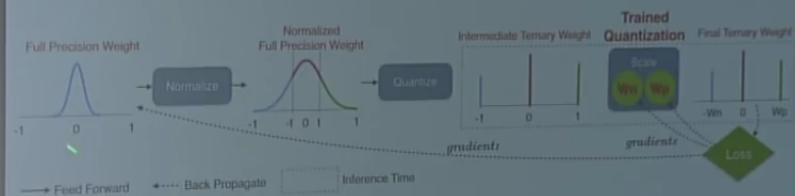
- Can we use only 2 or 3 weights to represent the Neural Network?

## Binary / Ternary Net: Motivation



Stanford University

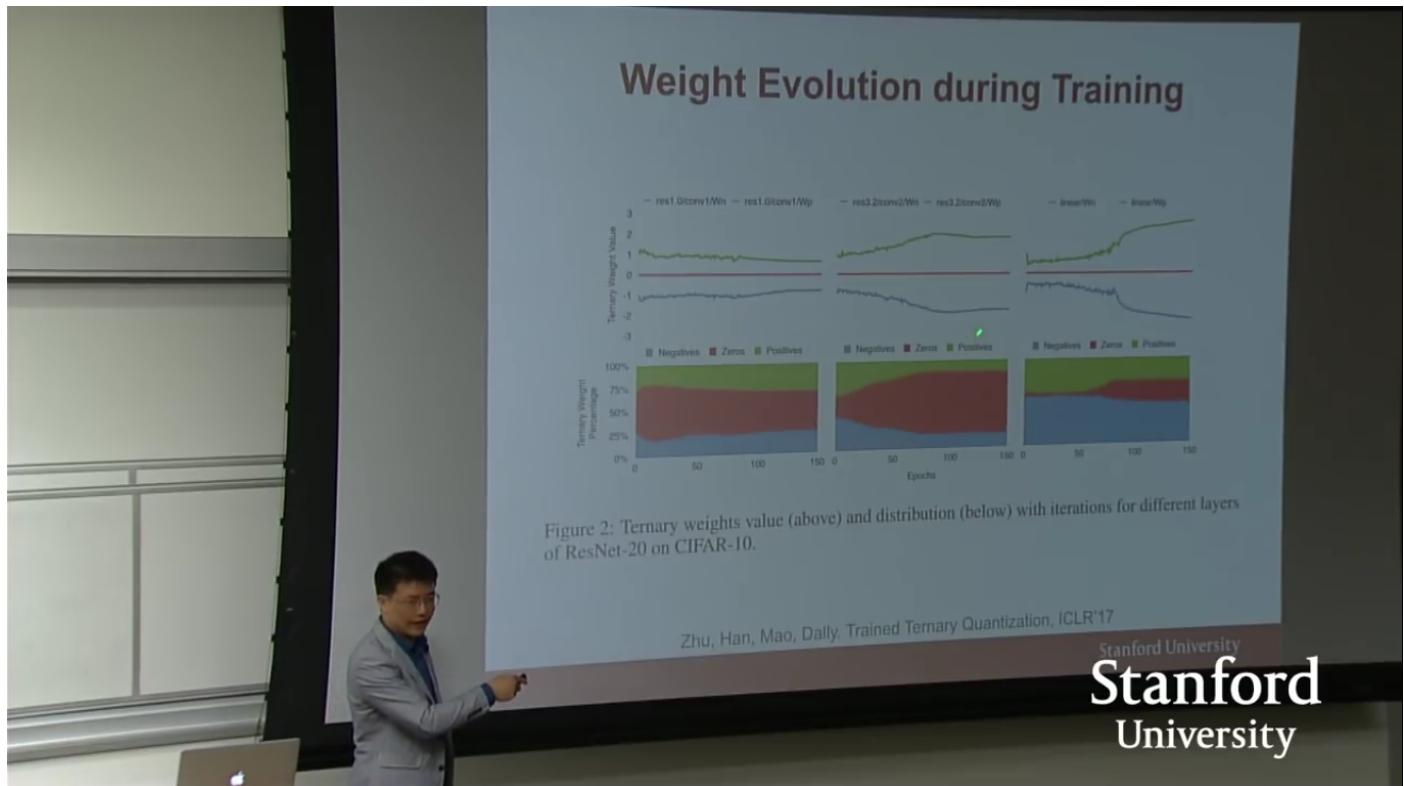
## Trained Ternary Quantization



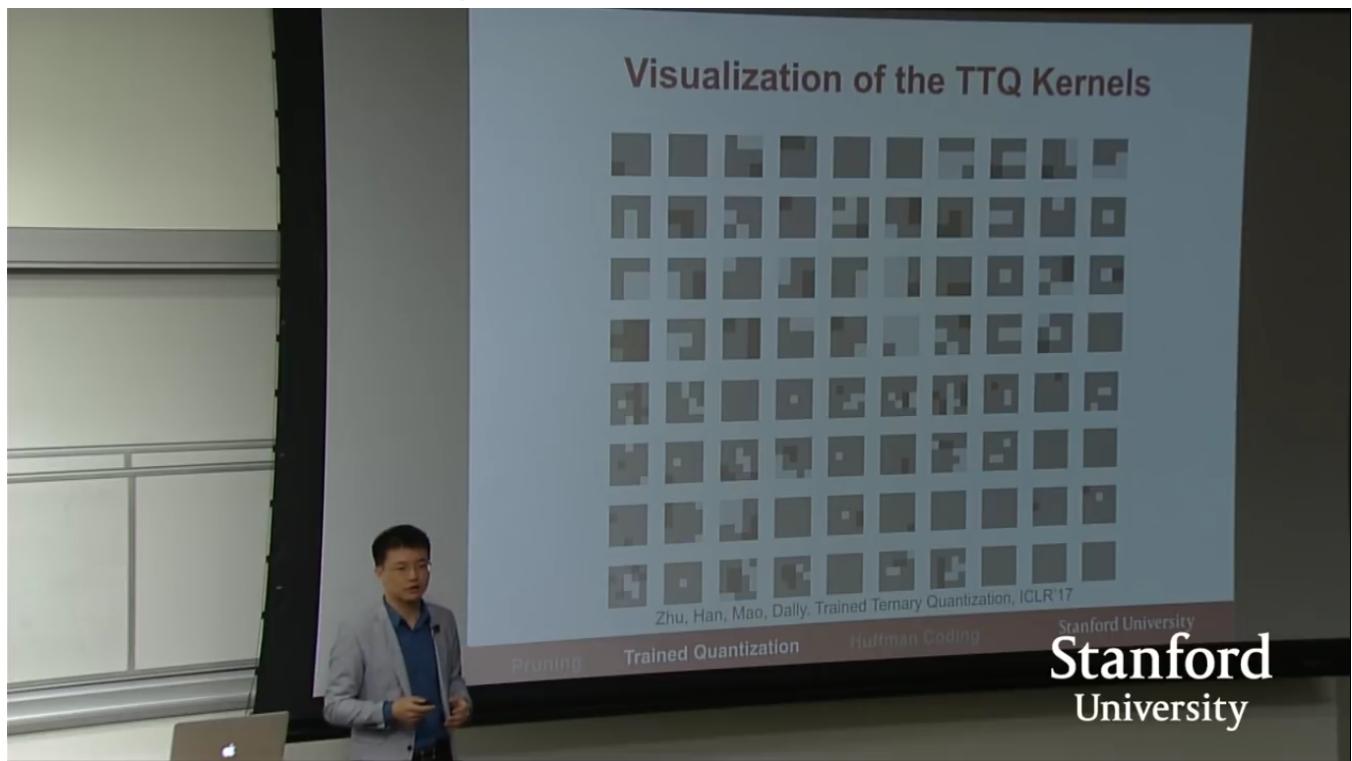
Zhu, Han, Mao, Daily, Trained Ternary Quantization, ICLR'17

Pruning      Trained Quantization      Huffman Coding

Stanford University



- Visualization of the Trained Ternary Quantization



- We can find many corner detection and edge detection kernels being learnt.

### Winograd Transformation:

**3x3 DIRECT Convolutions**

Compute Bound

Image Tensor

0	1	2
3	4	5
6	7	8

Filter

8	7	6
5	4	3
2	1	0

9xC FMAs/Output: Math Intensive

$$[8 \times 8 + 1 \times 7 + 7 \times 6 + 3 \times 1 + 5 \times 4 + 4 \times 3 + 2 \times 0 + 7 \times 1 + 6 \times 0] \times 4$$

9xK FMAs/Input: Good Data Reuse

8	0	7	1	6	3	2	4	5
5	3	4	2	1	0	7	6	8
2	6	7	8	5	4	3	0	1

Direct convolution: we need  $9 \times C \times 4 = 36 \times C$  FMAs for 4 outputs

Julien Demouth, Convolution OPTIMIZATION: Winograd, NVIDIA

Stanford University

**3x3 WINOGRAD Convolutions**

Transform Data to Reduce Math Intensity

4x4 Tile

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Filter

0	1	2
4	5	6
8	9	10

Data Transform

Filter Transform

$\sum$  over C

Point-wise multiplication

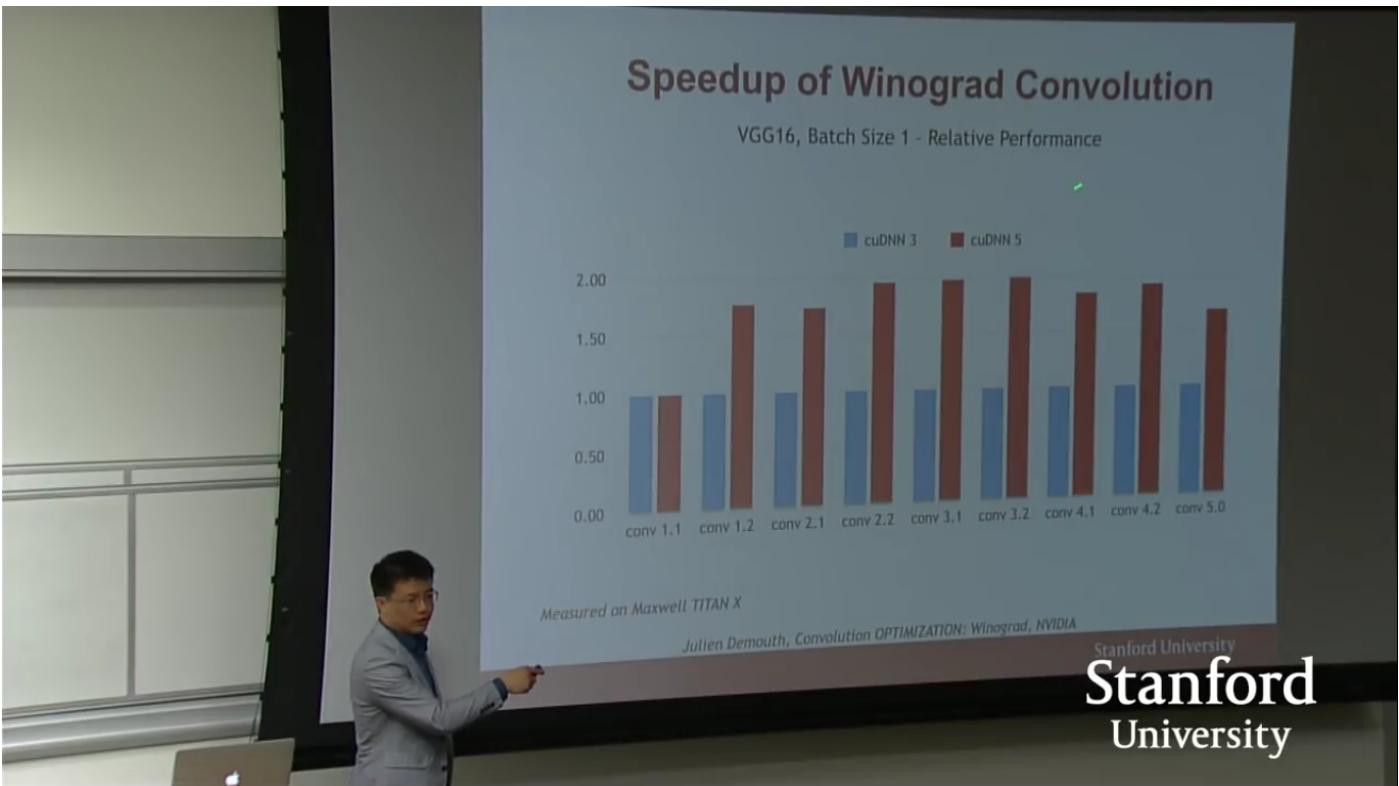
Output Transform

Direct convolution: we need  $9 \times C \times 4 = 36 \times C$  FMAs for 4 outputs  
Winograd convolution: we need  $16 \times C$  FMAs for 4 outputs: **2.25x** fewer FMAs

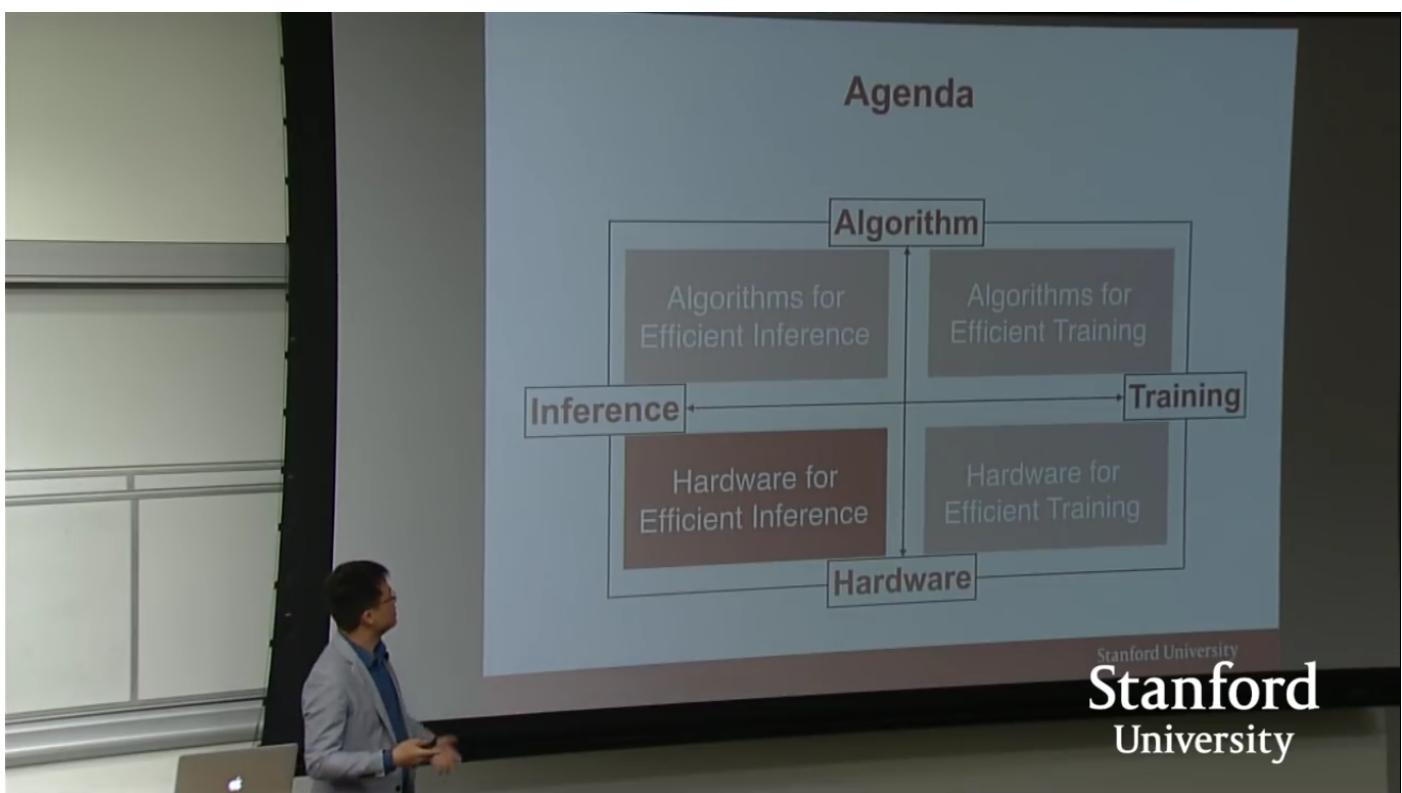
See A. Lavin & S. Gray, "Fast Algorithms for Convolutional Neural Networks"  
Julien Demouth, Convolution OPTIMIZATION: Winograd, NVIDIA

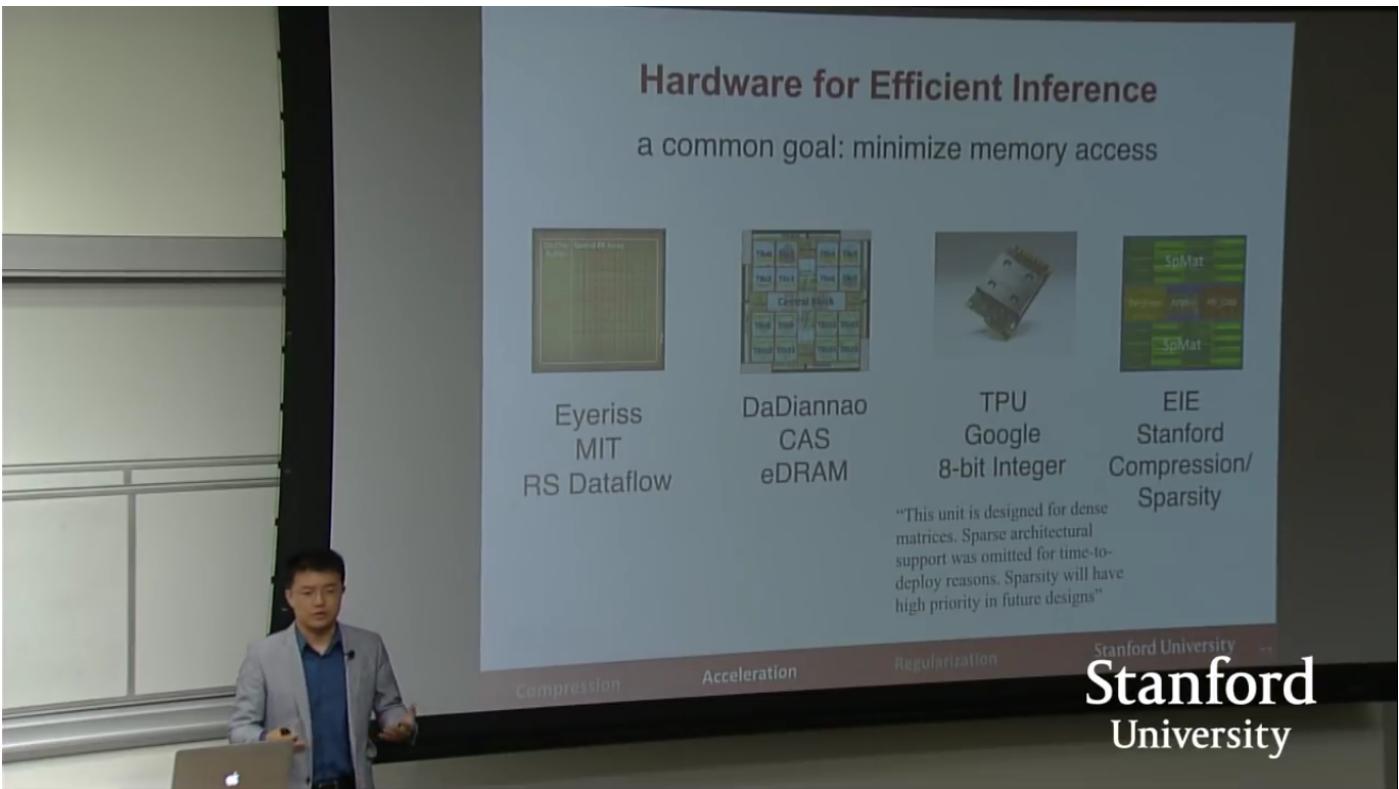
Stanford University

- This is not a lossy method, this is equivalent to the normal CONV operation.

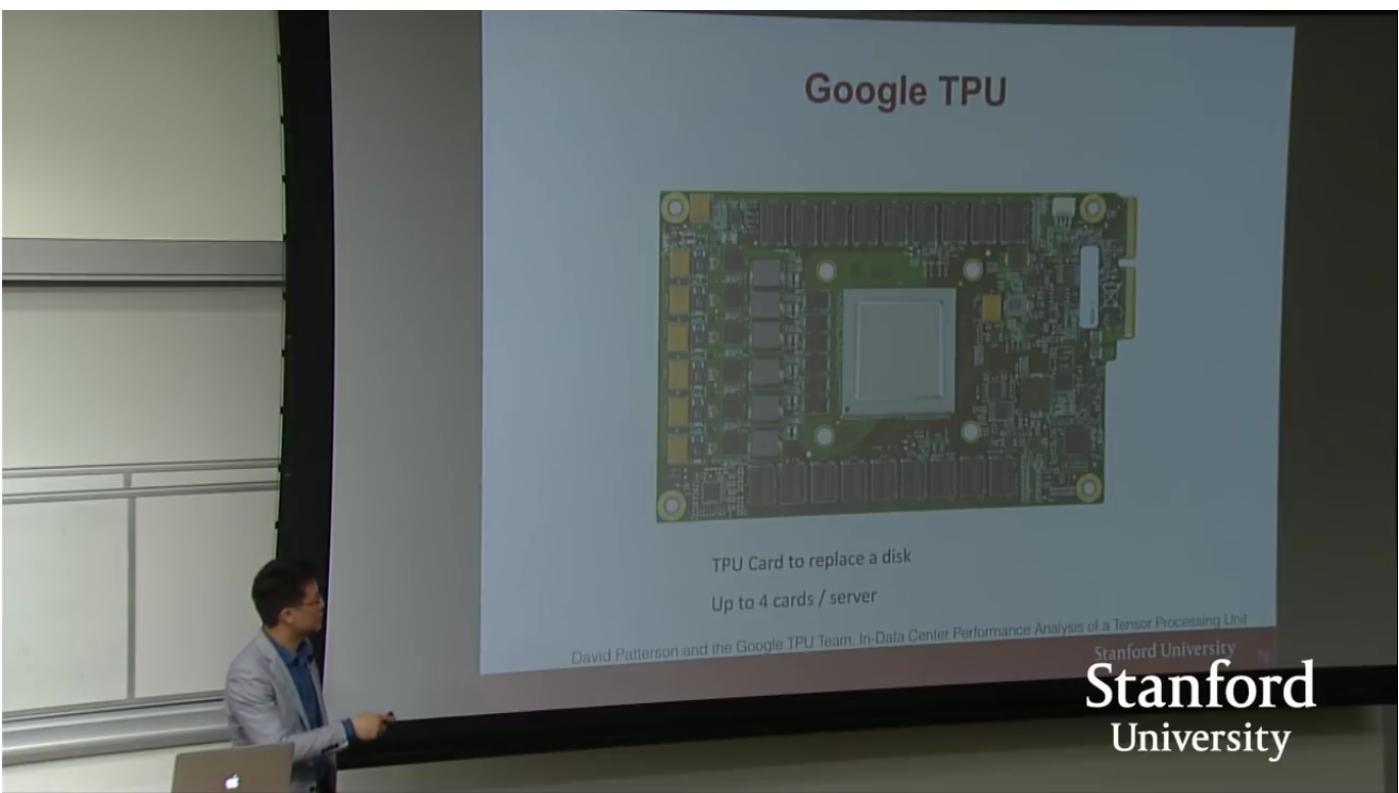


Optimal hardware for efficient inference:





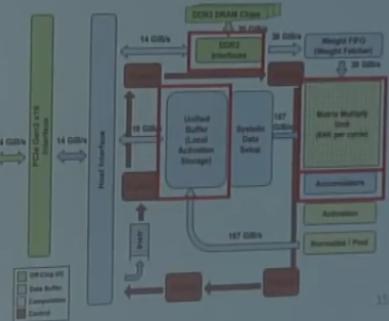
- The common goal is to minimize memory access.



## Google TPU

- The Matrix Unit: 65,536 (256x256)  
8-bit multiply-accumulate units
- 700 MHz clock rate
- Peak: 92T operations/second
  - $65,536 * 2 * 700M$
- >25X as many MACs vs GPU
- >100X as many MACs vs CPU
- 4 MiB of on-chip Accumulator memory
- 24 MiB of on-chip Unified Buffer, (activation memory)
- 3.5X as much on-chip memory vs GPU
- Two 2133MHz DDR3 DRAM channels
- 8 GiB of off-chip weight DRAM memory

### TPU: High-level Chip Architecture



David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

Stanford University  
University

- In 1 cycle the TPU can perform 65536 multiplication - accumulation operation.

## Google TPU

Processor	$mm^2$	Clock MHz	TDP Watts	Idle Watts	Memory GB/sec	Peak TOPS/chip	
						8b int.	32b FP
CPU: Haswell (18 core)	662	2300	145	41	51	2.6	1.3
GPU: Nvidia K80 (2 / card)	561	560	150	25	160	--	2.8
TPU	<331*	700	75	28	34	91.8	-

\*TPU is less than half die size of the Intel Haswell processor

K80 and TPU in 28 nm process; Haswell fabbed in Intel 22 nm process

These chips and platforms chosen for comparison because widely deployed in Google data centers

David Patterson and the Google TPU Team, In-Data Center Performance Analysis of a Tensor Processing Unit

Stanford University  
University

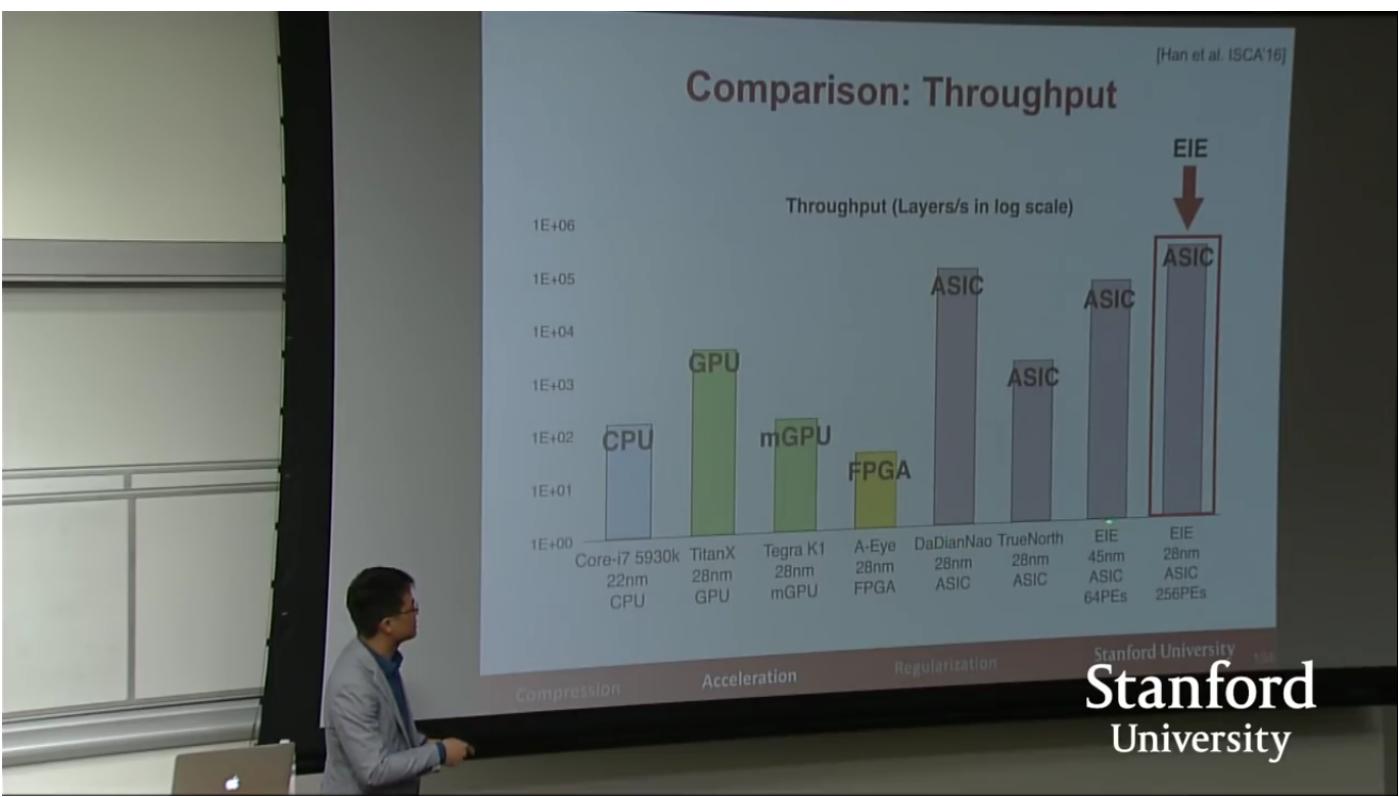
### Inference Datacenter Workload

Name	LOC	Layers				Nonlinear function	Weights	TPU Ops / Weight Byte	TPU Batch Size	% Deployed
		FC	Conv	Vector	Pool					
MLP0	0.1k	5			5	ReLU	20M	200	200	61%
MLP1	1k	4			4	ReLU	5M	168	168	
LSTM0	1k	24		34	58	sigmoid, tanh	52M	64	64	29%
LSTM1	1.5k	37		19	56	sigmoid, tanh	34M	96	96	
CNN0	1k		16		16	ReLU	8M	2888	8	5%
CNN1	1k	4	72		13	ReLU	100M	1750	32	

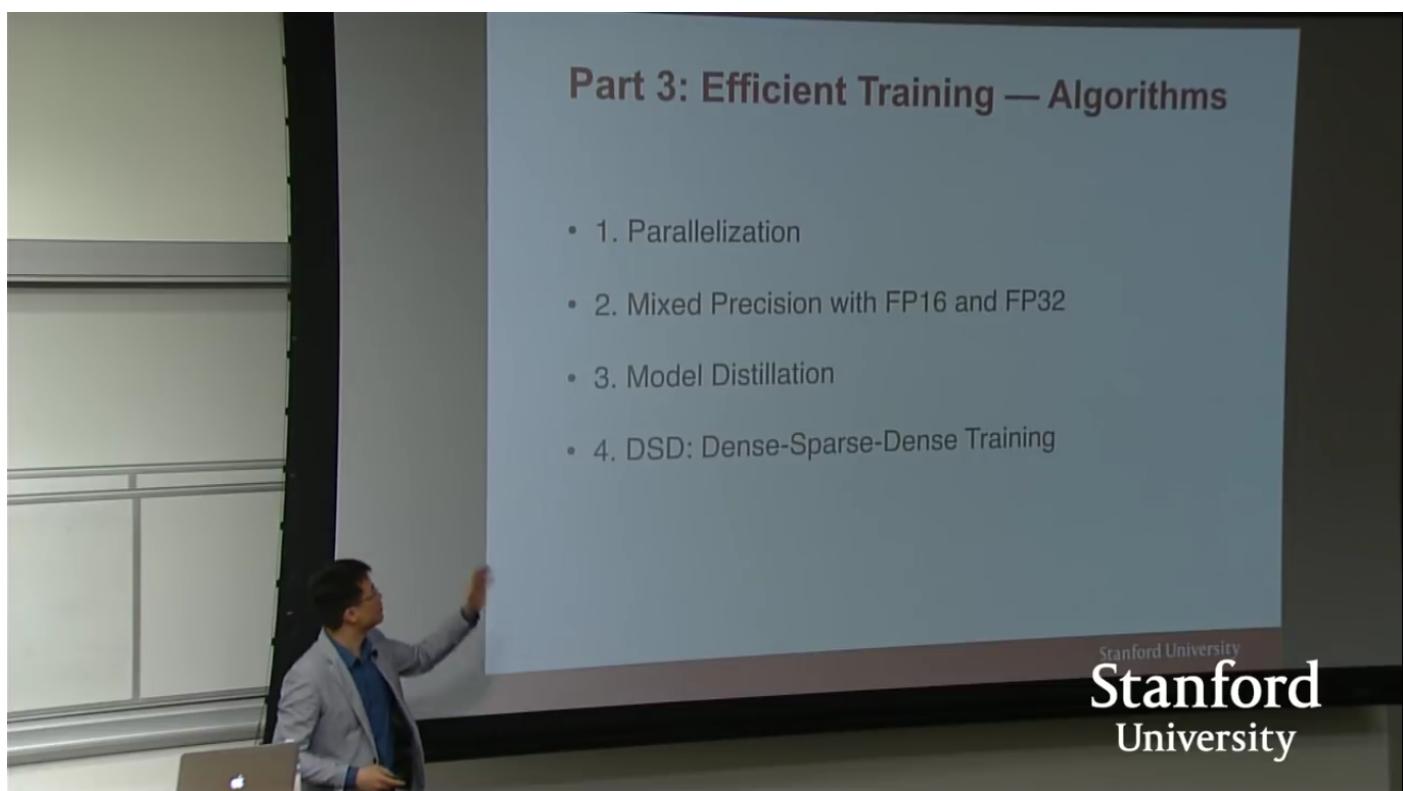
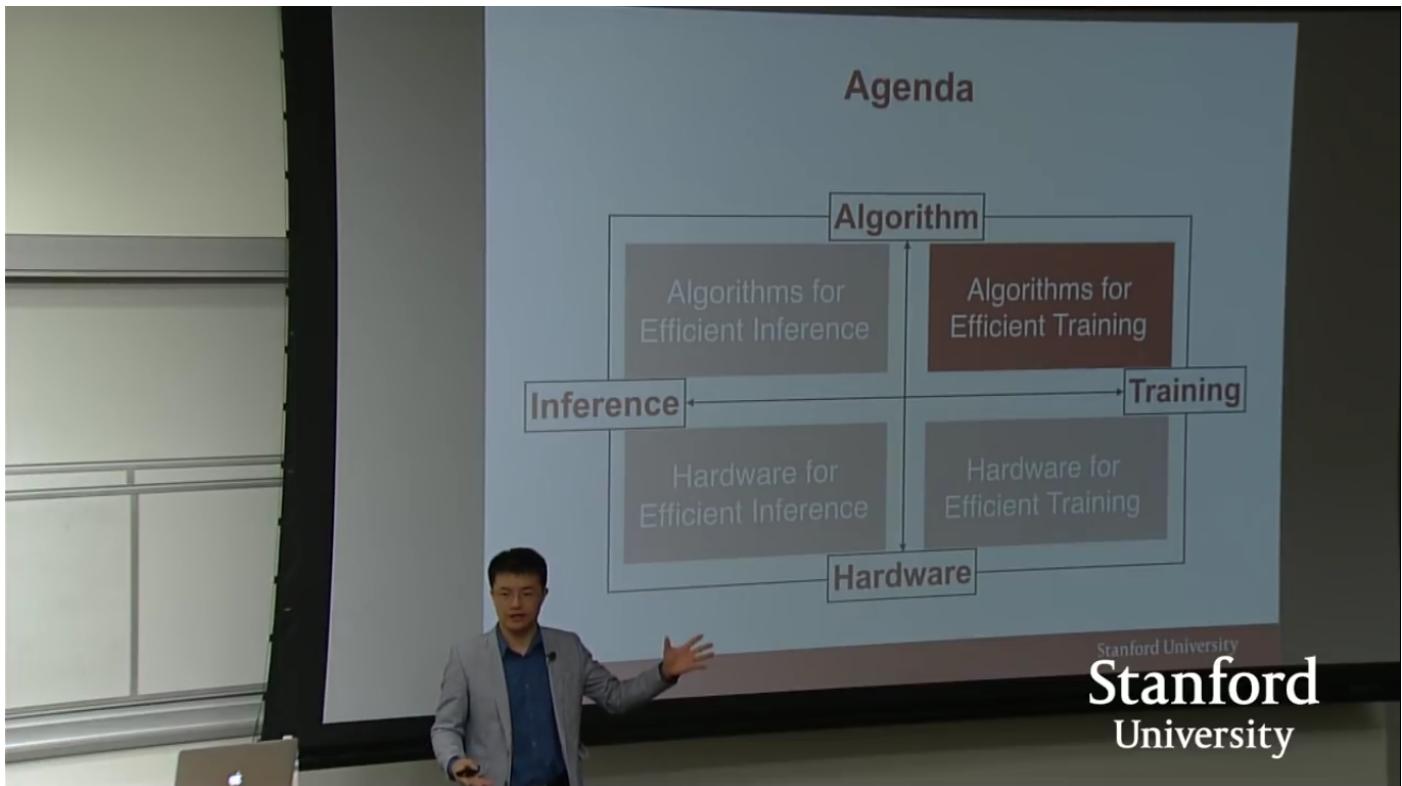
David Patterson and the Google TPU Team. In-Data Center Performance Analysis of a Tensor Processing Unit

Stanford University

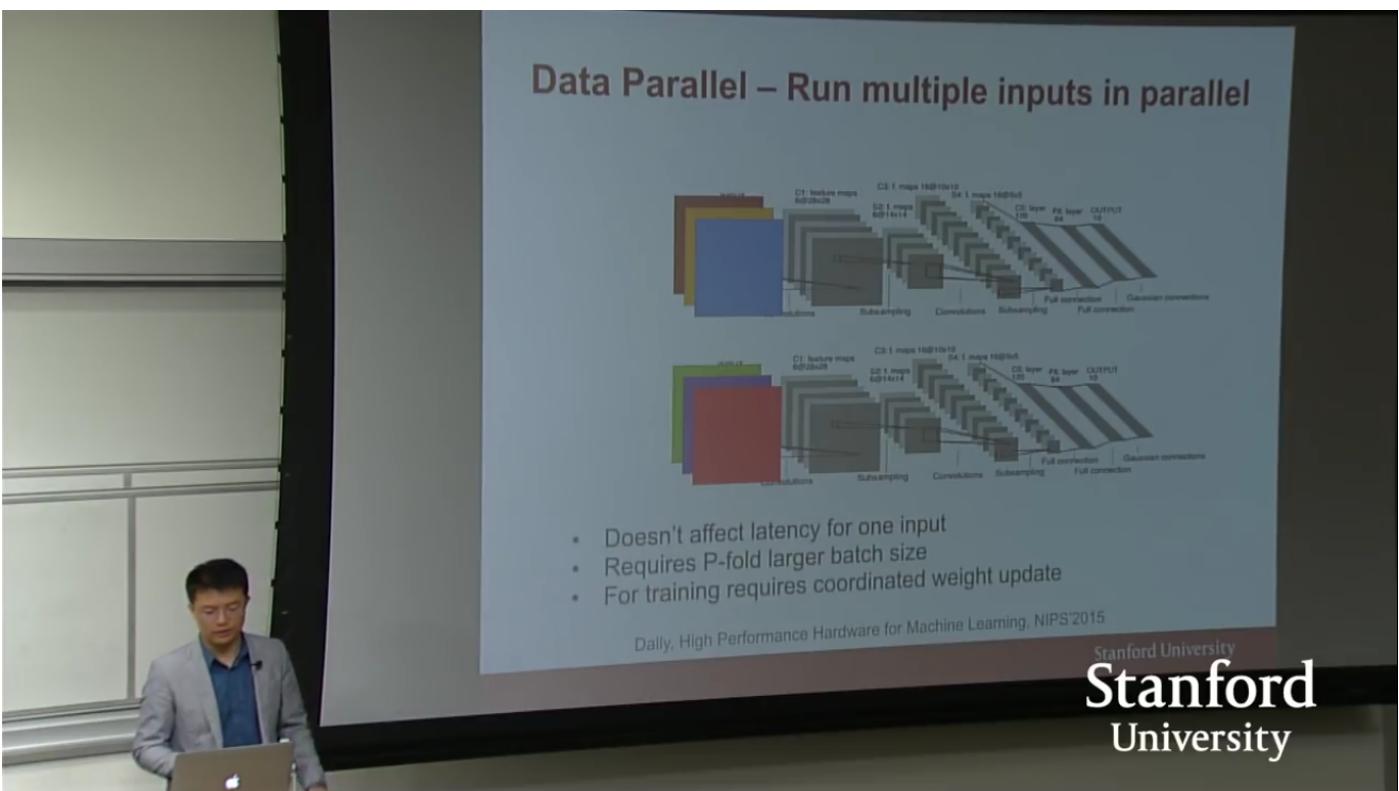
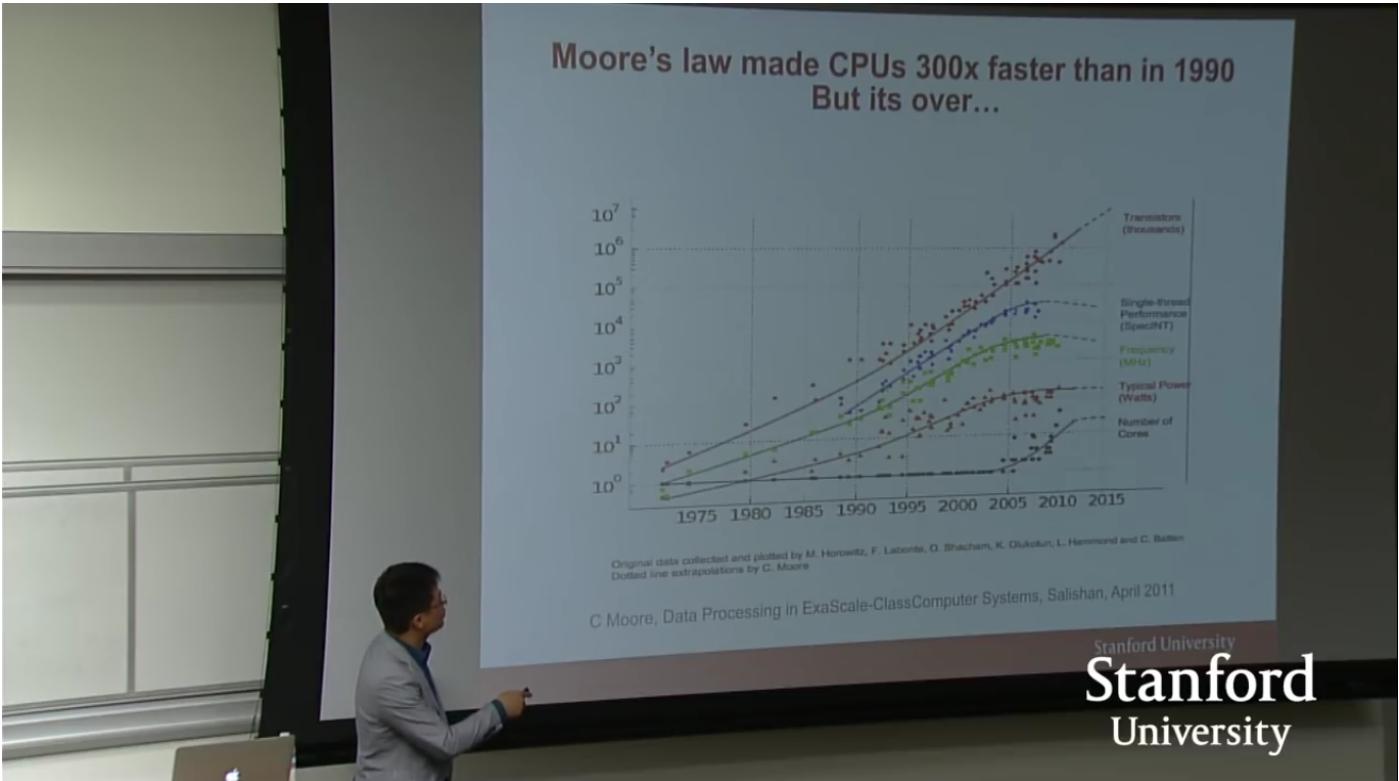
- We can notice that only 5% is being used for CNN operations whereas 29% for LSTM (maybe for speech recognition), 61% for FC (maybe for ad recommendation).



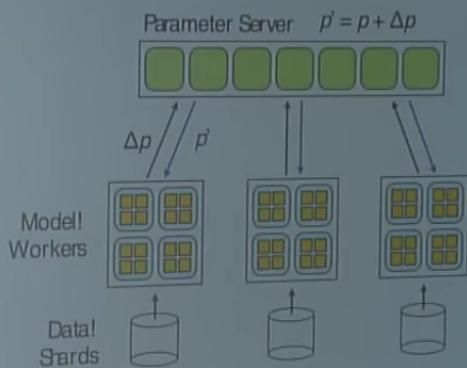
## Algorithms for Efficient Training:



**Parallelization:**



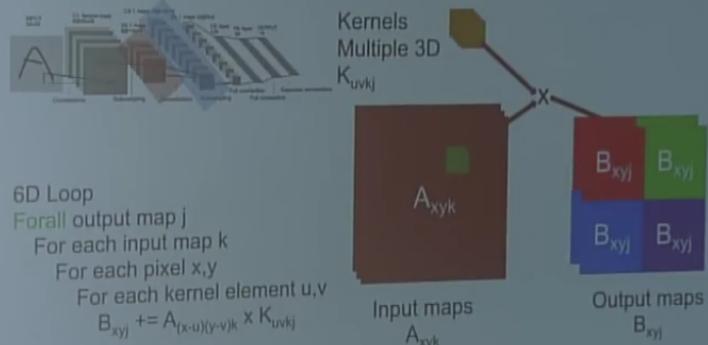
## Parameter Update



Large Scale Distributed Deep Networks, Jeff Dean et al., 2013

Stanford University

## Model-Parallel Convolution – by output region (x,y)



Dally, High Performance Hardware for Machine Learning, NIPS'2015

Stanford University

**Model Parallel Fully-Connected Layer ( $M \times V$ )**

The diagram illustrates a neural network architecture with four layers: input layer, hidden layer 1, hidden layer 2, and output layer. The input layer has 4 nodes, hidden layer 1 has 2 nodes, hidden layer 2 has 3 nodes, and the output layer has 2 nodes. The hidden layers have vertical connections between their nodes, while the input and output layers are fully connected to all nodes in the adjacent layers.

The mathematical equation below the diagram represents the computation of output activations:

$$\begin{bmatrix} b_i \\ b_i \end{bmatrix} = \begin{bmatrix} W_{ij} \\ W_{ij} \end{bmatrix} X \begin{bmatrix} a_j \end{bmatrix}$$

Legend:  $b_i$  = Output activations;  $W_{ij}$  = weight matrix;  $a_j$  = Input activations

Dally, High Performance Hardware for Machine Learning, NIPS'2015

Stanford University

**Summary of Parallelism**

- Lots of parallelism in DNNs
  - 16M independent multiplies in one FC layer
  - Limited by overhead to exploit a fraction of this
- Data parallel
  - Run multiple training examples in parallel
  - Limited by batch size
- Model parallel
  - Split model over multiple processors
  - By layer
  - Conv layers by map region
  - Fully connected layers by output activation
- Easy to get 16-64 GPUs training one model in parallel

Dally, High Performance Hardware for Machine Learning, NIPS'2015

Stanford University

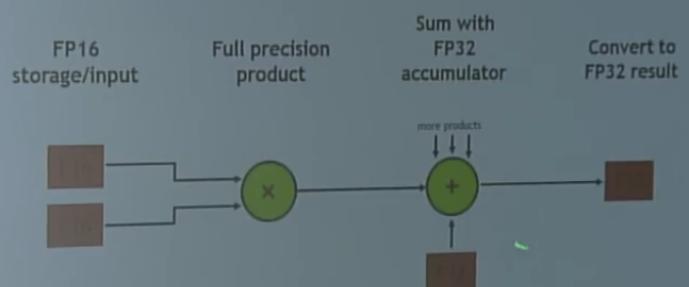
## Mixed Precision with FP-16 and FP-32

## Part 3: Efficient Training — Algorithms

- 1. Parallelization
- 2. Mixed Precision with FP16 and FP32
- 3. Model Distillation
- 4. DSD: Dense-Sparse-Dense Training

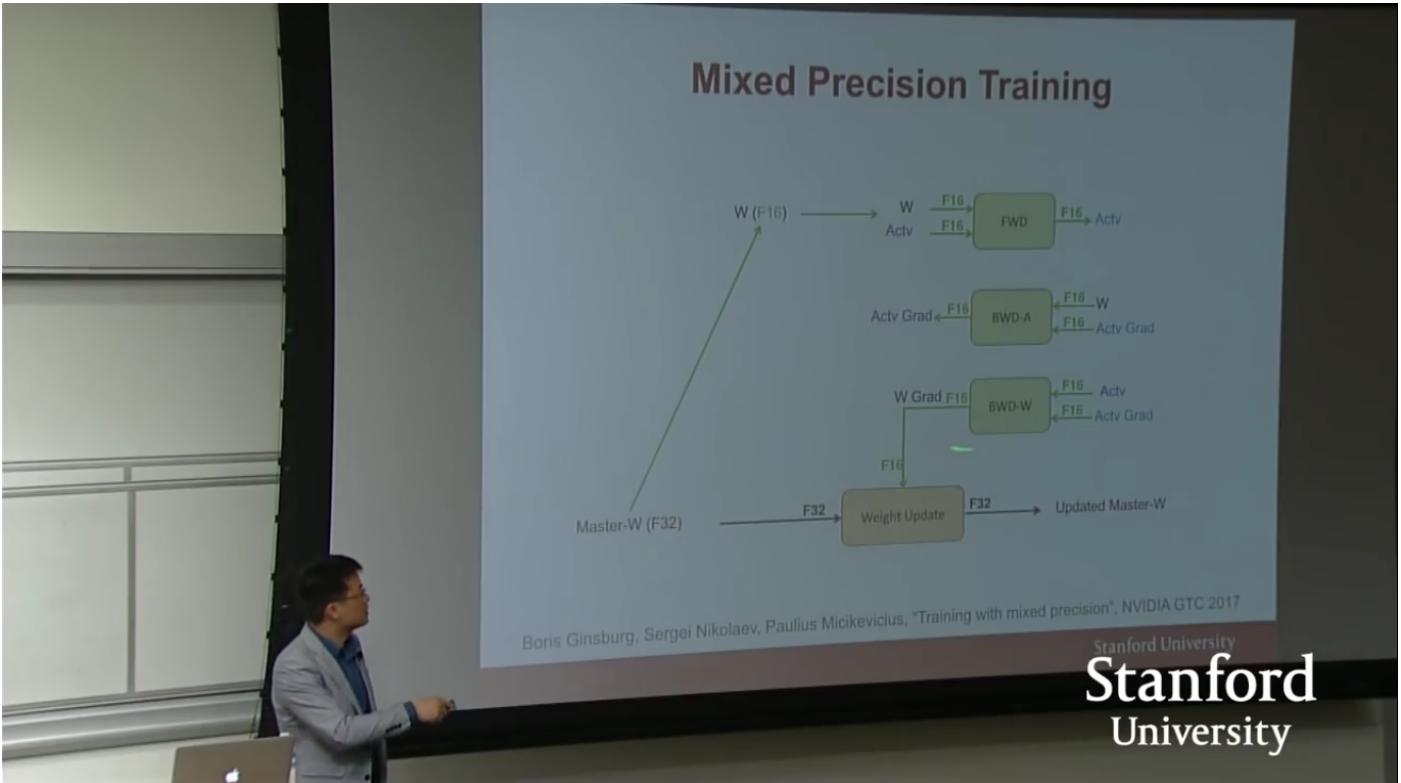
Stanford University  
**Stanford**  
University

### Mixed Precision

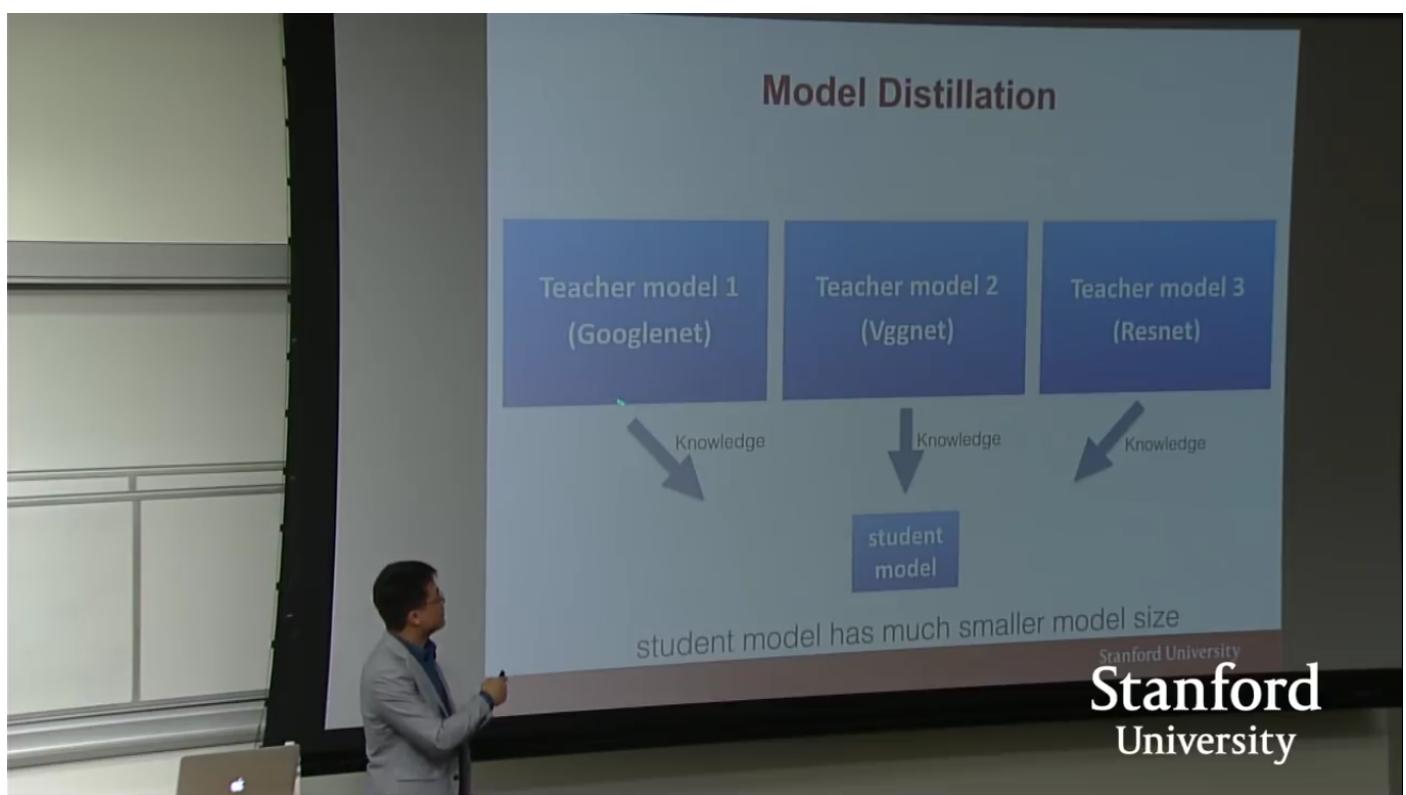


<https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/>

Stanford University  
**Stanford**  
University



### Model Distillation:



- The idea is to use better performing large parent networks and teach a small student network which is competitive to the parent networks.

**Softened outputs reveal the dark knowledge**

cow	dog	cat	car
0	1	0	0

original hard targets

cow	dog	cat	car
$10^{-6}$	.9	.1	$10^{-9}$

output of geometric ensemble

cow	dog	cat	car
.05	.3	.2	.005

softened output of ensemble

Hinton et al. Dark knowledge / Distilling the Knowledge in a Neural Network

Stanford University

**Softened outputs reveal the dark knowledge**

$$p_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

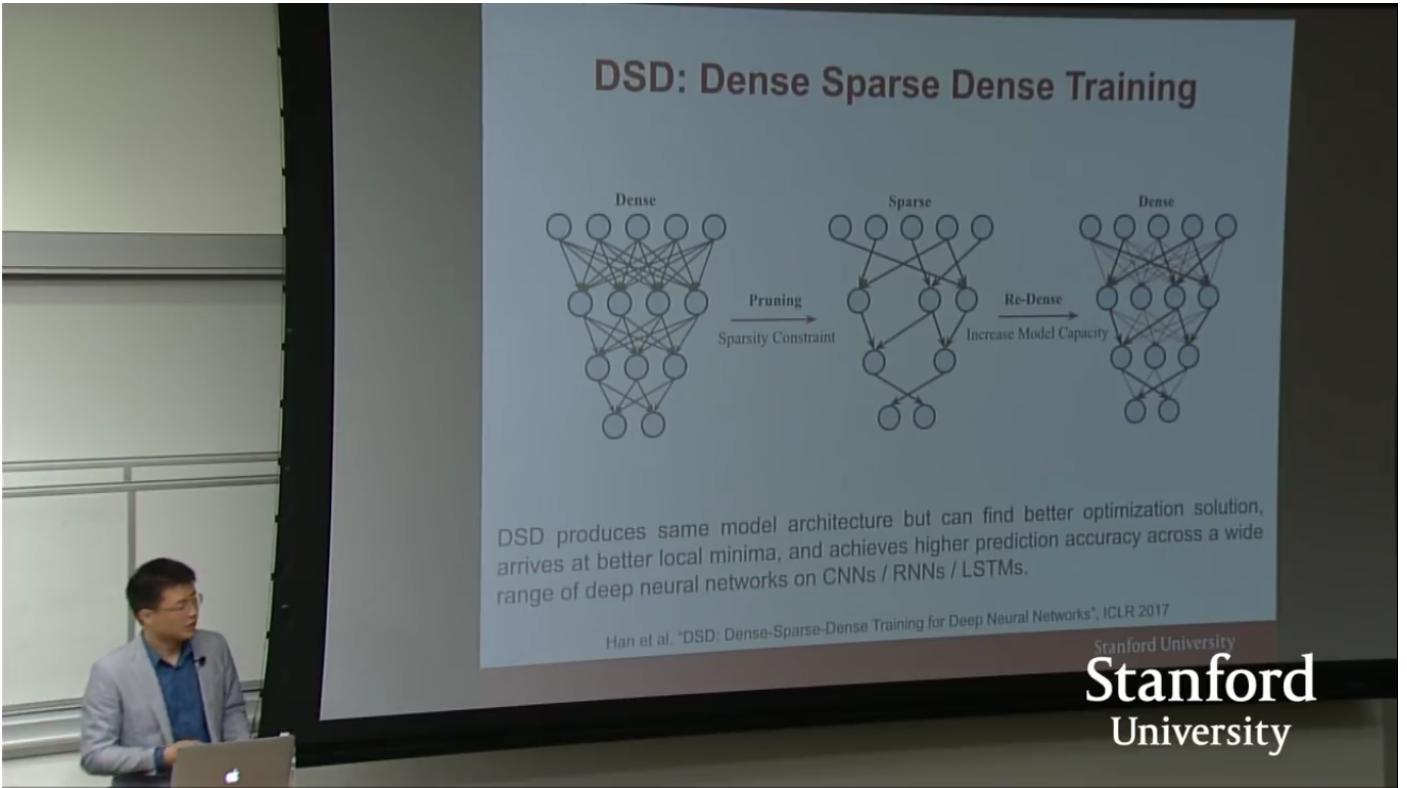
- Method: Divide score by a “temperature” to get a much softer distribution
- Result: Start with a trained model that classifies 58.9% of the test frames correctly. The new model converges to 57.0% correct even when it is only trained on 3% of the data

Hinton et al. Dark knowledge / Distilling the Knowledge in a Neural Network

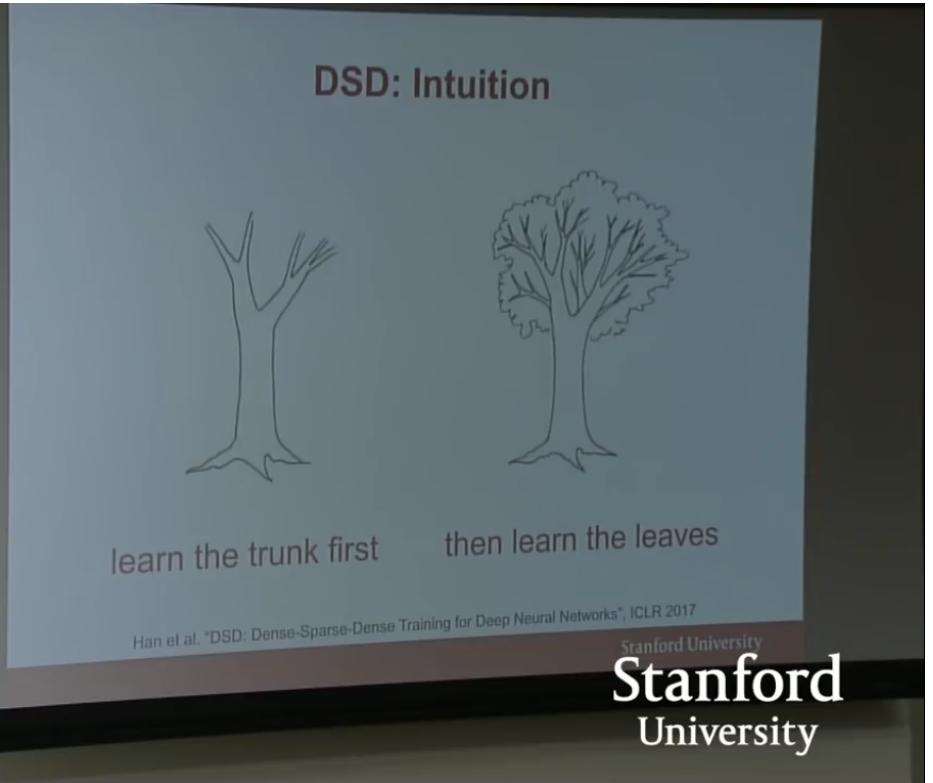
Stanford University

- We control the "softness" by the parameter T (Temperature)

### Dense - Sparse - Dense Training



- DSD produces same model architecture but can find better optimization solution, arrives at better local minima, and achieves higher prediction accuracy over a wide range of deep neural networks on CNNs, RNNs and LSTMs.

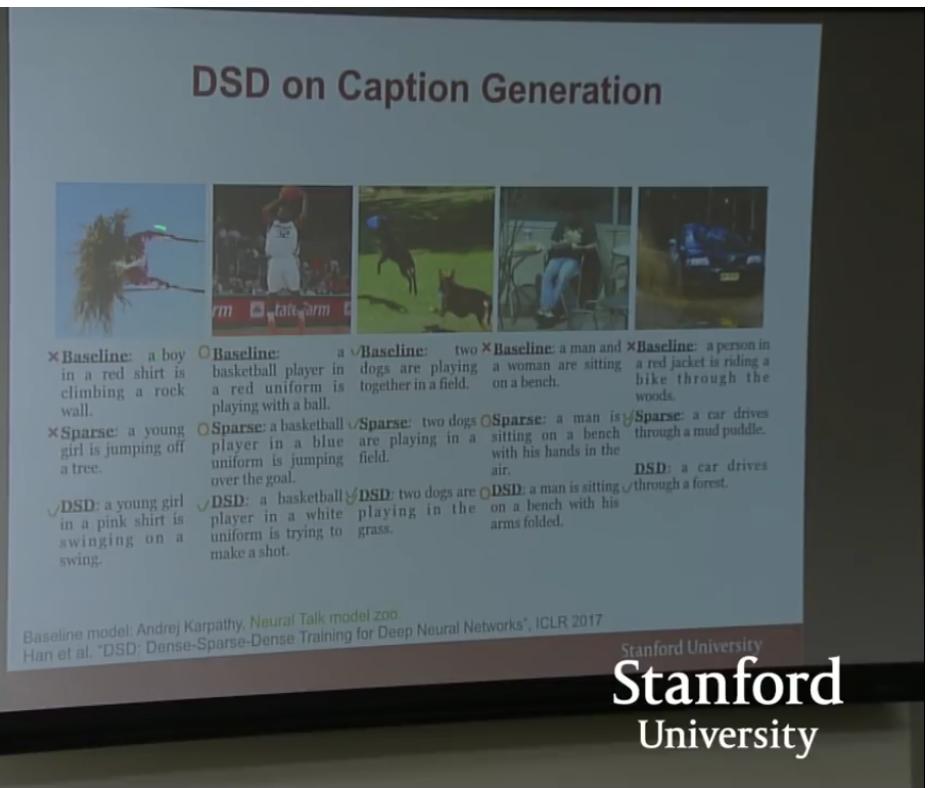
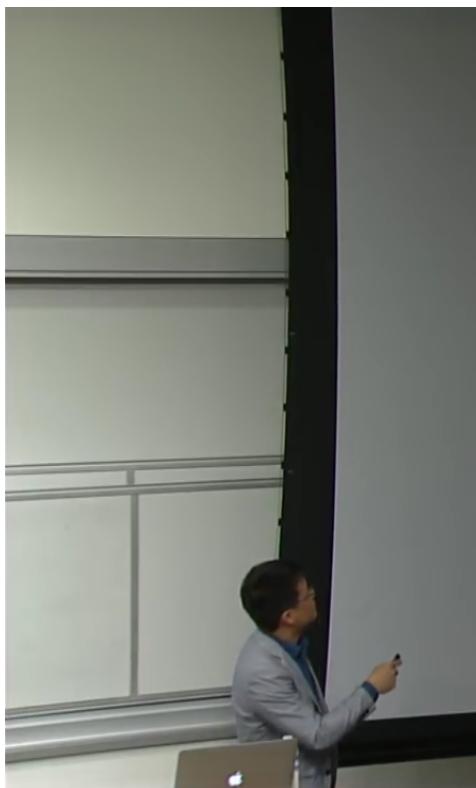


**DSD: Intuition**

learn the trunk first      then learn the leaves

Han et al. "DSD: Dense-Sparse-Dense Training for Deep Neural Networks", ICLR 2017

Stanford University



**DSD on Caption Generation**

✗ Baseline: a boy in a red shirt is climbing a rock wall.

✗ Baseline: a basketball player in a red uniform is playing with a ball.

✗ Baseline: two dogs are playing together in a field.

✗ Baseline: a person in a red jacket is riding a bike through the woods.

✗ Sparse: a young girl is jumping off a tree.

✗ Sparse: a basketball player in a blue uniform is jumping over the goal.

✗ Sparse: two dogs are playing in a field.

✗ Sparse: a man is sitting on a bench with his hands in the air.

✓ DSD: a young girl in a pink shirt is swinging on a swing.

✓ DSD: a basketball player in a white uniform is trying to make a shot.

✓ DSD: two dogs are playing in the grass.

✓ DSD: a man is sitting on a bench with his arms folded.

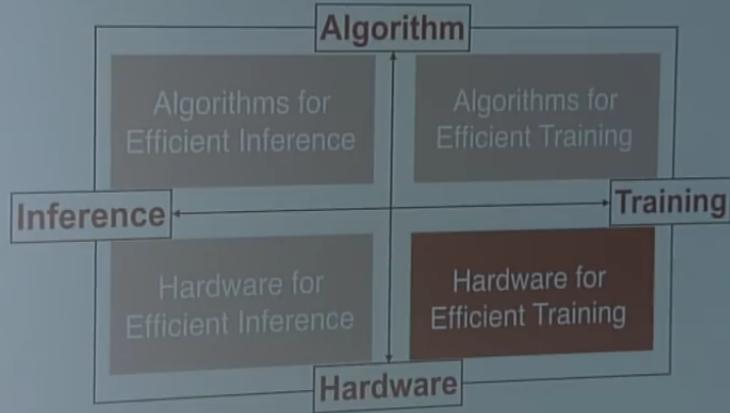
✓ DSD: a car drives through a forest.

Baseline model: Andrej Karpathy, Neural Talk model zoo  
Han et al. "DSD: Dense-Sparse-Dense Training for Deep Neural Networks", ICLR 2017

Stanford University

## Hardware for Efficient Training:

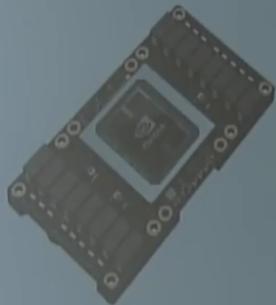
## Agenda



Stanford University  
**Stanford**  
University

## GPUs for Training

### Nvidia PASCAL GP100 (2016)



- 10/20 TFLOPS FP32/FP16
- 16GB HBM – 750 GB/s
- 300W TDP
- 67 GFLOPS/W (FP16)
- 16nm process
- 160GB/s NV Link

Slide Source: Sze et al Survey of DNN Hardware, MICRO'16 Tutorial.  
Data Source: NVIDIA

Stanford University  
**Stanford**  
University

## GPUs for Training

Nvidia Volta GV100 (2017)



Data Source: NVIDIA

- 15 FP32 TFLOPS
- 120 Tensor TFLOPS
- 16GB HBM2 @ 900GB/s
- 300W TDP
- 12nm process
- 21B Transistors
- die size: 815 mm<sup>2</sup>
- 300GB/s NVLink

Stanford University  
**Stanford**  
University

## What's new in Volta: Tensor Core

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}_{\text{FP16 or FP32}} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}_{\text{FP16}} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}_{\text{FP16 or FP32}}$$

a new instruction that performs 4x4x4 FMA mixed-precision operations per clock  
12X increase in throughput for the Volta V100 compared to the Pascal P100

<https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/>

Stanford University  
**Stanford**  
University

## Pascal v.s. Volta

cuBLAS Mixed Precision (FP16 Input, FP32 compute)

Matrix Size (M=N=K)	P100 (CUDA 8)	V100 Tensor Cores (CUDA 9)
512	~1.2	~3.0
1024	~1.2	~7.0
2048	~1.2	~9.3x
4096	~1.2	~9.3x

Relative Performance

Tesla V100 Tensor Cores and CUDA 9 deliver up to 9x higher performance for GEMM operations.  
<https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/>

Stanford University

## Pascal v.s. Volta

### ResNet-50 Training

GPU	Images per Second
P100 FP32	~250
V100 Tensor Cores	~600

2.4x faster

### ResNet-50 Inference

TensorRT - 7ms Latency

GPU	Images per Second
P100 FP16	~1500
V100 Tensor Cores	~5500

3.7x faster

Left: Tesla V100 trains the ResNet-50 deep neural network 2.4x faster than Tesla P100.  
Right: Given a target latency per image of 7ms, Tesla V100 is able to perform inference using the ResNet-50 deep neural network 3.7x faster than Tesla P100.

<https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/>

Stanford University

Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK110 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1455 MHz
Peak FP32 TFLOP/s*	5.04	6.8	10.6	15
Peak Tensor Core TFLOP/s*	-	-	-	120
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB	16 GB
TDP	235 Watts	250 Watts	300 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion	21.1 billion
GPU Die Size	551 mm <sup>2</sup>	601 mm <sup>2</sup>	610 mm <sup>2</sup>	815 mm <sup>2</sup>
Manufacturing Process	28 nm	28 nm	16 nm FinFET+	12 nm FFN

<https://devblogs.nvidia.com/parallelforall/cuda-9-features-revealed/>

Stanford University

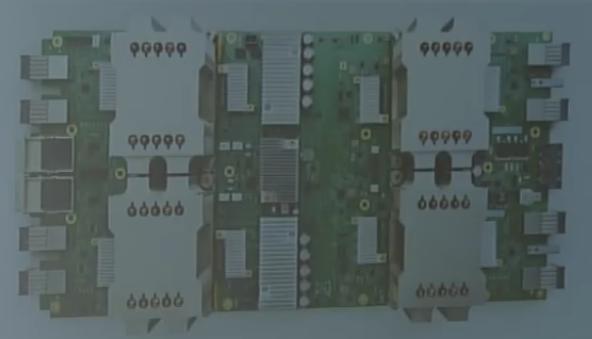
## GPU / TPU

	K80 2012	TPU 2015	P40 2016
Inferences/Sec <10ms latency	1/13X	1X	2X
Training TOPS	6 FP32	NA	12 FP32
Inference TOPS	6 FP32	90 INT8	48 INT8
On-chip Memory	16 MB	24 MB	11 MB
Power	300W	75W	250W
Bandwidth	320 GB/S	34 GB/S	350 GB/S

<https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/>

Stanford University

## Google Cloud TPU



Cloud TPU delivers up to 180 teraflops to train and run machine learning models.  
source: Google Blog

Stanford University  
**Stanford**  
University

## Google Cloud TPU



A “TPU pod” built with 64 second-generation TPUs delivers up to 11.5 petaflops of machine learning acceleration.

“One of our new large-scale translation models used to take a full day to train on 32 of the best commercially-available GPUs—now it trains to the same accuracy in an afternoon using just one eighth of a TPU pod.” — Google Blog

Stanford University  
**Stanford**  
University

