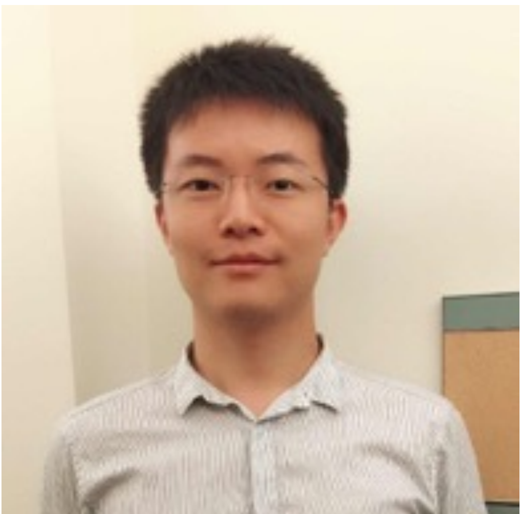
Deep Compression and EIE:

——Deep Neural Network Model Compression and Efficient Inference Engine

Song Han
CVA group, Stanford University
Jan 6, 2015

A few words about us



Song Han

- Fourth year PhD with Prof. Bill Dally at Stanford.
- Research interest is computer architecture for deep learning, to improve the energy efficiency of neural networks running on mobile and embedded systems.
- Recent work on “Deep Compression” and “EIE: Efficient Inference Engine” covered by TheNextPlatform.

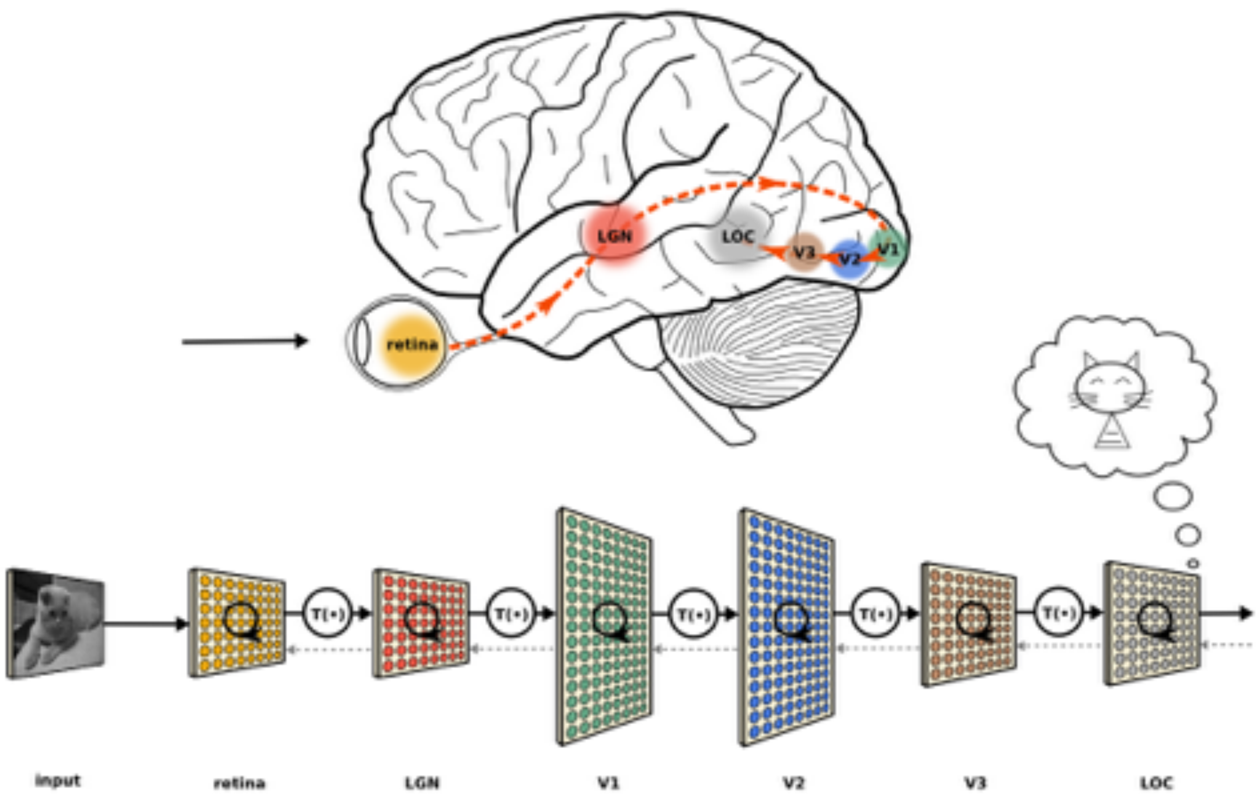
Bill Dally

- Professor at Stanford University and former chairman of CS department, leads the Concurrent VLSI Architecture Group.
- Chief Scientist of NVIDIA.
- Member of the National Academy of Engineering, Fellow of the American Academy of Arts & Sciences, Fellow of the IEEE, Fellow of the ACM.

This Talk:

- **Deep Compression**: A Deep Neural Network Model Compression Pipeline.
- **EIE Accelerator**: Efficient Inference Engine that Accelerates the Compressed Deep Neural Network Model.

Deep Learning: Next Wave of AI



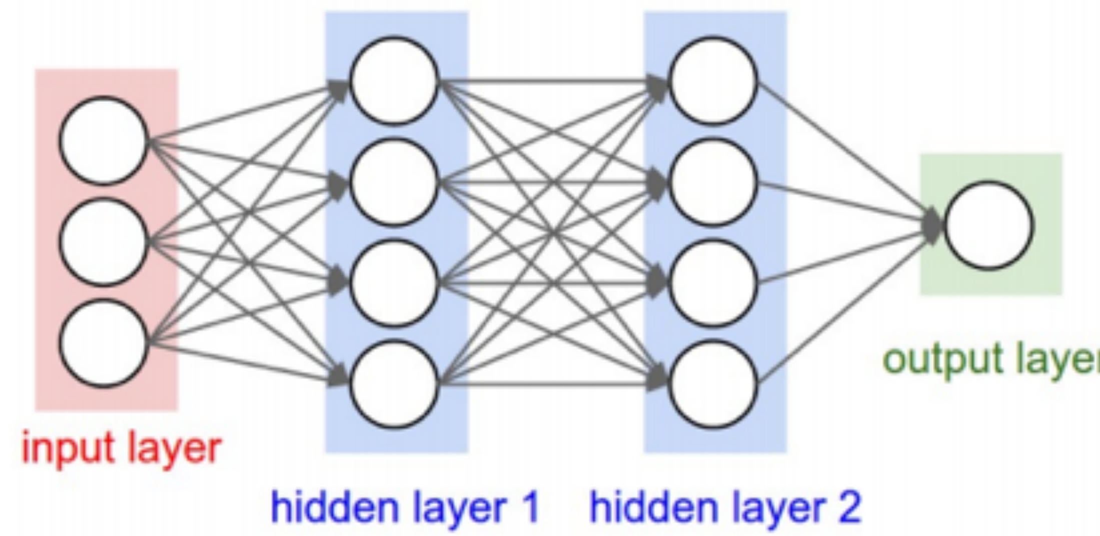
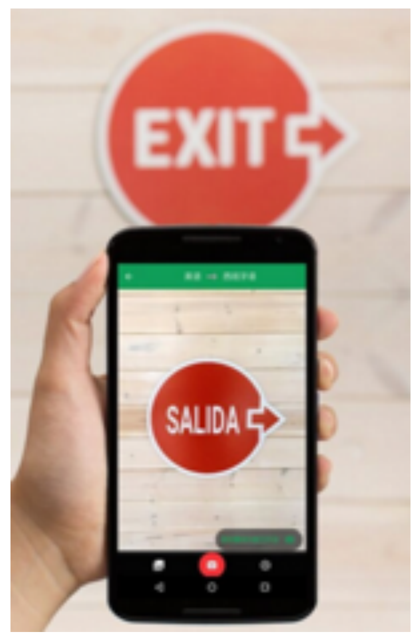


Image
Recognition

Speech
Recognition

Natural Language
Processing

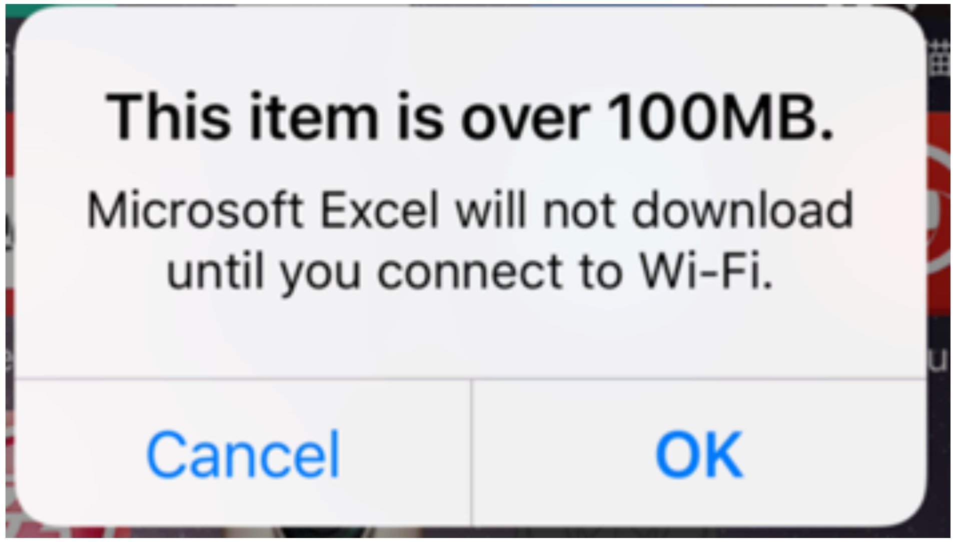
Applications



The Problem:

If Running DNN on Mobile…

App developers suffers from the model size



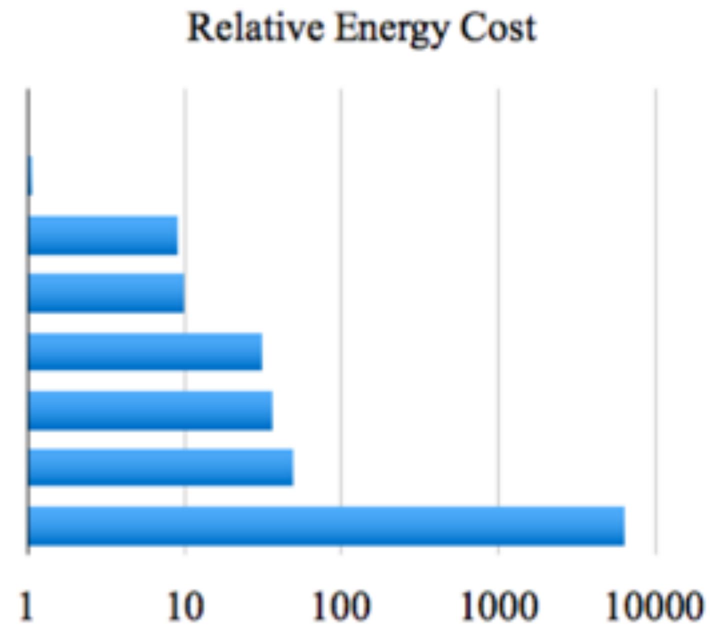
“At Baidu, our #1 motivation for compressing networks is to **bring down the size of the binary file**. As a mobile-first company, we frequently update various apps via different app stores. We've **very sensitive to the size of our binary files**, and a feature that increases the binary size by 100MB will receive much more scrutiny than one that increases it by 10MB.” —Andrew Ng

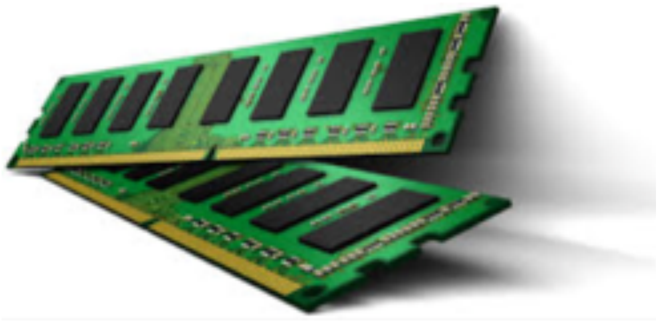
The Problem:

If Running DNN on Mobile…

Hardware engineer suffers from the model size (embedded system, limited resource)

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**





The Problem:

If Running DNN on the Cloud…

Network Delay

Power Budget

User Privacy

Intelligent but Inefficient

Solver 1: Deep Compression

Deep Neural Network Model Compression

Smaller Size
Compress Mobile App Size by 35x-50x

Accuracy
no loss of accuracy
improved accuracy

Speedup
make inference faster

Solve 2: EIE Accelerator

ASIC accelerator: EIE (Efficient Inference Engine)

Offline
No dependency on
network connection

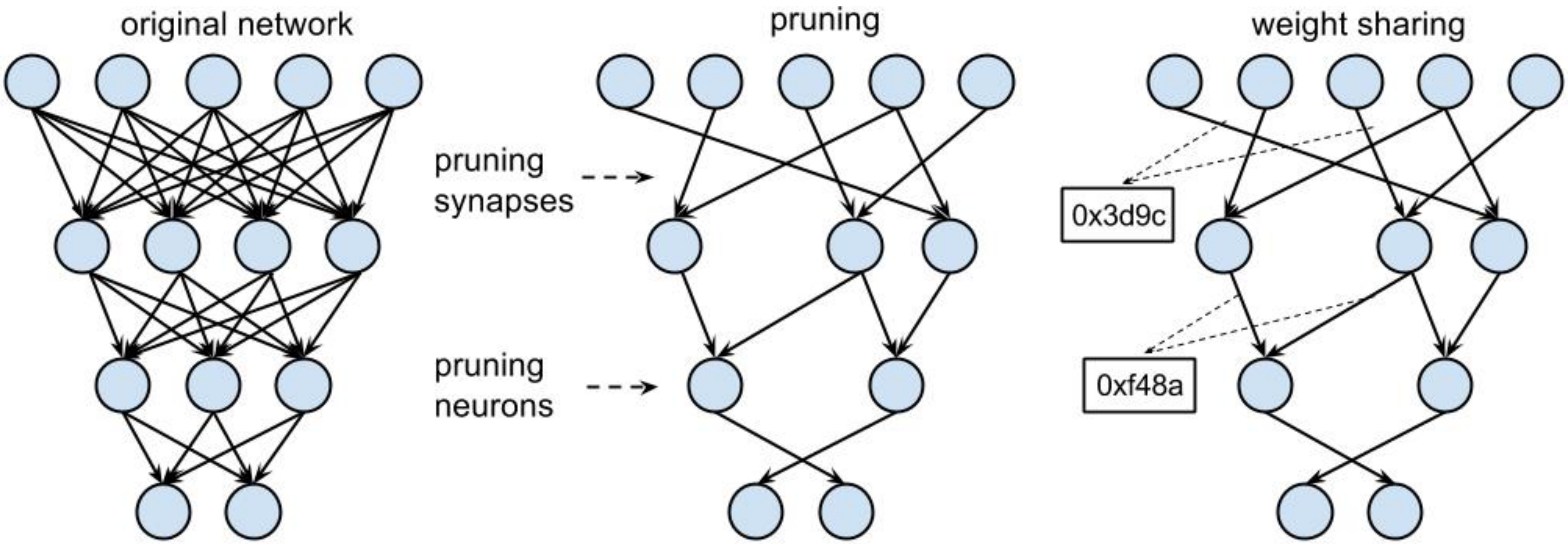
Real Time
No network delay
high frame rate

Low Power
High energy efficiency
that preserves battery

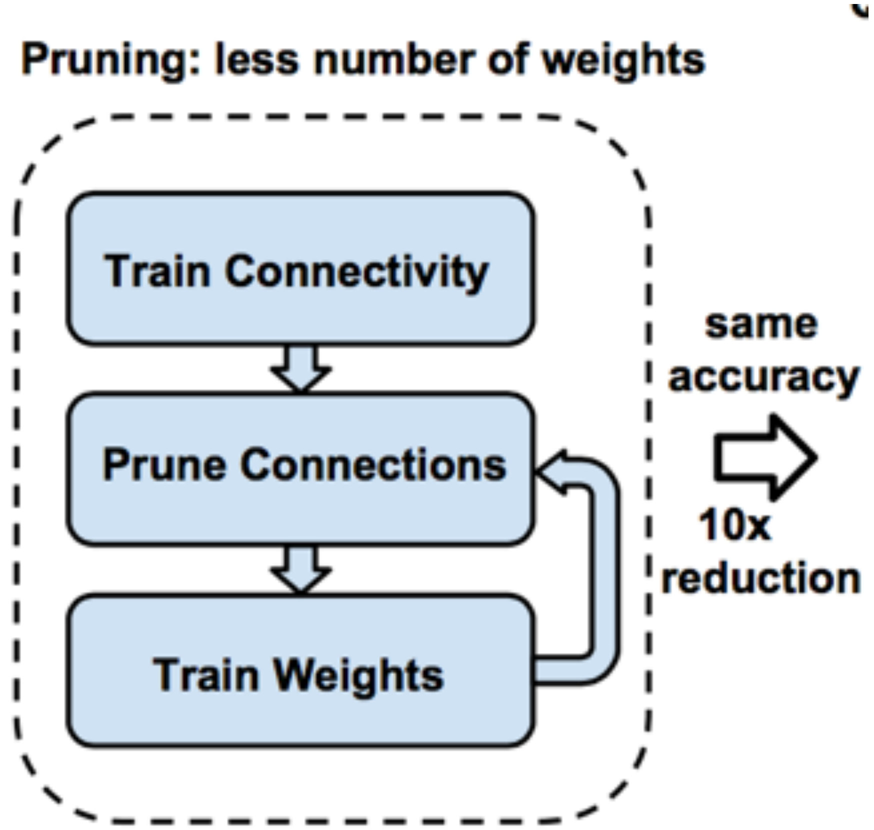
Deep Compression

- AlexNet: 35×, 240MB => 6.9MB
- VGG16: 49× 552MB => 11.3MB
- Both with no loss of accuracy on ImageNet12
- Weights fits on-chip SRAM, taking 120x less energy than DRAM

Compression Pipeline: Overview



1. Pruning



Pruning: less number of weights
Train Connectivity
Prune Connections
Train Weights
same accuracy
10x reduction

Pruning: Motivation

Age	Number of Connections	Stage
at birth	50 Trillion	newly formed
1 year old	1000 Trillion	peak
10 year old	500 Trillion	pruned and stabilized

Table 1: The synapses pruning mechanism in human brain development

- Trillion of synapses are generated in the human brain during the first few months of birth.
- **1 year old**, peaked at **1000 trillion**
- Pruning begins to occur.
- **10 years old**, a child has nearly **500 trillion** synapses
- This 'pruning' mechanism removes redundant connections in the brain.

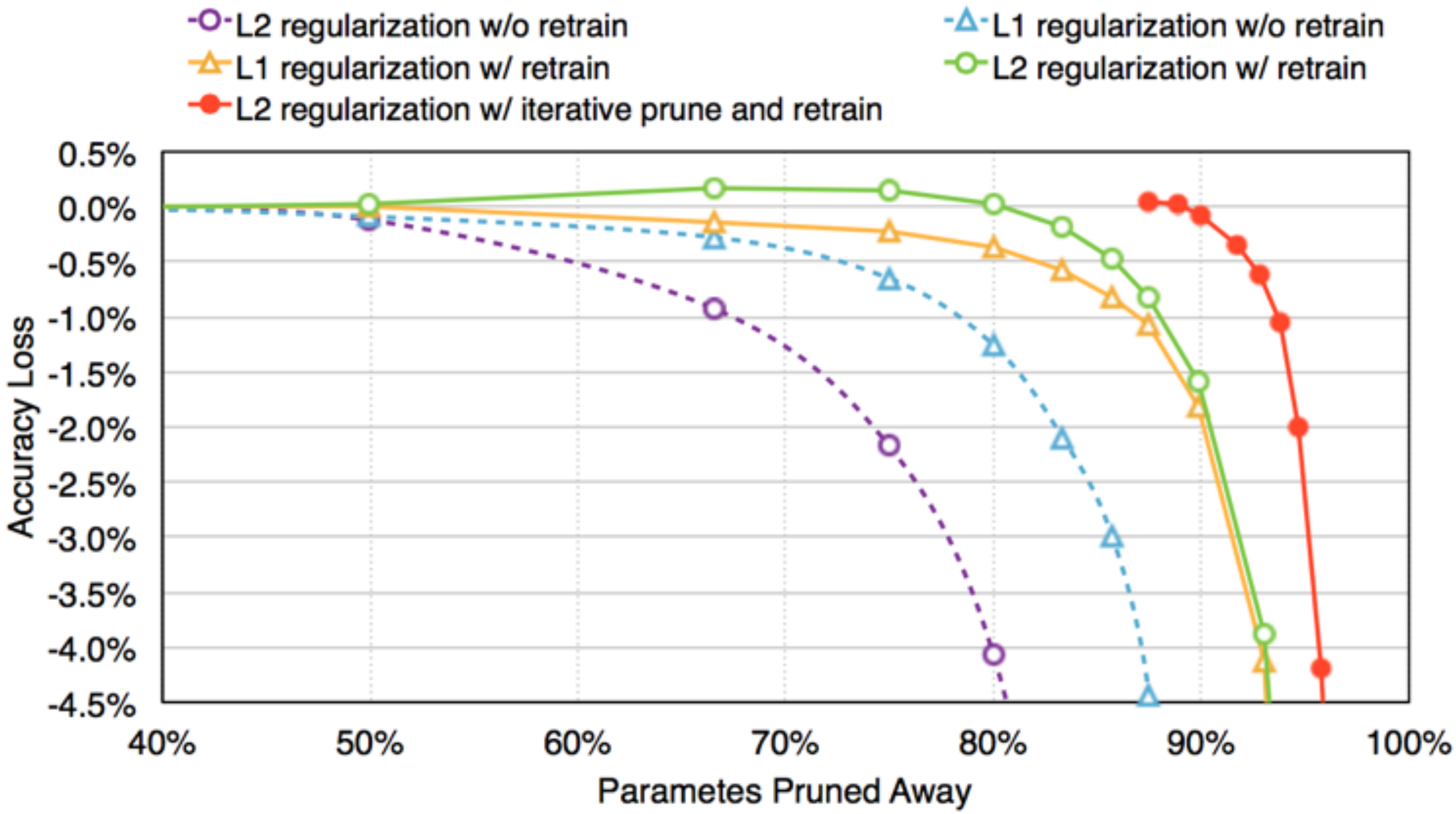
[1] Christopher A Walsh. Peter huttenlocher (1931-2013). *Nature*, 502(7470):172–172, 2013.

Pruning: Result on 4 Covnets

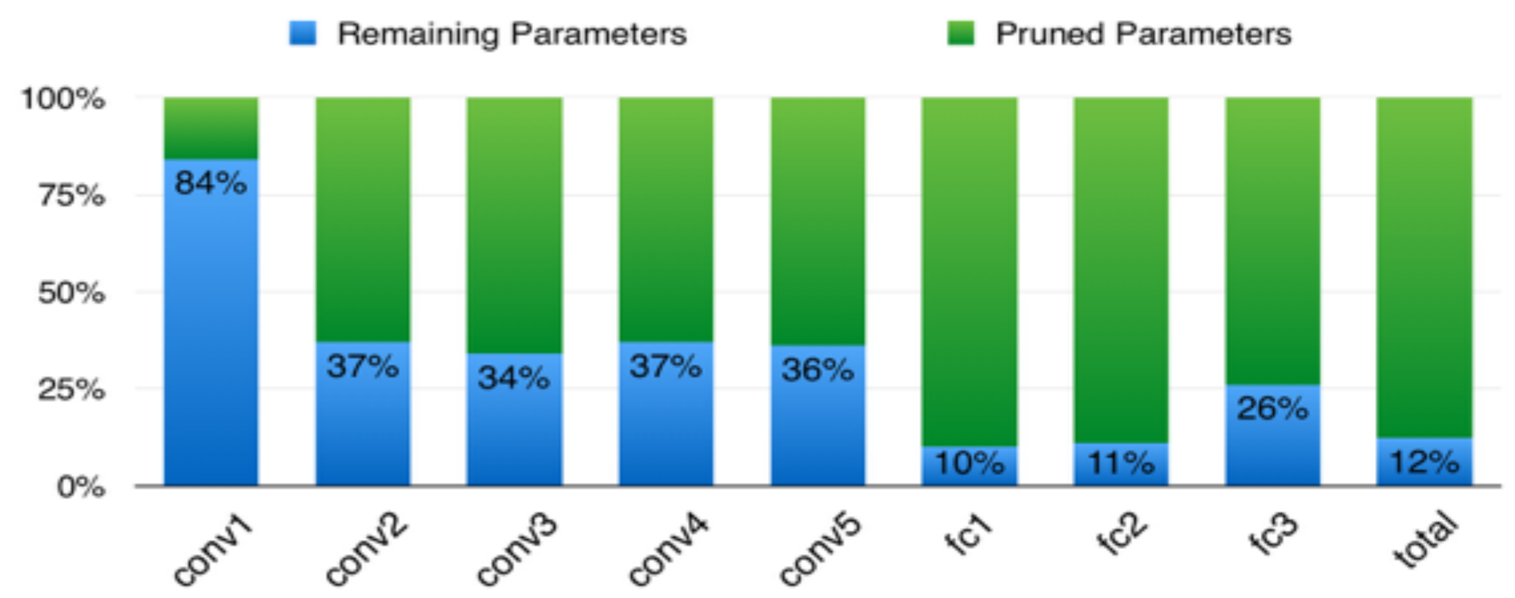
Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**×
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**×
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**×
VGG16 Ref	31.50%	11.32%	138M	
VGG16 Pruned	31.34%	10.88%	**10.3M**	**13**×

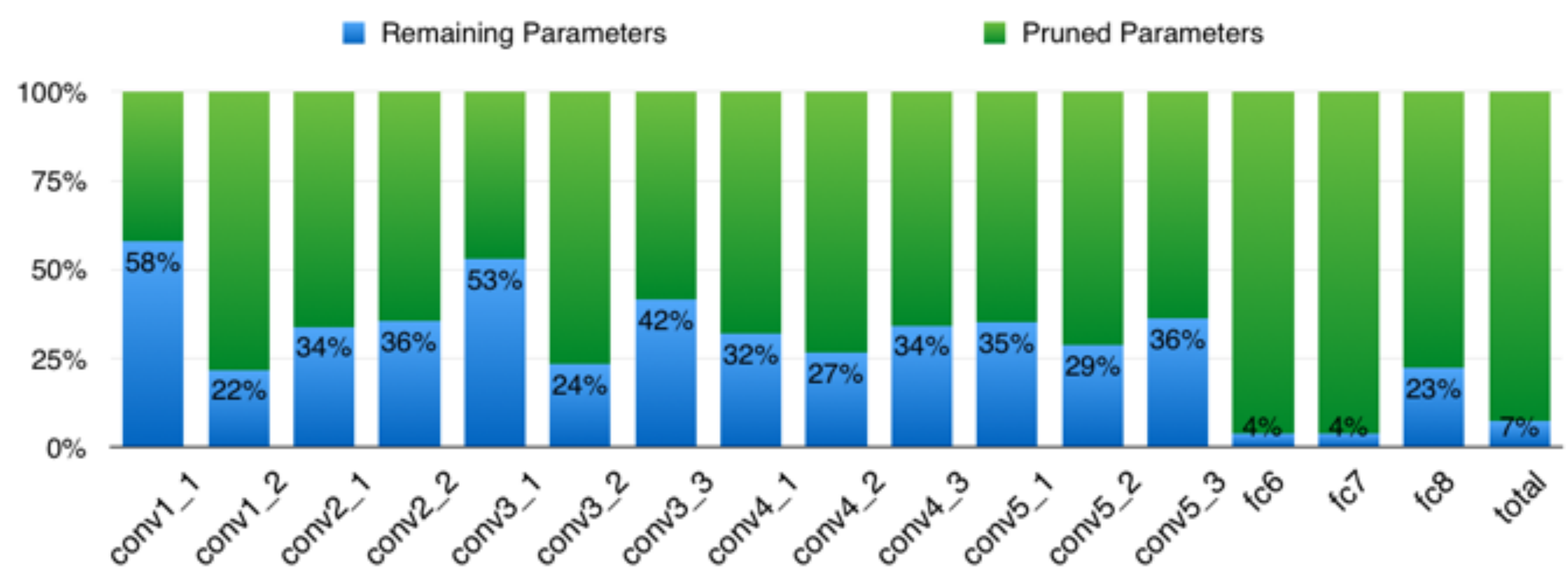
Table 1: Network pruning can save 9× to 13× parameters with no drop in predictive performance

Pruning: AlexNet

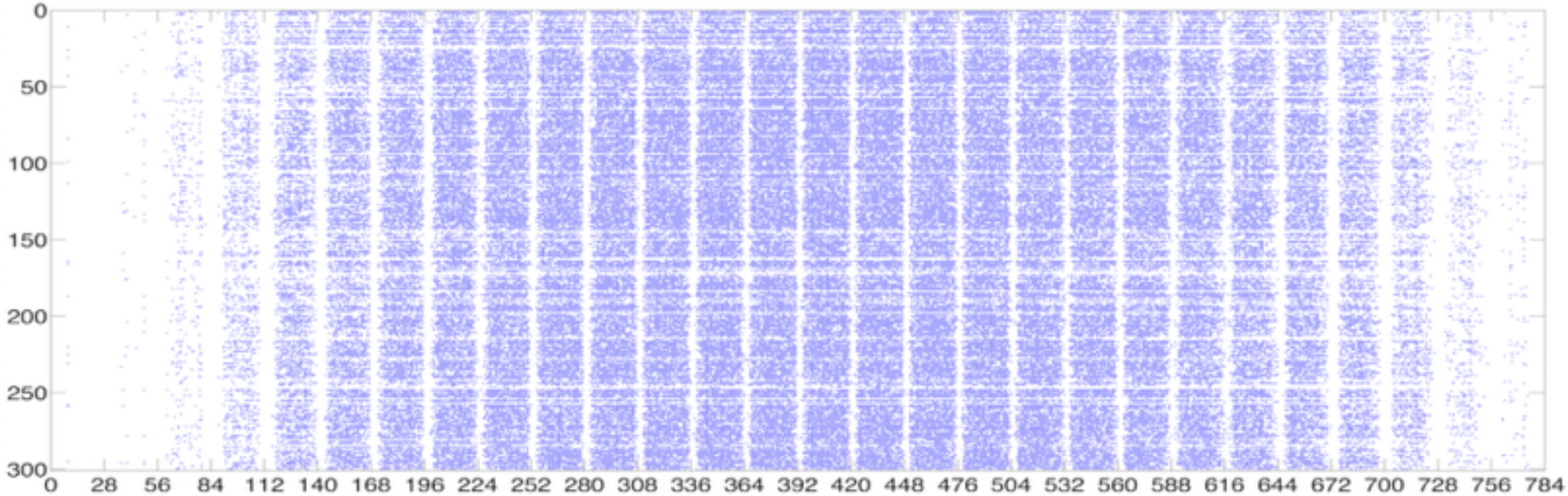


AlexNet & VGGNet



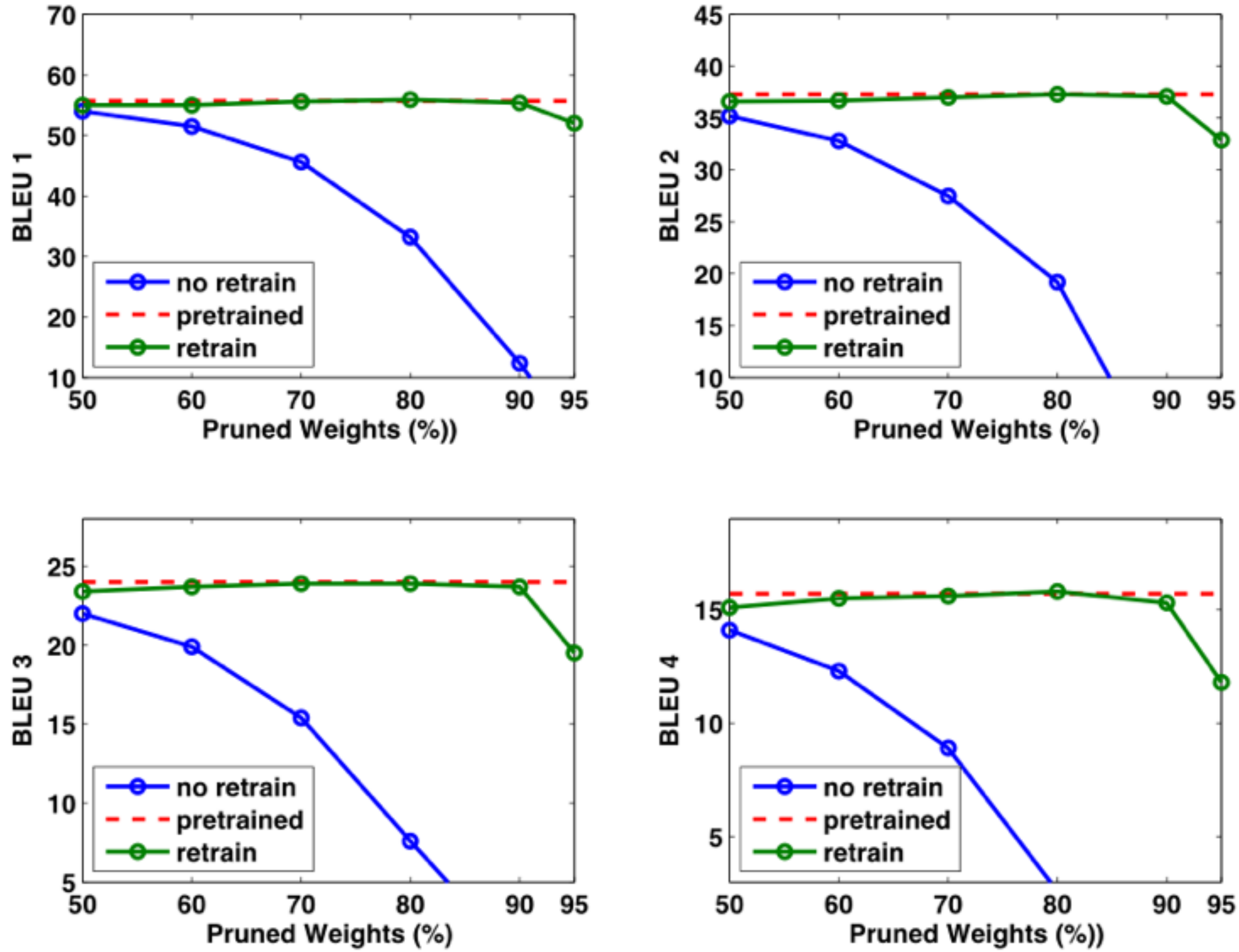


Mask Visualization



Visualization of the first FC layer's sparsity pattern of Lenet-300-100. It has a banded structure repeated 28 times, which correspond to the un-pruned parameters in the center of the images, since the digits are written in the center.

Pruning also works well on RNN+LSTM



[1] Thanks Shijian Tang pruning Neural Talk

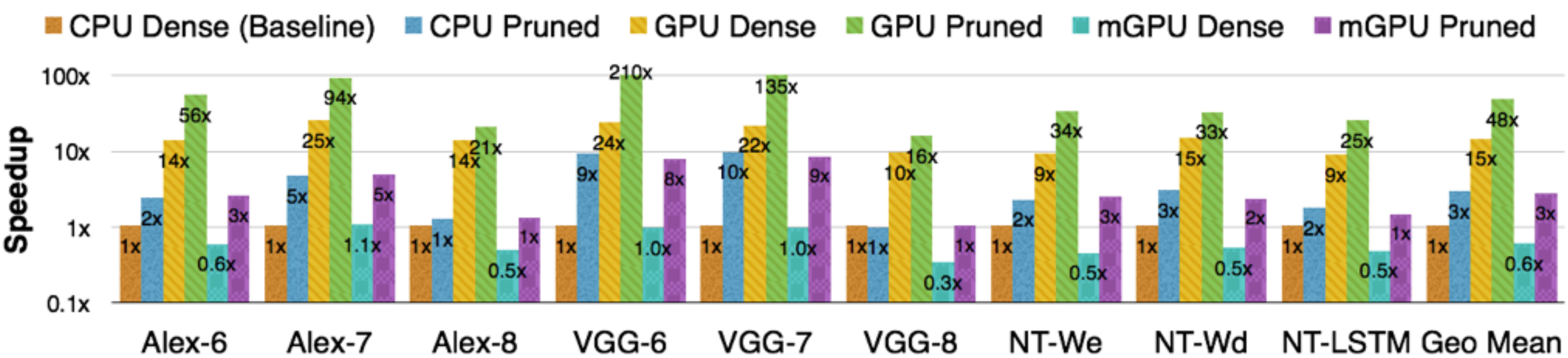
- **Original**: a basketball player in a white uniform is playing with a ball
- **Pruned 90%**: a basketball player in a white uniform is playing with a basketball

- **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

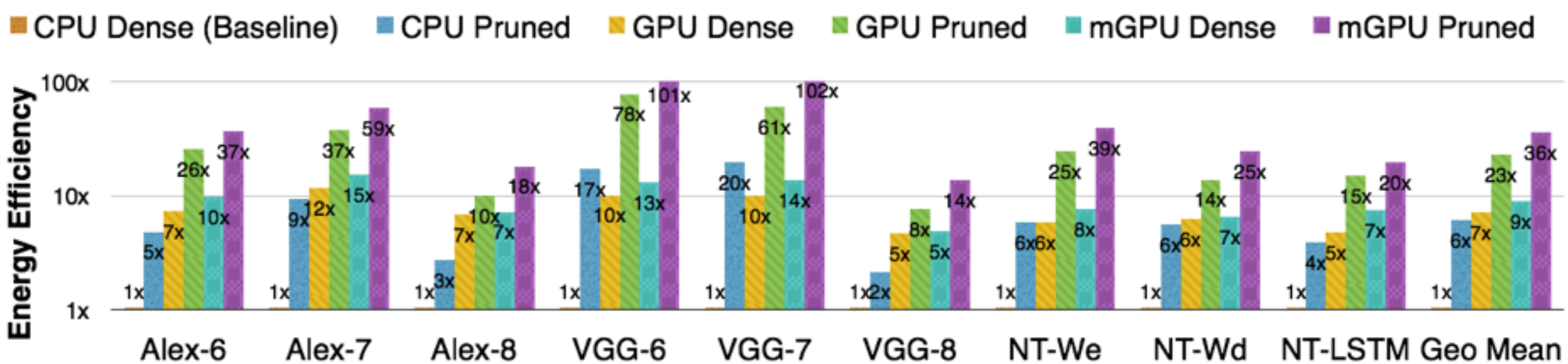
- **Original** : a soccer player in red is running in the field
- **Pruned <u>95%</u>**: a man in a red shirt and black and white black shirt is running through a field

Speedup (FC layer)



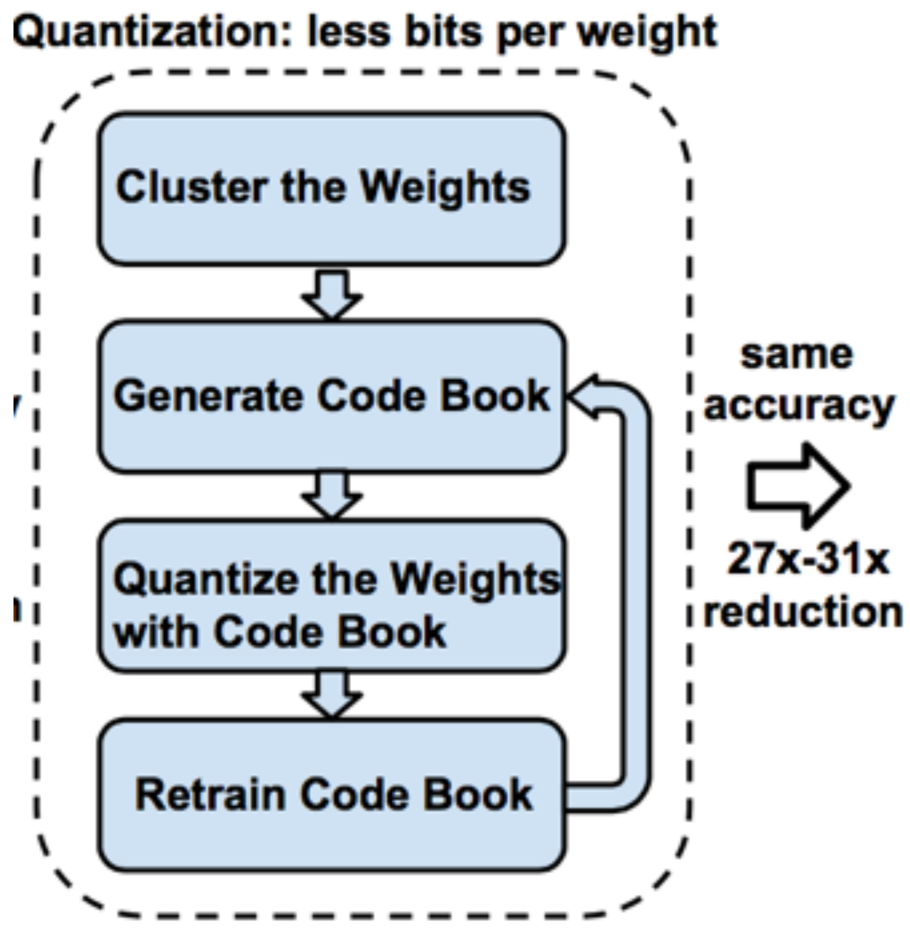
- Intel Core i7 5930K: MKL CBLAS GEMV, MKL SPBLAS CSRMV
- NVIDIA GeForce GTX Titan X: cuBLAS GEMV, cuSPARSE CSRMV
- NVIDIA Tegra K1: cuBLAS GEMV, cuSPARSE CSRMV

Energy Efficiency (FC layer)



- Intel Core i7 5930K: CPU socket and DRAM power are reported by pcm-power utility
- NVIDIA GeForce GTX Titan X: reported by nvidia-smi utility
- NVIDIA Tegra K1: measured the total power consumption with a power-meter, 15% AC to DC conversion loss, 85% regulator efficiency and 15% power consumed by peripheral components => 60% AP+DRAM power

2. Quantization and Weight Sharing



Weight Sharing: Overview

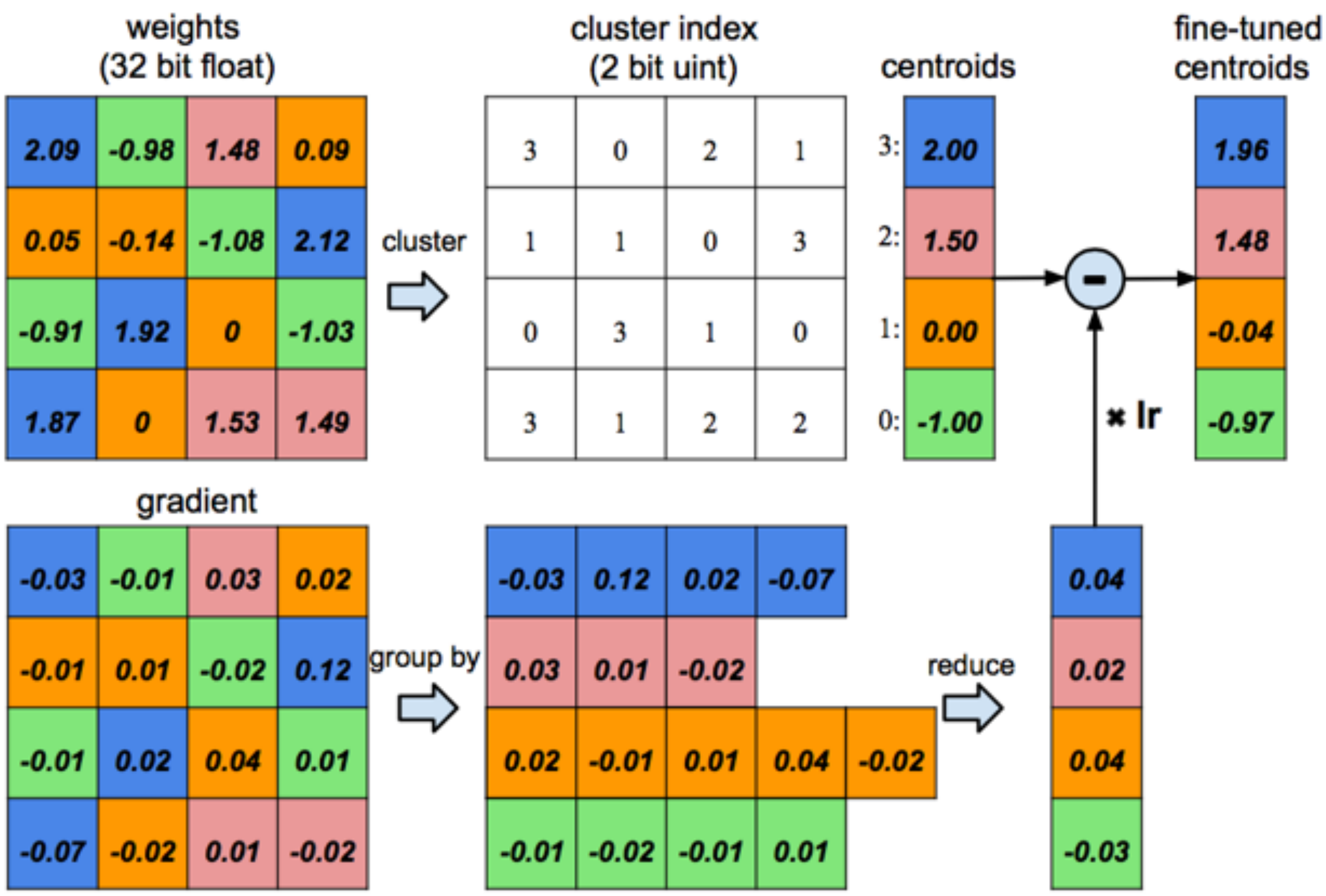
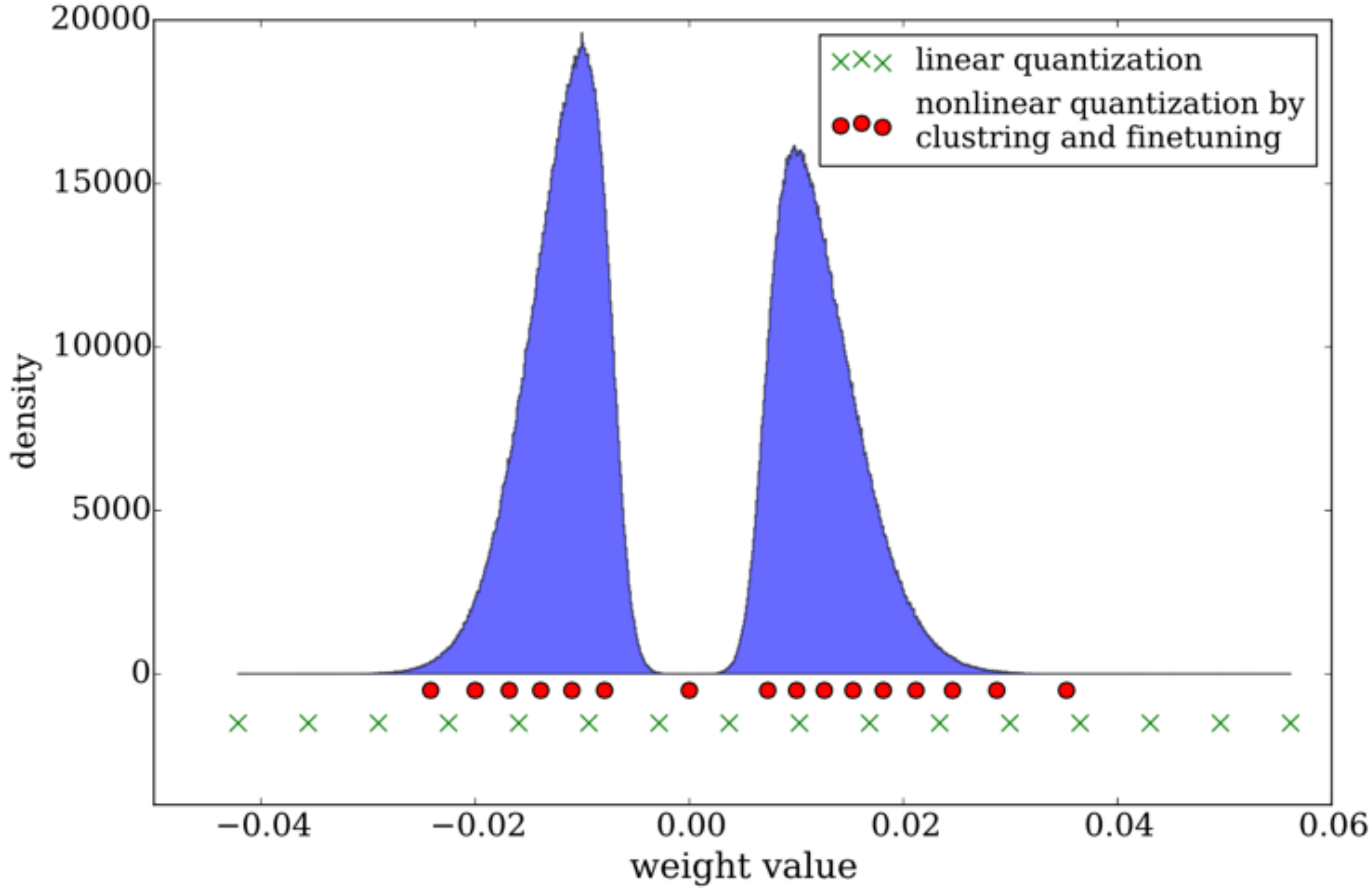


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom)

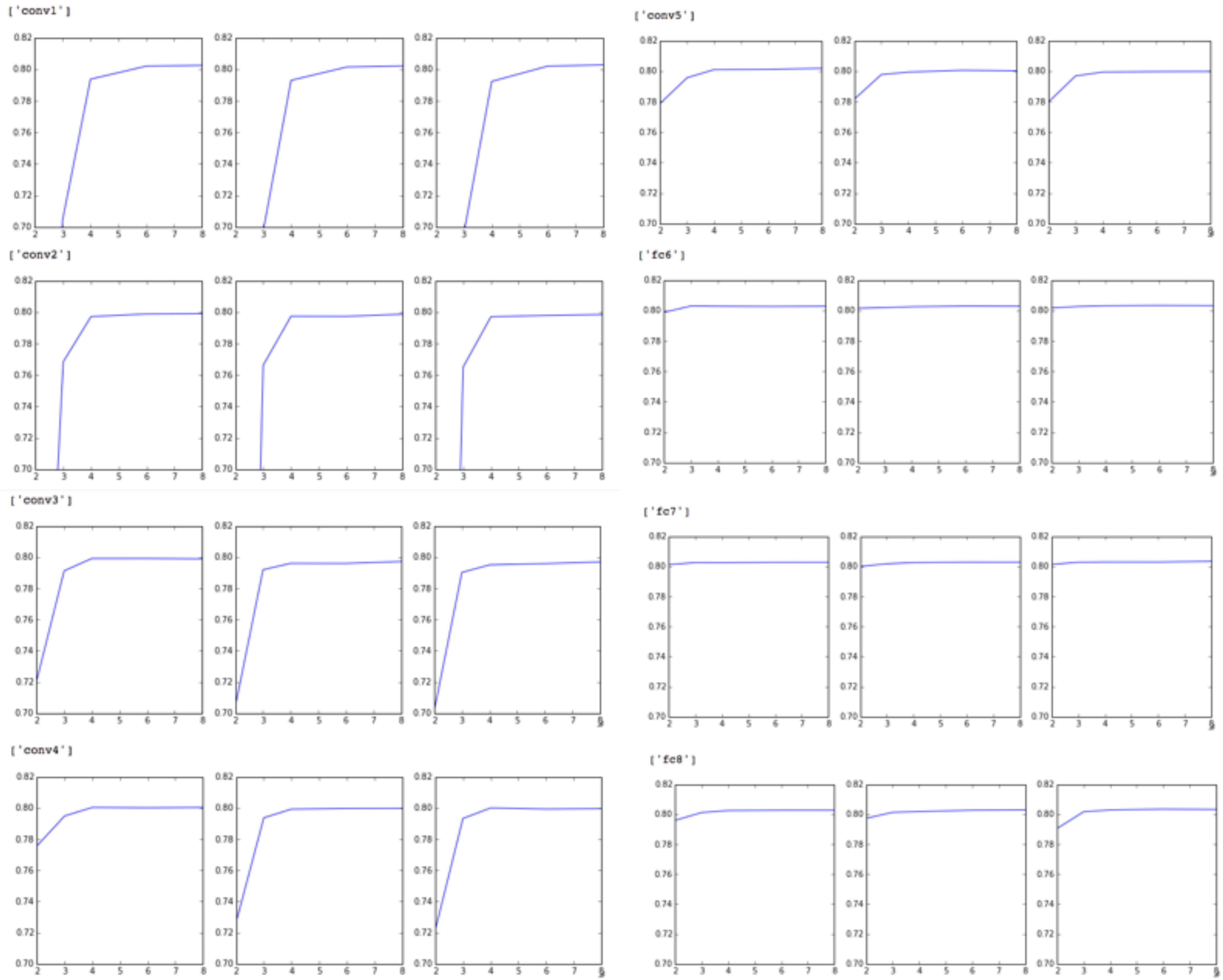
Finetune Centroids



Quantization: Result

- 16 Million => 2^4=16
- 8/5 bit quantization results in **no** accuracy loss
- 8/4 bit quantization results in no top-5 accuracy loss, **0.1%** top-1 accuracy loss
- 4/2 bit quantization results in **-1.99%** top-1 accuracy loss, and**-2.60%** top-5 accuracy loss, not that bad-:

Accuracy ~ #Bits on 5 Conv Layer + 3 FC Layer



Pruning and Quantization Works Well Together

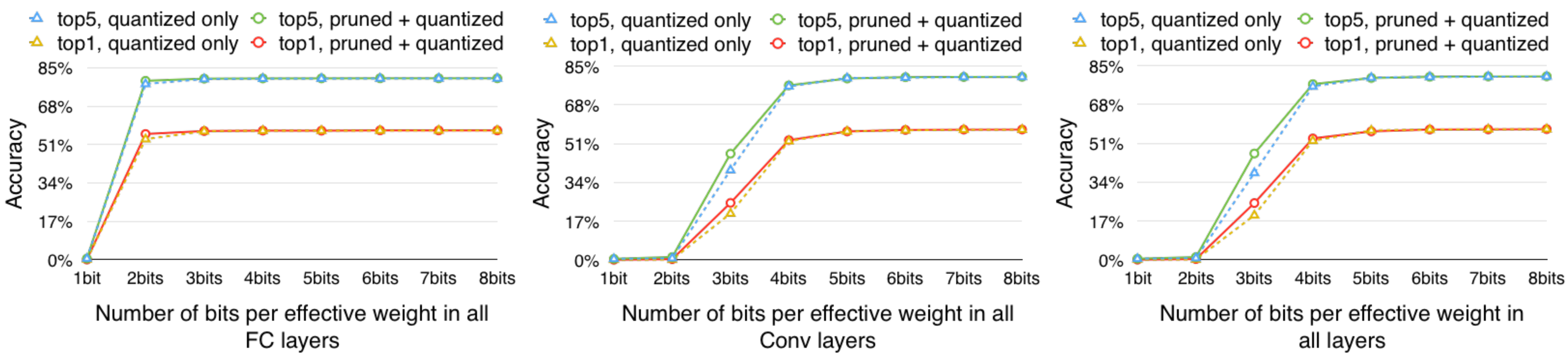
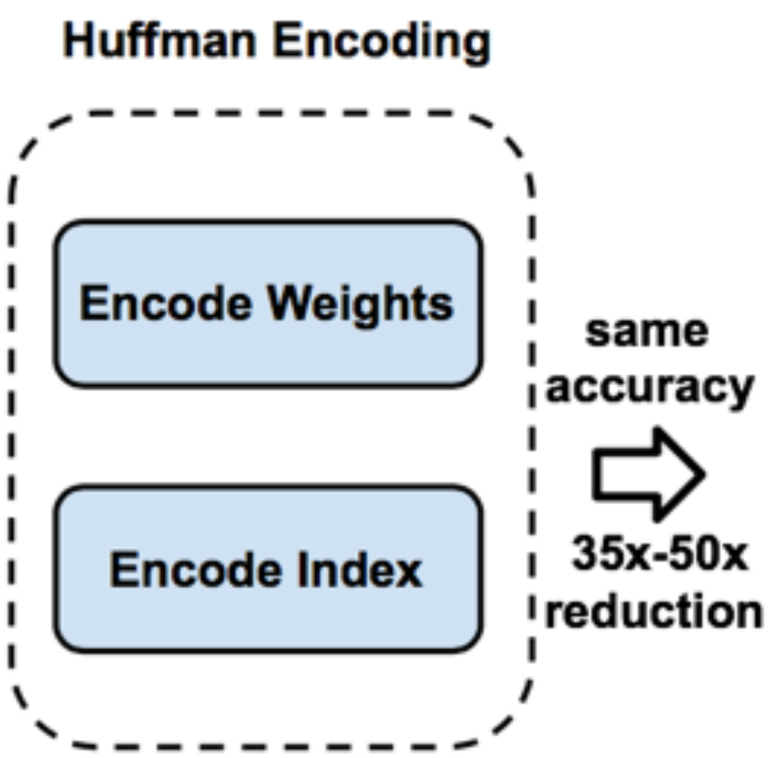


Figure 7: Pruning doesn't hurt quantization. Dashed: quantization on unpruned network. Solid: quantization on pruned network; Accuracy begins to drop at the same number of quantization bits whether or not the network has been pruned. Although pruning made the number of parameters less, quantization still works well, or even better(3 bits case on the left figure) as in the unpruned network.

3. Huffman Coding



Huffman Coding

Huffman code is a type of optimal prefix code that is commonly used for loss-less data compression. It produces a variable-length code table for encoding source symbol. The table is derived from the occurrence probability for each symbol. As in other entropy encoding methods, more common symbols are represented with fewer bits than less common symbols, thus save the total space.

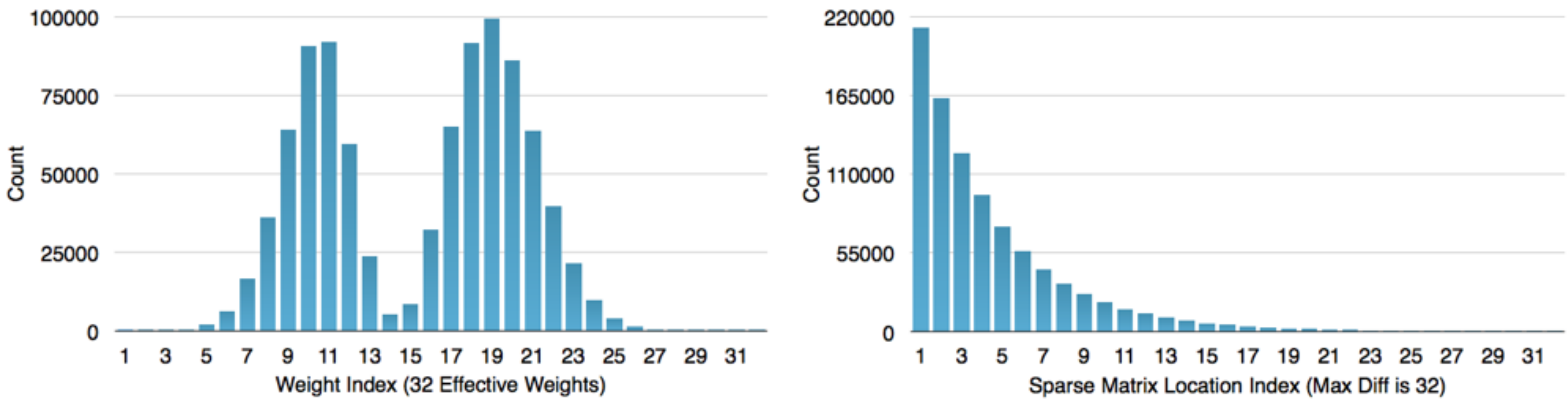


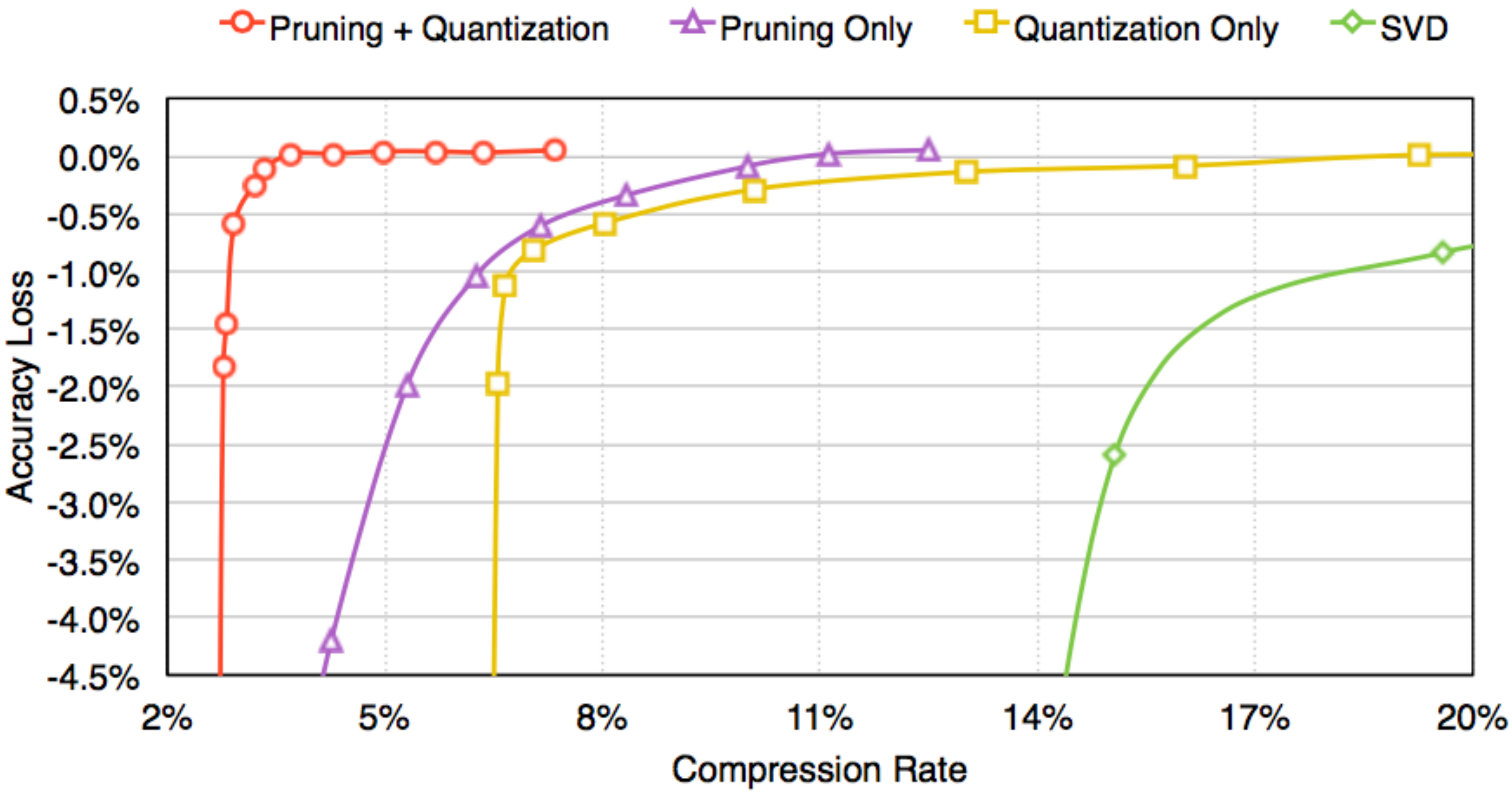
Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased and can be compressed by Huffman encoding

Deep Compression Result on 4 Convnets

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	**40**×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	**39**×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	**35**×
VGG16 Ref	31.50%	11.32%	552 MB	
VGG16 Compressed	31.17%	10.91%	**11.3 MB**	**49**×

Table 1: The compression pipeline can save 35× to 49× parameter storage with no drop in predictive performance

Result: AlexNet



AlexNet: Breakdown

Layer	#Weights	Weights% (P)	Weigh bits (P+Q)	Weight bits (+H)	Index bits (P+Q)	Index bits (+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	35K	84%	8	6.3	4	1.2	32.6%	20.53%
conv2	307K	38%	8	5.5	4	2.3	14.5%	9.43%
conv3	885K	35%	8	5.1	4	2.6	13.1%	8.44%
conv4	663K	37%	8	5.2	4	2.5	14.1%	9.11%
conv5	442K	37%	8	5.6	4	2.5	14.0%	9.43%
fc6	38M	9%	5	3.9	4	3.2	3.0%	2.39%
fc7	17M	9%	5	3.6	4	3.7	3.0%	2.46%
fc8	4M	25%	5	4	4	3.2	7.3%	5.85%
total	61M	11%	5.4	4	4	3.2	3.7%	2.88%

Table 4: Compression Statistics for Alexnet. P: pruning, Q:quantization, H:Huffman Encoding

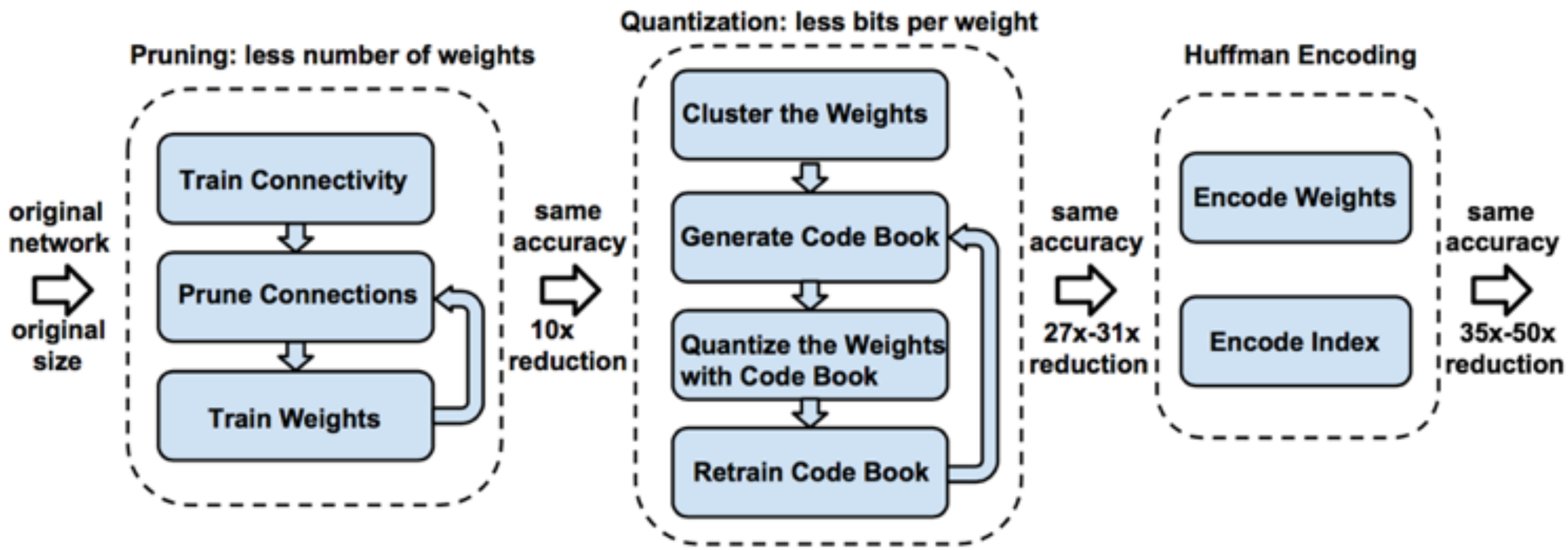
Comparison with other Compression Methods

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
Baseline Caffemodel [21]	42.78%	19.73%	240MB	1×
Fastfood-32-AD [22]	41.93%	-	131MB	2×
Fastfood-16-AD [22]	42.90%	-	64MB	3.7×
Collins & Kohli [23]	44.40%	-	61MB	4×
SVD [14]	44.02%	20.56%	55.2MB	5×
Pruning [6]	42.77%	19.67%	27MB	9×
Pruning+Quantization	42.78%	19.70%	8.9MB	27×
Pruning+Quantization+Huffman	**42.78%**	**19.70%**	**6.9MB**	**35**×

Table 6: Comparison with other model reduction methods on AlexNet. [23] reduced the parameters by 4× and with inferior accuracy. Deep Fried Convnets [22] worked on fully connected layers only and reduced the parameters by less than 4×. SVD save parameters but suffers from large accuracy loss as much as 2%. Network pruning [6] reduced the parameters by 9× without accuracy loss but the compression rate is only one third of this work. On other networks similar to AlexNet, [14] exploited linear structure of convnets and compressed the network by 2.4× to 13.4× layer wise, but had significant accuracy loss: as much as 0.9% even compressing a single layer. [15] experimented with vector quantization and compressed the network by 16 × −24×, but again incurred as much as 1% accuracy loss.

[14] EmilyLDenton,WojciechZaremba,JoanBruna,YannLeCun,andRobFergus.Exploitinglinearstructure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.
[15] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.
[21] Yangqing Jia. Bvlc caffe model zoo. ZichaoYang,MarcinMoczulski,MishaDenil,NandodeFreitas,AlexSmola,LeSong,andZiyuWang.
[22] Deep fried convnets. *arXiv preprint arXiv:1412.7149*, 2014.
[23] Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

Conclusion



- We have presented a method to compress neural networks without affecting accuracy by finding the right connections and quantizing the weights.
- Pruning the unimportant connections => quantizing the network and enforce weight sharing => apply Huffman encoding.
- We highlight our experiments on ImageNet, and reduced the weight storage by 35×, VGG16 by 49×, without loss of accuracy.
- Now weights can fit in cache

Product: A Model Compression Tool for Deep Learning Developers

- **Easy Version**:
  ✓ No training needed
  ✓ Fast
  x 5x - 10x compression rate
  x 1% loss of accuracy

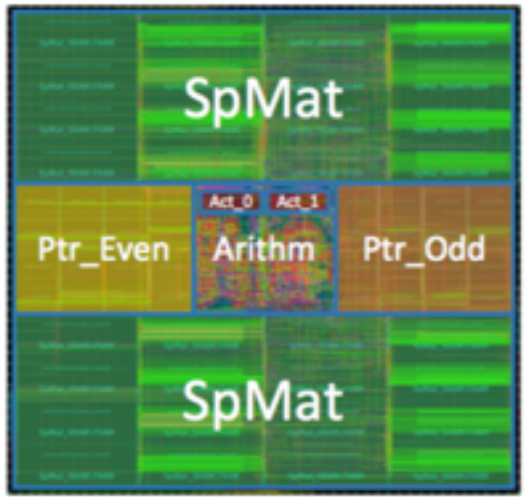
- **Advanced Version:**
  ✓ 35x - 50x compression rate
  ✓ no loss of accuracy
  x Training is needed
  x Slow

Demo:
Pocket AlexNet

EIE: Efficient Inference Engine on Compressed Deep Neural Network

Song Han
CVA group, Stanford University
Jan 6, 2015

ASIC Accelerator that Runs DNN on Mobile



Offline
No dependency on network connection

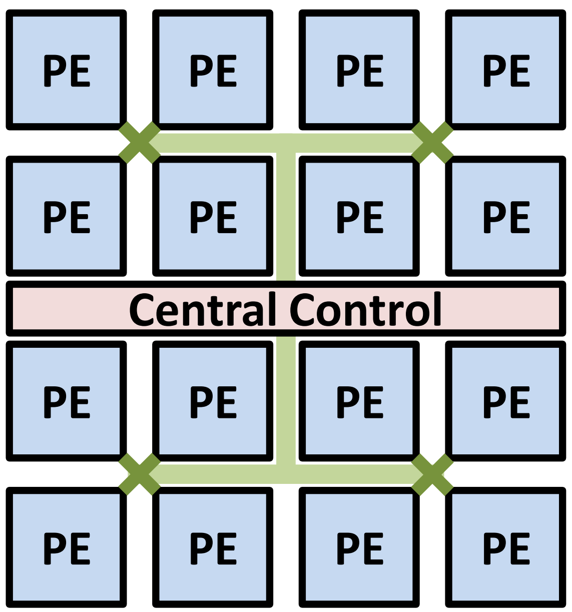
Real Time
No network delay high frame rate

Low Power
High energy efficiency that preserves battery

Solution: Everything on Chip

- We present the **sparse, indirectly indexed, weight shared** MxV accelerator.
- Large DNN models fit on-chip SRAM, 120× energy savings.
- EIE exploits the sparsity of activations (30% non-zero).
- EIE works on compressed model (30x model reduction)
- Distributed both storage and computation across multiple PEs, which achieves load balance and good scalability.
- Evaluated EIE on a wide range of deep learning models, including CNN for object detection, LSTM for natural language processing and image captioning. We also compare EIE to CPUs, GPUs, and other accelerators.

Distribute Storage and Processing



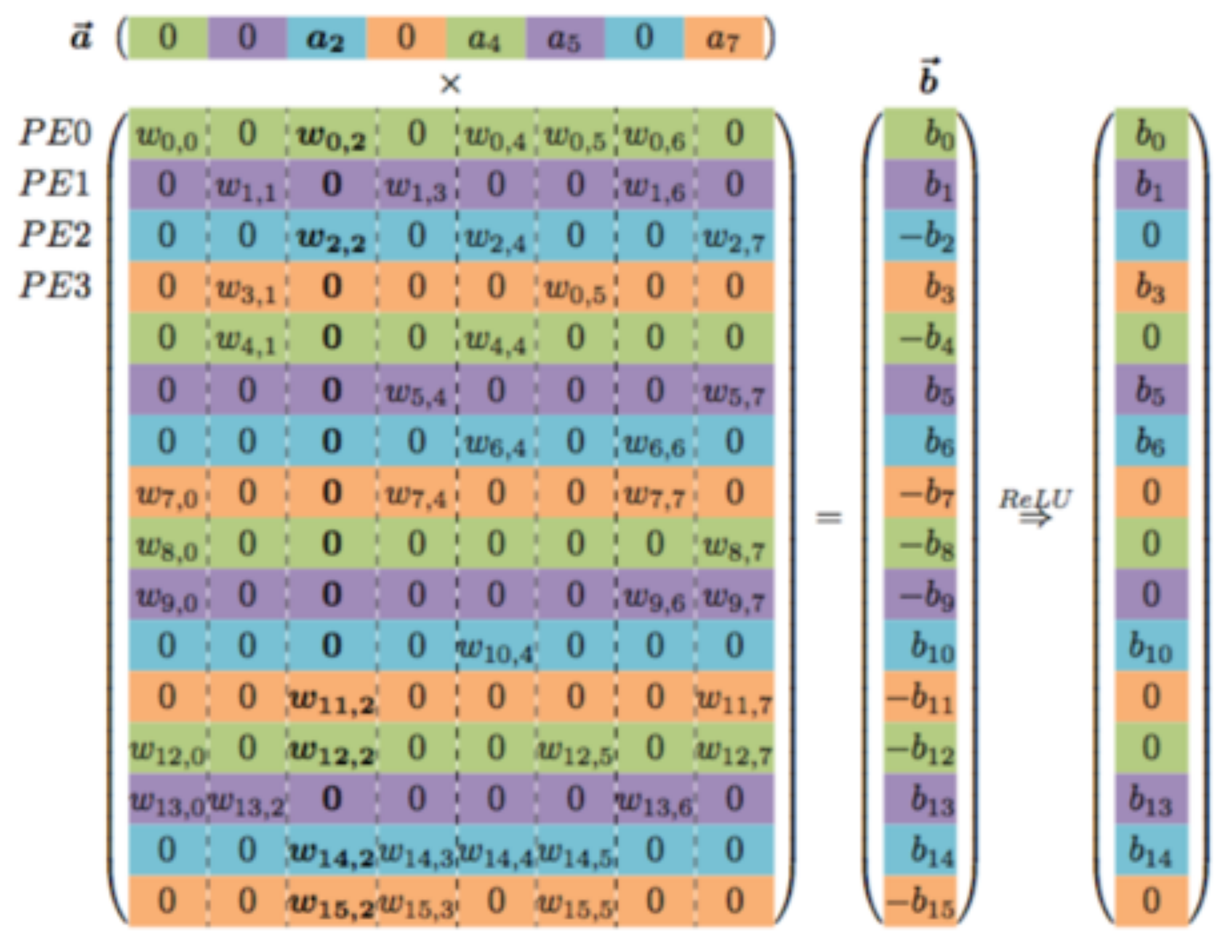
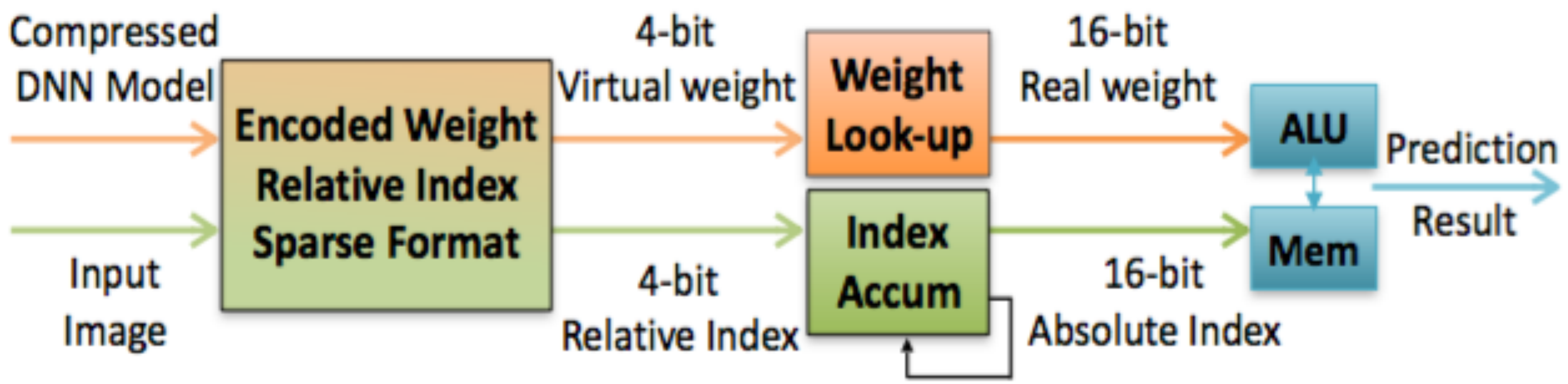


Figure 2: Matrix W and vectors a and b are interleaved over 4 PEs. Elements of the same color are stored in the same PE.

Inside each PE:



Evaluation

1. Cycle-accurate C++ simulator. Two abstract methods: Propagate and Update. Used for DSE and verification.

2. RTL in Verilog, verified its output result with the golden model in Modelsim.

3. Synthesized EIE using the Synopsys Design Compiler (DC) under the TSMC 45nm GP standard VT library with worst case PVT corner.

4. Placed and routed the PE using the Synopsys IC compiler (ICC). We used Cacti to get SRAM area and energy numbers.

5. Annotated the toggle rate from the RTL simulation to the gate-level netlist, which was dumped to switching activity interchange format (SAIF), and estimated the power using Prime-Time PX.

Baseline and Benchmark

- CPU: Intel Core-i7 5930k
- GPU: NVIDIA TitanX GPU
- Mobile GPU: Jetson TK1 with NVIDIA

Table 3: Benchmark from state-of-the-art DNN models

Layer	Size	Weight%	Act%	FLOP%	Description
Alex-6	9216, 4096	9%	35.1%	3%	Compressed AlexNet [1] for large scale image classification
Alex-7	4096, 4096	9%	35.3%	3%	
Alex-8	4096, 1000	25%	37.5%	10%	
VGG-6	25088, 4096	4%	18.3%	1%	Compressed VGG-16 [3] for large scale image classification and object detection
VGG-7	4096, 4096	4%	37.5%	2%	
VGG-8	4096, 1000	23%	41.1%	9%	
NT-We	4096, 600	10%	100%	10%	Compressed NeuralTalk [7] with RNN and LSTM for automatic image captioning
NT-Wd	600, 8791	11%	100%	11%	
NTLSTM	1201, 2400	10%	100%	11%	

Layout of an EIE PE

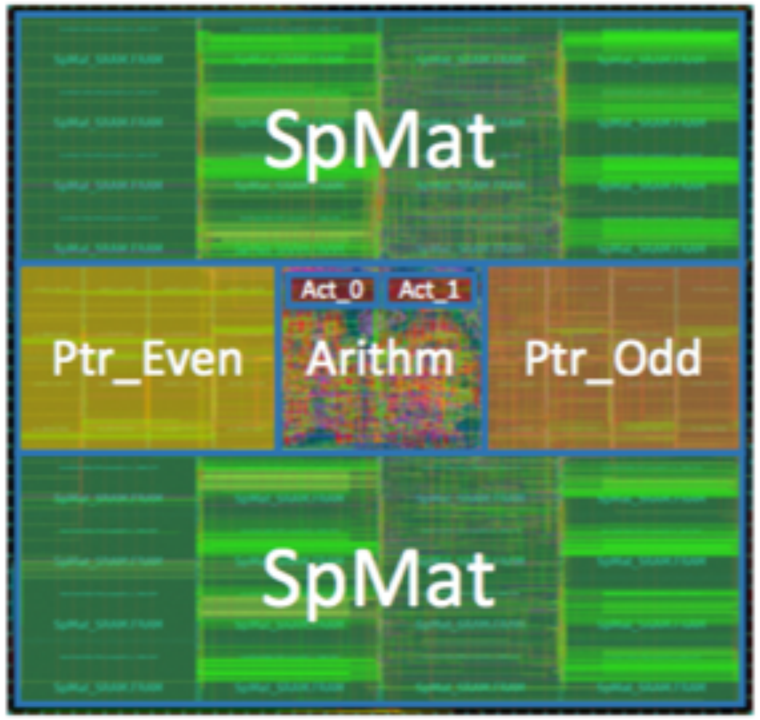
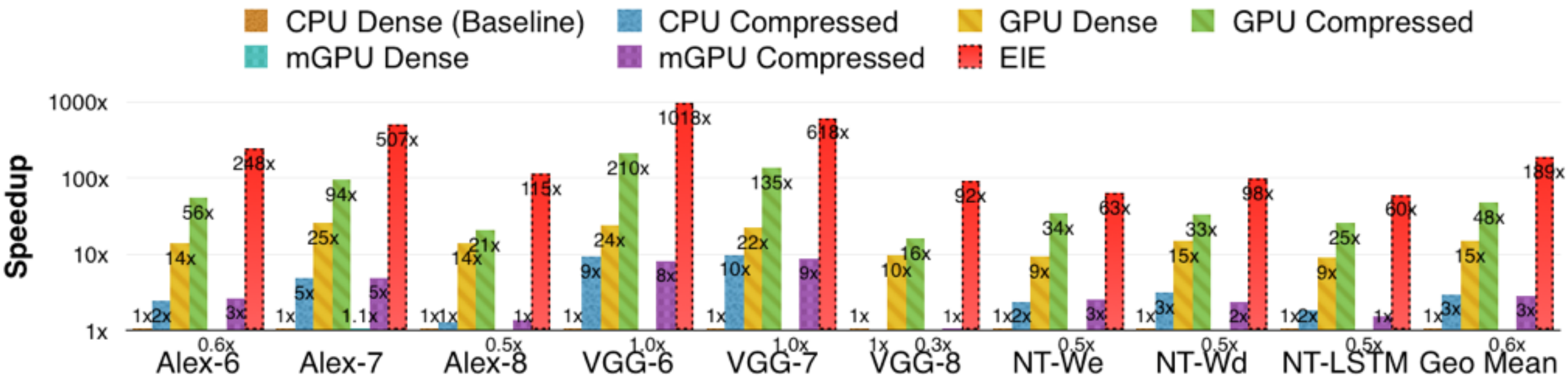


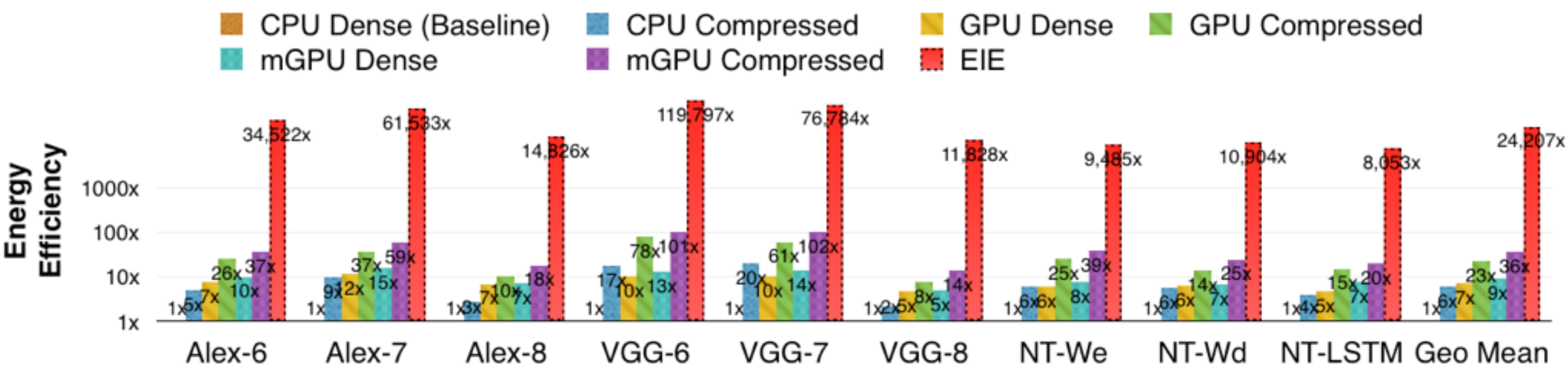
Figure 7: Layout of one PE in EIE under TSMC 45nm process.

	Power (mW)	(%)	Area (μm^2)	(%)
Total	9.157		638,024	
memory	5.416	(59.15%)	594,786	(93.22%)
clock network	1.874	(20.46%)	866	(0.14%)
register	1.026	(11.20%)	9,465	(1.48%)
combinational	0.841	(9.18%)	8,946	(1.40%)
filler cell			23,961	(3.76%)
Act_queue	0.112	(1.23%)	758	(0.12%)
PtrRead	1.807	(19.73%)	121,849	(19.10%)
SpmatRead	4.955	(54.11%)	469,412	(73.57%)
ArithmUnit	1.162	(12.68%)	3,110	(0.49%)
ActRW	1.122	(12.25%)	18,934	(2.97%)
filler cell			23,961	(3.76%)

Table 2: The implementation results of one PE in EIE and the breakdown by component type (line 3-7), by module (line 8-13). The critical path of EIE is 1.15*ns*

Result: Speedup / Energy Efficiency





Result: Speedup

Table 4: Performance comparison between CPU, GPU, mobile GPU implementations and EIE.

Platform	Batch Size	Matrix Type	AlexNet			VGG16			NT-		
			FC6	FC7	FC8	FC6	FC7	FC8	We	Wd	LSTM
CPU (Core i7-5930k)	1	dense	7516.2	6187.1	1134.9	35022.8	5372.8	774.2	605.0	1361.4	470.5
		sparse	3066.5	1282.1	890.5	3774.3	545.1	777.3	261.2	437.4	260.0
	64	dense	318.4	188.9	45.8	1056.0	188.3	45.7	28.7	69.0	28.8
		sparse	1417.6	682.1	407.7	1780.3	274.9	363.1	117.7	176.4	107.4
GPU (Titan X)	1	dense	541.5	243.0	80.5	1467.8	243.0	80.5	65	90.1	51.9
		sparse	134.8	65.8	54.6	167.0	39.8	48.0	17.7	41.1	18.5
	64	dense	19.8	8.9	5.9	53.6	8.9	5.9	3.2	2.3	2.5
		sparse	94.6	51.5	23.2	121.5	24.4	22.0	10.9	11.0	9.0
mGPU (Tegra K1)	1	dense	12437.2	5765.0	2252.1	35427.0	5544.3	2243.1	1316	2565.5	956.9
		sparse	2879.3	1256.5	837.0	4377.2	626.3	745.1	240.6	570.6	315
	64	dense	1663.6	2056.8	298.0	2001.4	2050.7	483.9	87.8	956.3	95.2
		sparse	4003.9	1372.8	576.7	8024.8	660.2	544.1	236.3	187.7	186.5
EIE	**Theoretical Time**		**28.1**	**11.7**	**8.9**	**28.1**	**7.9**	**7.3**	**5.2**	**13.0**	**6.5**
	Actual Time		**30.3**	**12.2**	**9.9**	**34.4**	**8.7**	**8.4**	**8.0**	**13.9**	**7.5**

Scalability

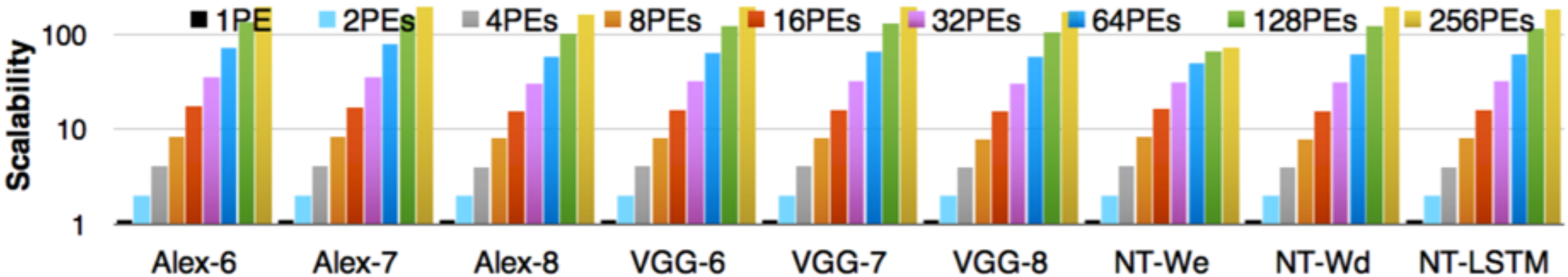


Figure 11: System scalability. The average efficiency of single PE finally decreases as the number of PEs increases. On some very sparse layers, having more PEs initially increases the efficiency a bit.

Useful Computation / Load Balance

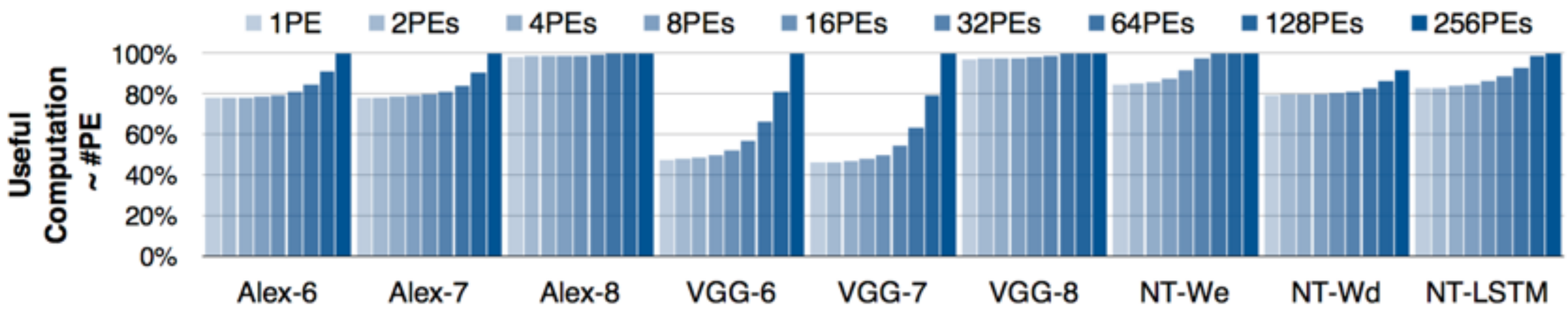


Figure 12: The number of padding zeros decreases as the number of PEs goes up, leading to less padding zeros and better compute efficiency.

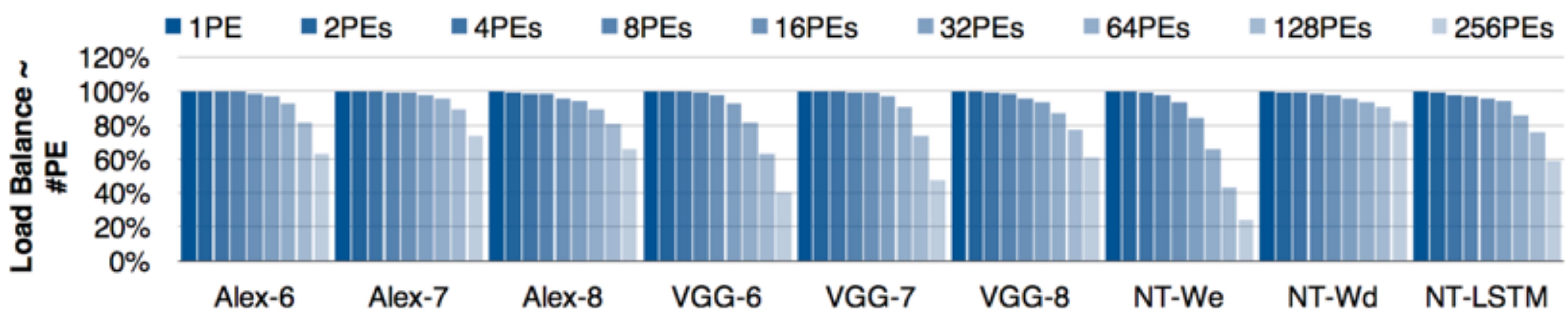


Figure 13: Load efficiency is measured by the ratio of stalled cycles over total cycles in ALU. More PEs lead to worse load imbalance accompanied with less load efficiency. This explains the sub-linear speedup at large number of PEs.

Load Balance

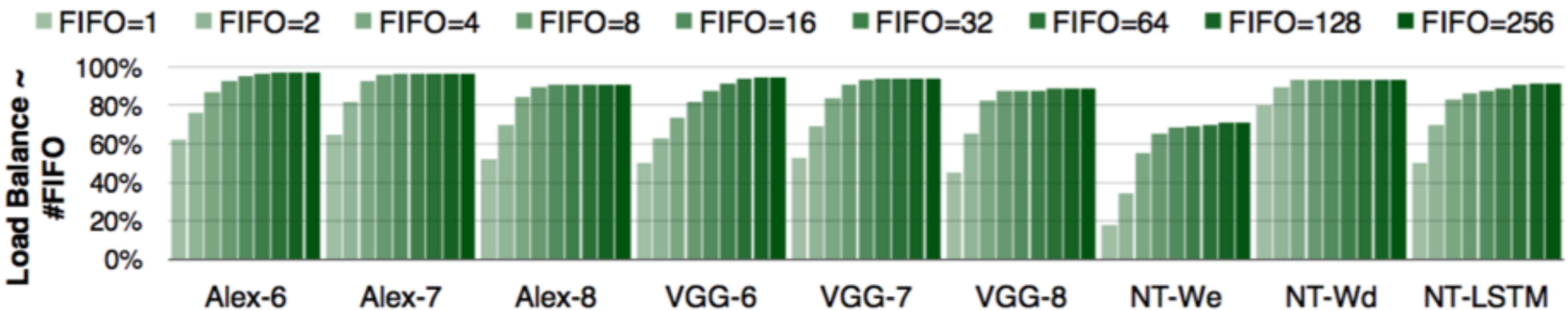


Figure 8: Load efficiency improves as FIFO size increases. When the size is larger than eight, the marginal gain quickly diminishes. So we choose FIFO depth to be eight.

Design Space Exploration

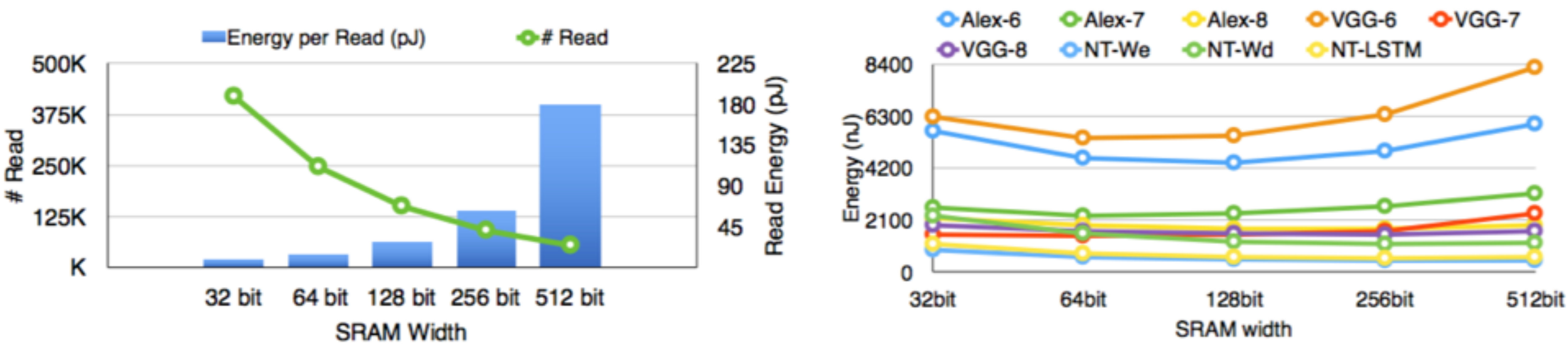


Figure 9: Left: SRAM read energy and number of reads benchmarked on AlexNet. Right: Multiplying the two curves in the left gives the total energy consumed by SRAM read.

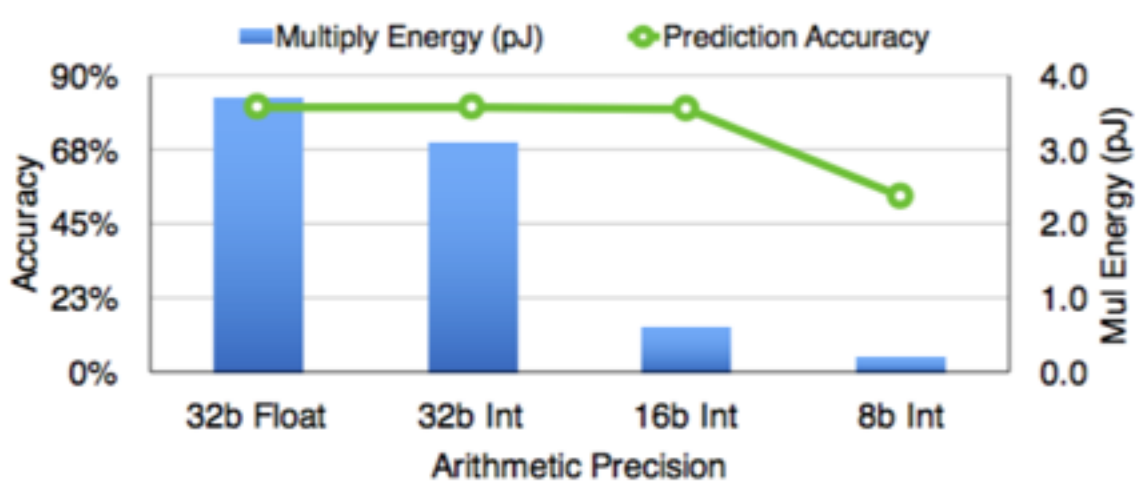


Figure 10: Prediction accuracy and multiplier energy with different arithmetic precision.

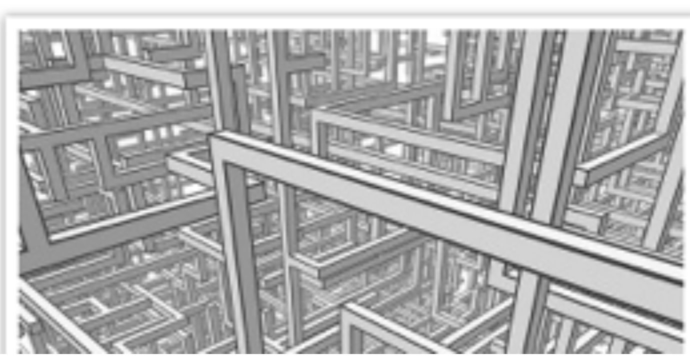
Media Coverage

THENEXTPLATFORM

HOME COMPUTE STORE CONNECT CONTROL CODE ANALYZE HPC ENTERPRIS

EMERGENT CHIP VASTLY ACCELERATES DEEP NEURAL NETWORKS

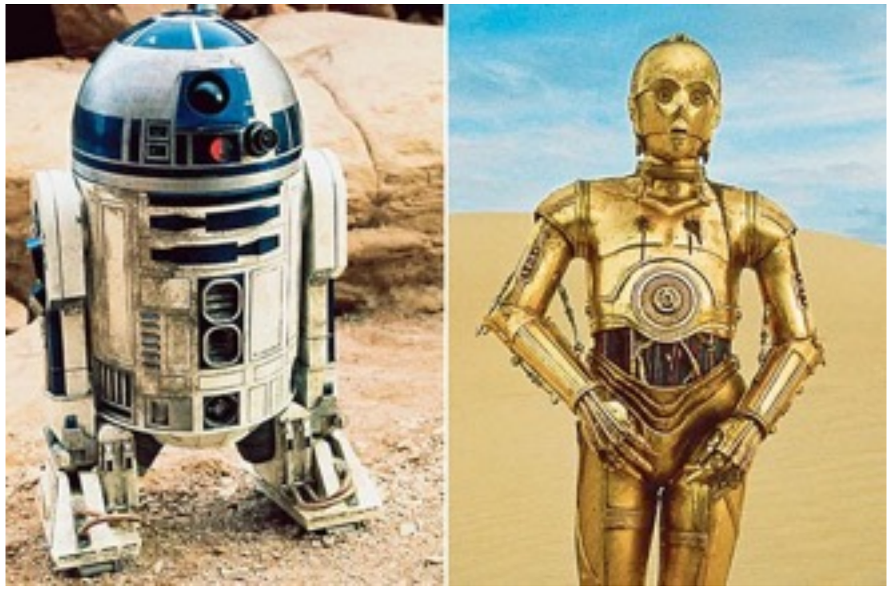
December 8, 2015 Nicole Hemsoth



Stanford University PhD candidate, Song Han, who works under advisor and networking pioneer, Dr. Bill Dally, responded in a most soft-spoken and thoughtful way to the question of whether the coupled software and hardware architecture he developed might change the world.

http://www.nextplatform.com/2015/12/08/emergent-chip-vastly-accelerates-deep-neural-networks/

Hardware for Deep Learning



PC | **Mobile** | **Intelligent Mobile**

intel

?

Computation → Mobile Computation → Intelligent Mobile Computation

Conclusion

- We present EIE, an energy-efficient engine optimized to operate on compressed deep neural networks.

- By leveraging sparsity in both the activations and the weights, EIE reduces the energy needed to compute a typical FC layer by 3,000×.

- Three factors for energy saving:
  matrix is compressed by 35×;
  DRAM => SRAM: 120×;
  take advantage of sparse activation: 3×;

Thank you!

songhan@stanford.edu