Lecture 5

Model Compression

Tien-Fu Chen

Dept. of Computer Science and Information Engineering

National Chiao Tung Univ.

CNN is the most complicated computation

- 90 % of the total computation time of CNN: convolutional layer

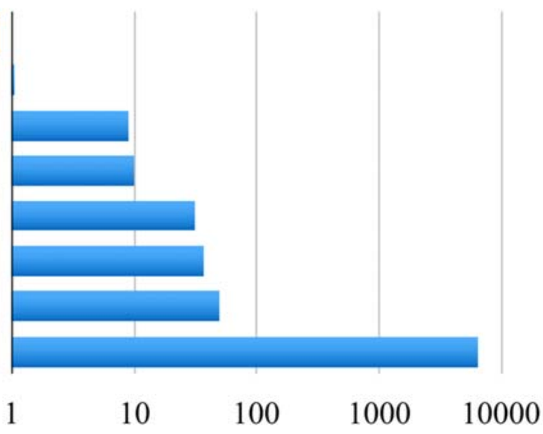
	AlexNet(Gops)	VGG16(Gops)
Convolutional layer	**1.52**	**30.7**
Fully Connected layer	0.12	0.5

  – **Accelerating Convolution** => Accelerating CNN

- Why smaller model?

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

Relative Energy Cost



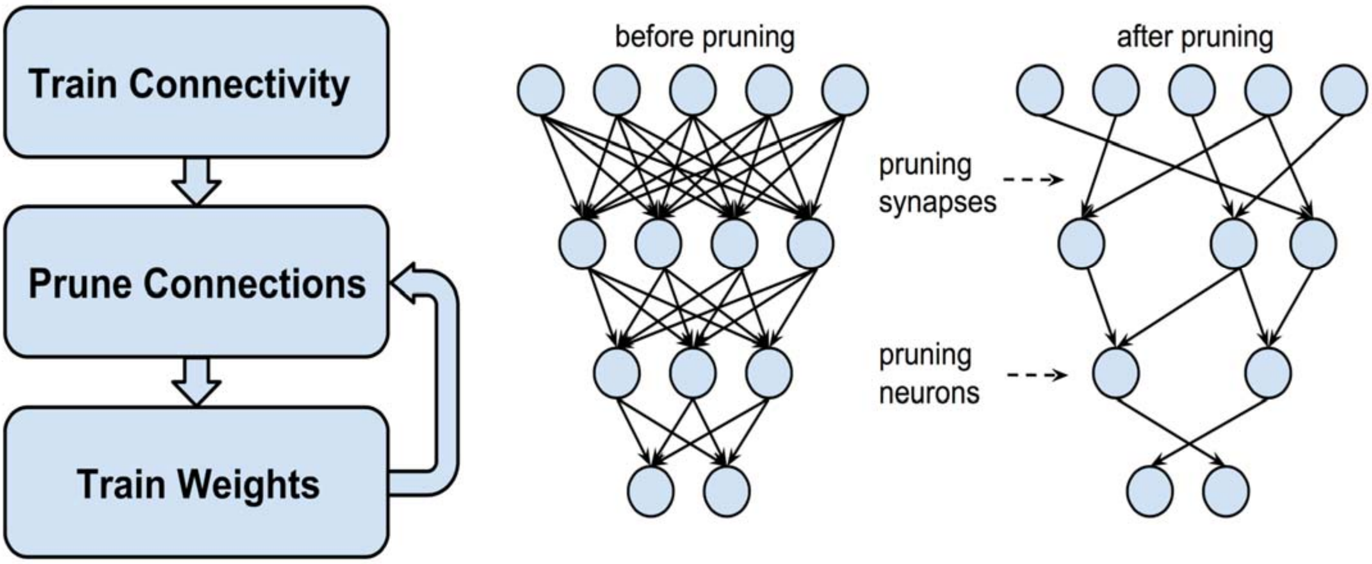
Source: http://isca2016.eecs.umich.edu/wp-content/uploads/2016/07/4A-1.pdf

Network Compression and Speedup

- Network Pruning
- Parameter Sharing, Quantization and Binarization
- Pruning + Quantization + Encoding
- Low Rank Matrix/Tensor Factorization
  - Singular Value Decomposition (SVD)
- Knowledge Distillation

- Where to Compress:
  - Weights
  - Activations
  - Gradients

Magnitude-based method: Iterative Pruning + Retraining



Han, Song, et al. "Learning both weights and connections for efficient neural network." NIPS. 2015.

Magnitude-based method: Iterative Pruning + Retraining (Algorithm)

1. Choose a neural network architecture.
2. Train the network until a reasonable solution is obtained.
3. Prune the weights of which magnitudes are less than a threshold τ.
4. Train the network until a reasonable solution is obtained.
5. Iterate to step 3.

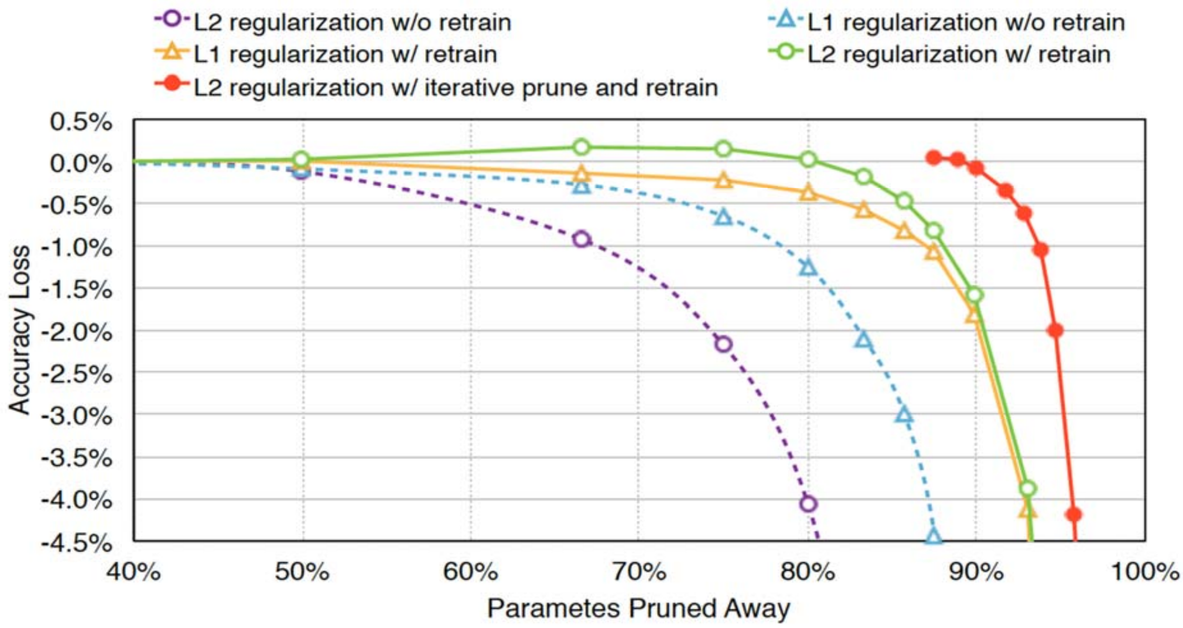
Han, Song, et al. "Learning both weights and connections for efficient neural network." NIPS. 2015.

Magnitude-based method: Iterative Pruning + Retraining (Experiment: Overall)

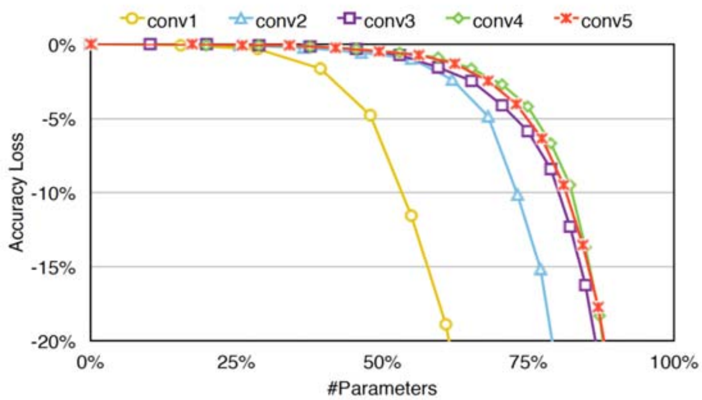
Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	12X
LeNet-300-100 Pruned	1.59%	-	22K	
LeNet-5 Ref	0.80%	-	431K	12X
LeNet-5 Pruned	0.77%	-	36K	
AlexNet Ref	42.78%	19.73%	61M	9X
AlexNet Pruned	42.77%	19.67%	6.7M	
VGG-16 Ref	31.50%	11.32%	138M	13X
VGG-16 Pruned	31.34%	10.88%	10.3M	

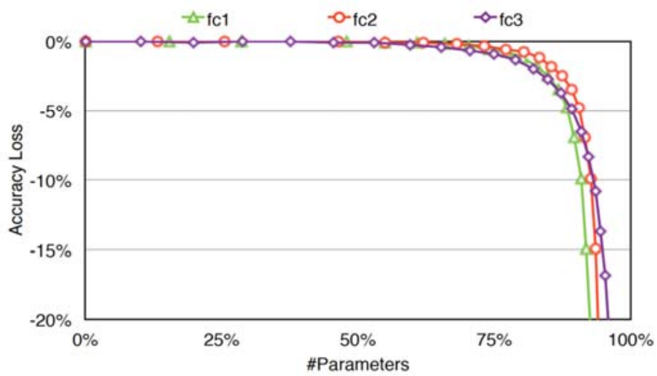
Han, Song, et al. "Learning both weights and connections for efficient neural network." NIPS. 2015.

Accuracy-pruning trade-off, effect of retraining and regularization

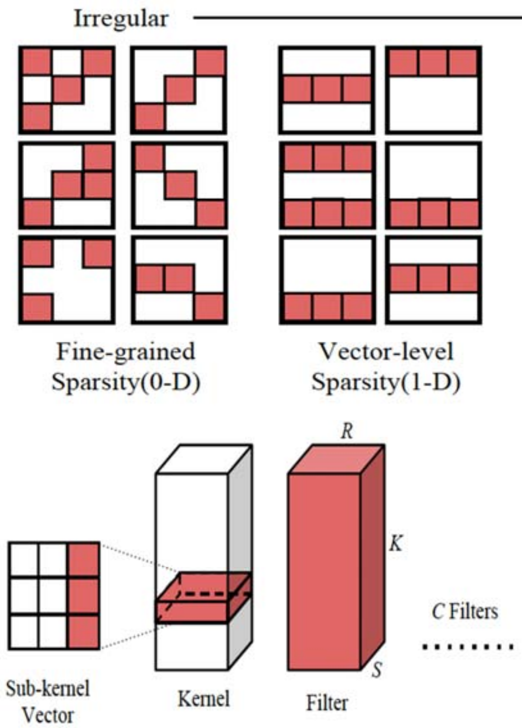


Layer type vs. sensitivity





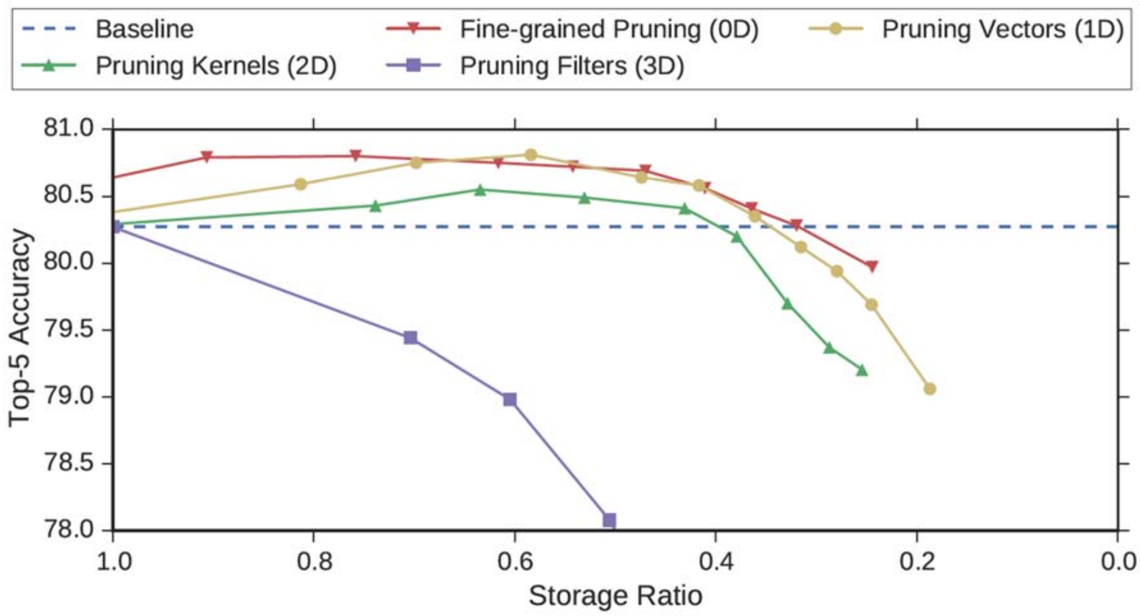
Irregular fine-grained vs. regular coarse grained pruning



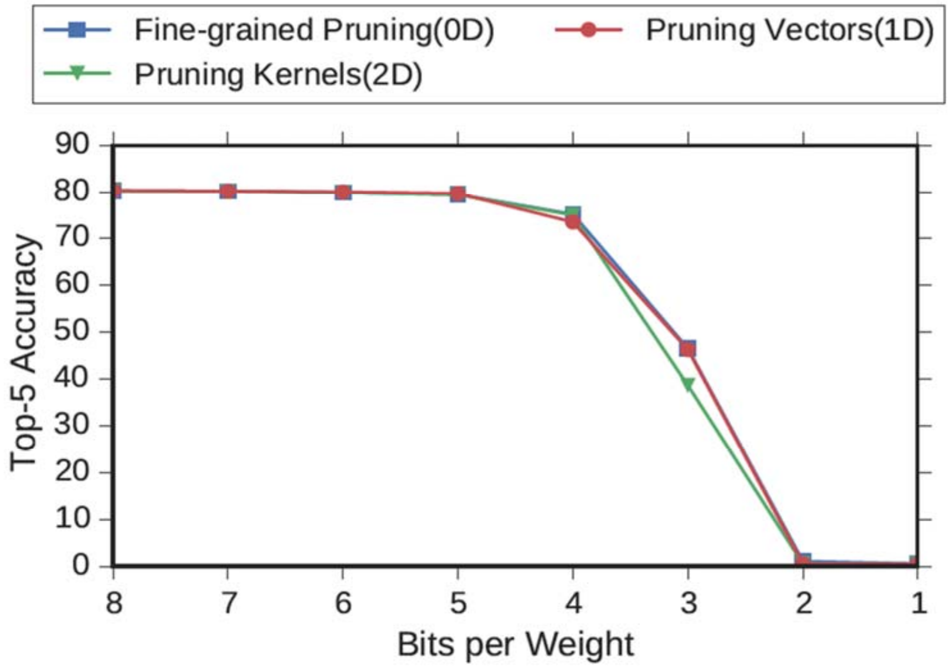

```
Weights = Array(C, K, R, S)

# Case: Dimension-level granularity
Filter(3-Dim) = Weights[c, :, :, :]
Kernel(2-Dim) = Weights[c, k, :, :]
Vector(1-Dim) = Weights[c, k, r, :]
Fine-grain(0-Dim) = Weights[c, k, r, s]
```

Exploring the Regularity of Sparse Structure in Convolutional Neural Networks. NIPS 2017

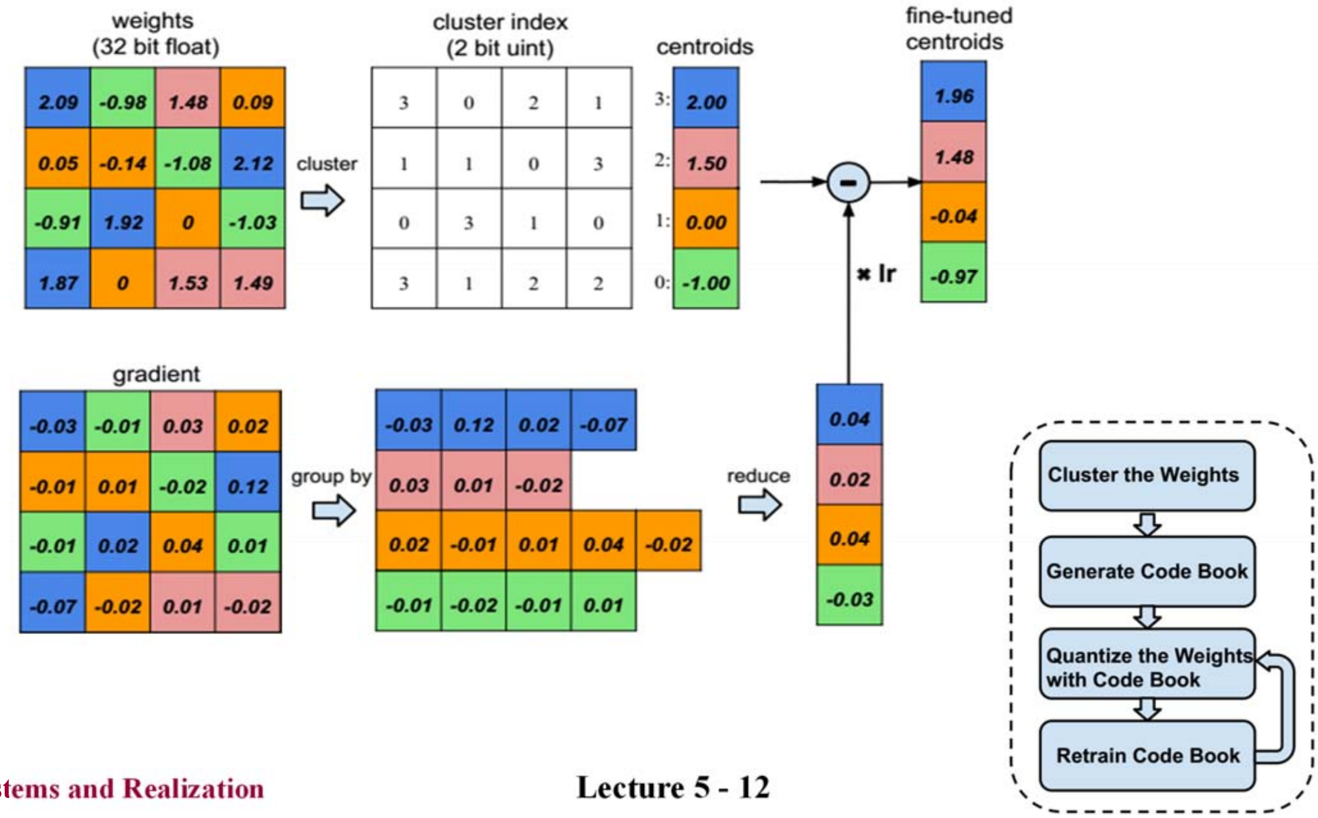
What granularity is best for model size (Alexnet)?



Quantization tests



Weight Sharing – Training Quantization

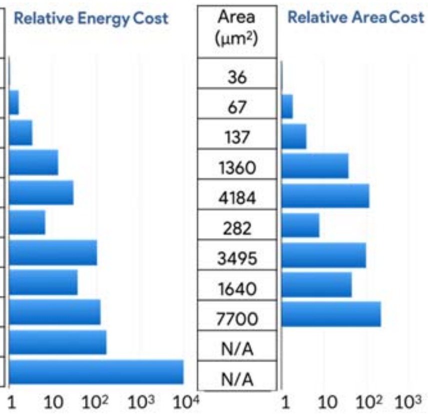


Quantization

- Neural networks are typically trained in floating point format (FP32) mainly for representation of small gradients (used to update weights in back-propagation)
- For inference, lower-precision is enough
- Benefits: reduce the area and energy cost fo storage & computation

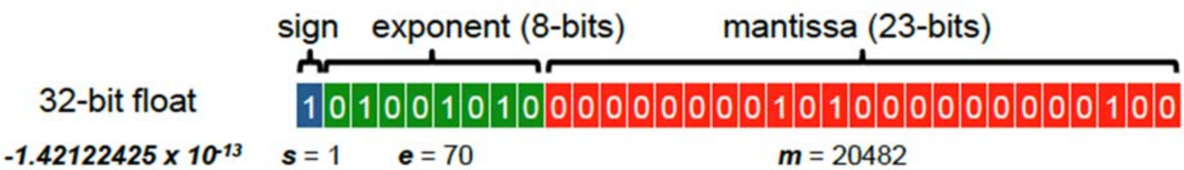
Rough energy & area cost for various operations in 45nm 0.9V

Operation:	Energy (pJ)	Area (µm²)
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



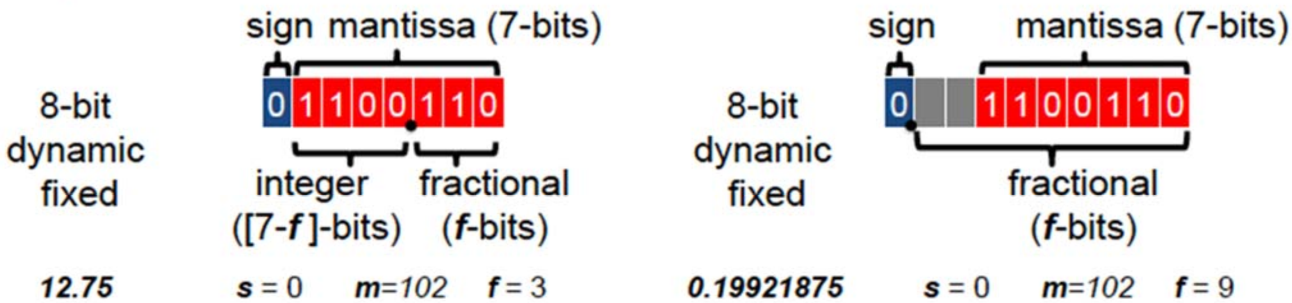
Mark Horowitz. "1.1 computing's energy problem (and what we can do about it)." ISSCC 2014.

Various methods of number representations



(a) 32-bit floating point example

$$(-1)^s \times m \times 2^{(e-127)}$$



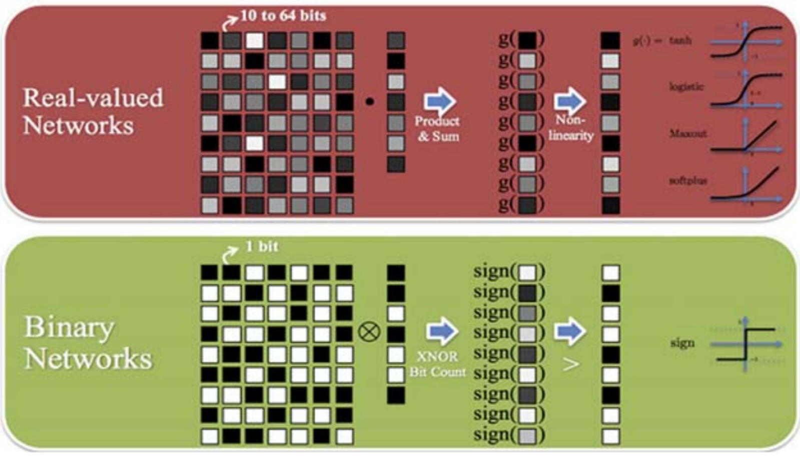
(b) 8-bit dynamic fixed point examples

$$(-1)^s \times m \times 2^{-f}$$

Binary Nets (1-bit)

- **Binary Connect (BC)**
  - **Weights: {-1, 1}, Activations: 32-bit float**
  - **MAC -> addition / subtraction**
  - **Accuracy loss: 19% on AlexNet**
- **Binarized Neural Networks (BNN)**
  - **Weights: {-1, 1}, Activations: {-1, 1}**
  - **MAC -> XNOR**
  - **Accuracy loss: 29% on AlexNet**

Can't quantize both weights & activations or suffers too much accuracy loss



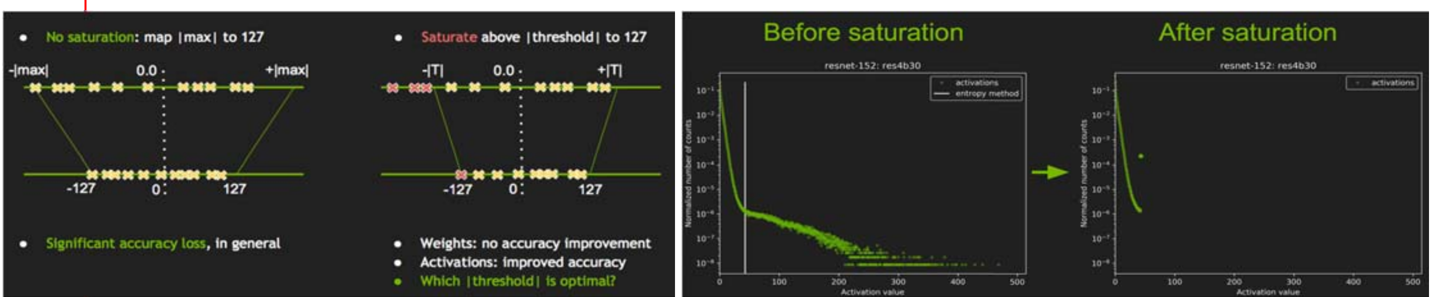
Matthieu Courbariaux, et al. "Binaryconnect." & "Binarized neural networks." NIPS 2015 & 2016.

8-bit Quantization in TensorRT

- Linear, symmetric mapping between FP32 and INT8

  Tensor Values = FP32 scale factor * INT8 value ~~+ FP32 bias~~

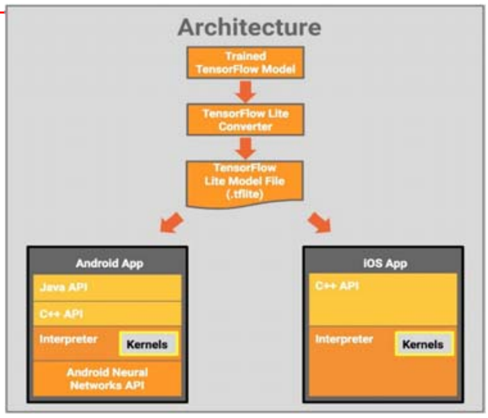
- Utilize Kullback-Leibler (KL) divergence (relative entropy) to decide thresholds that minimize loss of information for activations of each layer
- **Under 1% accuracy loss** for most CNNs on ImageNet
- **Doesn't require any additional fine-tuning or retraining**

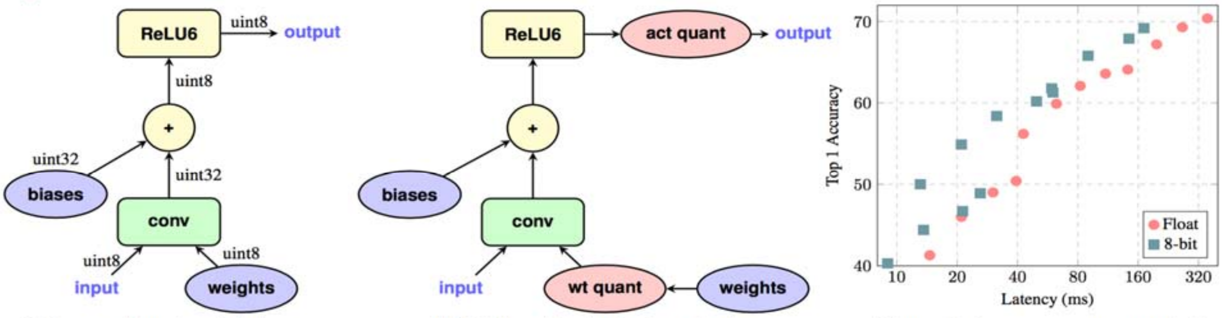


Szymon Migacz. "8-bit Inference with TensorRT." GTC 2017.

8-bit Quantization in TensorFlow

- For both training and inference
- TensorFlow Lite targets inference on mobile phones
- Google's gemmlowp library provides optimized 8-bit GEMM for ARM NEON





(a) Integer-arithmetic-only inference (b) Training with simulated quantization (c) ImageNet latency-vs-accuracy tradeoff

Benoit Jacob, et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference." CVPR 2018.

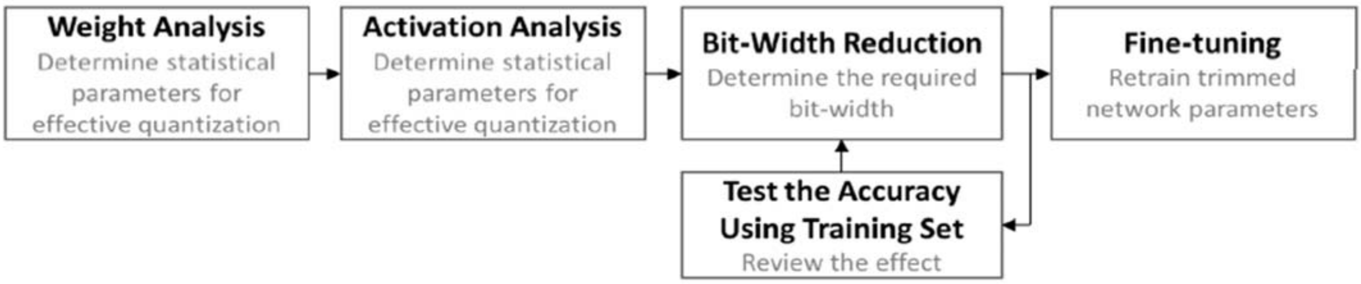
Recent Works on Quantization

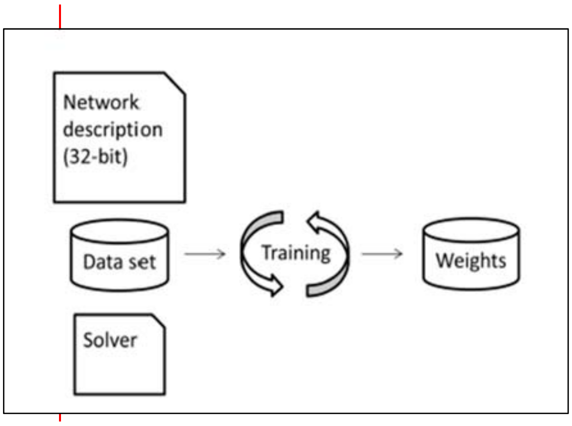
Category	Method	Bits for weights	Bits for activations	Accuracy loss @AlexNet (%)	Need fine-tune/retrain
Dynamic fixed point	Ristretto	8	8	0.9	v
Low-bit	BC	1	32	19.2	v
	BNN	1	1	29.8	v
	BWN	1*	32	0.8	v
	XNOR-Net	1	1*	11	v
	TWN	2*	32	3.7	v
	TTQ	2*	32	0.6	v
Non-uniform	LogNet	5(conv), 4(fc)	4	3.2	v
	Weight Sharing	8(conv), 4(fc)	16	0	v
Linear	TensorRT	8	8	0.03	x
	TensorFlow	8	8	-	v
	V-Quant	4+16**	4+16**	< 1%	v
	Ours	4+8	4+8	< 1%	x

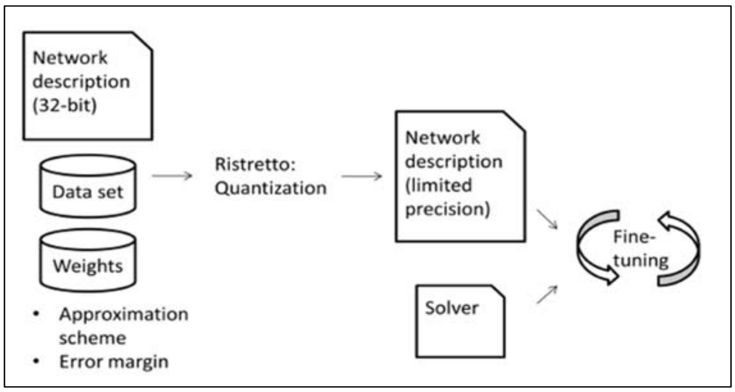
*fp32 for first and last layers

****fp16 for outlier weights**

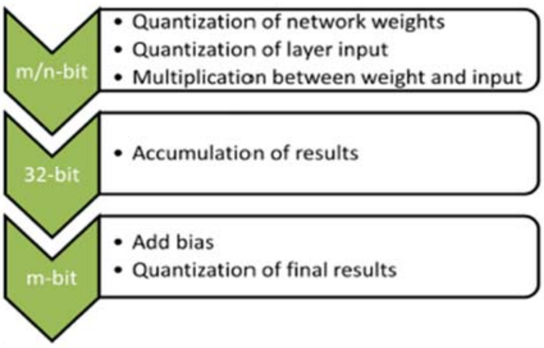
Ristretto: a quantization tool built on Caffe







Ristretto quantization methods: fixed point, dynamic fixed point, minifloat

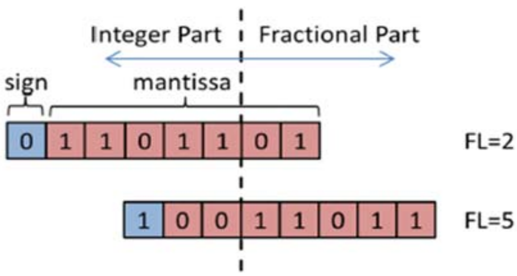


$$round(x) = \begin{cases} \lfloor x \rfloor, & \text{if } \lfloor x \rfloor \le x \le x + \frac{\epsilon}{2} \\ \lfloor x \rfloor + \epsilon, & \text{if } \lfloor x \rfloor + \frac{\epsilon}{2} < x \le x + \epsilon \end{cases}$$

Deterministic rounding

$$round(x) = \begin{cases} \lfloor x \rfloor, & \text{w.p. } 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon, & \text{w.p. } \frac{x - \lfloor x \rfloor}{\epsilon} \end{cases}$$

Stochastic rounding



Dynamic fixed point

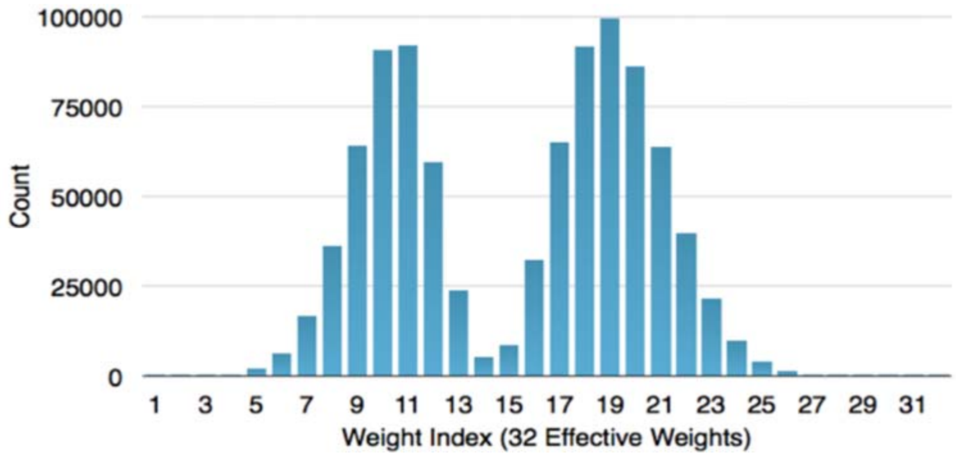
Summary of Quantization results

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

Network: Alexnet   Dataset: Imagenet   Accuracy measured: Top-5 error

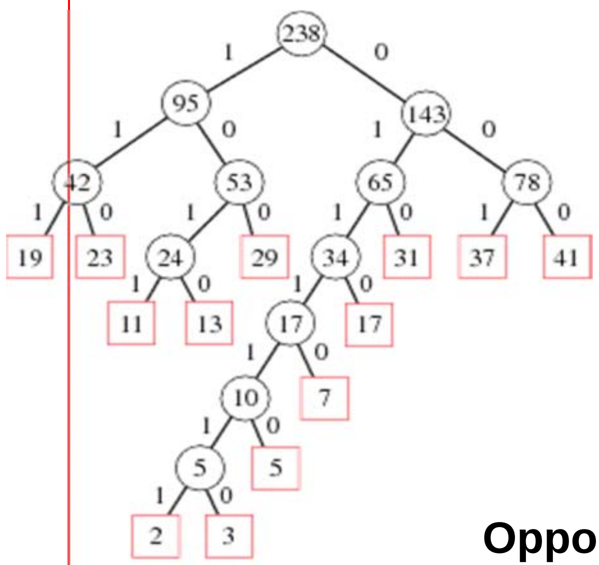
From V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," arXiv, 2017.

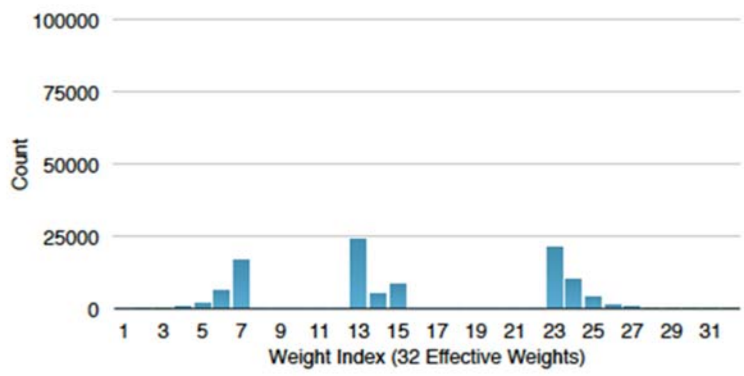
Huffman Coding



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

Huffman encoding

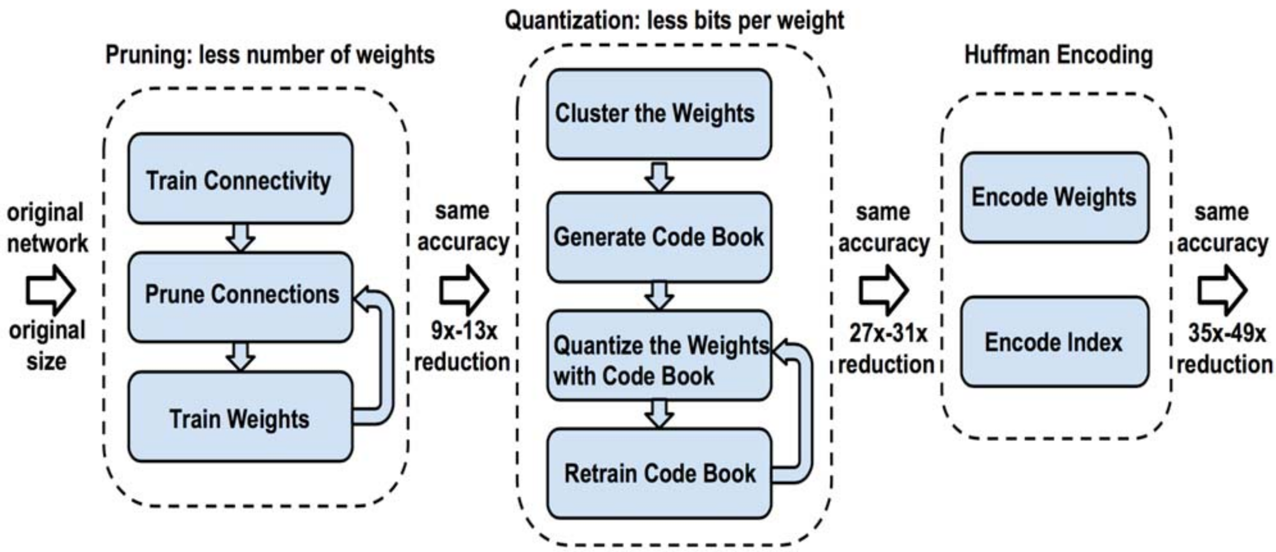




Opportunity for Huffman Encoding, as seen from the empirical values of weights

Pruning + Quantization + Encoding: Deep Compression

- **Weight pruning, quantization, and encoding are independent. We can use all three methods together for better compression ratio.**



Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding (ICLR 2016).

Pruning + Quantization + Encoding: Deep Compression

Table 1: The compression pipeline can save 35× to 49× parameter storage with no loss of accuracy.

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×
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	**11.3 MB**	49×

Table 2: Compression statistics for LeNet-300-100. P: pruning, Q:quantization, H:Huffman coding.

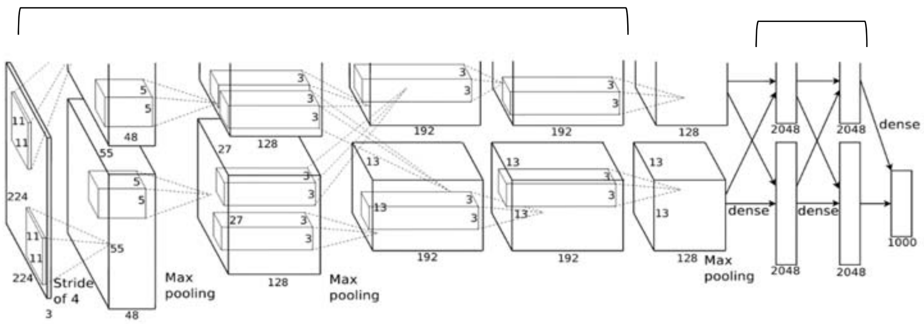
Layer	#Weights	Weights% (P)	Weight bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
ip1	235K	8%	6	4.4	5	3.7	3.1%	2.32%
ip2	30K	9%	6	4.4	5	4.3	3.8%	3.04%
ip3	1K	26%	6	4.3	5	3.2	15.7%	12.70%
Total	266K	8%(12×)	6	5.1	5	3.7	3.1% (**32**×)	2.49% (**40**×)

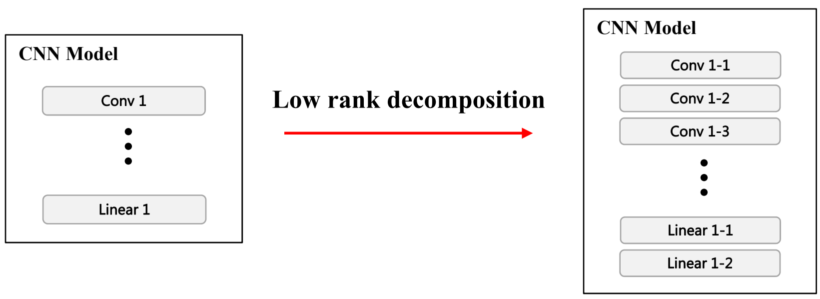
Table 3: Compression statistics for LeNet-5. P: pruning, Q:quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weight bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	0.5K	66%	8	7.2	5	1.5	78.5%	67.45%
conv2	25K	12%	8	7.2	5	3.9	6.0%	5.28%
ip1	400K	8%	5	4.5	5	4.5	2.7%	2.45%
ip2	5K	19%	5	5.2	5	3.7	6.9%	6.13%
Total	431K	8%(12×)	5.3	4.1	5	4.4	3.05% (**33**×)	2.55% (**39**×)

Low Rank Decomposition

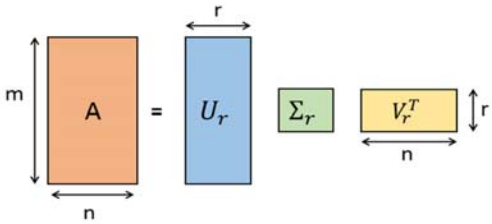
Conv layer , tucker decomposition **FC layer , SVD**





Low Rank Decomposition (cont'd)

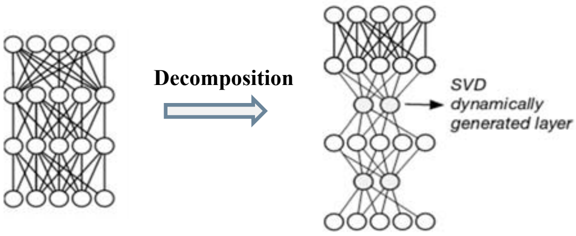
SVD : approximate weight matrix (A) as $U\Sigma V^T$



$$A_{mxn} = U_{mxm}\Sigma_{mxn}V^T_{nxn} = U_{mxr}\Sigma_{rxr}V^T_{rxn} \approx U_{mxk}\Sigma_{kxk}V^T_{kxn}$$

$m \, x \, n$ $\qquad$ $(m+n+1)r$ $\qquad$ $(m+n+1)k$

Rank controls the trade-off between performance(speed, memory, power) and accuracy loss

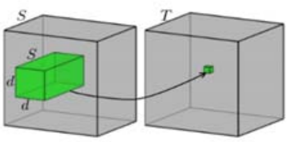


Tucker Decomposition : 1. higher order extension of SVD
2. compress the convolutional layers

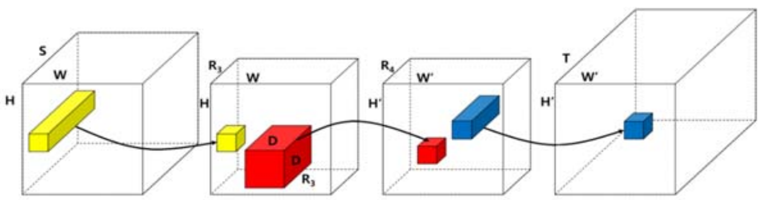
ND Lane, et al. "DeepX: A Software Accelerator for Low-Power Deep Learning." IPSN, 2016
Yong-Deok Kim, et al. "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Application" ICLR, 2016

Tucker Decomposition

- After training, perform **low-rank approximation by applying tensor decomposition** to weight kernel; then fine-tune weights for accuracy
- Can be applied to convolution and fully-connected layers
- Significantly reduce the resource cost (memory, latency, power) for inference



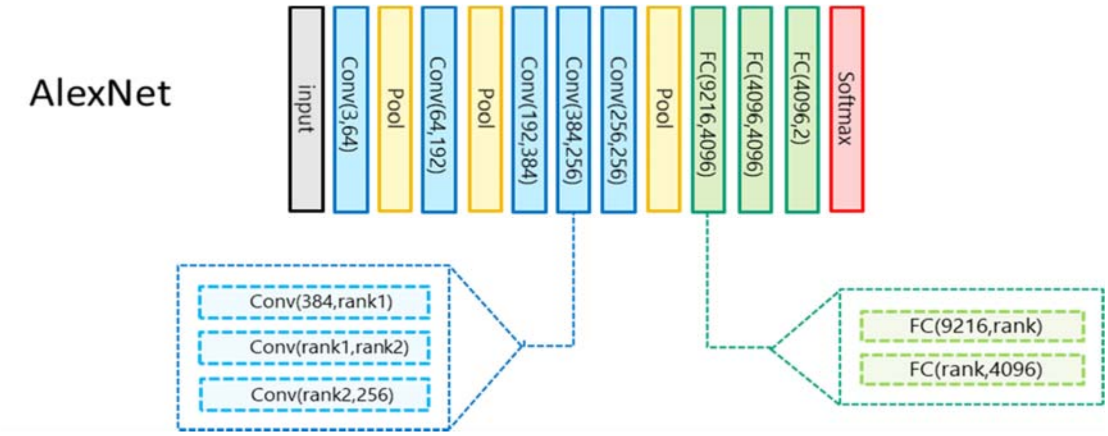
(a) Full convolution



Y.-D. Kim, et, al. "Compression of deep convolutional neural networks for fast and low power mobile applications." ICLR, 2016.

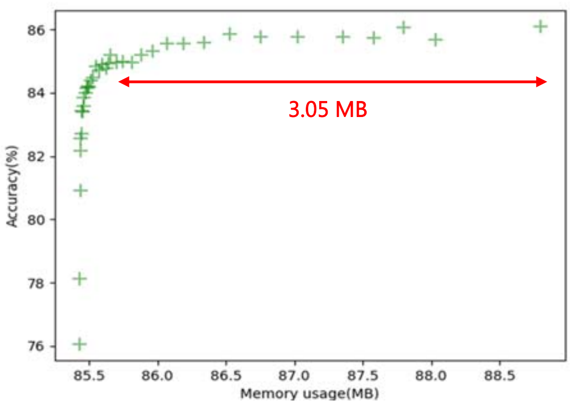
Variable Rank Decomposed AlexNet

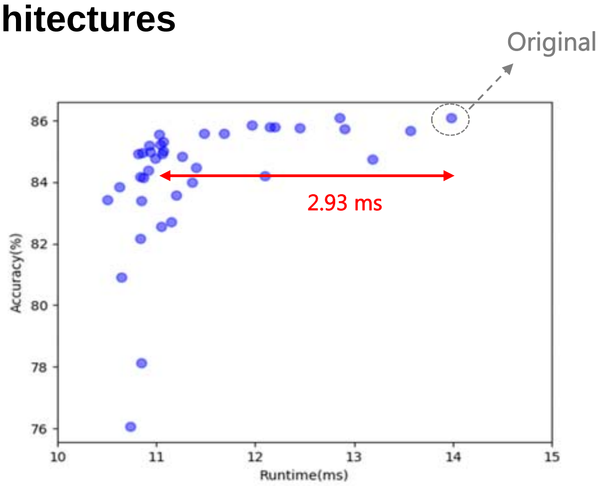
- **Adjust the rank of each layer to fit hardware resources**
- **How do we find the optimal rank that can significantly reduce resource requirement without accuracy loss ?**



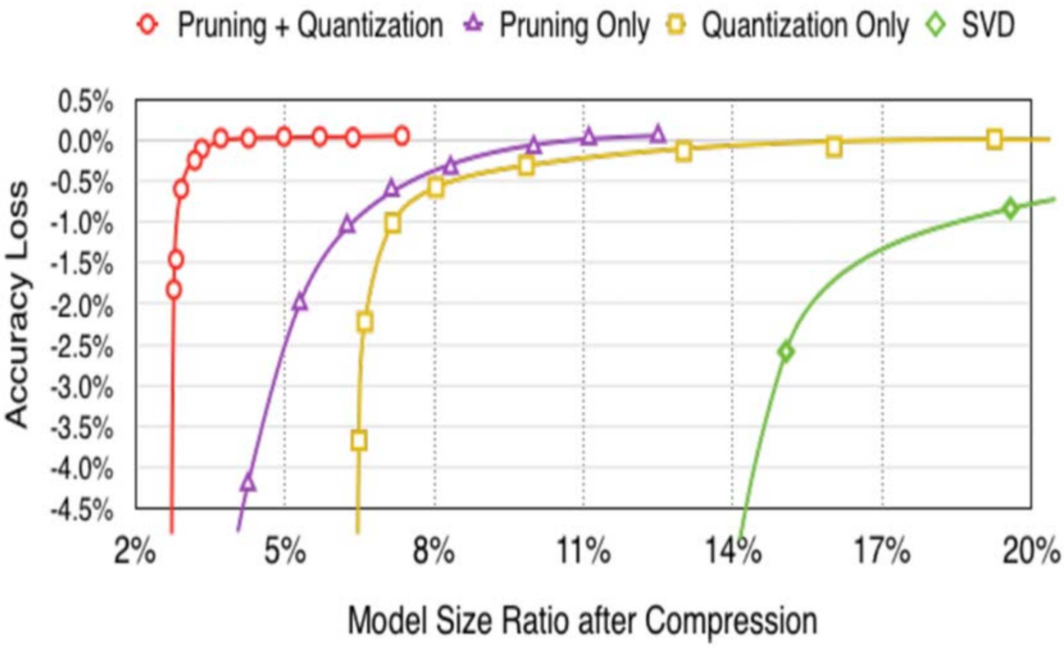
Accuracy vs. Performance for Variable Rank Decomposed AlexNet on CIFAR-10

- **Use grid search to enumerate all possible rank in one conv layer, and there exists optimal points**
- **For example, we need verify 384x256 combination in the biggest layer of AlexNet.**
- **We need a hyperparameters Planner that is more efficient than grid search to explore optimal CNN architectures**



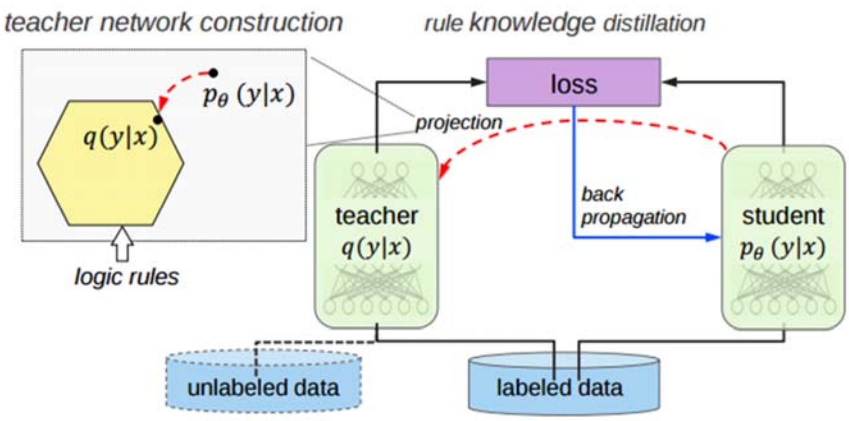


Benefits of pruning



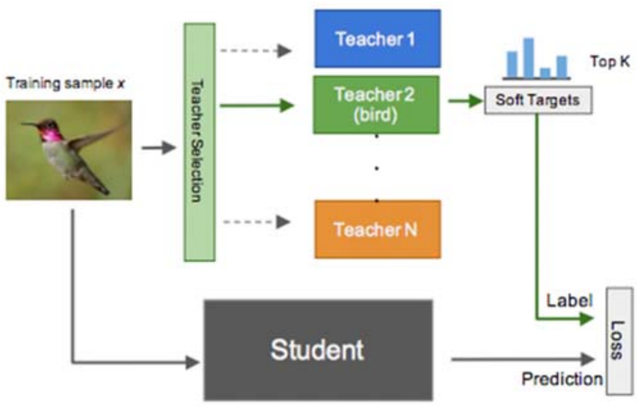
[Han, Mao, Dally, ICLR 2016]

Knowledge Distillation

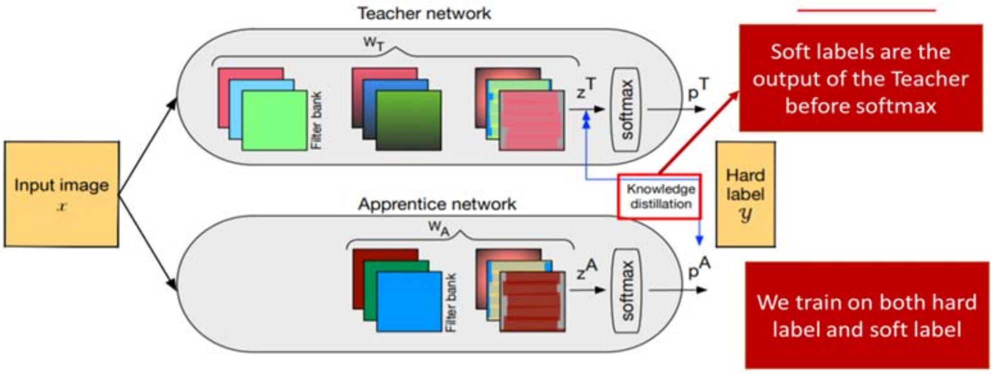


Softmax with temperature:

$$P_t(a) = \frac{\exp(q_t(a)/\tau)}{\sum_{i=1}^{n} \exp(q_t(i)/\tau)},$$



Knowledge Distillation



2 hyper-parameters

- T: How soft
- λ: ratio between hard labels and soft labels (Teacher)