Network Compression and Speedup

SHUOCHAO YAO, YIWEN XU, DANIEL CALZADA

Deep Learning on Mobile

Phones


Drones


Robots


Glasses


Self Driving Cars

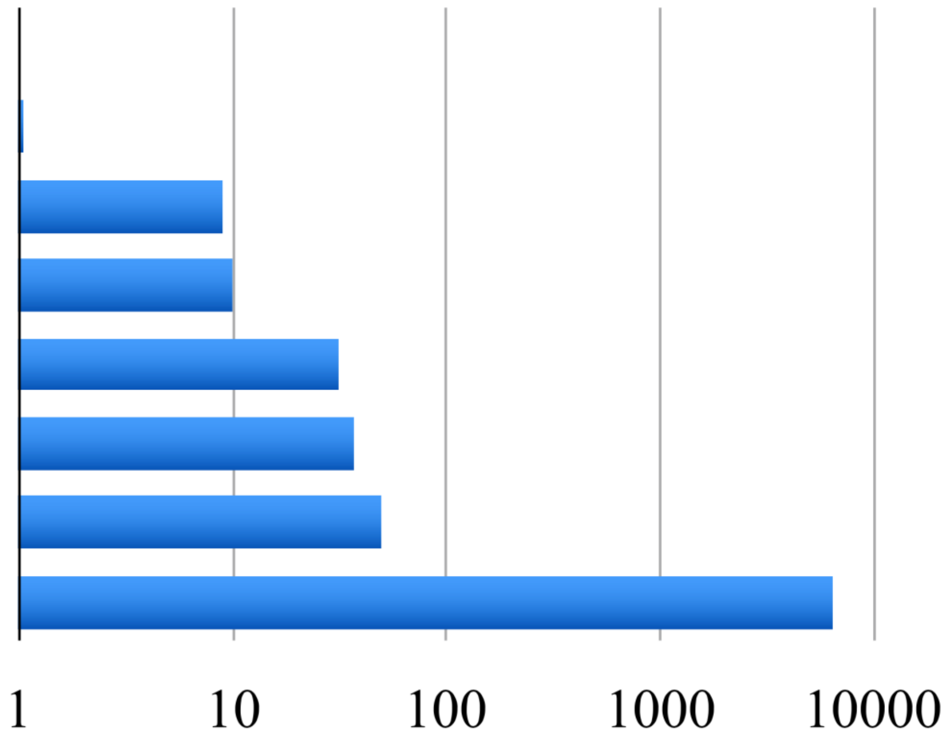
Battery Constrained!

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

Why smaller models?

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

Outline

- Matrix Factorization
- Weight Pruning
- Quantization method
- Pruning + Quantization + Encoding
- Design small architecture: SqueezeNet

Outline

Matrix Factorization

- Singular Value Decomposition (SVD)
- Flattened Convolutions

Weight Pruning

Quantization method

Pruning + Quantization + Encoding

Design small architecture: SqueezeNet

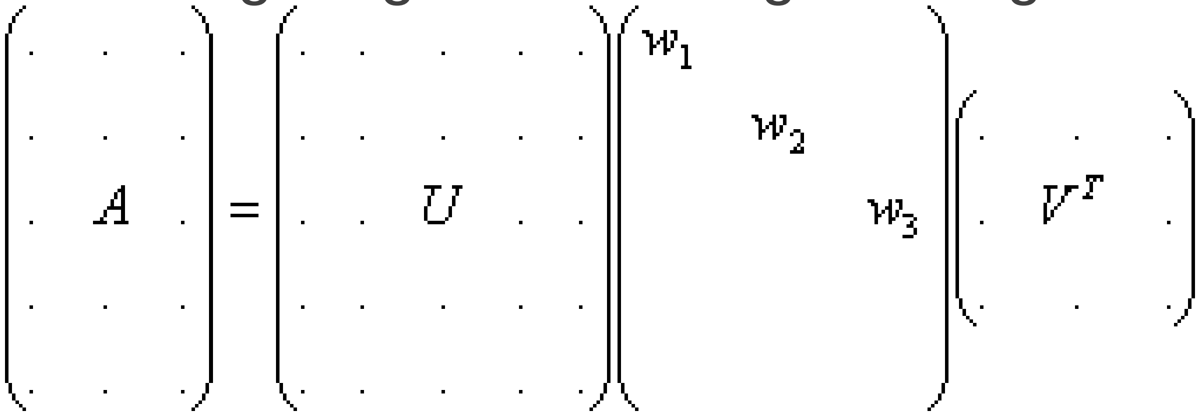
Fully Connected Layers: Singular Value Decomposition

Most weights are in the fully connected layers (according to Denton et al.)

$W = USV^\top$

- $W \in \mathbb{R}^{m \times k}, U \in \mathbb{R}^{m \times m}, S \in \mathbb{R}^{m \times k}, V^\top \in \mathbb{R}^{k \times k}$

S is diagonal, decreasing magnitudes along the diagonal



http://www.alglib.net/matrixops/general/i/svd1.gif

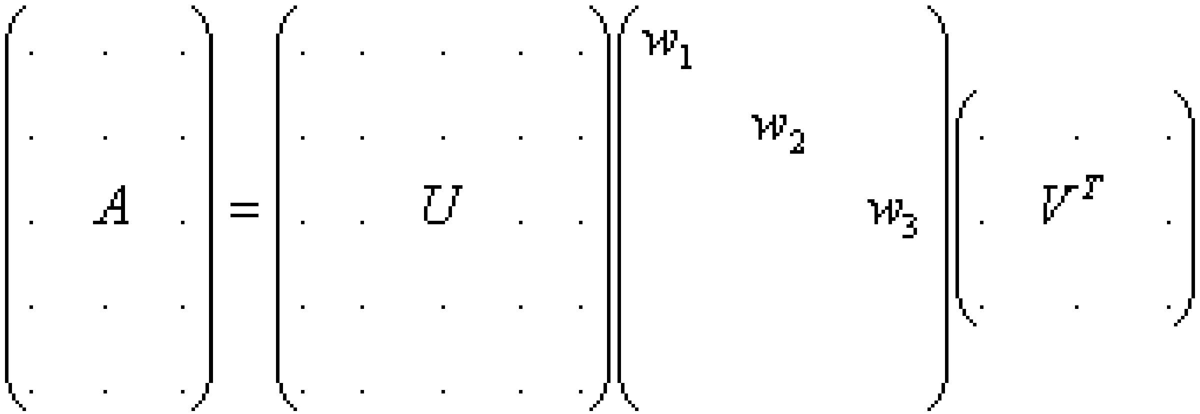
Singular Value Decomposition

By only keeping the t singular values with largest magnitude:

$\widetilde{W} = \widetilde{U}\tilde{S}\widetilde{V}^{\top}$

- $\widetilde{W} \in \mathbb{R}^{m \times k}, \widetilde{U} \in \mathbb{R}^{m \times t}, \tilde{S} \in \mathbb{R}^{t \times t}, \widetilde{V}^{\top} \in \mathbb{R}^{t \times k}$

$Rank(\widetilde{W}) = t$



http://www.alglib.net/matrixops/general/i/svd1.gif

SVD: Compression

$W = USV^\top, W \in \mathbb{R}^{m \times k}, U \in \mathbb{R}^{m \times m}, S \in \mathbb{R}^{m \times k}, V^\top \in \mathbb{R}^{k \times k}$

$\widetilde{W} = \widetilde{U}\tilde{S}\widetilde{V}^\top, \widetilde{W} \in R^{m \times k}, \widetilde{U} \in R^{m \times t}, \tilde{S} \in R^{t \times t}, \widetilde{V}^\top \in R^{t \times k}$

Storage for W: $O(mk)$

Storage for $\widetilde{W}$: $O(mt + t + tk)$

Compression Rate: $O\left(\frac{mk}{t(m+k+1)}\right)$

Theoretical error: $\left\|A\widetilde{W} - AW\right\|_F \leq s_{t+1}\|A\|_F$

Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." *arXiv preprint arXiv:1412.6115* (2014).

SVD: Compression Results

Trained on ImageNet 2012 database, then compressed

5 convolutional layers, 3 fully connected layers, softmax output layer

Approximation method	Number of parameters	Approximation hyperparameters	Reduction in weights	Increase in error
Standard FC	NM			
FC layer 1: Matrix SVD	$NK + KM$	$K = 250$ $K = 950$	$13.4\times$ $3.5\times$	0.8394% 0.09%
FC layer 2: Matrix SVD	$NK + KM$	$K = 350$ $K = 650$	$5.8\times$ $3.14\times$	0.19% 0.06%
FC layer 3: Matrix SVD	$NK + KM$	$K = 250$ $K = 850$	$8.1\times$ $2.4\times$	0.67% 0.02%

K refers to rank of approximation, t in the previous slides.

Denton, Emily L., et al. "Exploiting linear structure within convolutional networks for efficient evaluation." *Advances in Neural Information Processing Systems*. 2014.

SVD: Side Benefits

Reduced memory footprint

- Reduced in the dense layers by 5-13x

Speedup: $A\widetilde{W}, A \in \mathbb{R}^{n \times m}$, computed in $O(nmt + nt^2 + ntk)$ instead of $O(nmk)$

- Speedup factor is $O\left(\frac{mk}{t(m+t+k)}\right)$

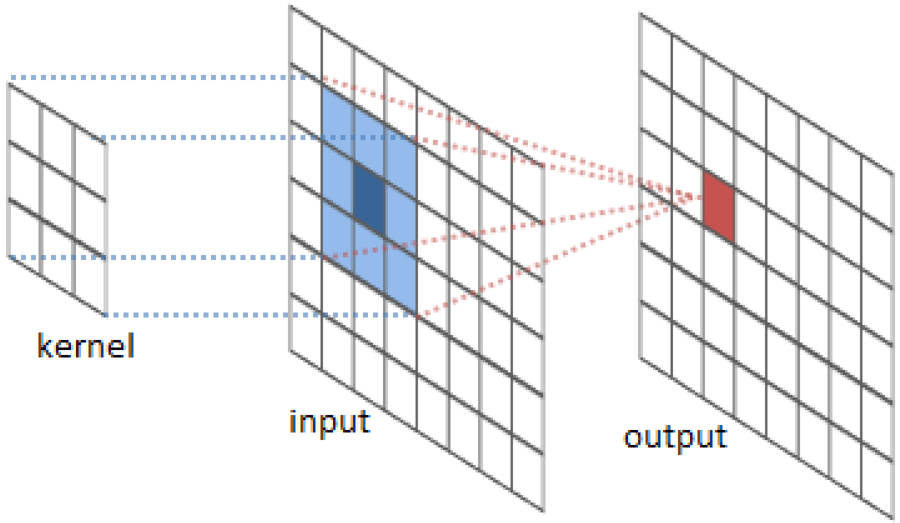
Regularization

- "Low-rank projections effectively decrease number of learnable parameters, suggesting that they might improve generalization ability."
- Paper applies SVD after training

Denton, Emily L., et al. "Exploiting linear structure within convolutional networks for efficient evaluation." *Advances in Neural Information Processing Systems*. 2014.

Convolutions: Matrix Multiplication

Most time is spent in the convolutional layers

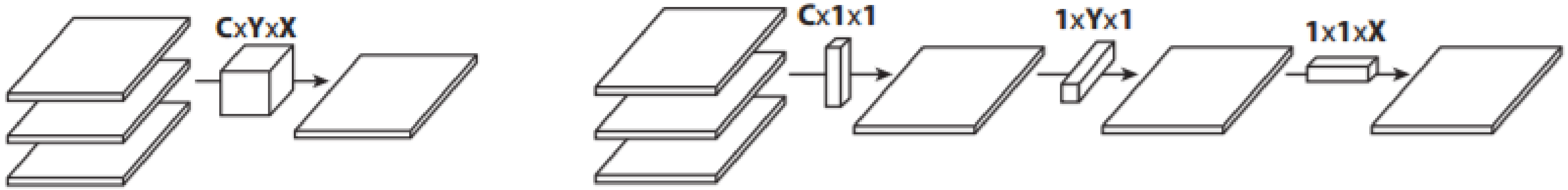


$$F(x, y) = I * W$$

http://stackoverflow.com/questions/15356153/how-do-convolution-matrices-work

Flattened Convolutions

Replace $c \times y \times x$ convolutions with $c \times 1 \times 1$, $1 \times y \times 1$, and $1 \times 1 \times x$ convolutions



(a) 3D convolution (b) 1D convolutions over different directions

Jin, Jonghoon, Aysegul Dundar, and Eugenio Culurciello. "Flattened convolutional neural networks for feedforward acceleration." *arXiv preprint arXiv:1412.5474* (2014).

Flattened Convolutions

$$\hat{F}(x,y) = I * \hat{W} = \sum_{x'=1}^{X}\left(\sum_{y'=1}^{Y}\left(\sum_{c=1}^{C} I(c, x-x', y-y')\alpha(c)\right)\beta(y')\right)\gamma(x')$$

$\alpha \in \mathbb{R}^C, \beta \in \mathbb{R}^Y, \gamma \in \mathbb{R}^X$

Compression and Speedup:

- Parameter reduction: $O(XYC)$ to $O(X + Y + C)$
- Operation reduction: $O(mnCXY)$ to $O\big(mn(C + X + Y)\big)$ (where $W_f \in \mathbb{R}^{m \times n}$)

Jin, Jonghoon, Aysegul Dundar, and Eugenio Culurciello. "Flattened convolutional neural networks for feedforward acceleration." *arXiv preprint arXiv:1412.5474* (2014).

Flattening = MF

$$\hat{F}(x,y) = \sum_{x=1}^{X}\sum_{y'=1}^{Y}\sum_{c=1}^{C} I(c, x-x', y-y')\alpha(c)\,\beta(y')\,\gamma(x')$$

$$= \sum_{x=1}^{X}\sum_{y'=1}^{Y}\sum_{c=1}^{C} I(c, x-x', y-y')\,\widehat{W}(c, x', y')$$

$\widehat{W} = \alpha \otimes \beta \otimes \gamma, Rank\left(\widehat{W}\right) = 1$

$\widehat{W}_S = \sum_{k=1}^{K} \alpha_k \otimes \beta_k \otimes \gamma_k$, Rank K

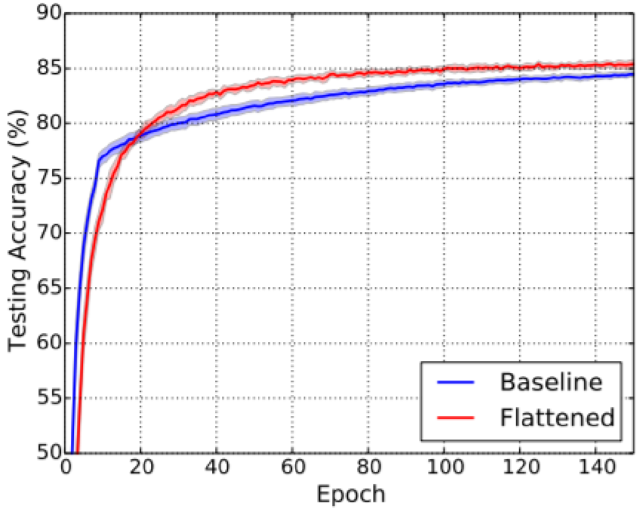
SVD: Can reconstruct the original matrix as $A = \sum_{k=1}^{K} w_k u_k \otimes v_k$

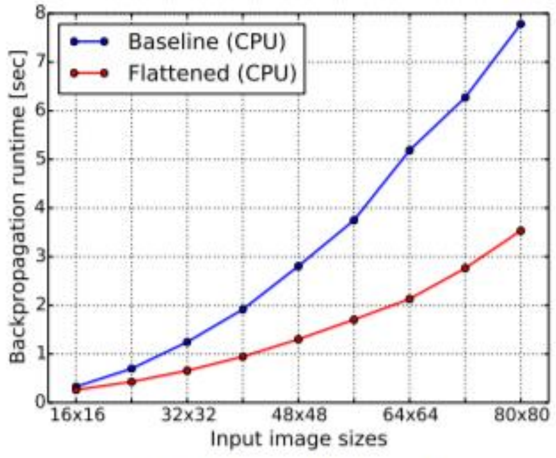
Denton, Emily L., et al. "Exploiting linear structure within convolutional networks for efficient evaluation." *Advances in Neural Information Processing Systems*. 2014.

Flattening: Speedup Results

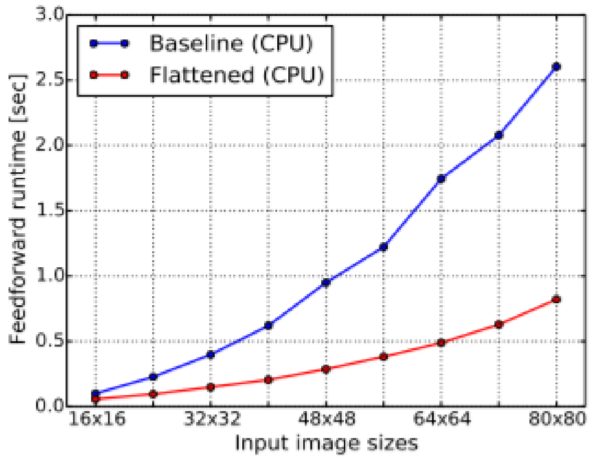
3 convolutional layers (5x5 filters) with 96, 128, and 256 channels

Used stacks of 2 rank-1 convolutions





(c) Backpropagation on CPU



(a) Feedforward on CPU

Jin, Jonghoon, Aysegul Dundar, and Eugenio Culurciello. "Flattened convolutional neural networks for feedforward acceleration." *arXiv preprint arXiv:1412.5474* (2014).

Outline

Matrix Factorization

Weight Pruning

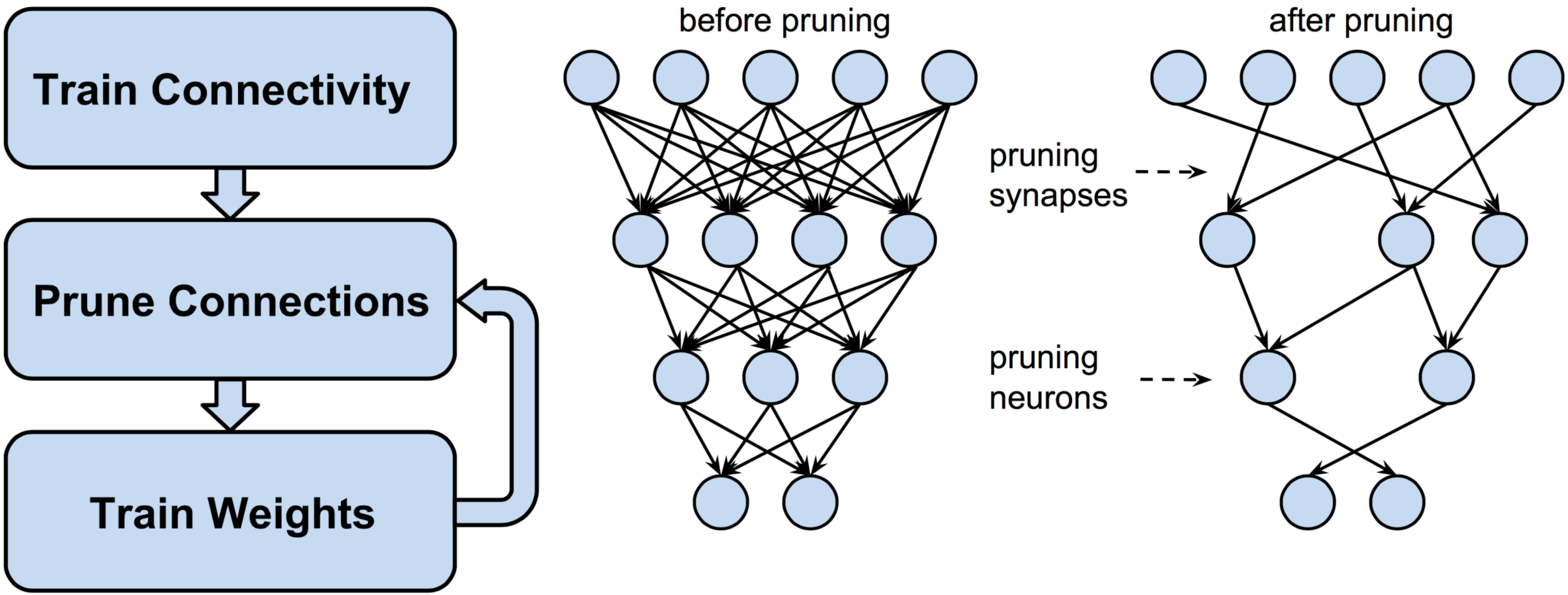
- Magnitude-based method
  - Iterative pruning + Retraining
  - Pruning with rehabilitation
- Hessian-based method

Quantization method

Pruning + Quantization + Encoding

Design small architecture: SqueezeNet

Magnitude-based method: Iterative Pruning + Retraining



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

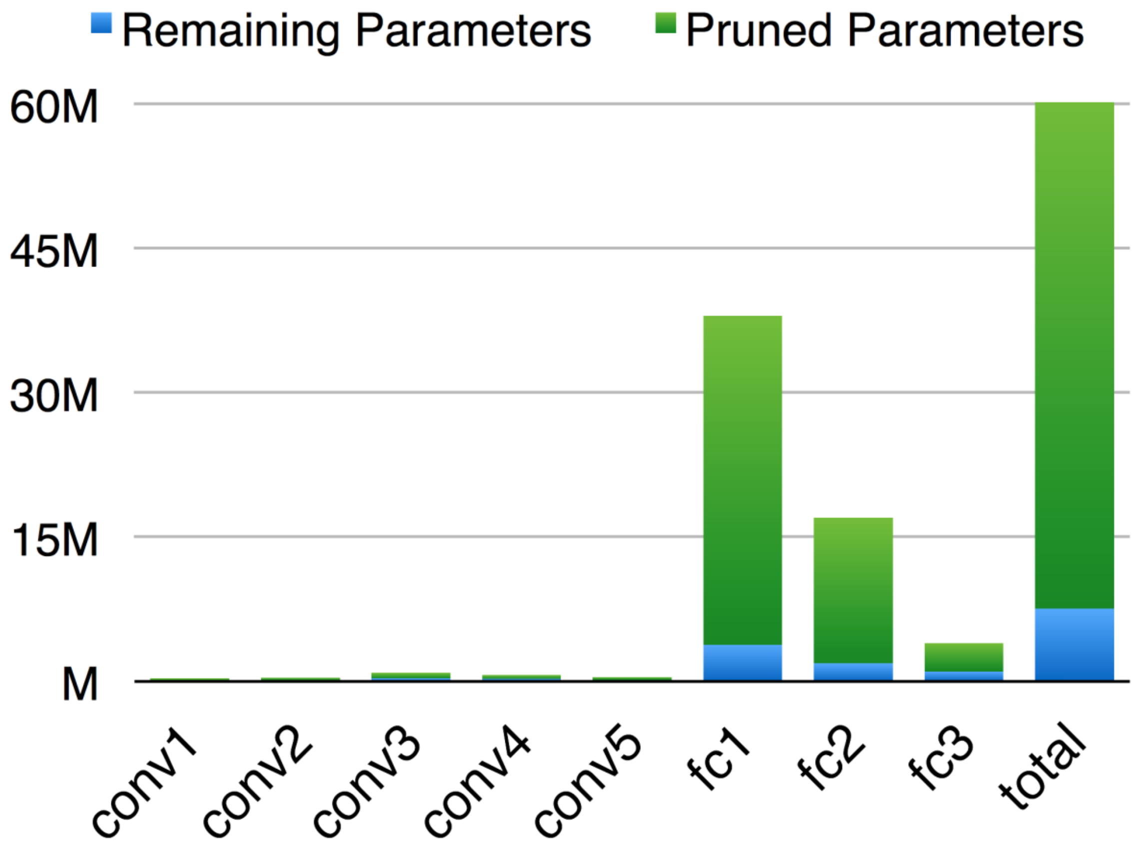
Magnitude-based method: Iterative Pruning + Retraining (Algorithm)

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.

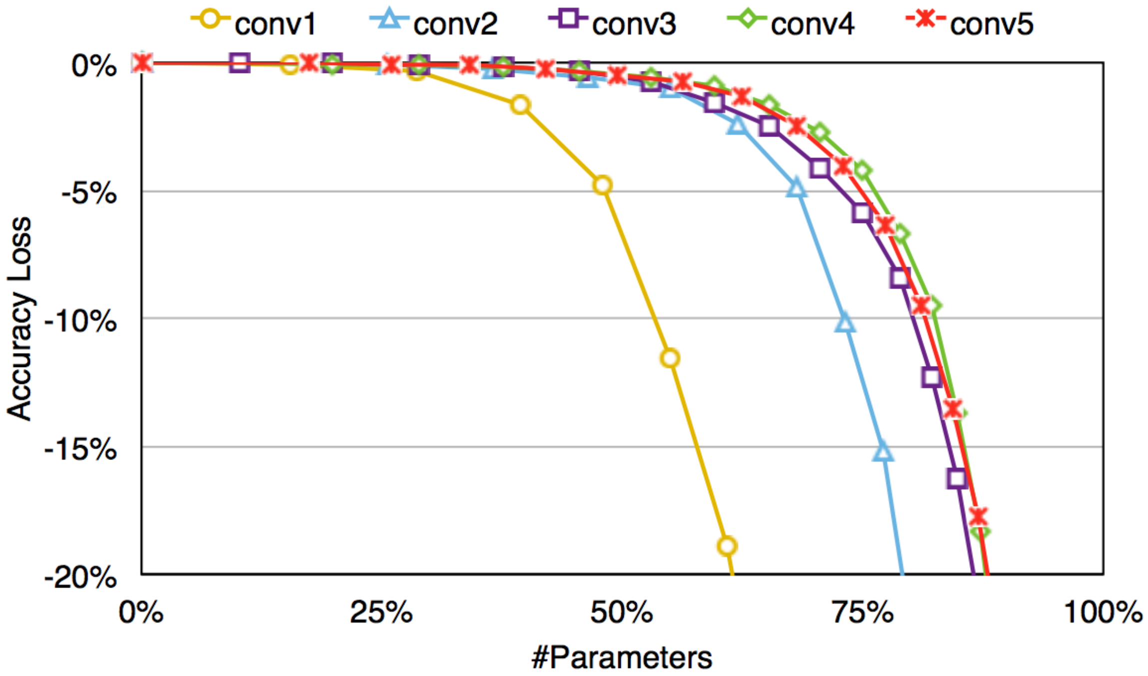
Magnitude-based method: Iterative Pruning + Retraining (Experiment: AlexNet)

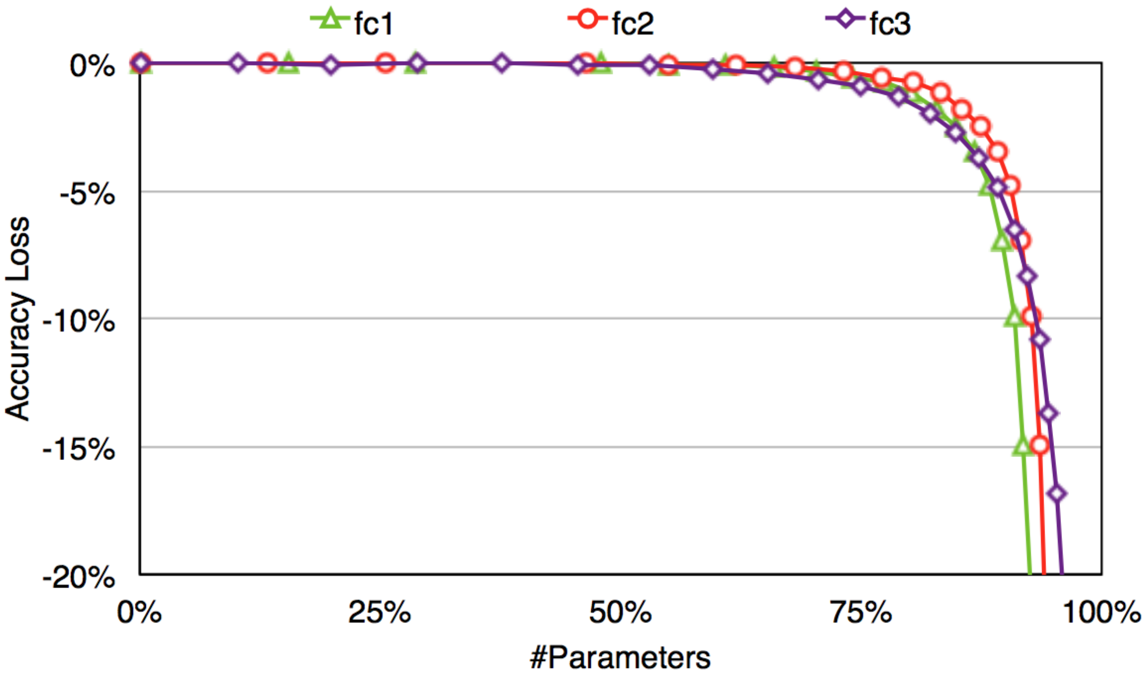
Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10
Total	61M	1.5B	54%	11%	30%



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

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





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

Pruning with rehabilitation: Dynamic Network Surgery (Motivation)

Pruned connections have no chance to come back.

Incorrect pruning may cause severe accuracy loss.

Avoid the risk of irretrievable network damage .

Improve the learning efficiency.

Guo, Yiwen, et al. "Dynamic Network Surgery for Efficient DNNs." NIPS. 2016.

Pruning with rehabilitation: Dynamic Network Surgery (Formulation)

W_k denotes the weights, and T_k denotes the corresponding 0/1 masks.

$$\min_{W_k,T_k} L(W_k \odot T_k) \quad s.t. \; {T_k}^{(i,j)} = h_k\left({W_k}^{(i,j)}\right), \forall (i,j) \in \mathfrak{T}$$

- $\odot$ is the element-wise product. $L(\cdot)$ is the loss function.

Dynamic network surgery updates only W_k. T_k is updated based on $h_k(\cdot)$.

$$h_k\left({W_k}^{(i,j)}\right) = \begin{cases} 0 & a_k \geq \left|{W_k}^{(i,j)}\right| \\ {T_k}^{(i,j)} & a_k \leq \left|{W_k}^{(i,j)}\right| \leq b_k \\ 1 & b_k \leq \left|{W_k}^{(i,j)}\right| \end{cases}$$

- a_k is the pruning threshold. $b_k = a_k + t$, where t is a pre-defined small margin.

Guo, Yiwen, et al. "Dynamic Network Surgery for Efficient DNNs." NIPS. 2016.

Pruning with rehabilitation: Dynamic Network Surgery (Algorithm)

1. Choose a neural network architecture.
2. Train the network until a reasonable solution is obtained.
3. Update T_k based on $h_k(\cdot)$.
4. Update W_k based on back-propagation.
5. Iterate to step 3.

Guo, Yiwen, et al. "Dynamic Network Surgery for Efficient DNNs." NIPS. 2016.

Pruning with rehabilitation: Dynamic Network Surgery (Experiment on AlexNet)

Layer	Parameters	Parameters (Han et al. 2015)	Parameters (DNS)
conv1	35K	84%	53.8%
conv2	307K	38%	40.6%
conv3	885K	35%	29.0%
conv4	664K	37%	32.3%
conv5	443K	37%	32.5%
fc1	38M	9%	3.7%
fc2	17M	9%	6.6%
fc3	4M	25%	4.6%
Total	61M	11%	5.7%

Guo, Yiwen, et al. "Dynamic Network Surgery for Efficient DNNs." NIPS. 2016.

Outline

- Matrix Factorization
- Weight Pruning
  - Magnitude-based method
  - Hessian-based method
    - Diagonal Hessian-based method
    - Full Hessian-based method
- Quantization method
- Pruning + Quantization + Encoding
- Design small architecture: SqueezeNet

Diagonal Hessian-based method: Optimal Brain Damage

The idea of model compression & speed up: traced by to 1990.

Actually theoretically more "optimal" compared with the current state of the art, but much more computational inefficient.

Delete parameters with small "saliency".

- Saliency: effect on the training error

Propose a theoretically justified saliency measure.

Diagonal Hessian-based method: Optimal Brain Damage (Formulation)

Approximate objective function E with Taylor series:

$$\delta E = \sum_i \frac{\partial E}{\partial u_i} \delta u_i + \frac{1}{2} \sum_i \frac{\partial^2 E}{\partial^2 u_i} \delta^2 u_i + \frac{1}{2} \sum_i \frac{\partial^2 E}{\partial u_i \partial u_j} \delta u_i \delta u_j + \mathrm{O}(\|\delta U\|^3)$$

Deletion after training has converged: local minimum with gradients equal 0.

Neglect cross terms

$$\delta E = \frac{1}{2} \sum_i \frac{\partial^2 E}{\partial^2 u_i} \delta^2 u_i$$

LeCun, Yann, et al. "Optimal brain damage." NIPs. Vol. 2. 1989.

Diagonal Hessian-based method: Optimal Brain Damage (Algorithm)

1. Choose a neural network architecture.

2. Train the network until a reasonable solution is obtained.

3. Compute the second derivatives for each parameters.

4. Compute the saliencies for each parameter $S_k = \frac{\partial^2 E}{\partial^2 u_k} {u_k}^2$.

5. Sort the parameters by saliency and delete some low-saliency parameters

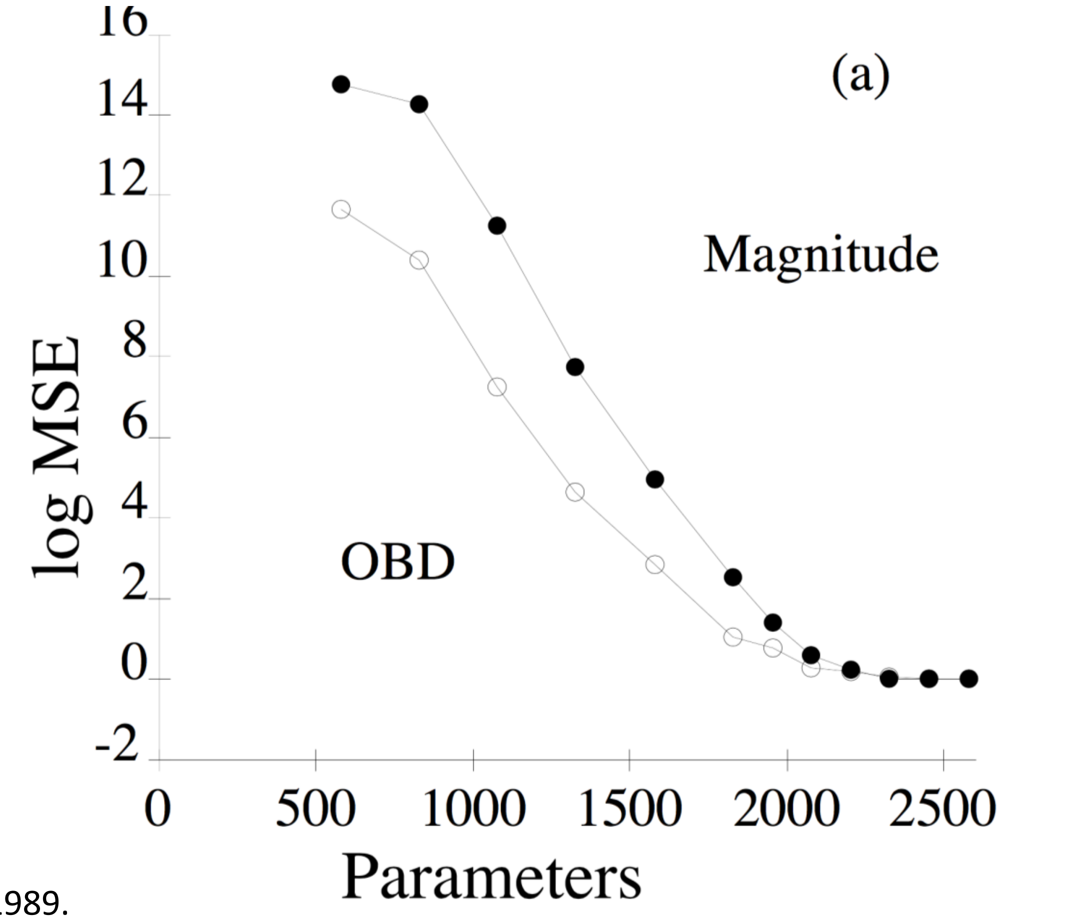
6. Iterate to step 2

LeCun, Yann, et al. "Optimal brain damage." NIPs. Vol. 2. 1989.

Diagonal Hessian-based method: Optimal Brain Damage (Experiment: OBD vs. Magnitude)

OBD vs. Magnitude

Deletion based on saliency performs better



LeCun, Yann, et al. "Optimal brain damage." NIPs. Vol. 2. 1989.

Diagonal Hessian-based method: Optimal Brain Damage (Experiment: Retraining)

How retraining helps?

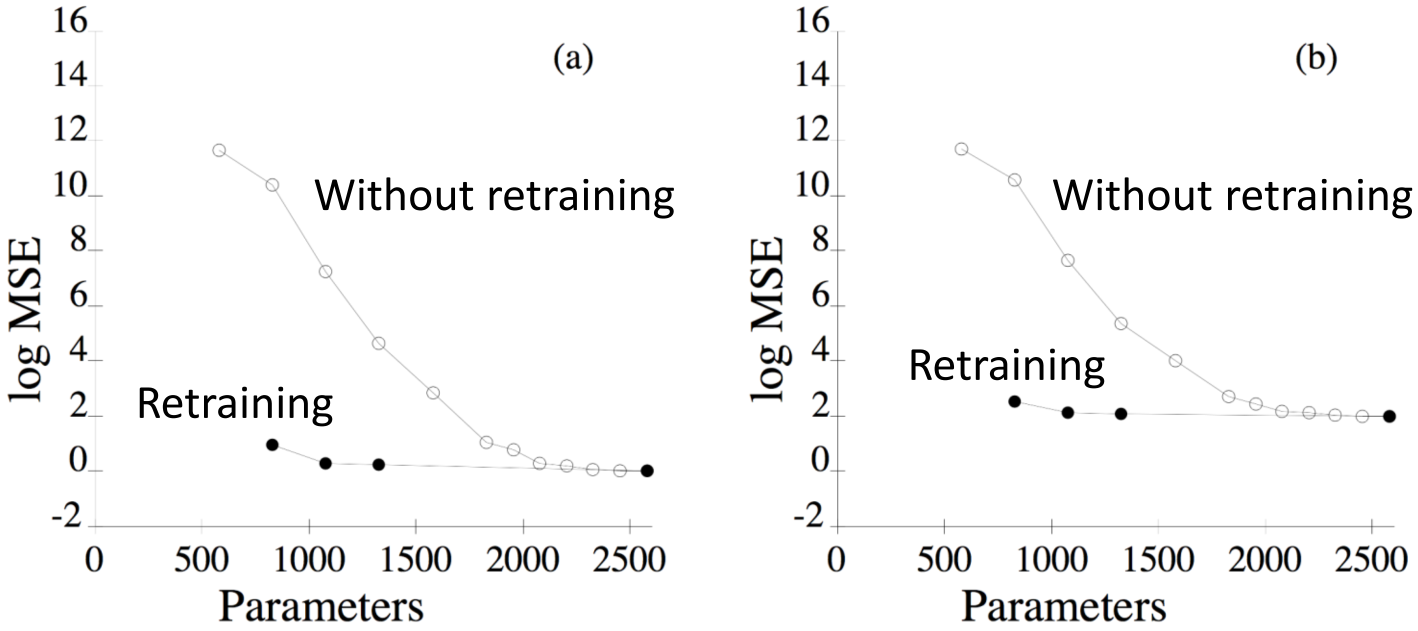


Figure 2: Objective function (in dB) versus number of parameters, without retraining (upper curve), and after retraining (lower curve). Curves are given for the training set (a) and the test set (b).

LeCun, Yann, et al. "Optimal brain damage." NIPs. Vol. 2. 1989.

Full Hessian-based method: Optimal Brain Surgeon

Motivation:

- A more accurate estimation of saliency.
- Optimal weight updates.

Advantage:

- More accuracy estimation with saliency.
- Directly provide the weight updates, which minimize the change of objective function.

Disadvantage

- More computation compared with OBD.
- Weight updates are not based on minimizing the objective function.

Hassibi, Babak, and David G. Stork. "Second order derivatives for network pruning: Optimal brain surgeon." NIPS, 1993

Full Hessian-based method: Optimal Brain Surgeon (Formulation)

Approximate objective function E with Taylor series:

$$\delta E = \left(\frac{\partial E}{\partial w}\right)^T \cdot \delta w + \frac{1}{2}\delta w^T \cdot H \cdot \delta w + O(\|\delta W\|^3)$$

- with constraint $e_q{}^T \cdot \delta w + w_q = 0$

We assume the trained network with local minimum and ignore high order terms. Solve it through Lagrangian form:

$$\delta \mathrm{w} = -\frac{w_q}{[H^{-1}]_{qq}} H^{-1} \cdot e_q \text{ and } L_q = \frac{w_q{}^2}{2 \cdot [H^{-1}]_{qq}}$$

- L_q is saliency for weight w_q

Hassibi, Babak, and David G. Stork. "Second order derivatives for network pruning: Optimal brain surgeon." NIPS, 1993

Full Hessian-based method: Optimal Brain Surgeon (Algorithm)

1. Choose a neural network architecture.

2. Train the network until a reasonable solution is obtained.

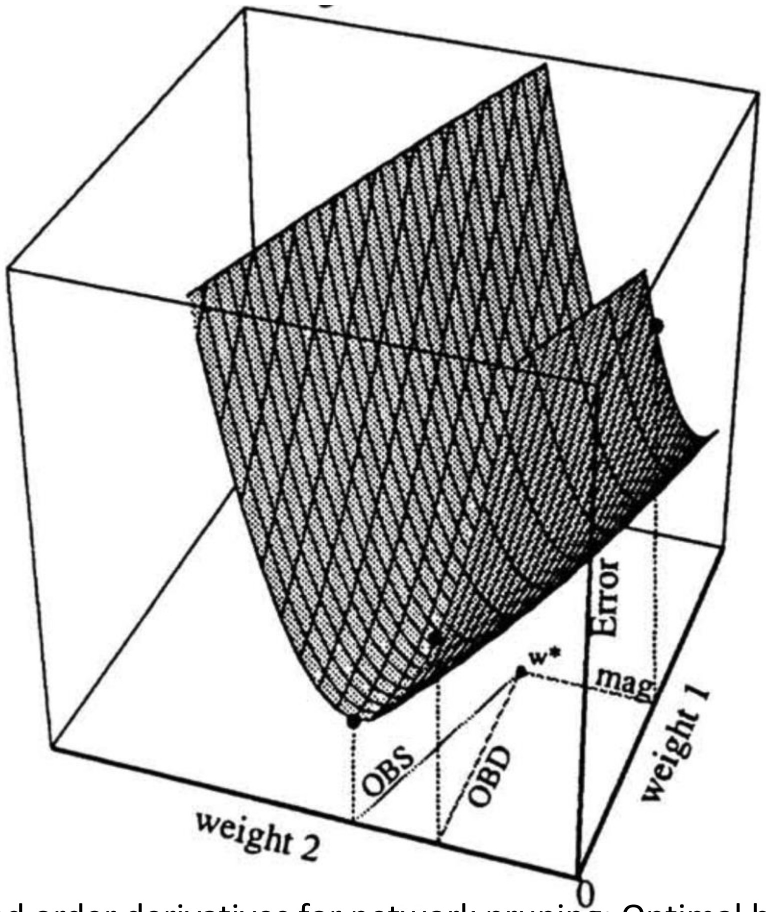
3. Find the q that gives the smallest saliency L_q, and decide to delete q or stop pruning.

4. Update all weights based on calculated $\delta\mathrm{w}$.

5. Iterate to step 3.

Hassibi, Babak, and David G. Stork. "Second order derivatives for network pruning: Optimal brain surgeon." NIPS, 1993

Full Hessian-based method: Optimal Brain Surgeon



Hassibi, Babak, and David G. Stork. "Second order derivatives for network pruning: Optimal brain surgeon." NIPS, 1993

Outline

Matrix Factorization

Weight Pruning

Quantization method

- Full Quantization
  - Fixed-point format
  - Code book
- Quantization with full-precision copy

Pruning + Quantization + Encoding

Design small architecture: SqueezeNet

Full Quantization : Fixed-point format

Limited Precision Arithmetic

- $[QI.QF]$, where QI and QF correspond to the integer and the fractional part of the number.
- The number of integer bits (IL) plus the number of fractional bits (FL) yields the total number of bits used to represent the number.
- WL = IL + FL.
- Can be represented as $\langle IL, FL \rangle$.
- $\langle IL, FL \rangle$ limits the precision to FL bits.
- $\langle IL, FL \rangle$ sets the range to $[-2^{IL-1}, 2^{IL-1} - 2^{-FL}]$.

Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.

Full Quantization : Fixed-point format (Rounding Modes)

Define $\lfloor x \rfloor$ as the largest integer multiple of $\epsilon = 2^{-FL}$.

Round-to-nearest:

- $Round(x, \langle IL, FL \rangle) = \begin{cases} \lfloor x \rfloor & \lfloor x \rfloor \leq x \leq \lfloor x \rfloor + \frac{\epsilon}{2} \\ \lfloor x \rfloor + \epsilon & \lfloor x \rfloor + \frac{\epsilon}{2} \leq x \leq \lfloor x \rfloor + \epsilon \end{cases}$

Stochastic rounding (unbiased):

- $Round(x, \langle IL, FL \rangle) = \begin{cases} \lfloor x \rfloor & w.p. \quad 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & w.p. \quad \frac{x - \lfloor x \rfloor}{\epsilon} \end{cases}$

If x lies outside the range of $\langle IL, FL \rangle$, we saturate the result to either the lower or the upper limit of $\langle IL, FL \rangle$:

Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.

Multiply and accumulate (MACC) operation

During training:

1. $\boldsymbol{a}$ and $\boldsymbol{b}$ are two vectors with fixed point format $\langle IL, FL \rangle$.

2. Compute $z = \sum_{i=1}^{d} a_i b_i$.
   - Results a fixed point number with format $\langle 2 \times IL, 2 \times FL \rangle$.

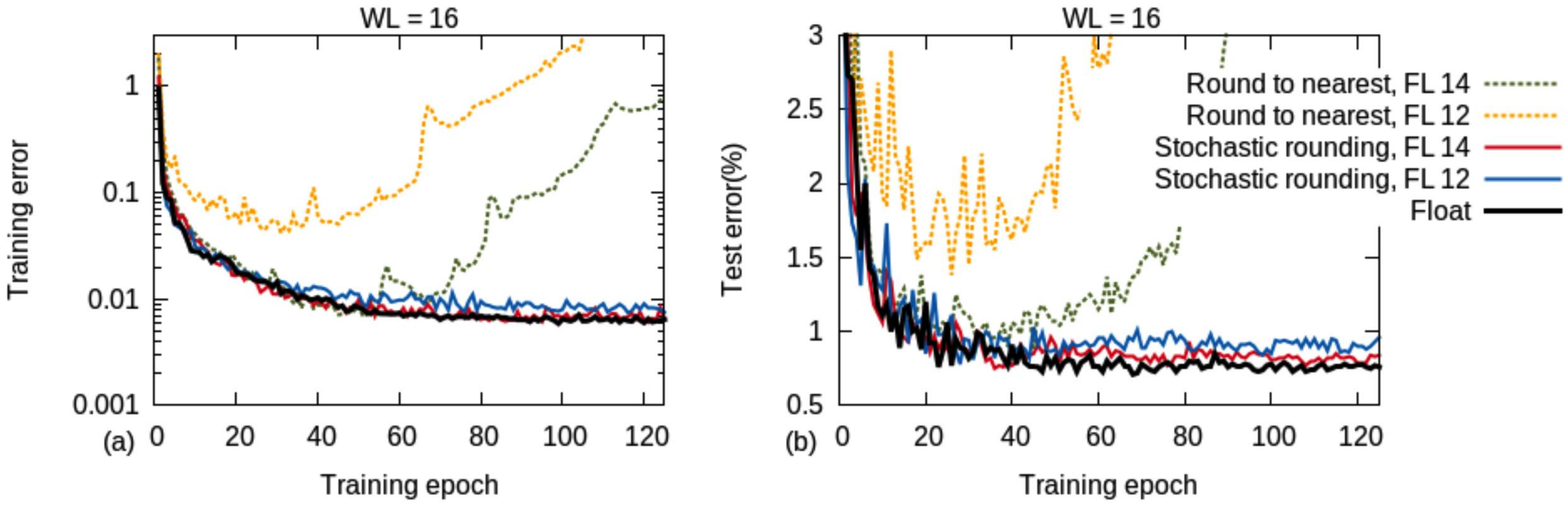
3. Covert and round z back to fixed point format $\langle IL, FL \rangle$.

During testing:

With fixed point format $\langle IL, FL \rangle$.

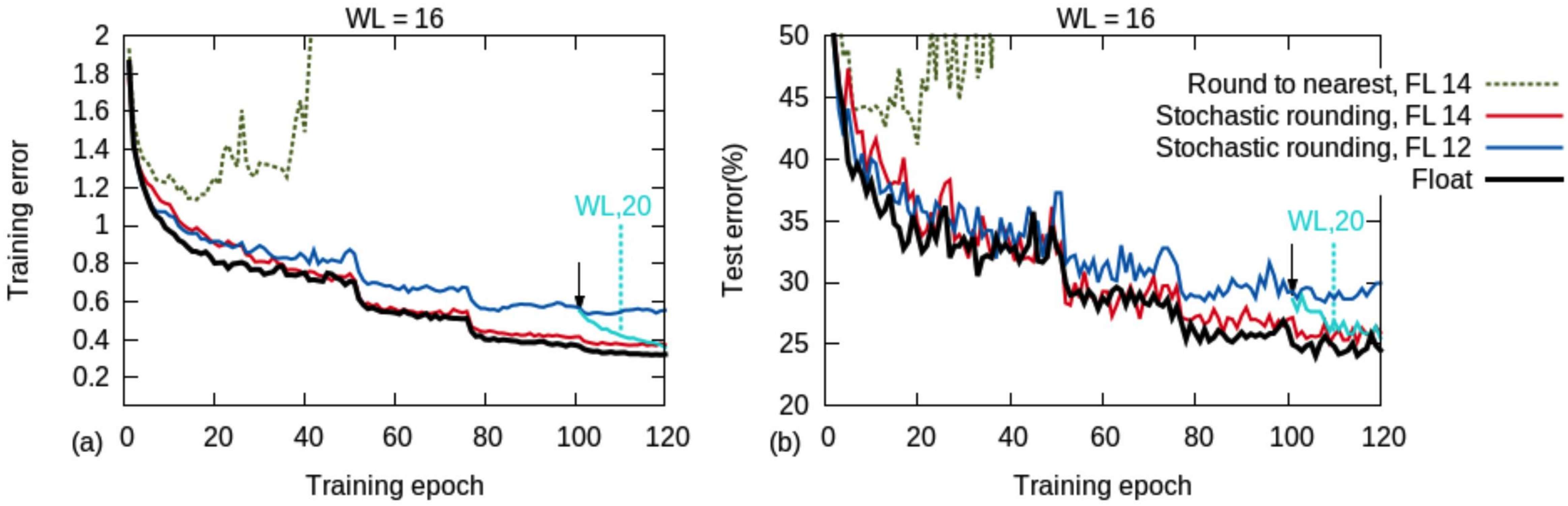
Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.

Full Quantization: Fixed-point format (Experiment on MNIST with CNNs)



Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.

Full Quantization: Fixed-point format (Experiment on CIFAR10 with fully connected DNNs)



Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.

Full Quantization: Code book

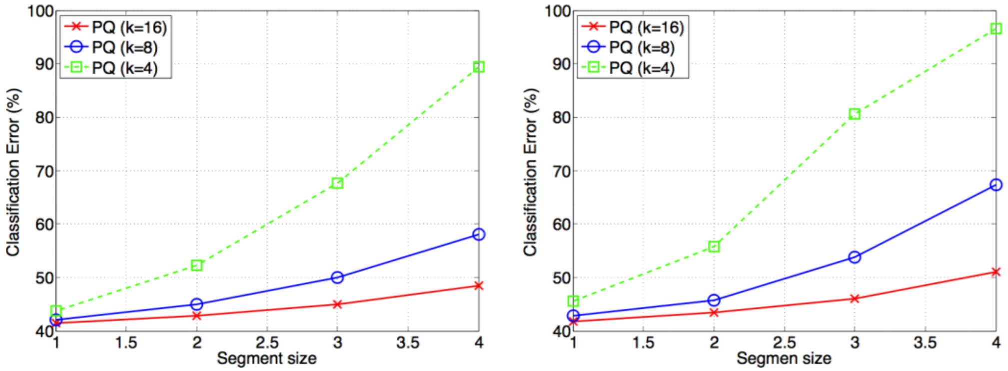
Quantization using k-means
- Perform k-means to find k centers $\{c_z\}$ for weights W.
- $\widehat{W_{ij}} = c_z$ where $\min_z \|W_{ij} - c_z\|^2$.
- Compression ratio: $32/\log_2 k$ (codebook itself is negligible).

Product Quantization
- Partition $W \in \mathbb{R}^{m \times n}$ colum-wise into s submatrices $W = [W^1, W^2, \cdots, W^s]$.
- Perform k-means for elements in W^i to find k centers $\{c_z^i\}$.
- $\widehat{W_j^i} = c_z^i$ where $\min_z \|W_j^i - c_z^i\|^2$.
- Compression ratio: $32mn/(32kn + \log_2 k\, ms)$

Residual Quantization
- Quantize the vectors into k centers.
- Then recursively quantize the residuals for t iterations.
- Compression ratio: $m/(tk + \log_2 k \cdot tn)$

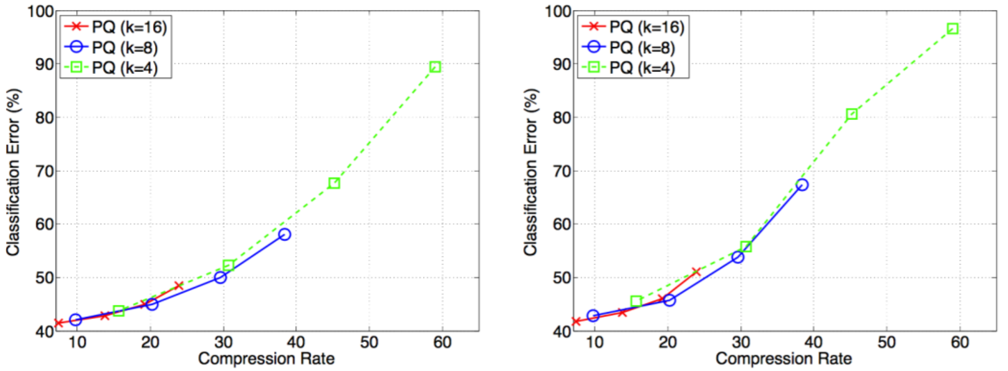
Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." arXiv preprint arXiv:1412.6115 (2014).

Full Quantization: Code book (Experiment on PQ)



(a) PQ for X-axis (b) PQ for Y-axis

Figure 1: Comparison of PQ compression with aligned segment size for accuracy@1.



(a) PQ for X-axis (b) PQ for Y-axis

Figure 2: Comparison of PQ compression with aligned compression rate for accuracy@1. We can clearly find when taking codebook size into account, using more centers do not necessarily lead to better accuracy with same compression rate. See text for detailed discussion.

Full Quantization: Code book

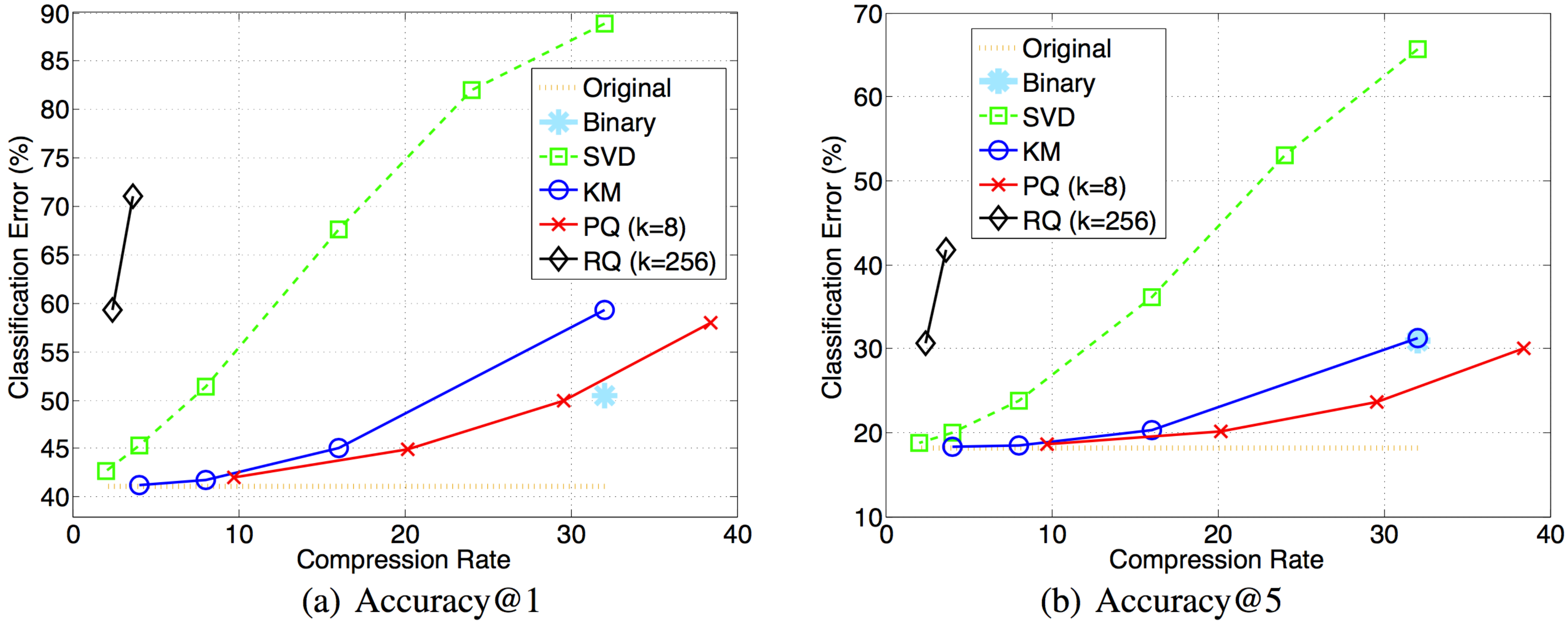


Figure 3: Comparison of different compression methods on ILSVRC dataset.

Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." arXiv preprint arXiv:1412.6115 (2014).

Outline

Matrix Factorization

Weight Pruning

Quantization method

- Full quantization
- Quantization with full-precision copy
  - Binnaryconnect
  - BNN

Design small architecture: SqueezeNet

Quantization with full-precision copy: Binaryconnect (Motivation)

Use only two possible value (e.g. +1 or -1) for weights.

Replace many multiply-accumulate operations by simple accumulations.

Fixed-point adders are much less expensive both in terms of area and energy than fixed-point multiply-accumulators.

Courbariaux, et al. "Binaryconnect: Training deep neural networks with binary weights during propagations." NIPS. 2015

Quantization with full-precision copy: Binaryconnect (Binarization)

Deterministic Binarization:

- $w_b = \begin{cases} +1 & if\ w \geq 0 \\ -1 & otherwise \end{cases}$

Stochastic Binarization:

- $w_b = \begin{cases} +1 & with\ probability\ p = \sigma(w_b) \\ -1 & with\ probability\ 1 - p \end{cases}$
- $\sigma(x) = clip\left(\frac{x+1}{2}, 0, 1\right) = max\left(0, min\left(1, \frac{x+1}{2}\right)\right)$

Stochastic binarization is more theoretically appealing than the deterministic one, but harder to implement as it requires the hardware to generate random bits when quantizing.

Courbariaux, et al. "Binaryconnect: Training deep neural networks with binary weights during propagations." NIPS. 2015

Quantization with full-precision copy: Binaryconnect

1. Given the DNN input, compute the unit activations layer by layer, leading to the top layer which is the output of the DNN, given its input. This step is referred as the **forward propagation**.

2. Given the DNN target, compute the training objective's gradient w.r.t. each layer's activations, starting from the top layer and going down layer by layer until the first hidden layer. This step is referred to as the **backward propagation or backward phase of back-propagation**.

3. Compute the gradient w.r.t. each layer's parameters and then update the parameters using their computed gradients and their previous values. This step is referred to as the **parameter update**.

Courbariaux, et al. "Binaryconnect: Training deep neural networks with binary weights during propagations." NIPS. 2015

Quantization with full-precision copy: Binaryconnect

BinaryConnect only **binarize** the weights during the **forward** and **backward** propagations (steps 1 and 2) but **not** during the **parameter update** (step 3).

Courbariaux, et al. "Binaryconnect: Training deep neural networks with binary weights during propagations." NIPS. 2015

Quantization with full-precision copy: Binaryconnect

1. Binarize weights and perform forward pass.

2. Back propagate gradient based on binarized weights.

3. Update the full-precision weights.

4. Iterate to step 1.

Courbariaux, et al. "Binaryconnect: Training deep neural networks with binary weights during propagations." NIPS. 2015

Quantization with full-precision copy: Binaryconnect

Method	MNIST	CIFAR-10	SVHN
No regularizer	$1.30 \pm 0.04\%$	10.64%	2.44%
BinaryConnect (det.)	$1.29 \pm 0.08\%$	9.90%	2.30%
BinaryConnect (stoch.)	$1.18 \pm 0.04\%$	**8.27%**	2.15%
50% Dropout	$1.01 \pm 0.04\%$		
Maxout Networks [29]	0.94%	11.68%	2.47%
Deep L2-SVM [30]	**0.87%**		
Network in Network [31]		10.41%	2.35%
DropConnect [21]			1.94%
Deeply-Supervised Nets [32]		9.78%	**1.92%**

Courbariaux, et al. "Binaryconnect: Training deep neural networks with binary weights during propagations." NIPS. 2015

Quantization with full-precision copy: Binarized Neural Networks (Motivation)

Neural networks with **both binary weights and activations** at run-time and when computing the parameters' gradient at train time.

Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

Quantization with full-precision copy: Binarized Neural Networks

Propagating Gradients Through Discretization ("straight-through estimator")

- $q = Sign(r)$
- Estimator g_q of the gradient $\frac{\partial C}{\partial q}$
- Straight-through estimator of $\frac{\partial C}{\partial r}$:
  - $g_r = g_q 1_{|r| \leq 1}$
  - Can be viewed as propagating the gradient through *hard tanh*

Replace multiplications with bit-shift

- Replace batch normalization with shift-based batch normalization
- Replace ADAM with shift-based AdaMax

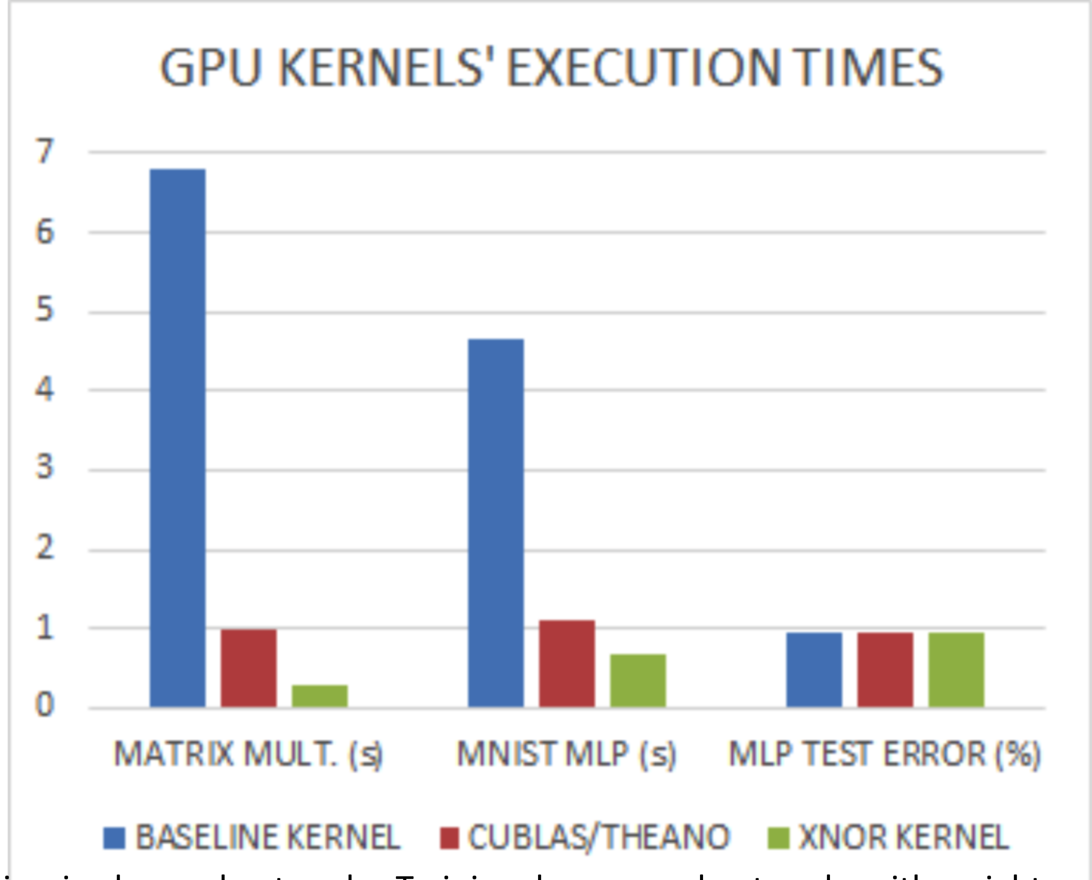
Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

Quantization with full-precision copy: Binarized Neural Networks

Data set	MNIST	SVHN	CIFAR-10
Binarized activations+weights, during training and test			
BNN (Torch7)	1.40%	2.53%	10.15%
BNN (Theano)	0.96%	2.80%	11.40%
Committee Machines' Array (Baldassi et al., 2015)	1.35%	-	-
Binarized weights, during training and test			
BinaryConnect (Courbariaux et al., 2015)	1.29± 0.08%	2.30%	9.90%
Binarized activations+weights, during test			
EBP (Cheng et al., 2015)	2.2± 0.1%	-	-
Bitwise DNNs (Kim & Smaragdis, 2016)	1.33%	-	-
Ternary weights, binary activations, during test			
(Hwang & Sung, 2014)	1.45%	-	-
No binarization (standard results)			
Maxout Networks (Goodfellow et al.)	0.94%	2.47%	11.68%
Network in Network (Lin et al.)	-	2.35%	10.41%
Gated pooling (Lee et al., 2015)	-	1.69%	7.62%

Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

Quantization with full-precision copy: Binarized Neural Networks



Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

Outline

Matrix Factorization

Weight Pruning

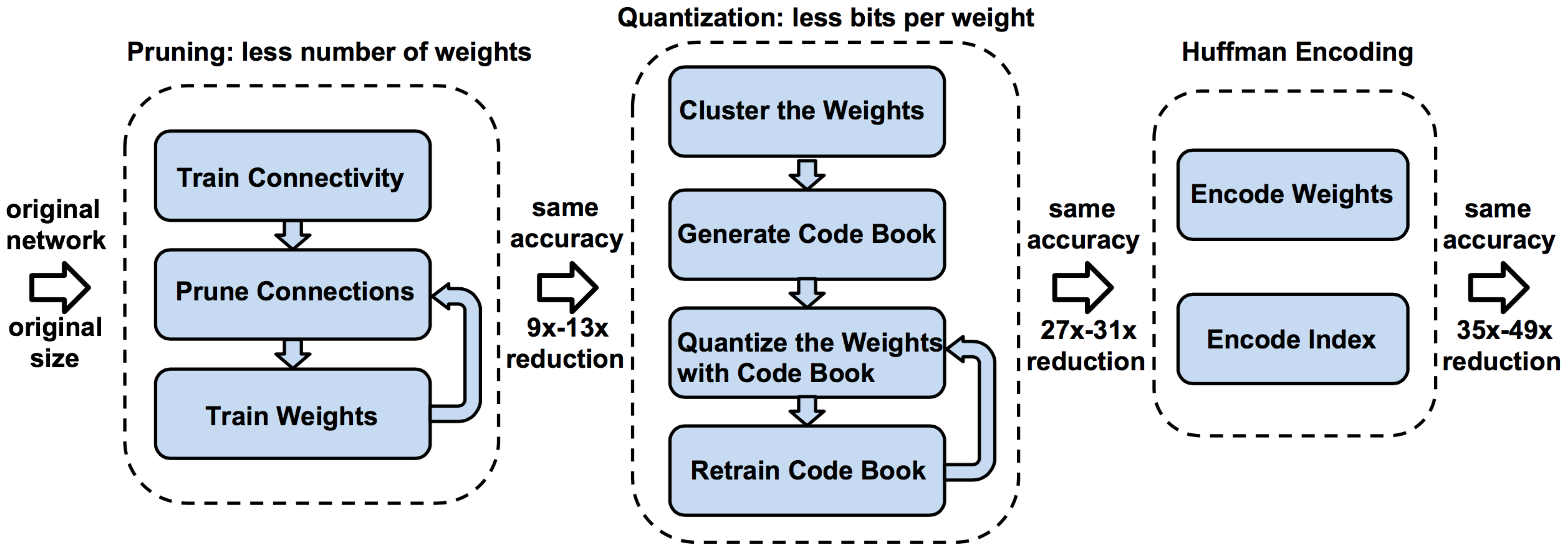
Quantization method

Pruning + Quantization + Encoding

- Deep Compression

Design small architecture: SqueezeNet

Pruning + Quantization + Encoding: Deep Compression



Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

Pruning + Quantization + Encoding: Deep Compression

1. Choose a neural network architecture.

2. Train the network until a reasonable solution is obtained.

3. Prune the network with magnitude-based method until a reasonable solution is obtained.

4. Quantize the network with k-means based method until a reasonable solution is obtained.

5. Further compress the network with Huffman coding.

Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

Pruning + Quantization + Encoding: Deep Compression

Table 4: Compression statistics for AlexNet. 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	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%(9×)	5.4	4	4	3.2	3.7% (**27**×)	2.88% (**35**×)

Table 5: Compression statistics for VGG-16. P: pruning, Q:quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weigh 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_1	2K	58%	8	6.8	5	1.7	40.0%	29.97%
conv1_2	37K	22%	8	6.5	5	2.6	9.8%	6.99%
conv2_1	74K	34%	8	5.6	5	2.4	14.3%	8.91%
conv2_2	148K	36%	8	5.9	5	2.3	14.7%	9.31%
conv3_1	295K	53%	8	4.8	5	1.8	21.7%	11.15%
conv3_2	590K	24%	8	4.6	5	2.9	9.7%	5.67%
conv3_3	590K	42%	8	4.6	5	2.2	17.0%	8.96%
conv4_1	1M	32%	8	4.6	5	2.6	13.1%	7.29%
conv4_2	2M	27%	8	4.2	5	2.9	10.9%	5.93%
conv4_3	2M	34%	8	4.4	5	2.5	14.0%	7.47%
conv5_1	2M	35%	8	4.7	5	2.5	14.3%	8.00%
conv5_2	2M	29%	8	4.6	5	2.7	11.7%	6.52%
conv5_3	2M	36%	8	4.6	5	2.3	14.8%	7.79%
fc6	103M	4%	5	3.6	5	3.5	1.6%	1.10%
fc7	17M	4%	5	4	5	4.3	1.5%	1.25%
fc8	4M	23%	5	4	5	3.4	7.1%	5.24%
Total	138M	7.5%(13×)	6.4	4.1	5	3.1	3.2% (**31**×)	2.05% (**49**×)

Outline

- Matrix Factorization
- Weight Pruning
- Quantization method
- Pruning + Quantization + Encoding
- Design small architecture: SqueezeNet

Design small architecture: SqueezeNet

Compression scheme on **pre-trained model**

VS

Design **small CNN architecture** from scratch
(also preserve accuracy?)

SqueezeNet Design Strategies

Strategy 1. Replace 3x3 filters with 1x1 filters

- Parameters per filter: (3x3 filter) = 9 * (1x1 filter)

Strategy 2. Decrease the number of input channels to 3x3 filters

- Total # of parameters: (# of input channels) * (# of filters) * (# of parameters per filter)

Strategy 3. Downsample late in the network so that convolution layers have large activation maps

- Size of activation maps: the size of input data, the choice of layers in which to downsample in the CNN architecture

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

Microarchitecture – Fire Module

Fire module is consist of:

- A *squeeze* convolution layer
  - full of s_{1x1} # of 1x1 filters

- An *expand* layer
  - mixture of e_{1x1} # of 1x1 and e_{3x3} # of 3x3 filters

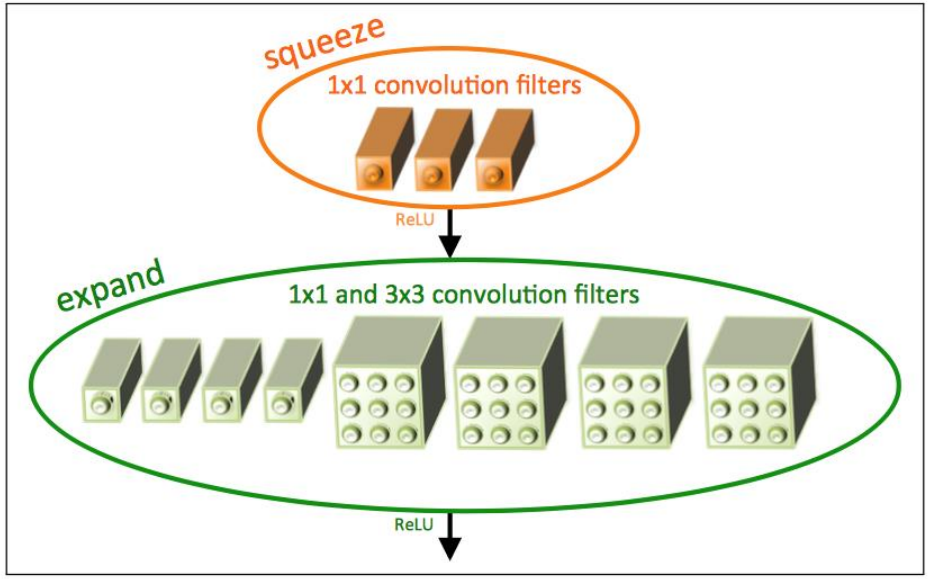


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

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

Microarchitecture – Fire Module

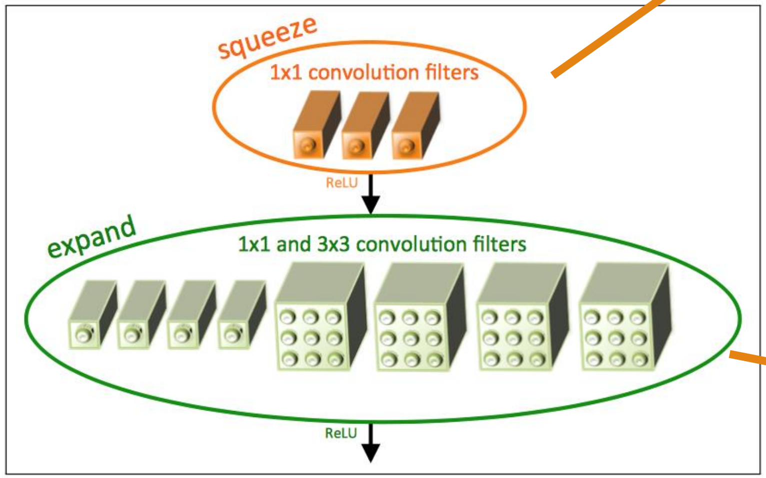


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

Strategy 2. Decrease the number of input channels to 3x3 filters

Total # of parameters: (# of input channels) * (# of filters) * (# of parameters per filter)

Squeeze Layer

Set s_{1x1} < (e_{1x1} + e_{3x3}), limits the # of input channels to 3*3 filters

How much can we limit s_{1x1}?

Strategy 1. Replace 3*3 filters with 1*1 filters

Parameters per filter: (3*3 filter) = 9 * (1*1 filter)

How much can we replace 3*3 with 1*1?
(e_{1x1} vs e_{3x3})?

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

Parameters in Fire Module

The # of expanded filter(e_i)

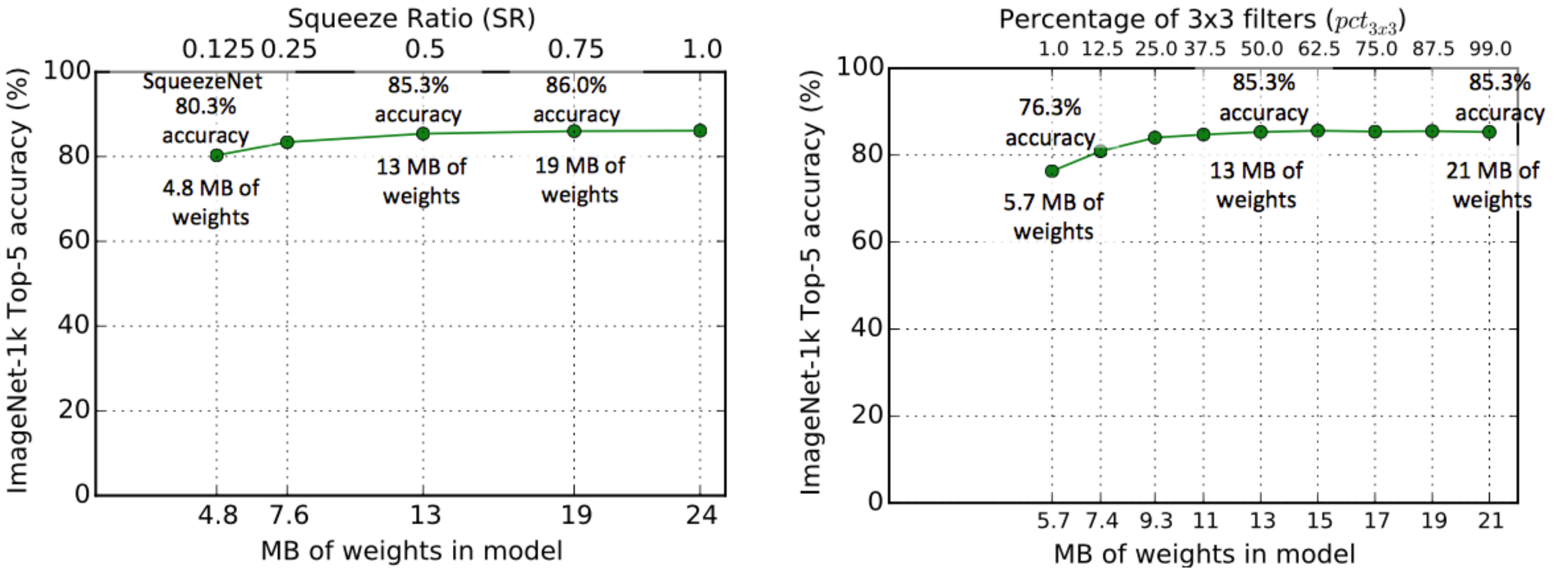
$$e_i = e_{i,1x1} + e_{i,3x3}$$

The % of 3x3 filter in expanded layer(pct_{3x3})

$$e_{i,3x3} = \boldsymbol{pct_{3x3}} * e_i$$

The Squeeze Ratio(SR)

$$s_{i,1x1} = \boldsymbol{SR} * e_i$$



(a) Exploring the impact of the squeeze ratio (SR) on model size and accuracy.

(b) Exploring the impact of the ratio of 3x3 filters in expand layers (pct_{3x3}) on model size and accuracy.

Figure 3: Microarchitectural design space exploration.

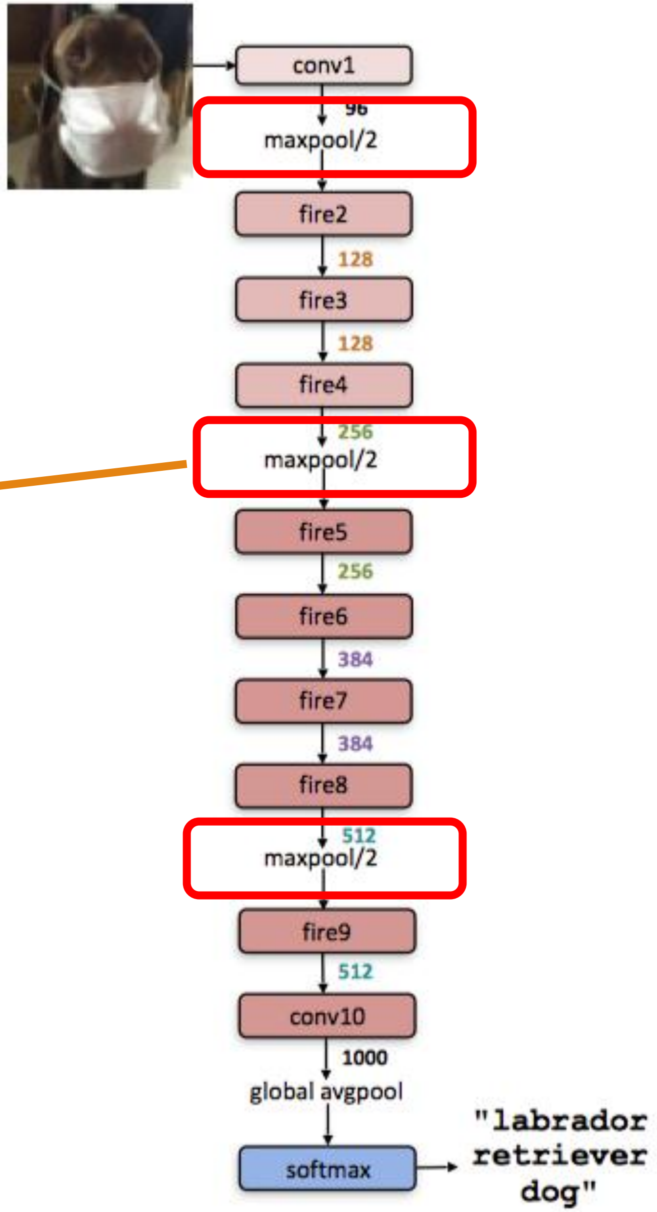
Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size."

Macroarchitecture

Strategy 3. Downsample late in the network so that convolution layers have large activation maps

Size of activation maps: the size of input data, the choice of layers in which to downsample in the CNN architecture

These relative late placements of pooling concentrates activation maps at later phase to **preserve higher accuracy**

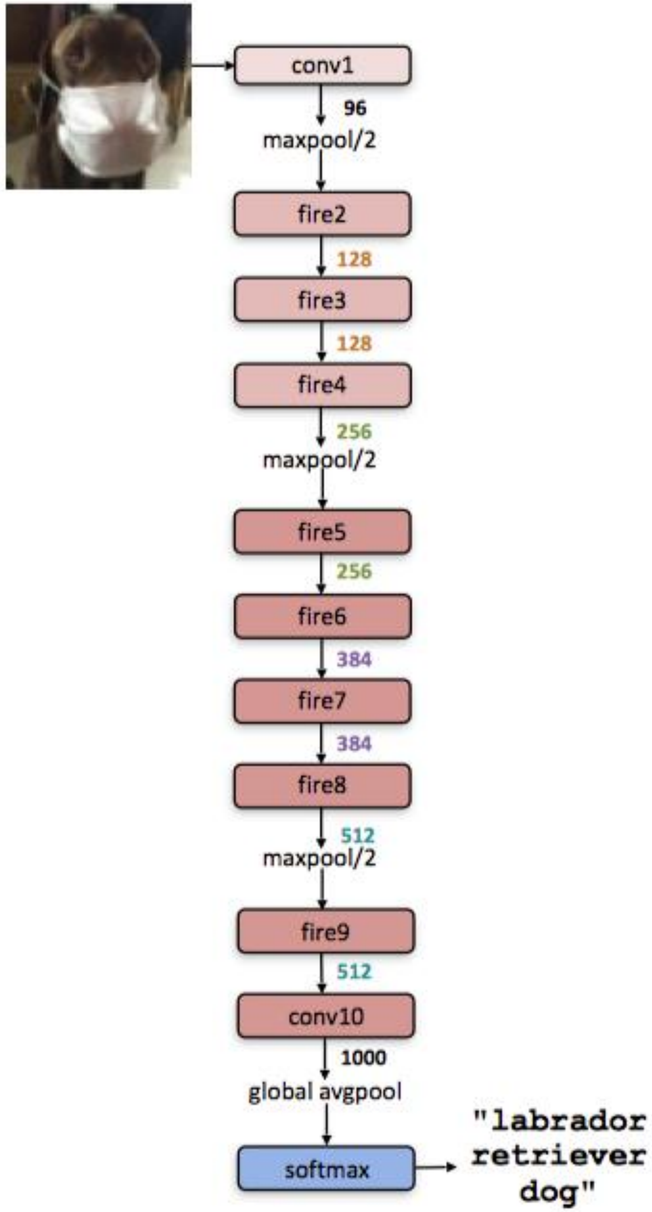


Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size."

Macroarchitecture

Table 1: SqueezeNet architectural dimensions. (The formatting of this table was inspired by the Inception2 paper (Ioffe & Szegedy, 2015).)

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				**20%** (3x3)			6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations	parameters					compression info				1,248,424 (total)	**421,098** (total)



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

Evaluation of Results

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%

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

Further Compression on 4.8M?

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

Further Compression

- Deep Compression + Quantization

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

Takeaway Points

Compress Pre-trained Networks

- **On Single Layer:**
  - Fully connected layer: SVD
  - Convolutional layer: Flattened Convolutions
- **Weight Pruning:**
  - Magnitude-based pruning method is simple and effective, which is the first choice for weight pruning.
  - Retraining is important for model compression.
  - Weight quantization with the full-precision copy can prevent gradient vanishing.
  - Weight pruning, quantization, and encoding are independent. We can use all three methods together for better compression ratio.

Design a smaller CNN architecture

- Example: SqueezeNet
  - Use of Fire module, delay pooling at later stage

Reading List

- Denton, Emily L., et al. "Exploiting linear structure within convolutional networks for efficient evaluation." Advances in Neural Information Processing Systems. 2014.
- Jin, Jonghoon, Aysegul Dundar, and Eugenio Culurciello. "Flattened convolutional neural networks for feedforward acceleration." arXiv preprint arXiv:1412.5474 (2014).
- Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." arXiv preprint arXiv:1412.6115 (2014).
- Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in Neural Information Processing Systems. 2015.
- Guo, Yiwen, Anbang Yao, and Yurong Chen. "Dynamic Network Surgery for Efficient DNNs." Advances In Neural Information Processing Systems. 2016.
- Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.
- Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations." Advances in Neural Information Processing Systems. 2015.
- Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).
- Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).
- Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." arXiv preprint arXiv:1602.07360 (2016).