EEP590 Spring 2022

Deep Learning for Embedded Real Time Intelligence

Lecture 7: Neural Network Compression: Simplifcation, Quantization, Pruning, Encoding, Squeezing, …

Prof. Richard Shi
Department of Electrical and Computer Engineering
(cjshi@uw.edu)

ELECTRICAL & COMPUTER ENGINEERING

Source Acknowledgement:

- S. Yao, Y. Xu and D. Calzada, “Network Compression and Speed up” Lecture Notes 6 PPT
- S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, W. J. Daly, “EIE: Efficient Inference Engine on Compressed Deep Neural Network”, Proceedings of ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA 2016). (PPT)
- Chong Li and C-J Richard Shi, “Constrained Optimization Based Low-Rank Approximation of Deep Neural Networks”, ECCV 2018.
- Arjun Sarkar, Understanding Depthwise Separable Convolutions and the efficiency of MobileNets, June 2021: https://towardsdatascience.com/understanding-depthwise-separable-convolutions-and-the-efficiency-of-mobilenets-6de3d6b62503

References

- 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).

Industry Resources for Implementing Embedded ML

- Intel
  - Intel Neural Compressor: Deploy Low-Precision Inference Solutions on Popular Frameworks; https://www.intel.com/content/www/us/en/developer/tools/oneapi/neural-compressor.html
- Google:
  - TensorFlowLite: https://www.tensorflow.org/lite
- Meta: Facebook Neural Compression Tool Kits
  - https://github.com/facebookresearch/NeuralCompression

More Resources on Embedded ML

- Google's TensorFlow Lite on MCUs:
  - https://www.tensorflow.org/lite/microcontrollers
  - https://www.digikey.com/en/articles/run-machine-learning-code-in-an-embedded-iot-node
- ARM's Work on TinyML on MCUs:
  - https://www.arm.com/blogs/blueprint/tinyml
- ML on MCUs:
  - https://makezine.com/projects/exploring-the-microverse-machine-learning-on-microcontrollers/

Outline

1. Introduction
2. Matrix Factorization
3. Weight Pruning
4. Quantization
5. Deep Compression: Pruning + Quantization + Encoding
6. Design Small Architectures-- SqueezeNet
7. Knowledge Distillation with Teacher-Student Network

Deep Learning on Mobile

Phones

Drones

Robots

Glasses

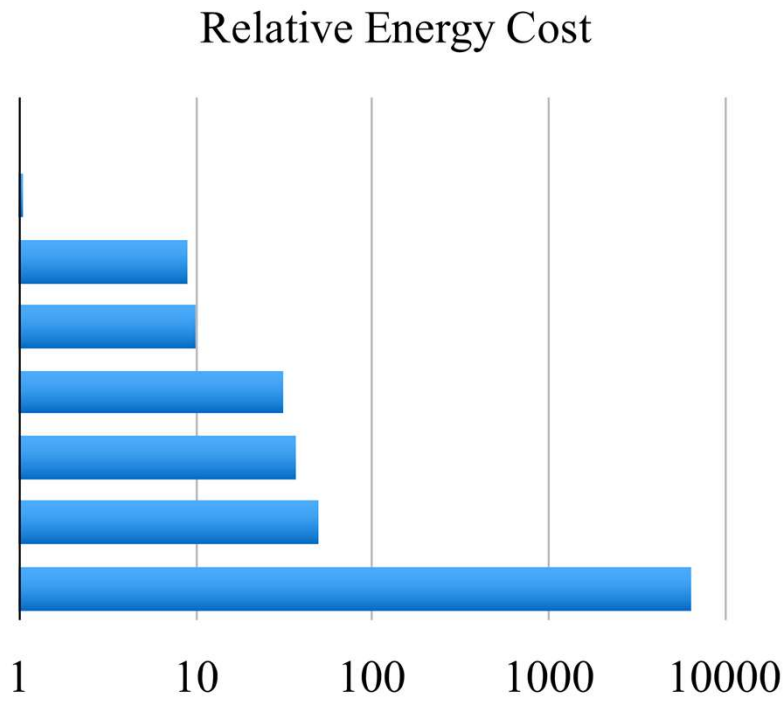
Self Driving Cars

Battery Constrained!

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

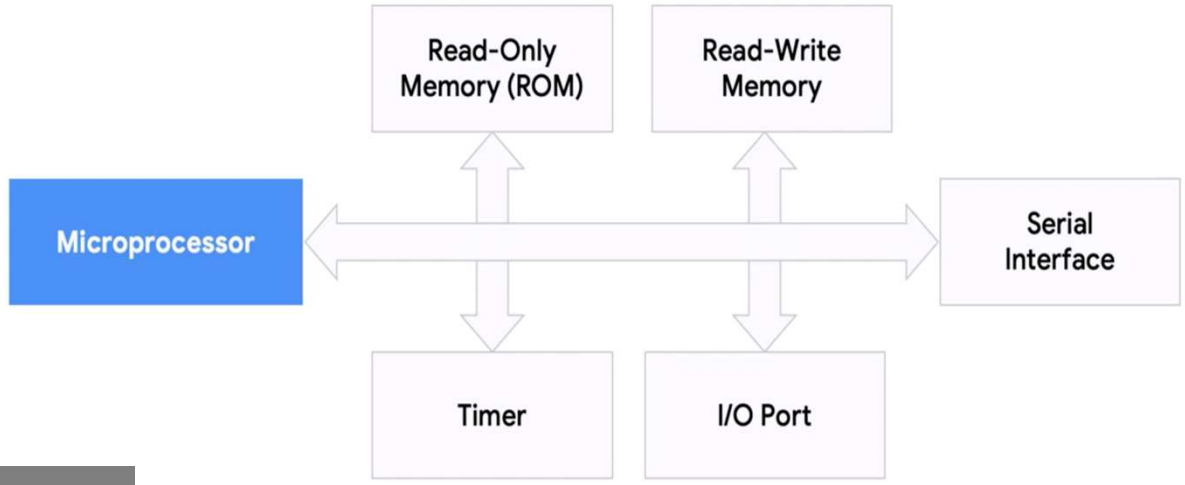
Why Smaller Models?

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



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

Microprocessor vs Microcontroller



Microcontroller

CPU	Read-Only Memory (ROM)	Read-Write Memory
Timer	I/O Port	Serial Interface

Microprocessor vs Microcontollerr

	Microprocessor	>	Microcontroller
Platform			
Compute	1GHz–4GHz	~10X	1MHz–400MHz
Memory	512MB–64GB	~10000X	2KB–512KB
Storage	64GB–4TB	~100000X	32KB–2MB
Power	30W–100W	~1000X	150µW–23.5mW

Ref: https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning

Exponential Growth of DL Model Complexity

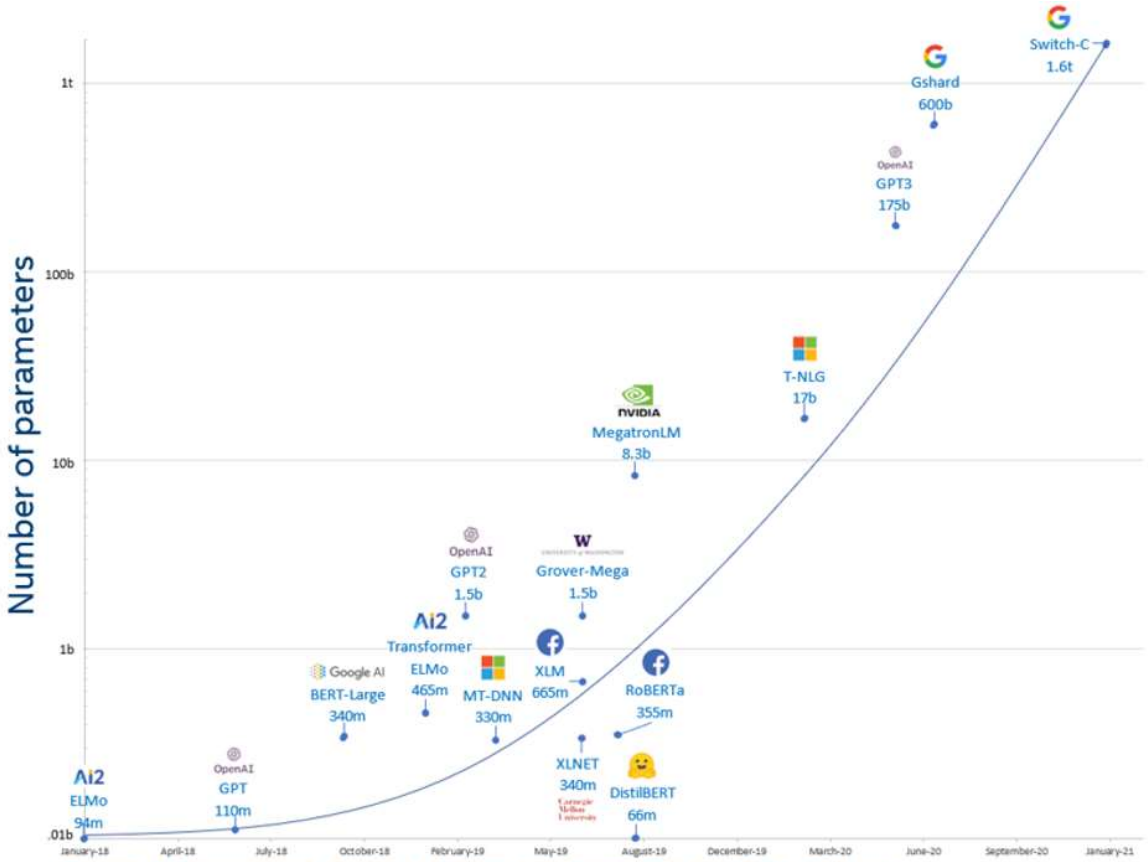
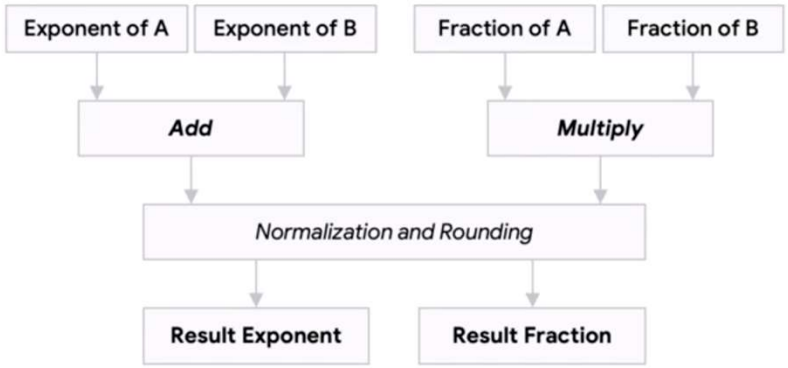


Figure 1: Exponential growth of number of parameters in DL models

ELECTRICAL & COMPUTER ENGINEERING

Floating Point Operation

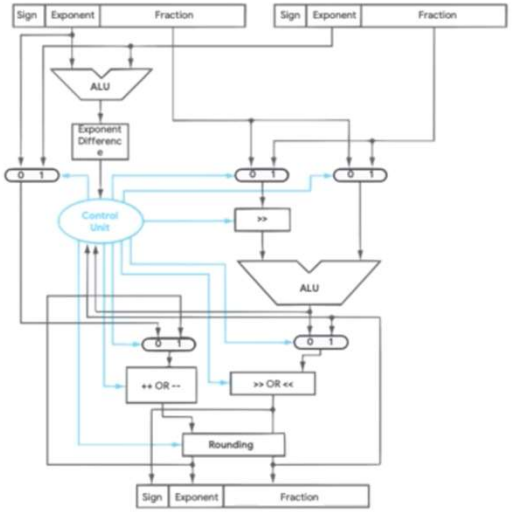
Single Precision
IEEE 754 Floating-Point Standard



Ref: https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning

Floating Point Operation

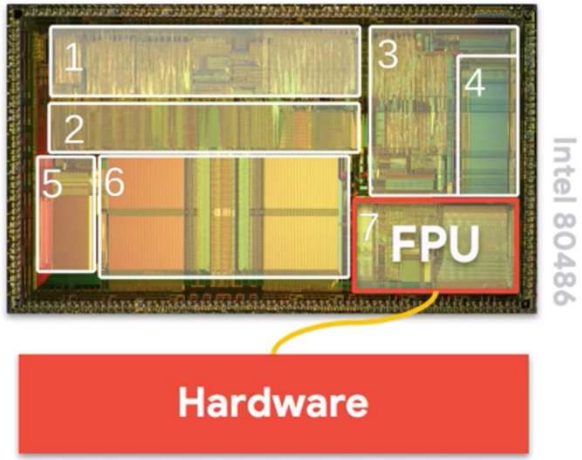
Single Precision
IEEE 754 Floating-Point Standard



Ref: https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning

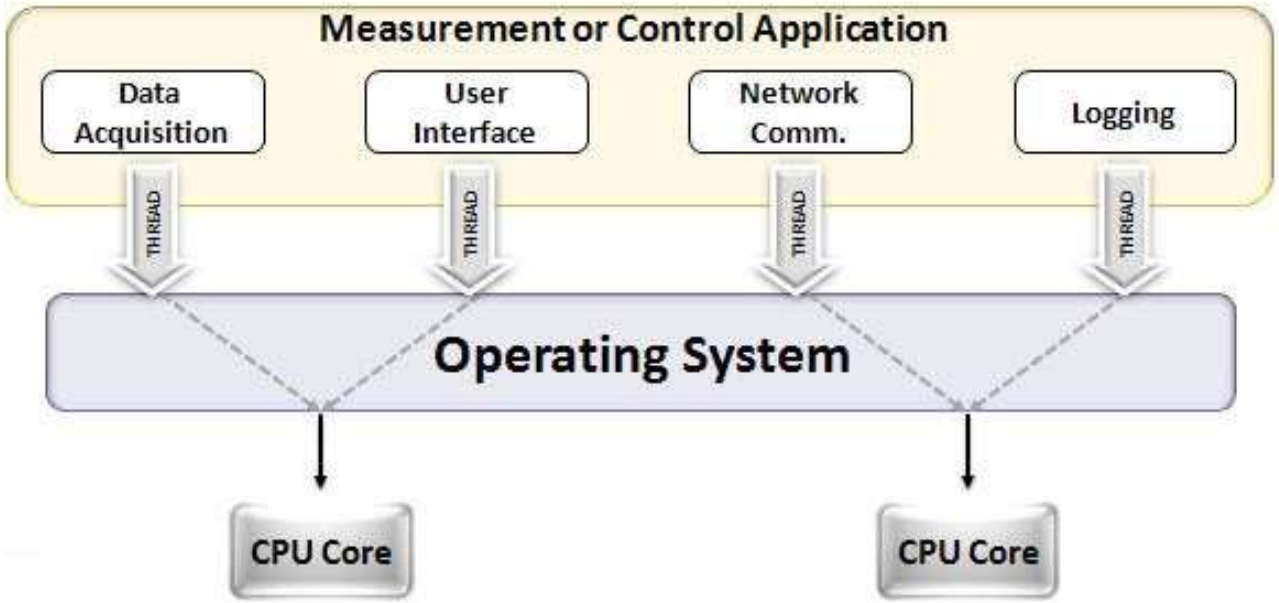
Floating Point Operation

Single Precision
IEEE 754 Floating-Point Standard

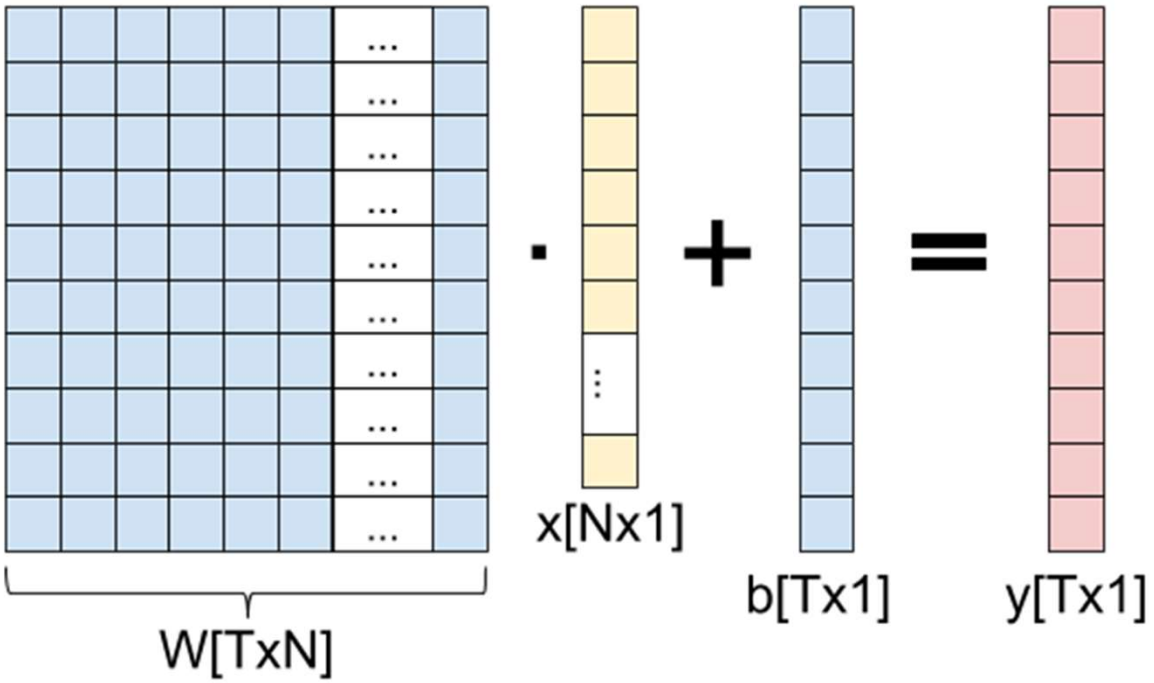


Ref: https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning

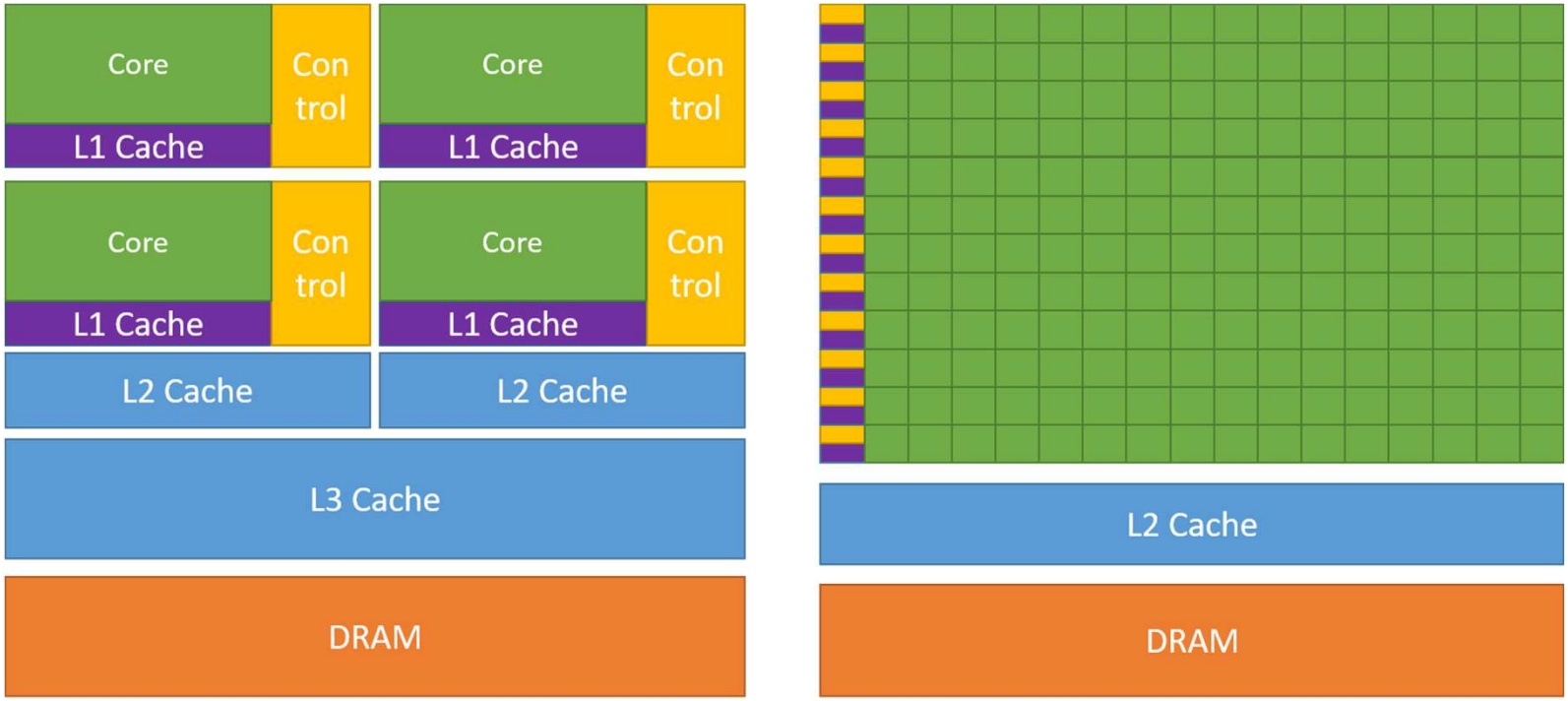
CPU vs GPU for Parallel Operations



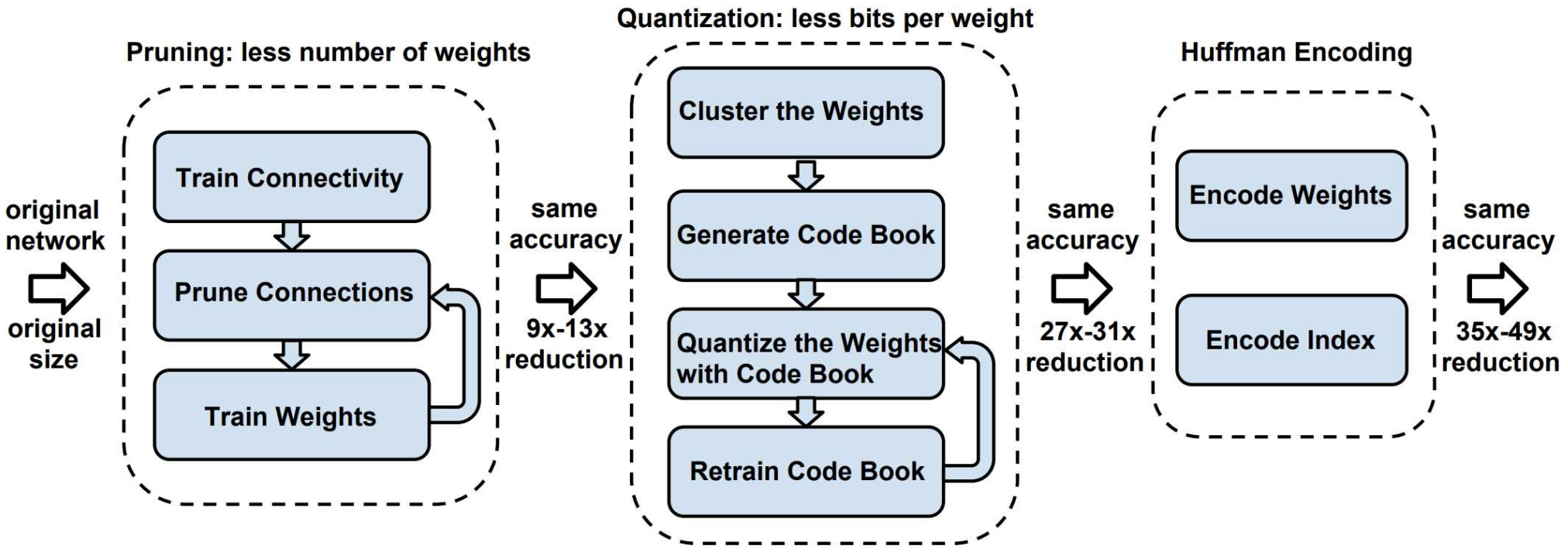
CPU vs GPU for Parallel Operations



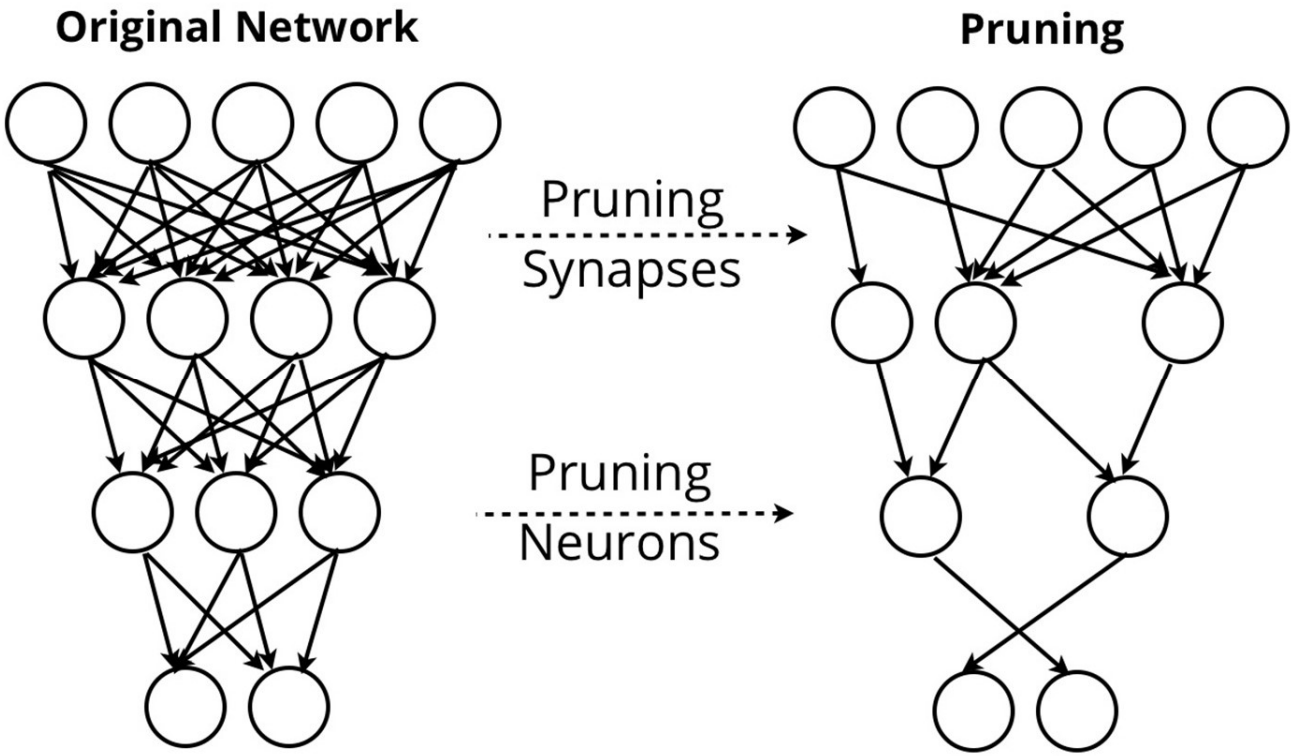
CPU vs GPU



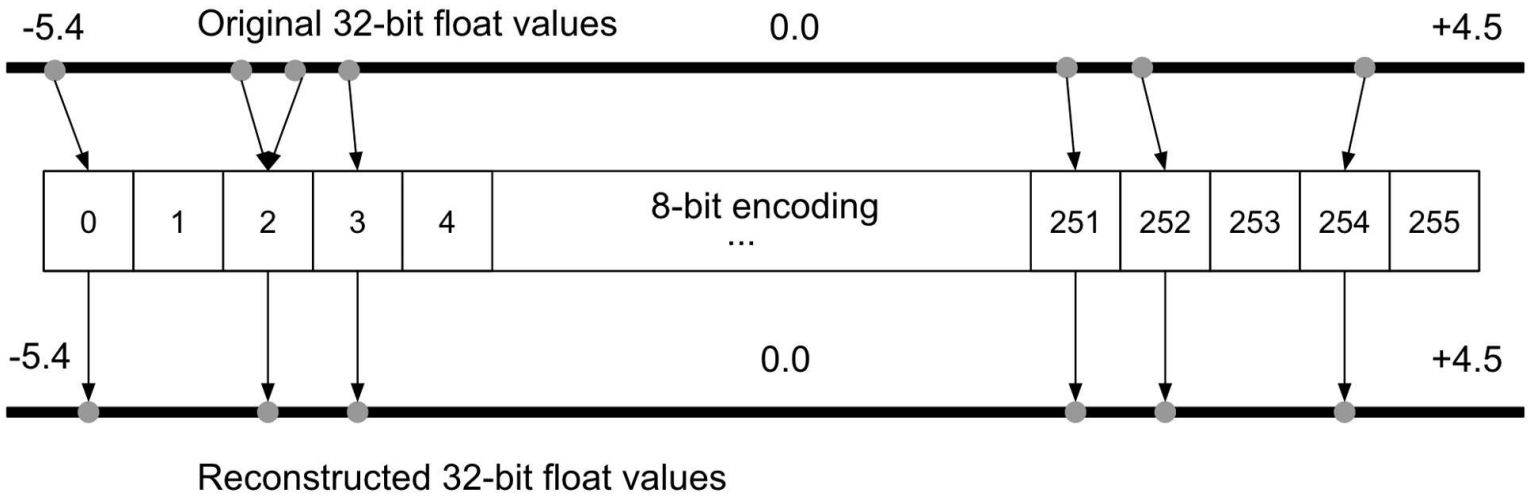
Compression of Deep Learning Models (TinyML example)



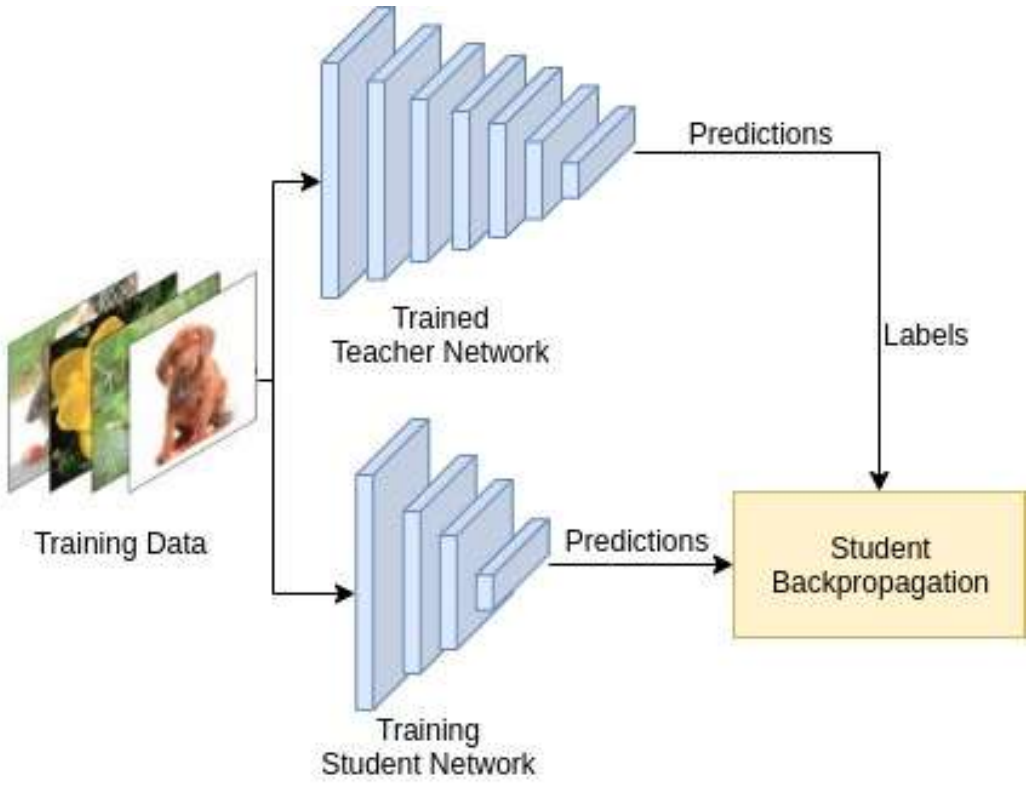
Pruning


Original Network
Pruning
Pruning
Synapses
Pruning
Neurons

Quantization



Knowledge Distillation



Knowledge Distillation

Noname manuscript No.
(will be inserted by the editor)

Knowledge Distillation: A Survey

Jianping Gou[1] · **Baosheng Yu** [1] · **Stephen J. Maybank** [2] · **Dacheng Tao**[1]

Received: date / Accepted: date

Abstract In recent years, deep neural networks have been successful in both industry and academia, especially for computer vision tasks. The great success of deep learning is mainly due to its scalability to

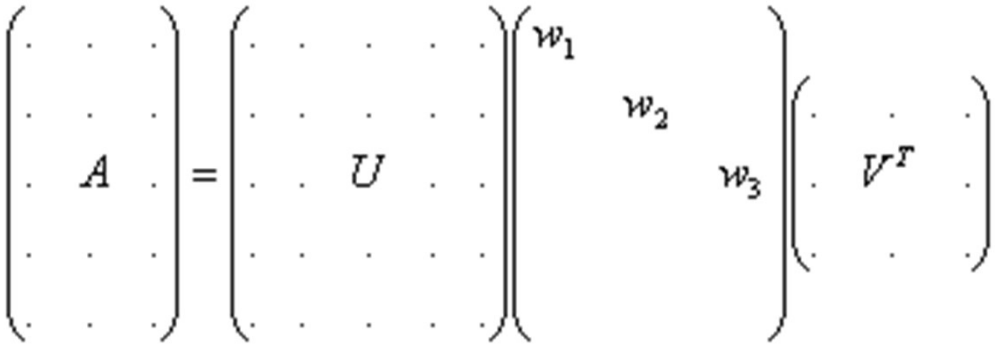
Keywords Deep neural networks · Model compression · Knowledge distillation · Knowledge transfer · Teacher-student architecture.

Outline

1. Introduction
2. **Matrix Factorization**
   - Singular Value Decomposition (SVD)
   - Constrained Optimization for Low-Rank Approximation
   - Flattened Convolutions
3. Weight Pruning
4. Quantization
5. Pruning + Quantization + Encoding
6. 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:
- $\widetilde{W} = \widetilde{U}\tilde{S}\widetilde{V}^{\top}$
  - $\widetilde{W} \in \mathbb{R}^{m\times k}, \widetilde{U} \in \mathbb{R}^{m\times t}, \tilde{S} \in \mathbb{R}^{t\times t}, \tilde{V}^{\top} \in \mathbb{R}^{t\times k}$
- $Rank(\widetilde{W}) \neq t$

$$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & A & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & U & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} w_1 & & \\ & w_2 & \\ & & w_3 \end{pmatrix} \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & V^T & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$$

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

SVD: Compression

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

- 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

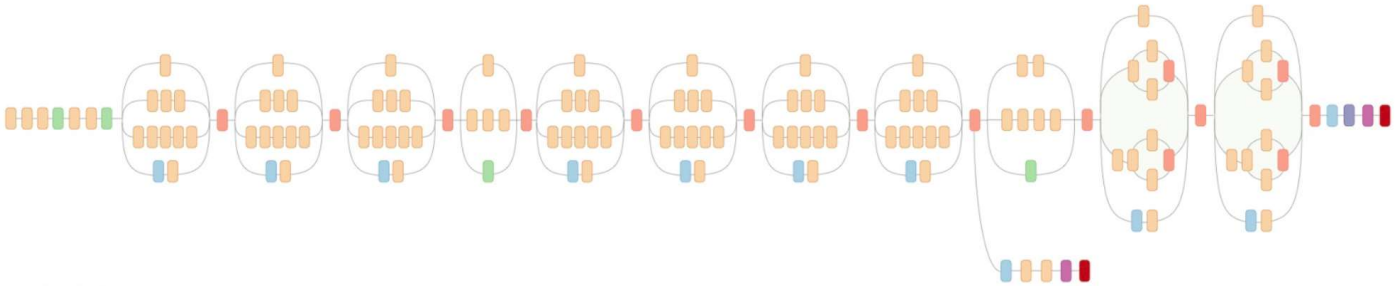
- Reduced memory footprint
  - Reduced in the dense layers by 5-13x
- Speedup: $A\widetilde{W}, A \in \mathbb{R}^{n\times m}$, computed in $O\left(nmt + nt^2 + ntk\right)$ instead of $O(nmk)$
  - Speedup factor is $O\left(\frac{mk}{t(m+t+k)}\right)$
- Regularization
  - “Low-rank projections effectively 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.

Outline

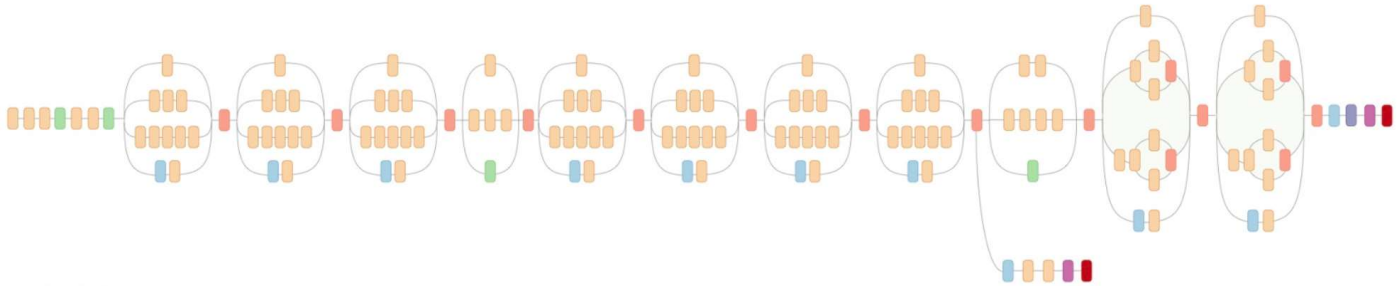
1. Introduction
2. **Matrix Factorization**
   - Singular Value Decomposition (SVD)
   - **Constrained Optimization of Low-Rank Approximation**
   - Flattened Convolutions
3. Weight Pruning
4. Quantization
5. Pruning + Quantization + Encoding
6. Design small architecture: SqueezeNet

Motivation



Model Deployment

Motivation (2)



Basic Idea: To approximate of network weights $\widetilde{W} \approx W$, such that the approximated network is less expensive to evaluate.

Low-Rank Approximation of Matrix

- A is the network weight of a layer, B is the input to the layer

$A \in \mathbb{R}^{k \times l}$ $B \in \mathbb{R}^{l \times p}$

$A \cdot B$ requires $\mathcal{O}(klp)$ operations.

U V $B \in \mathbb{R}^{l \times p}$

With a pre-computed rank r approximation $\widehat{A}$, only requires $\mathcal{O}((k+l)pr)$ operations to approximately compute $A \cdot B$.

Optimal Low-Rank Approximation

$$\widehat{A} = \sum_{j \le r} \sigma_j \cdot U_j \cdot (V_j)^T \text{ is a rank } r \text{ minimizer of } ||A - \widehat{A}||$$

$$\left[\, U_1, U_2, U_3, U_4, U_5 \,\right] \cdot \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_5 & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1^T \\ V_2^T \\ V_3^T \\ V_4^T \\ V_5^T \\ V_6^T \end{bmatrix}$$

$r = 3$

Optimal Low-Rank Approximation is NP-hard

- There is no guarantee that $\widehat{A} = \sum_{j \leq r} \sigma_j \cdot U_j \cdot (V_j)^T$ is a rank r minimizer of $||A \cdot B - \widehat{A} \cdot B||$.
- Weighted low-rank approximation is NP-Hard.
- Consider a slightly relaex problem:

$$\begin{bmatrix} U_1, U_2, U_3, U_4, U_5 \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_5 & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1^T \\ V_2^T \\ V_3^T \\ V_4^T \\ V_5^T \\ V_6^T \end{bmatrix}$$

$r = 2$

With U and V fixed, select r singular values, and set the reset to 0

System-level Budget of Network Evaluation

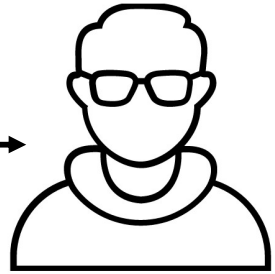
Customer Research

Pricing Strategy

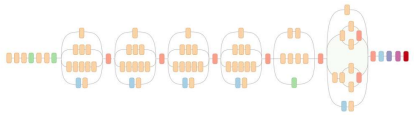
System-Level Budget:

1. Power consumption
2. Latency
3. Model size

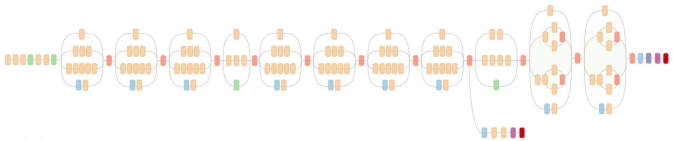
…



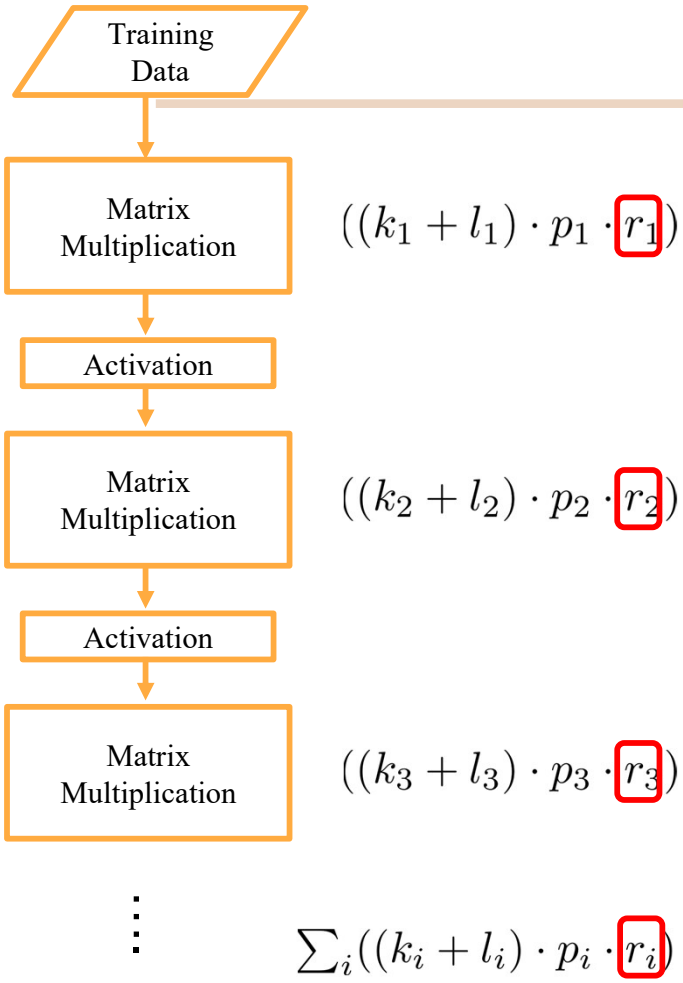
Practitioner



Approximated Network



Trained Network



Objective:

To identify: the optimal *r* value for each layer

To minimize: the loss of the network

Subject to: an upper bound in the number of multiply-accumulate (MAC) operation

$$\sum_i((k_i + l_i) \cdot p_i \cdot r_i) \leq N_{C,max}$$

Hardness of Low-rank Approximation in Neural Networks

A single weight (layer) with known r:

- $\widehat{A} = \arg\min ||A - \widehat{A}||,\ \text{rank}(\widehat{A}) \leq r$ can be easily computed via SVD: $\widehat{A} = \sum_{j \leq r} \sigma_j \cdot U_j \cdot (V_j)^T$
- $\widehat{A} = \arg\min ||A \cdot B - \widehat{A} \cdot B||,\ \text{rank}(\widehat{A}) \leq r$ is NP-Hard.
- $\widehat{A} = \arg\min ||A \cdot B - \widehat{A} \cdot B||,\ \widehat{A} = \sum_{j \in S} \sigma_j \cdot U_j \cdot (V_j)^T, |S| \leq r$ is a subset selection problem.

All the weights (layers) with total resource constraint (unknown r for each layer):

- Key Insight: Assigning a proper r value for each layer subject to resource constraint is a *resource allocation* problem.

Proposed Solution

$$\left[\, U_1, U_2, \boxed{U_3}, U_4, \boxed{U_5} \,\right] \cdot \begin{bmatrix} m_1 \cdot \sigma_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_2 \cdot \sigma_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boxed{m_3 \cdot \sigma_3} & 0 & 0 & 0 \\ 0 & 0 & 0 & m_4 \cdot \sigma_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{m_5 \cdot \sigma_5} & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1^T \\ V_2^T \\ \boxed{V_3^T} \\ V_4^T \\ \boxed{V_5^T} \\ V_6^T \end{bmatrix}$$

- m_i takes binary value {0,1}
- The rank of the matrix is the number of $m_i$that takes value 1 (sum of the masking variables)
- The computation/memory cost of evaluating the approximated network is a function of m_i

$$\begin{aligned} \underset{\boldsymbol{m},\mathcal{P},\mathcal{V}}{\text{minimize}} \quad & E(\boldsymbol{m},\mathcal{P},\mathcal{V}) \\ \text{subject to} \quad & N_C(\boldsymbol{m}) \leq N_{C,max} \\ & N_M(\boldsymbol{m}) \leq N_{M,max} \\ & m_{i,j} \in \{0,1\} \end{aligned} \tag{1}$$

If we were given an optimal solution $\boldsymbol{m}^*$, then the optimal target rank for the ith convolutional layer r_i is simply $\Sigma_j \boldsymbol{m}_{\boldsymbol{i}}^*$

- $E(\vec{\boldsymbol{m}},\mathcal{P},\mathcal{V})$: empirical risk of the network
- $N_C(\vec{\boldsymbol{m}})$: computation cost
- $N_{C,max}$: upper bound of computation cost
- $N_M(\vec{\boldsymbol{m}})$: memory cost
- $N_{M,max}$: upper bound of memory cost
- $m_{i,j}$: masking variable for the jth singular value of the ith layer

$$
\begin{array}{lll}
\underset{\boldsymbol{m},\mathcal{P},\mathcal{V}}{\text{minimize}} & E(\boldsymbol{m},\mathcal{P},\mathcal{V}) & \underset{\boldsymbol{m}}{\text{minimize}} \quad E_{\mathcal{P},\mathcal{V}}(\boldsymbol{m}) \\
\text{subject to} & N_C(\boldsymbol{m}) \leq N_{C,max} & \text{subject to} \quad N_C(\boldsymbol{m}) \leq N_{C,max} \\
& N_M(\boldsymbol{m}) \leq N_{M,max} & \qquad\quad N_M(\boldsymbol{m}) \leq N_{M,max} \\
& m_{i,j} \in \{0,1\} \xrightarrow{\text{Integer Relaxation}} & 0 \leq m_{i,j} \leq 1
\end{array}
\tag{2}
$$

Solution is $\boldsymbol{m}^*$ Solution is $\hat{\boldsymbol{m}}^*$

- A locally optimal solution, denoted by $\hat{\boldsymbol{m}}^*$, can be identified by a constrained non-linear optimization algorithm such as SQP.
- Intuitively, $\hat{\boldsymbol{m}}^*$ quantifies the relative importance of each singular value (and its corresponding singular vectors) in the approximation with a scalar between 0 and 1.
- $\lfloor \Sigma \hat{\boldsymbol{m}}_{\boldsymbol{i}}{}^* \rceil^r$ serves as a surrogate for $\Sigma \boldsymbol{m}_{\boldsymbol{i}}^*$

Architecture of Implementation

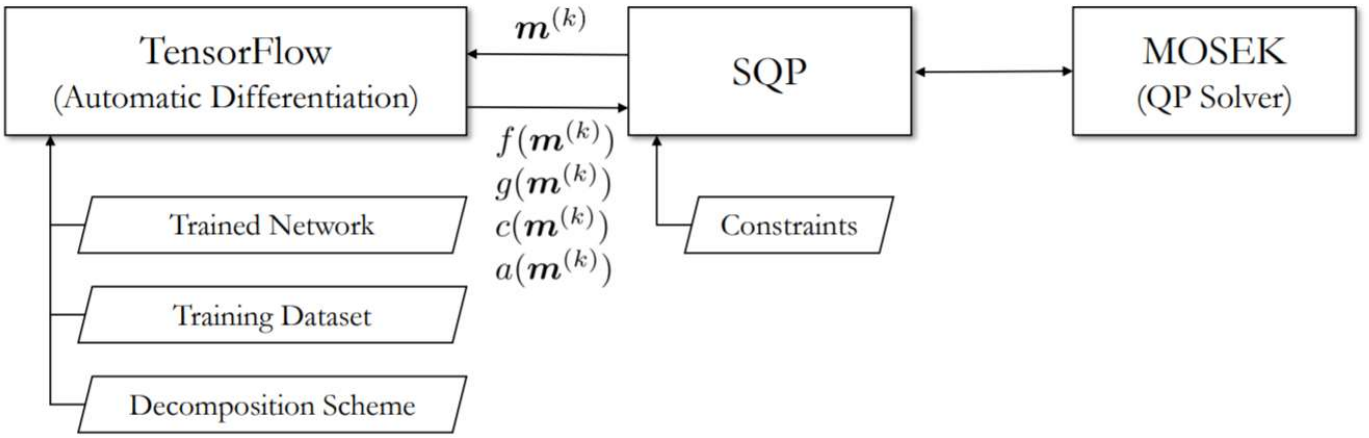


Fig. 4: System overview of COBLA. $\boldsymbol{m}^{(k)}$ is the value of the masking variables at the kth SQP iteration. $f(\boldsymbol{m}^{(k)})$ is the loss, $g(\boldsymbol{m}^{(k)})$ is the gradient of the loss with respect to the masking variables, $c(\boldsymbol{m}^{(k)})$ is the value of the constraint functions, and $a(\boldsymbol{m}^{(k)})$ is the Jacobian of the constraints.

Previous Literature

How the constrained resource is allocated to each layer is defined by the target rank *r* of each layer:

- Manual trial-and-error [Jaderberg2014, Denton 2014, Novikov 2017]
- Certain Percentage of Total Variation (CPTV) [Tai 2016]

$$\sum_{j=1}^{k_i} \sigma_{i,j}^2 \leq \beta \cdot \sum_{\forall j} \sigma_{i,j}^2$$

$\sigma_{i,j}$ is the jth singular value of the ith layer.

Previous Literature (2)

- Product of Sum of Variation (POS) [Zhang 2017]

  maximize the following objective subject to the resources constraint

$$\prod_{\forall i}\left(\sum_{j=1}^{k_i}\sigma_{i,j}^2\right)$$

$\sigma_{i,j}$ is the jth singular value of the ith layer.

- Greedy algorithm is used in [Zhang 2017] to solve this constrained optimization problem.

Previous Literature (3)

- We can significantly improve the greedy algorithm heuristic in [Zhang 2017] using the masking variable formulation

$$\begin{aligned} \underset{\boldsymbol{m}}{\text{maximize}} \quad & \prod_{\forall i} \left(\sum_{\forall j} m_{i,j} \cdot \sigma_{i,j}^2 \right) \\ \text{subject to} \quad & N_C(\boldsymbol{m}) \leq N_{C,max} \\ & N_M(\boldsymbol{m}) \leq N_{M,max} \\ & m_{i,j} \in \{0,1\} \end{aligned}$$

- Modern numerical s with provable optimality

Previous Literature (4)

- Variational Bayesian Matrix Factorization [Nakajima 2013, Kim 2017]

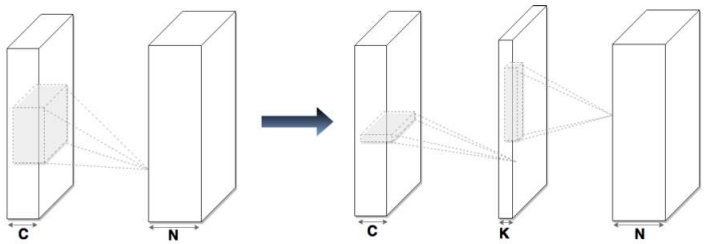
$$V = U + \epsilon$$
$$U = A \cdot B^T$$

  Assume prior distribution and conditional distribution P(V|A,B) are known.

- Low-rank Signal Recovery [Gavish 2013]
  - A signal matrix of unknown *r* is contaminated by noise with known variance
  - The optimal hard threshold of singular value is $4/\sqrt{3} \cdot \sqrt{n} \cdot \sigma$

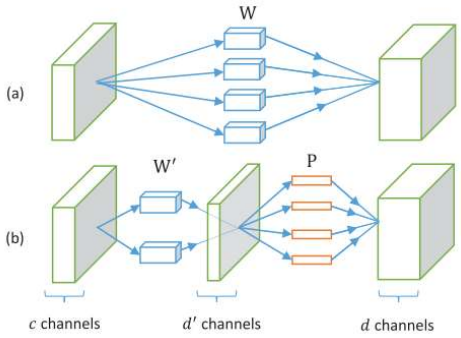
Matricization of the 4-D Weights

- W is 4-D tensor (C,X,Y,N)
- 2-D matrix (CX, YN) (Tai 2016)



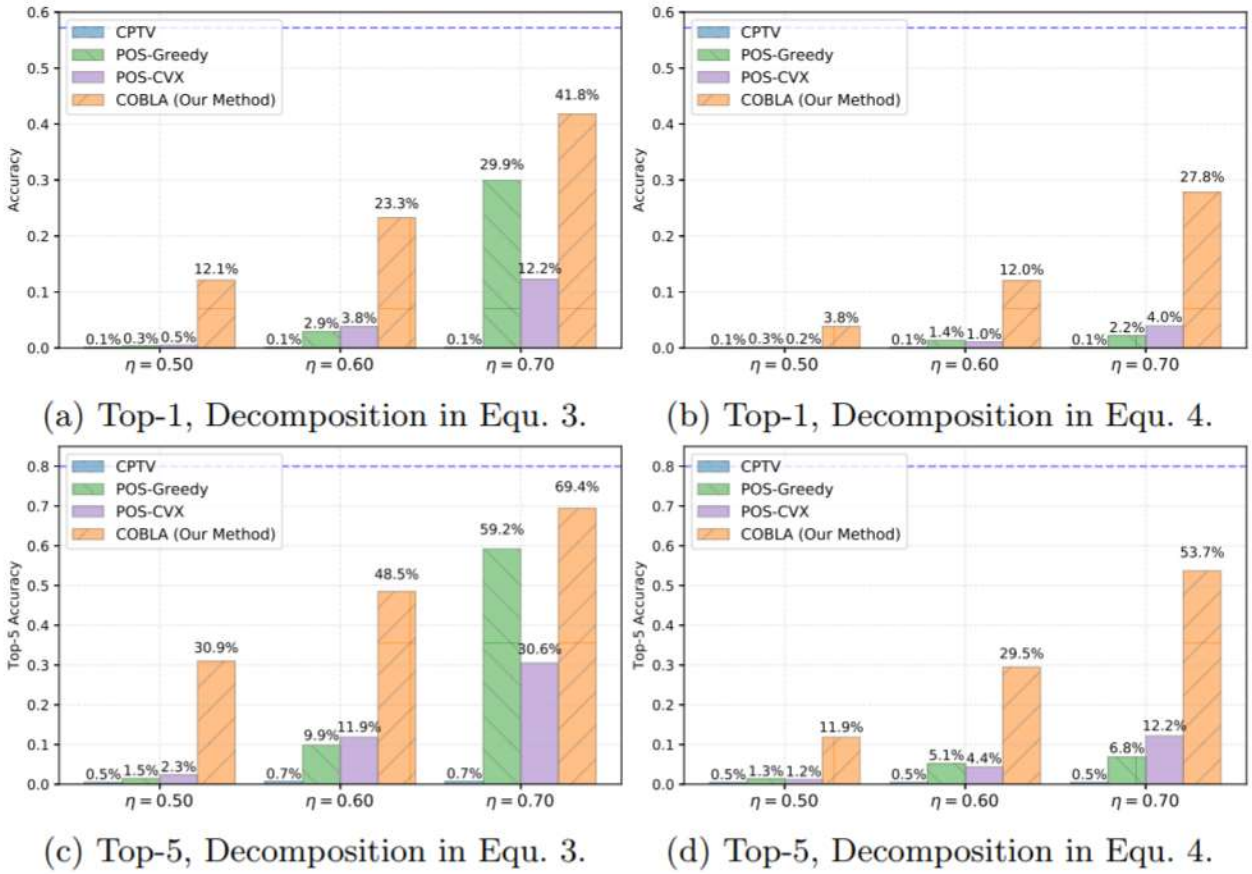
$$W[c',:,:,f'] \approx \sum_{j \in S_{\sigma,i}} P_{f'}^{j} \cdot (V_j^{c'})^T = \sum_{\forall j} m_{ij} \cdot (P_{f'}^{j} \cdot (V_j^{c'})^T) \qquad (3)$$

- 2-D matrix (CXY, N) (Zhang 2016)



$$W_M = \sum_{\forall j} P_j \cdot (V_j)^T \approx \sum_{j \in S_{\sigma,i}} P_j \cdot (V_j)^T = \sum_{\forall j} m_{ij} \cdot (P_j \cdot (V_j)^T) \qquad (4)$$

Accuracy of Compressed SqueezeNet: before fine-tuning



(a) Top-1, Decomposition in Equ. 3.

(b) Top-1, Decomposition in Equ. 4.

(c) Top-5, Decomposition in Equ. 3.

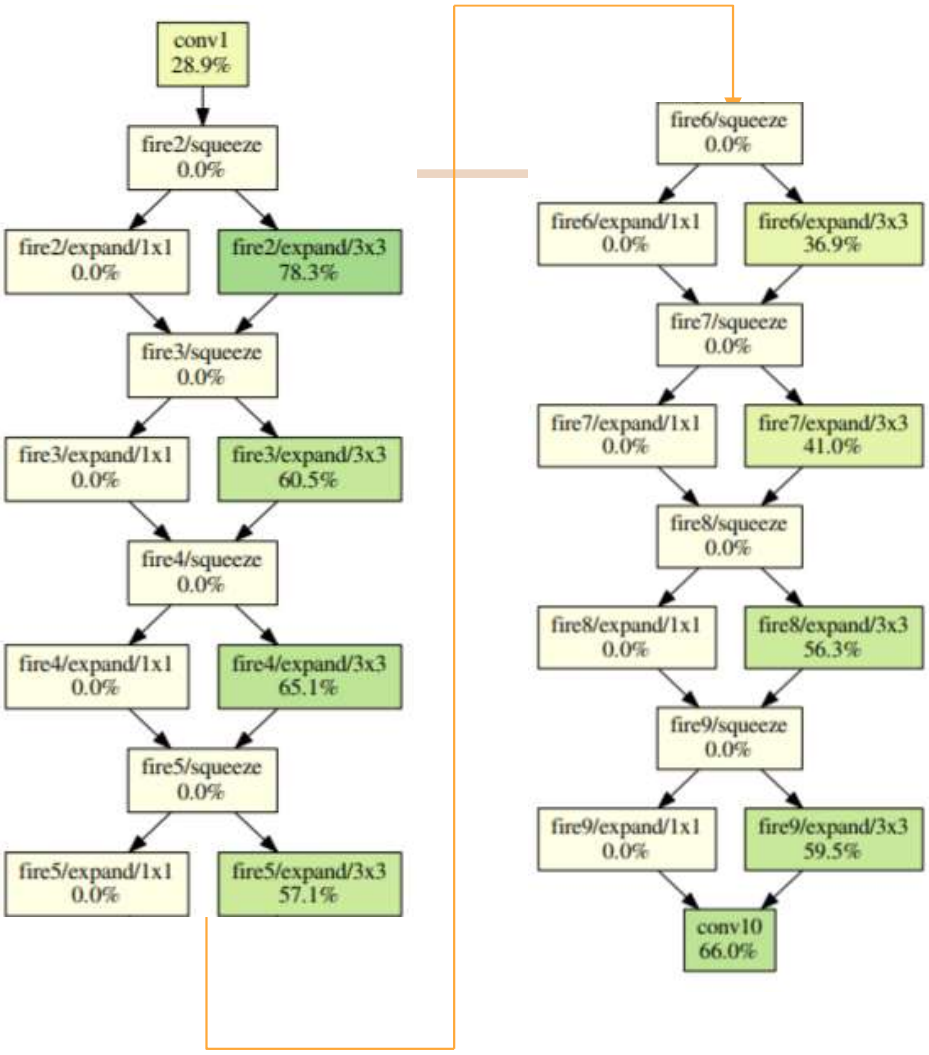
(d) Top-5, Decomposition in Equ. 4.

Accuracy of Compressed SqueezeNet: after fine-tuning

	$N_{C,max}$	$N_{M,max}$	Decomposition Scheme	Top-1 COBLA	Top-1 Baseline	Top-5 COBLA	Top-5 Baseline
1	$0.7 \cdot N_{C,O}$	$0.7 \cdot N_{M,O}$	Equ. 3	55.7%	-2.0%	79.2%	-1.1%
2	$0.7 \cdot N_{C,O}$	$0.7 \cdot N_{M,O}$	Equ. 4	55.4%	-2.4%	78.8%	-1.6%
3	$0.6 \cdot N_{C,O}$	$0.6 \cdot N_{M,O}$	Equ. 3	54.4%	-4.1%	78.2%	-2.7%
4	$0.6 \cdot N_{C,O}$	$0.6 \cdot N_{M,O}$	Equ. 4	54.3%	-3.8%	77.9%	-2.7%
5	$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	Equ. 3	52.6%	-7.4%	77.0%	-5.7%
6	$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	Equ. 4	51.7%	-5.5%	76.2%	-4.1%

Accuracy of Compressed VGG-16

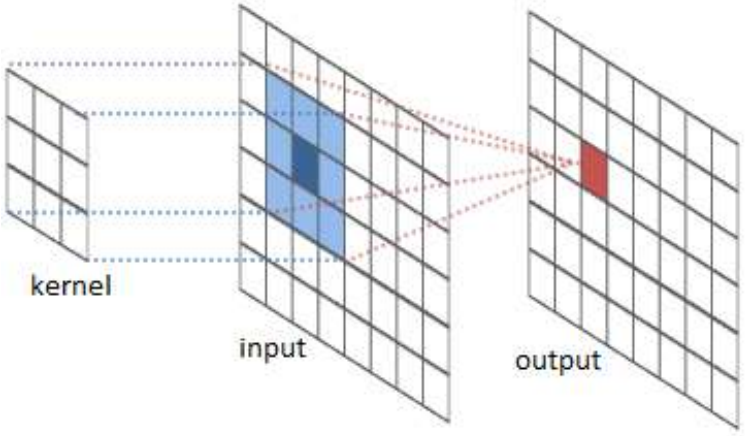
	Computation (Reduction)	Memory (Reduction)	Top-5 Accuracy	Target Rank of Decomposed Layers
Baseline [27]	$0.33 \cdot N_{C,O}$ -	$0.36 \cdot N_{M,O}$ -	89.8% -	5, 24, 48, 48, 64, 128, 160 192, 192, 256, 320, 320, 320
COBLA	$0.29 \cdot N_{C,O}$ (-12%)	$0.32 \cdot N_{M,O}$ (-12%)	89.8% (+0.0%)	5, 17, 41, 54, 77, 109, 133 155, 180, 239, 274, 283, 314
COBLA	$0.23 \cdot N_{C,O}$ (-30%)	$0.25 \cdot N_{M,O}$ (-30%)	88.9% (-0.9%)	5, 16, 32, 48, 64, 81, 95 116, 126, 203, 211, 215, 232



- Percentage number of the amount of computation that can be reduced.
- Green means much computation can be reduced.
- White means exact computation is required.
- The redundancy of the layers is much more complex than hypothesized in [Park 2017]

Convolutions: Matrix Multiplication

Most time is spent in the convolutional layers

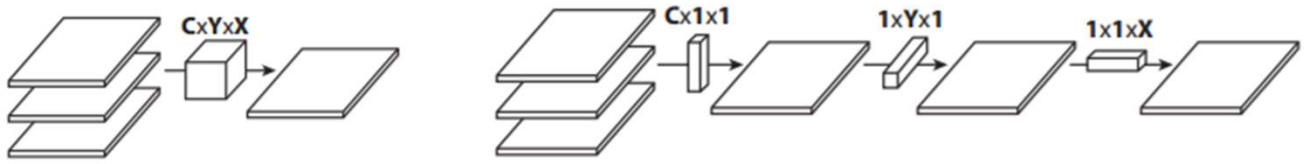


$$F(x, y) = I * W$$

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

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

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

- 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

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

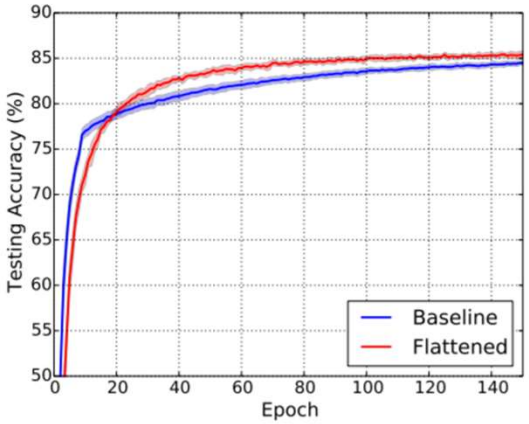
$$= \sum_{x=1}^{X} \sum_{y'=1}^{Y} \sum_{c=1}^{C} I(c, x-x', y-y')\, \widehat{W}(c, x', y')$$

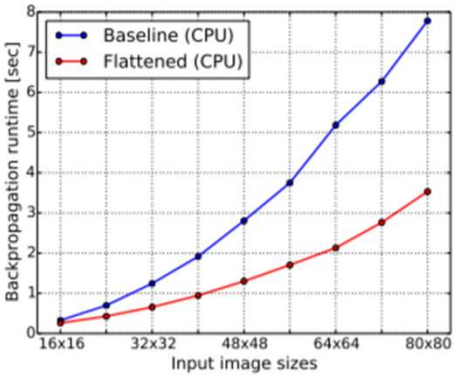
- $\widehat{W} = \alpha \otimes \beta \otimes \gamma, Rank(\widehat{W}) = 1$
- $\widehat{W}_S = \sum_{k=1}^{K} \alpha_k \otimes \beta_k \otimes \gamma_k$, Rank K
- SVD: Can reconstruct the original matrix as $A = \sum_{k=1}^{K} w_k u_k \otimes v_k$

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

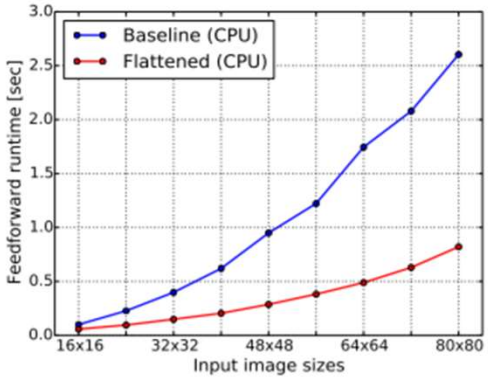
Flattening: Speedup Results

- 3 convolutional layers (5x5 filters) with 96, 128, and 256 channels
- Used stacks of 2 rank-1 convolutions





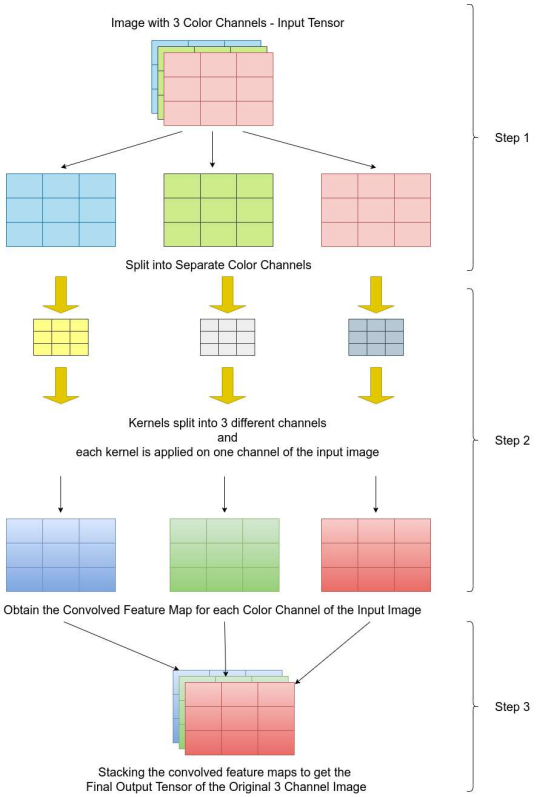
(c) Backpropagation on CPU



(a) Feedforward on CPU

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

Depthwise Convolution



The main difference between 2D convolutions and Depthwise Convolution is that 2D convolutions are performed over all/multiple input channels, whereas in Depthwise convolution, each channel is kept separate.

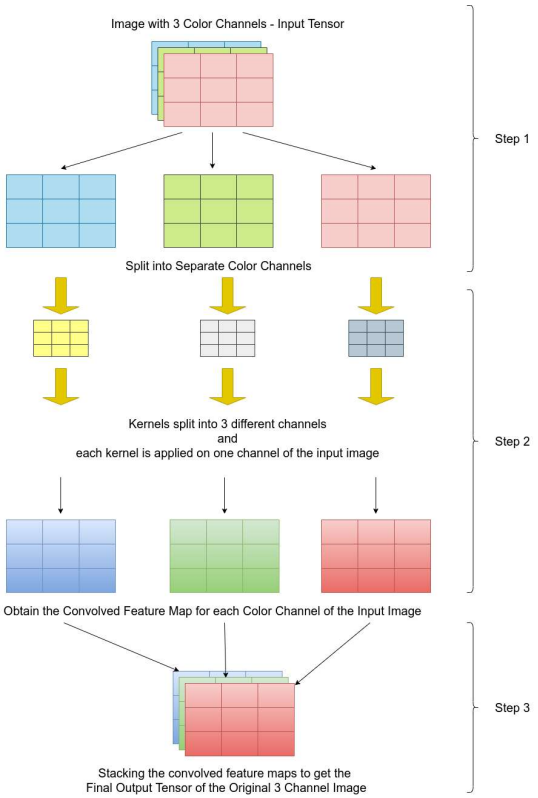
Approach —

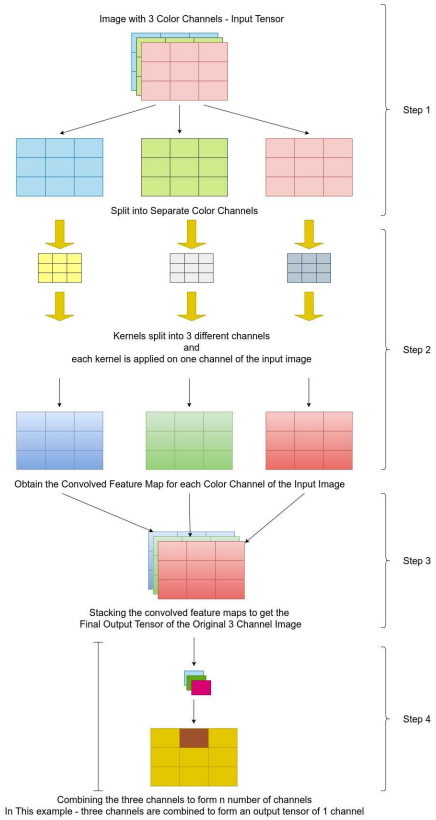
1.Input tensor of 3 dimensions is split into separate channels

2.For each channel, the input is convolved with a filter (2D)

3.The output of each channel is then stacked together to get the output on the entire 3D tensor

Depthwise Separable Convolution





Depthwise convolutions are generally applied in combination with another step — Depthwise Separable Convolution.

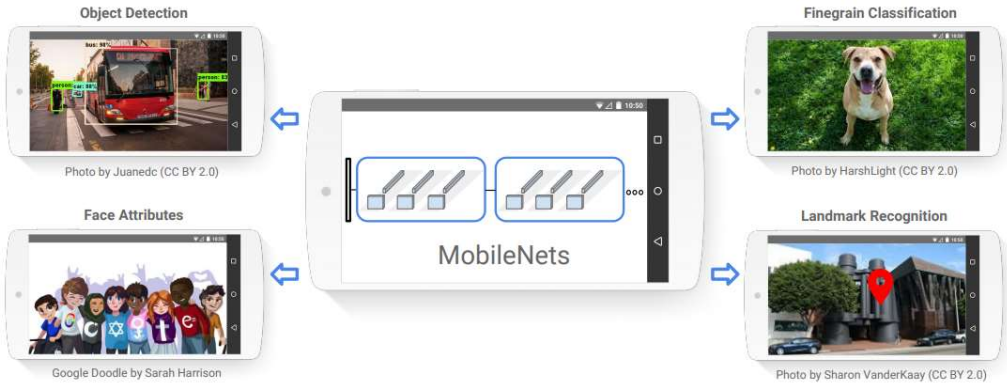
This contains two parts— 1. Filtering (all the previous steps) and 2. Combining (combining the 3 color channels to form ’n’ number of channels, as desired — in the example below we see how the 3 channel can be combined to form a 1 channel output).

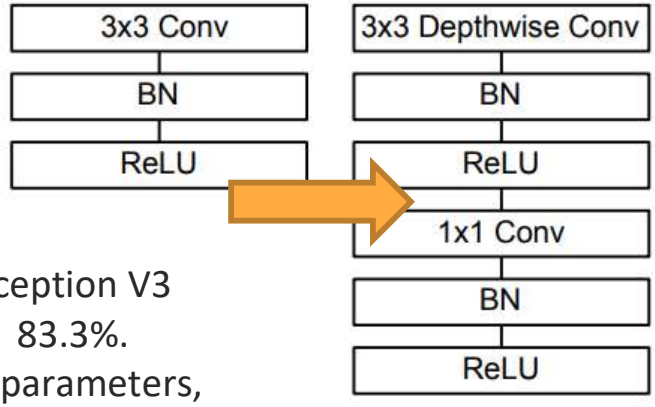
ELECTRICAL & COMPUTER ENGINEERING

Why Depthwise Separable Convolution So Efficient?

Depthwise Convolution is -1x1 convolutions across all channels
Let's assume that we have an input tensor of size — 8x8x3,
And the desired output tensor is of size — 8x8x256
In 2D Convolutions —
Number of multiplications required — (8x8) x (5x5x3) x (256) = ***1,228,800***
In Depthwise Separable Convolutions —
The number of multiplications required:
a. **Filtering** — Split into single channels, so 5x5x1 filter is required in place of 5x5x3, and since there are three channels, so the total number of 5x5x1 filters required is 3, so,(8x8) x (5x5x1) x (3) = 3,800
b. **Combining** — Total number of channels required is 256, so,(8x8) x (1x1x3) x (256) = 49,152
Total number of multiplications = 3,800+49,152 = ***53,952***
So a 2D convolution will require 1,228,800 multiplications, while a Depthwise Separable convolution will require only 53,952 multiplications to reach the same output.
Finally,
1,228,800/53,952 = 23x less multiplications required
These are the layers implemented in the MobileNet architecture.

MobileNets





Stanford dogs dataset, Inception V3 accuracy 84%, MobileNet 83.3%. Inception V3 23.2 million parameters, MobileNet 3.3 million parameters.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

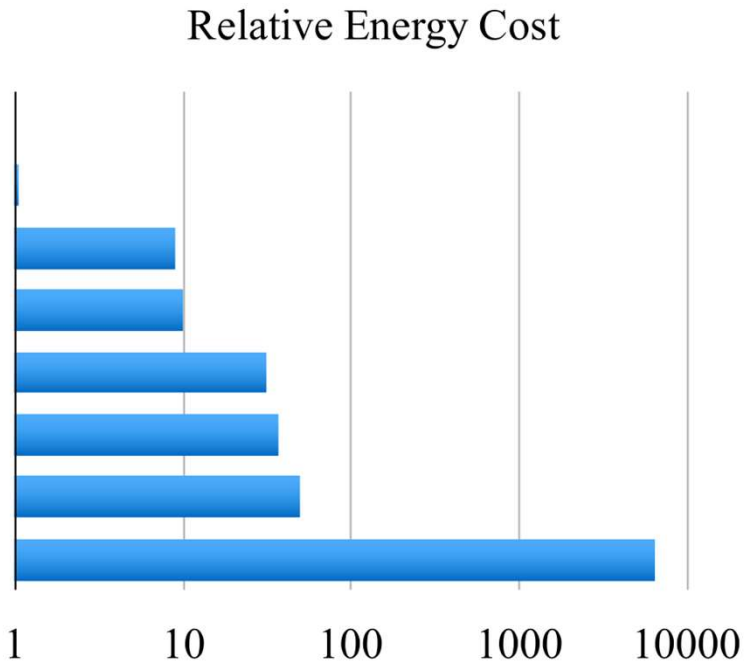
Outline

1. Introduction
2. Matrix Factorization
3. **Weight Pruning**
    - **Magnitude-based method**
        - **Iterative pruning + Retraining**
        - **Pruning with rehabilitation**
    - **Hessian-based method**
4. Quantization
5. Pruning + Quantization + Encoding
6. Design small architecture: SqueezeNet

Magnitude-Based: Iterative Pruning + Retraining

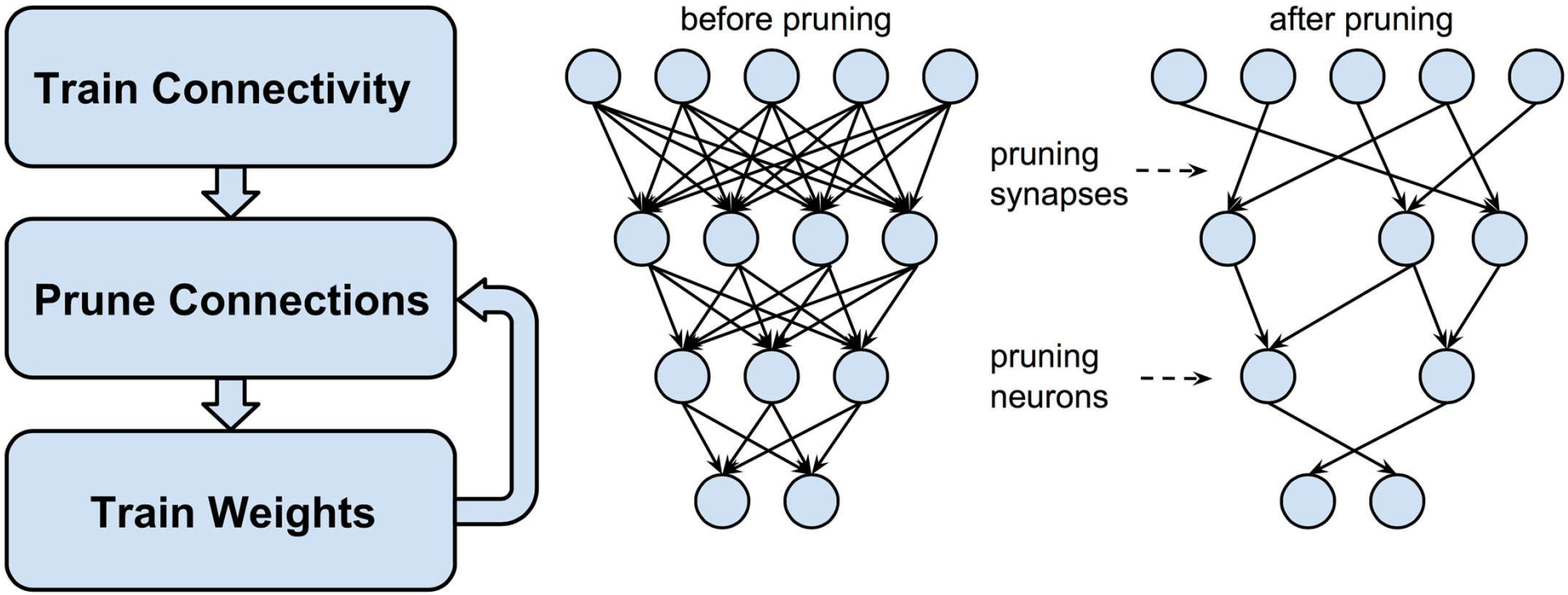
- Pruning connection with small magnitude.
- Iterative pruning an re-training.

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



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

Magnitude-Based: Iterative Pruning + Retraining



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

Magnitude-Based: Iterative Pruning + Retraining

- 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: Iterative Pruning + Retraining

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.

Magnitude-Based: Iterative Pruning + Retraining (Experiment: Lenet)

Lenet-300-100

Layer	Weights	FLOP	Act%	Weights%	FLOP%
fc1	235K	470K	38%	8%	8%
fc2	30K	60K	65%	9%	4%
fc3	1K	2K	100%	26%	17%
Total	266K	532K	46%	8%	8%

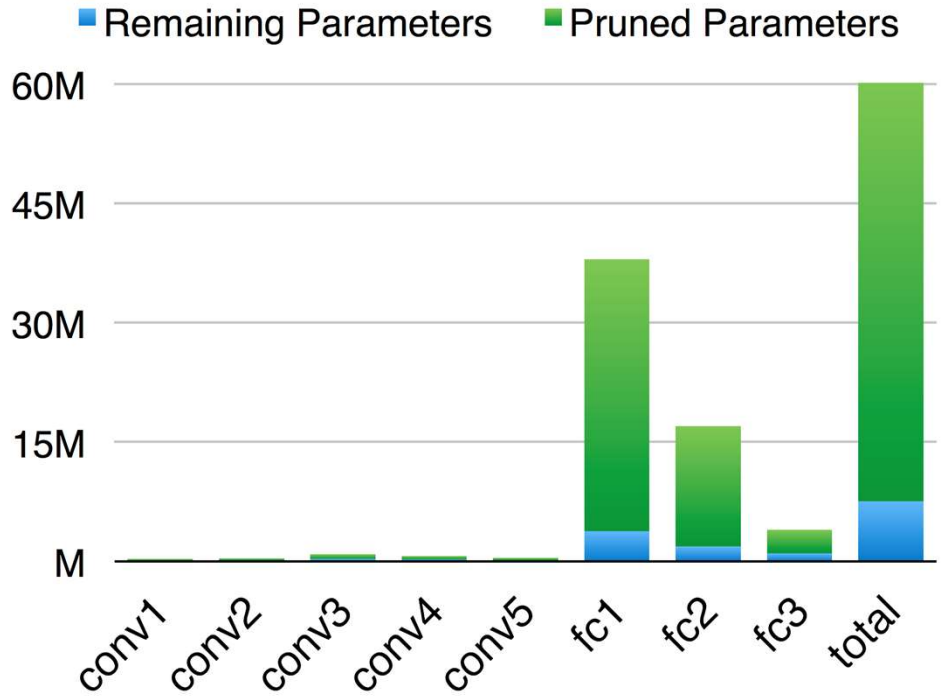
Lenet-5

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	0.5K	576K	82%	66%	66%
conv2	25K	3200K	72%	12%	10%
fc1	400K	800K	55%	8%	6%
fc2	5K	10K	100%	19%	10%
Total	431K	4586K	77%	8%	16%

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

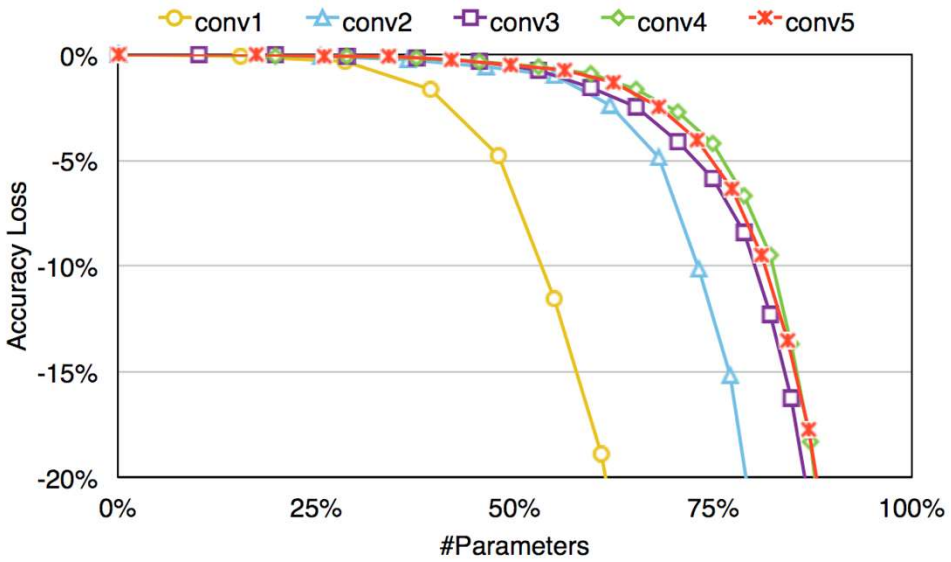
Magnitude-Based: Iterative Pruning + Retraining (Experiment: AlexNet)

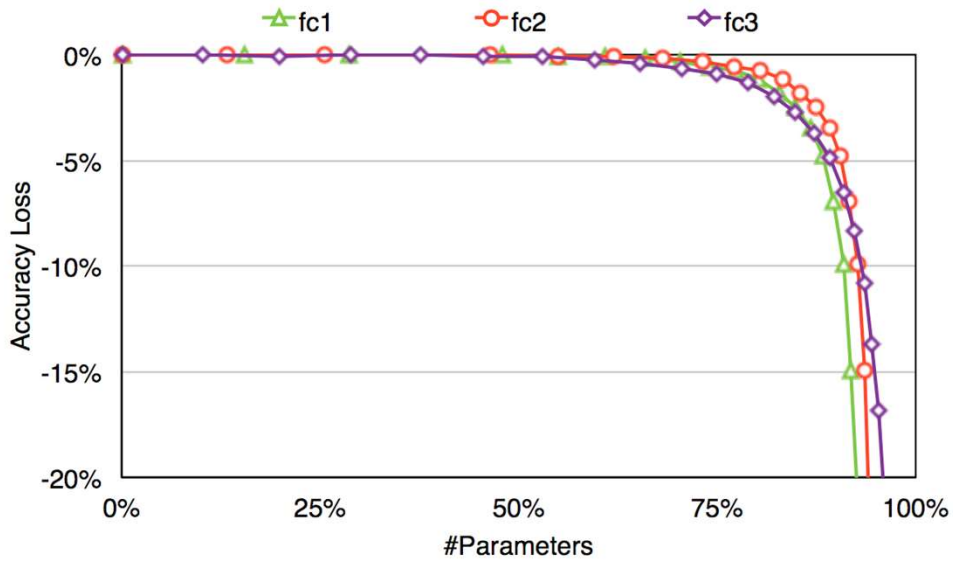
Layer	Weights	FLOP	Act%	Weights %	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10
Total	61M	1.5B	54%	11%	30%



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

Magnitude-Based: 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 LeNet)

Model	Layer	Parameters	Parameters (DNS)
LeNet-5	conv1	0.5K	14.2%
	conv2	25K	3.1%
	fc1	400K	0.7%
	fc2	5K	4.3%
	Total	431K	0.9%
LeNet-300-100	fc1	236K	1.8%
	fc2	30K	1.8%
	fc3	1K	5.5%
	Total	267K	1.8%

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

1. Introduction
2. Matrix Factorization
3. **Weight Pruning**
    - **Magnitude-based method**
        - **Iterative pruning + Retraining**
        - **Pruning with rehabilitation**
    - **Hessian-based method**
4. Quantization
5. Pruning + Quantization + Encoding
6. Design small architecture: SqueezeNet

Diagonal Hessian-Based: 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:
- $\delta E = \sum_i \frac{\partial E}{\partial u_i} \delta u_i + \frac{1}{2} \sum_i \frac{\partial^2 E}{\partial^2 u_i} \delta^2 u_i + \frac{1}{2} \sum_i \frac{\partial^2 E}{\partial u_i \partial u_j} \delta u_i \delta u_j + \mathrm{O}\left(\|\delta U\|^3\right)$

- 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.

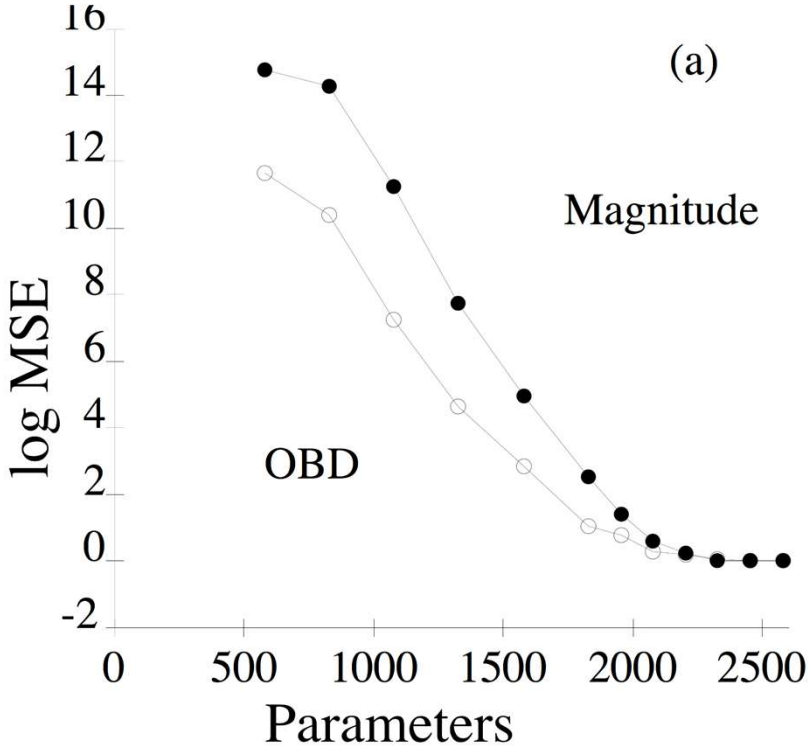
Diagonal Hessian-Based: Optimal Brain Damage (Algorithm)

- 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: 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: Optimal Brain Damage (Experiment: Retraining)

- How retraining helps?

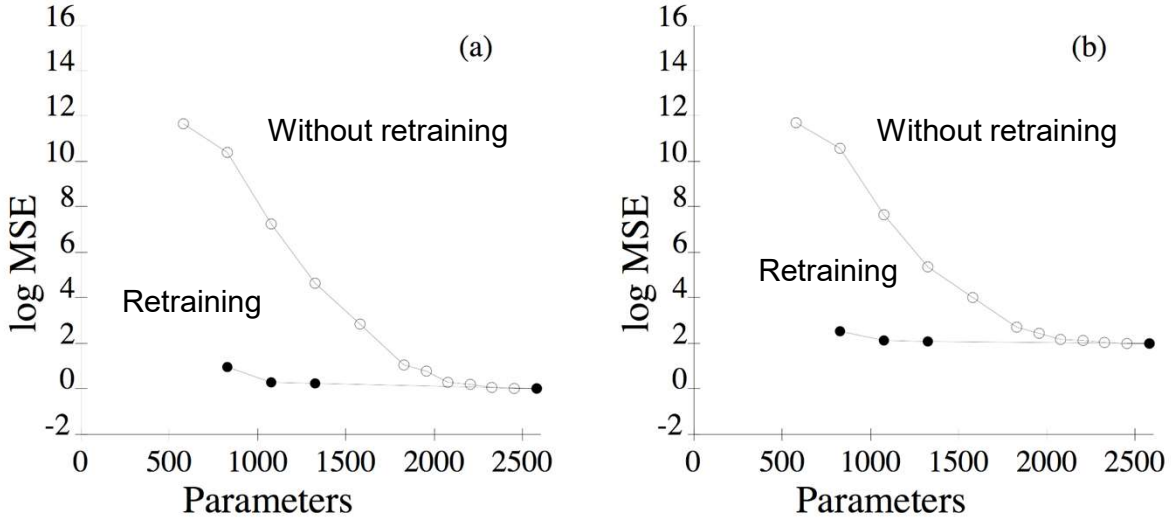


Figure 2: Objective function (in dB) versus number of parameters, without retraining (upper curve), and after retraining (lower curve). Curves are given for the training set (a) and the test set (b).

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

Full Hessian-Based: 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:
- $\delta E = \left(\frac{\partial E}{\partial w}\right)^T \cdot \delta w + \frac{1}{2}\delta w^T \cdot H \cdot \delta w + \mathrm{O}\left(\|\delta W\|^3\right)$
  - 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:
- $\delta \mathrm{w} = -\frac{w_q}{[H^{-1}]_{qq}} H^{-1} \cdot e_q$ and $L_q = \frac{{w_q}^2}{2 \cdot [H^{-1}]_{qq}}$
  - L_q is saliency for weight w_q

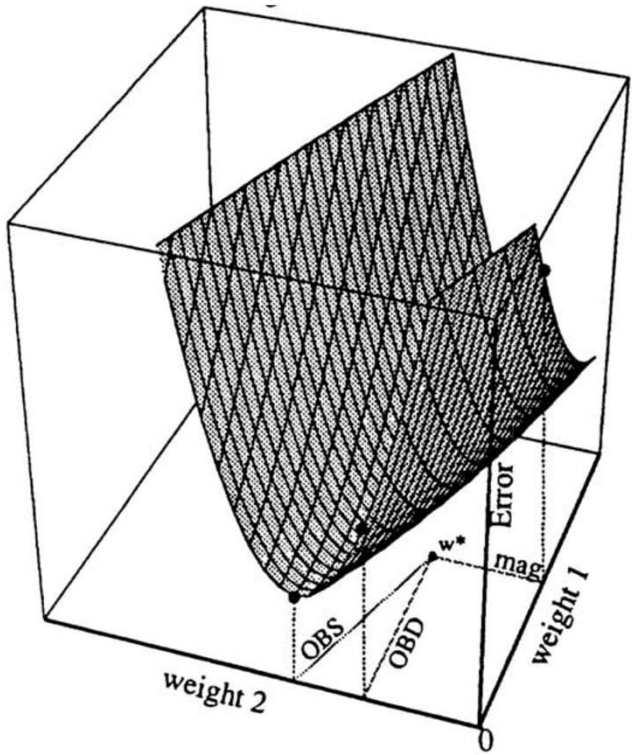
Hassibi, Babak, and David G. Stork. "Second order derivatives for network pruning: Optimal brain surgeon." NIPS, 1993

Full Hessian-Based: 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: Optimal Brain Surgeon



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

Outline

1. Introduction
2. Matrix Factorization
3. Weight Pruning
4. **Quantization**
   - **Full Quantization**
     - **Fixed-point format**
     - **Code book**
   - **Quantization with full-precision copy**
5. Pruning + Quantization + Encoding
6. 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 $\left[-2^{IL-1}, 2^{IL-1} - 2^{-FL}\right]$.

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 $\langle IL, FL \rangle$:

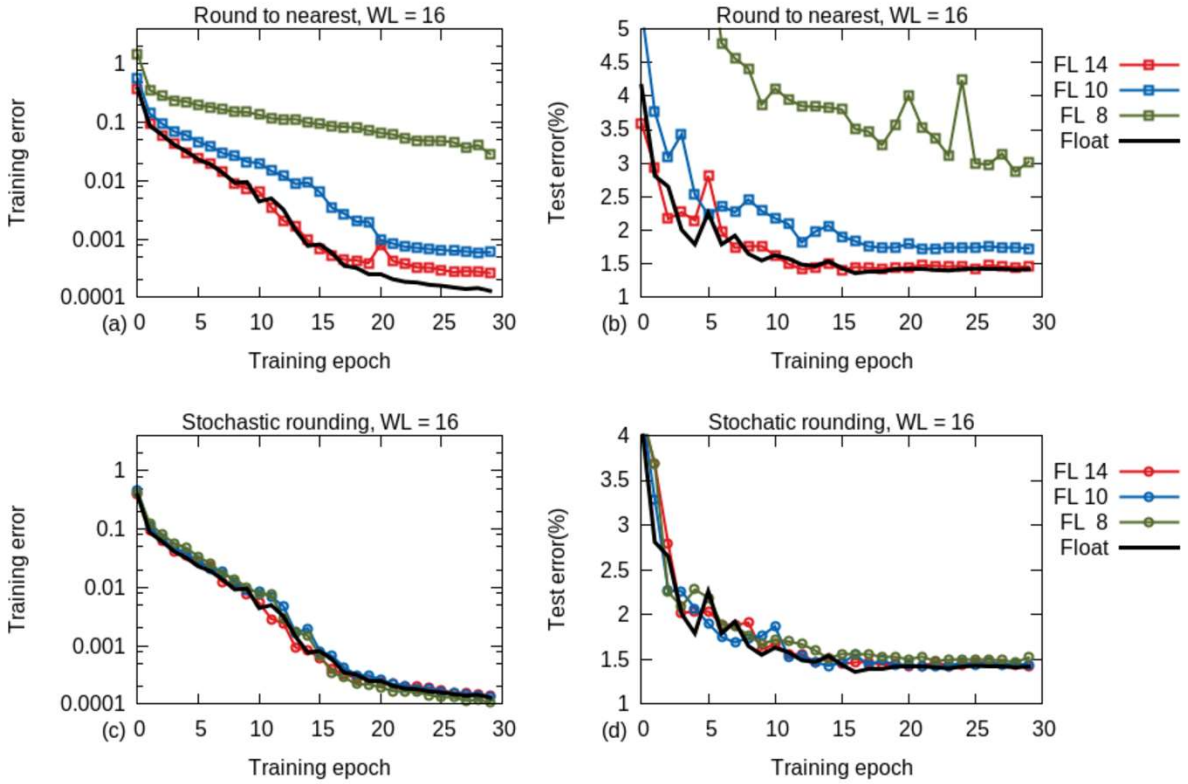
Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.

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 $\langle IL, FL \rangle$.

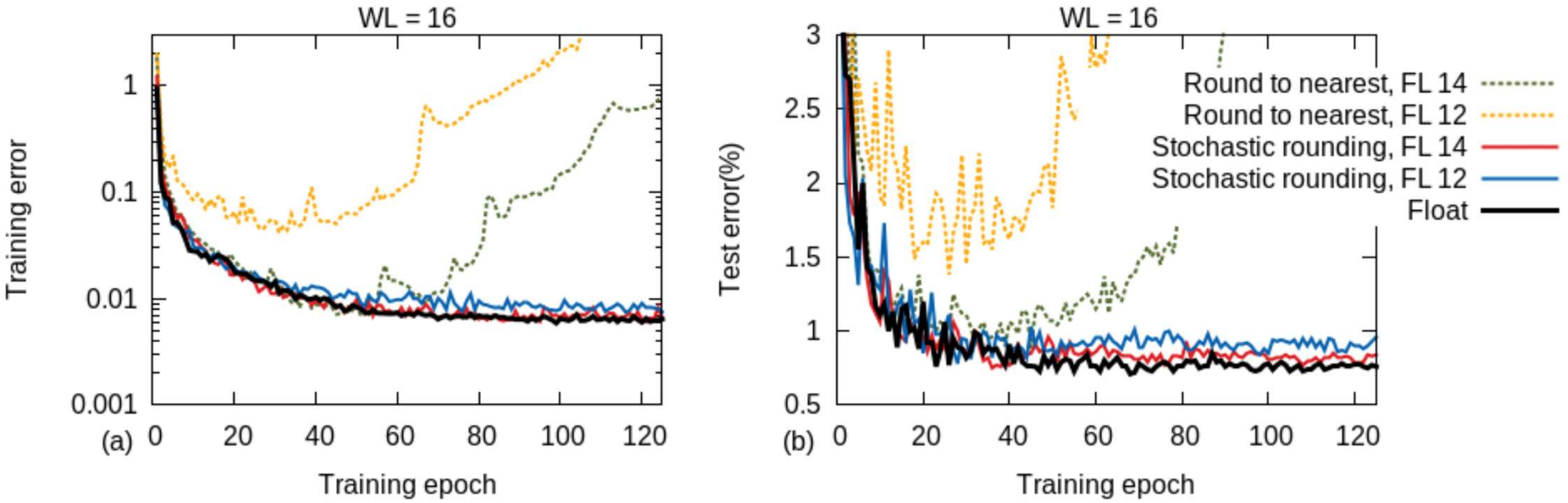
Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.

Full Quantization: Fixed-Point Format (Experiment on MNIST with fully connected DNNs)



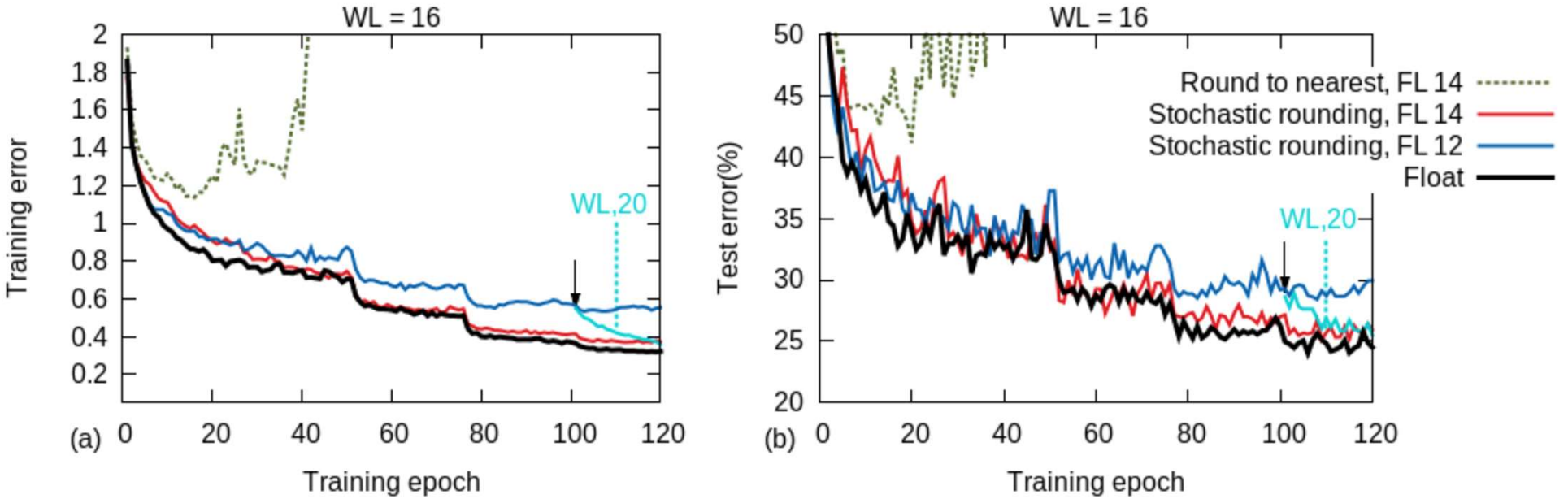
ELECTRICAL & COMPUTER ENGINEERING

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)



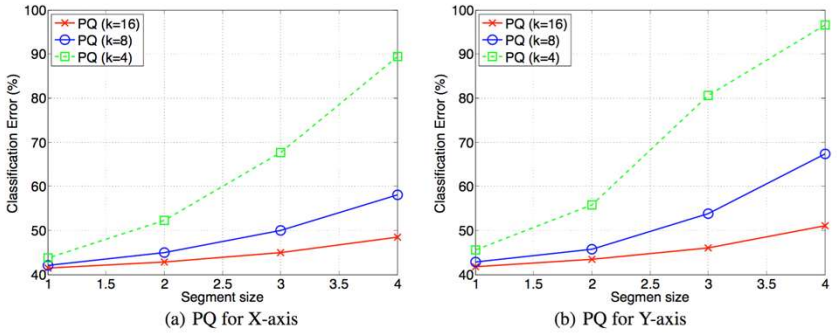
Gupta, Suyog, et al. "Deep Learning with Limited Numerical Precision." ICML. 2015.

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 $\min_z \left\|W_j^i - c_z^i\right\|^2$.
  - 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: $m/(tk + \log_2 k \cdot tn)$

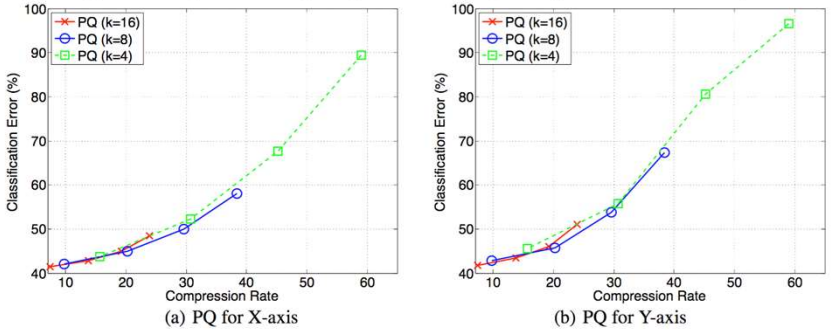
Gong, Yunchao, et al. "Compressing deep convolutional networks using vector quantization." arXiv preprint arXiv:1412.6115 (2014).

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

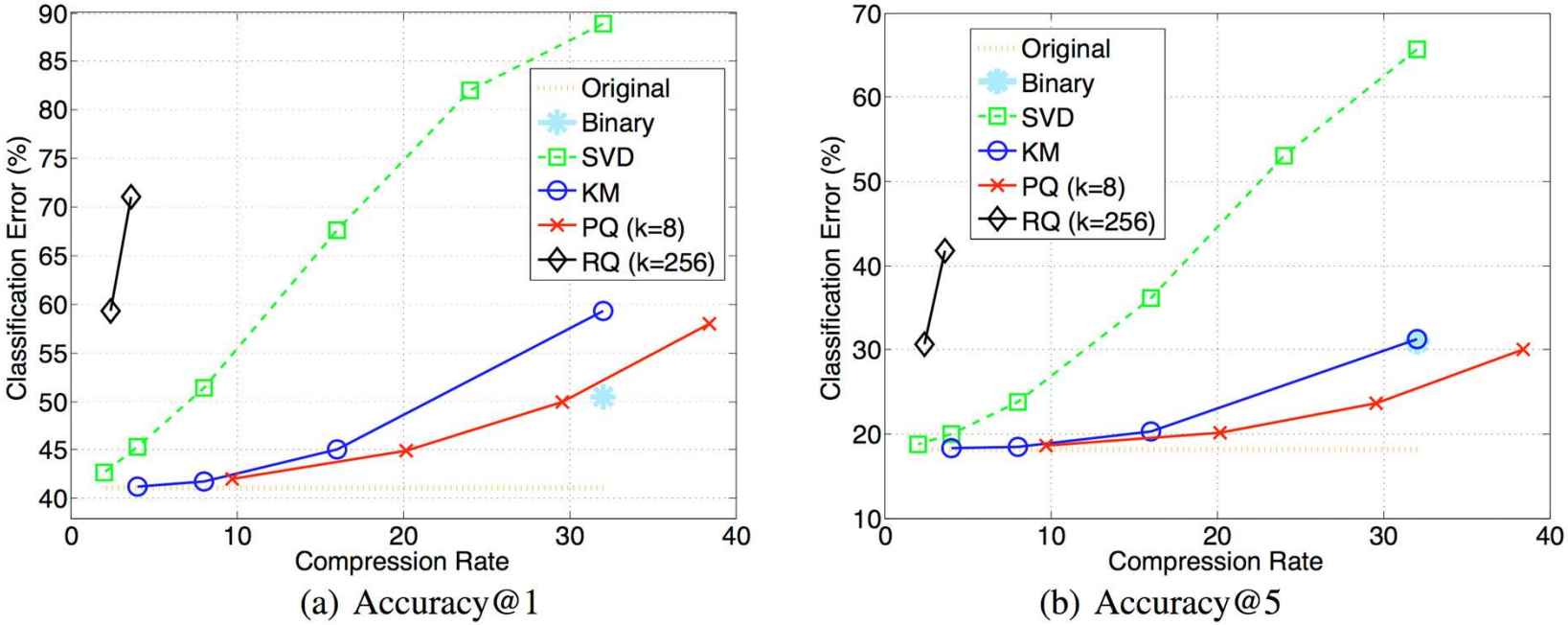


Figure 3: Comparison of different compression methods on ILSVRC dataset.

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

Outline

1. Introduction
2. Matrix Factorization
3. Weight Pruning
4. Quantization
   - Full Quantization
   - Quantization with full-precision copy
     - BinaryConnect
     - BNN
5. Pruning + Quantization + Encoding
6. 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

Algorithm 1 SGD training with BinaryConnect. C is the cost function for minibatch and the functions binarize(w) and clip(w) specify how to binarize and clip weights. L is the number of layers.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η.

Ensure: updated parameters w_t and b_t.

1. Forward propagation:

$w_b \leftarrow \text{binarize}(w_{t-1})$

For $k = 1$ to L, compute a_k knowing a_{k-1}, w_b and b_{t-1}

2. Backward propagation:

Initialize output layer's activations gradient $\frac{\partial C}{\partial a_L}$

For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b

3. Parameter update:

Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{db_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}

$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$

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).

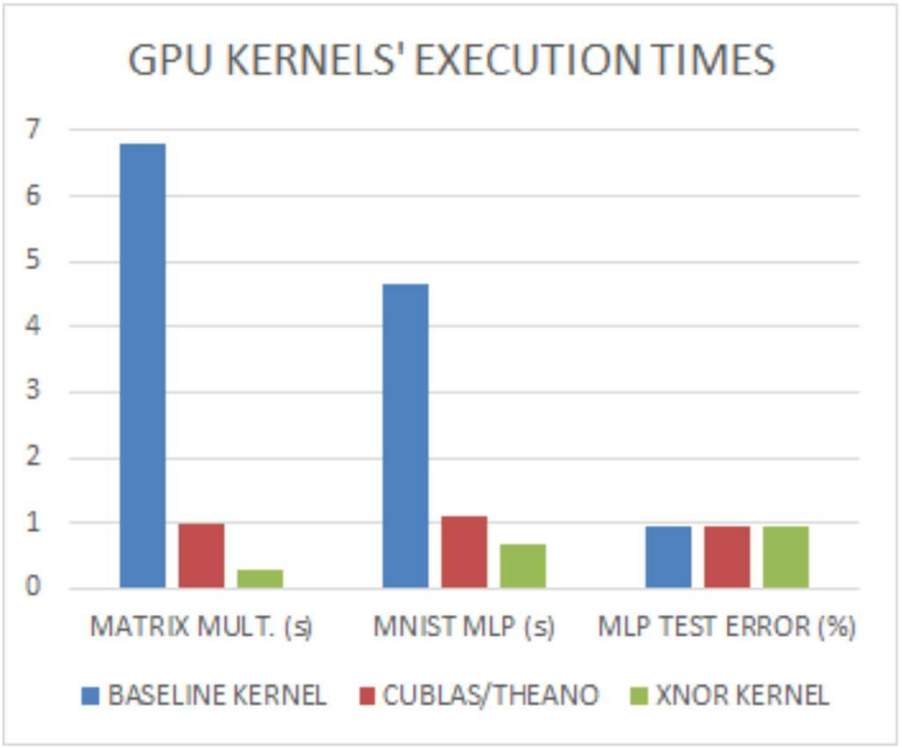
Quantization with full-precision copy: Binarized Neural Networks

Operation	MUL	ADD
8bit Integer	0.2pJ	0.03pJ
32bit Integer	3.1pJ	0.1pJ
16bit Floating Point	1.1pJ	0.4pJ
32bit Floating Point	3.7pJ	0.9pJ

Memory size	64-bit memory access
8k	10pJ
32k	20pJ
1M	100pJ
DRAM	1.3-2.6nJ

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

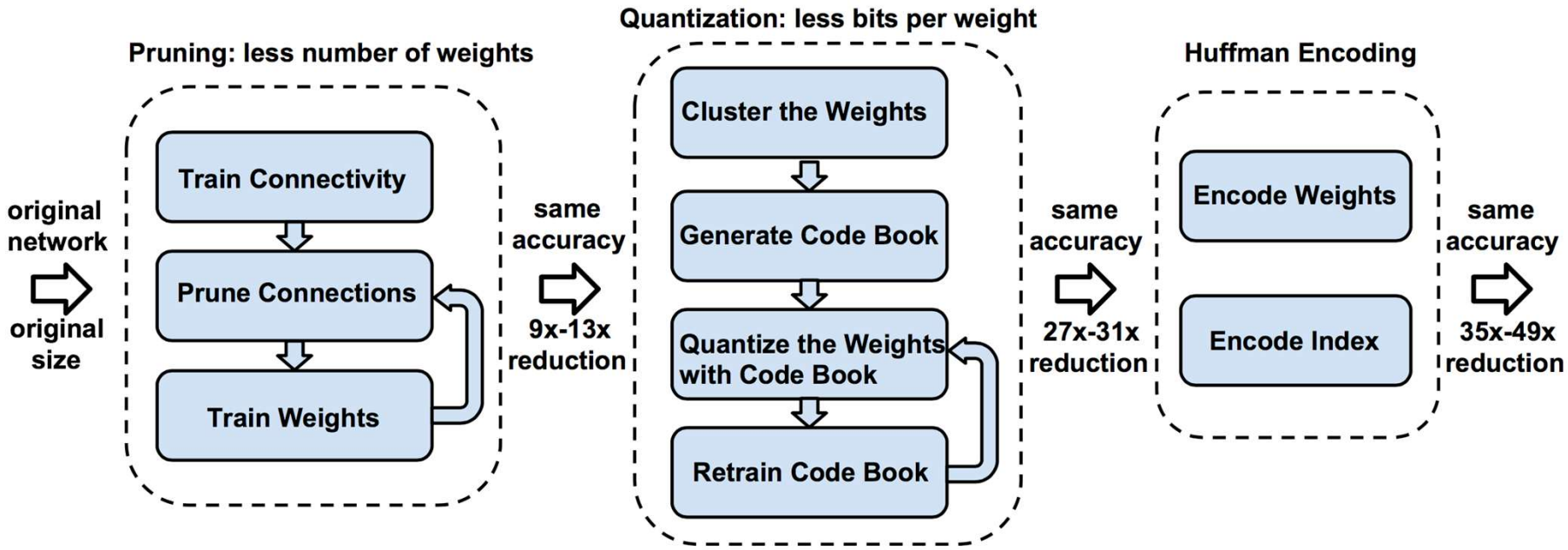


Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

Outline

1. Introduction
2. Matrix Factorization
3. Weight Pruning
4. Quantization
5. **Deep Compression: Pruning + Quantization + Encoding**
6. Design Small Architectures: SqueezeNet

Pruning + Quantization + Encoding: Deep Compression



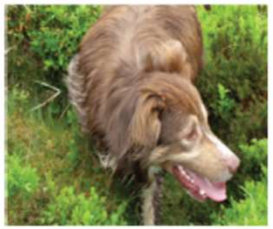
Courbariaux, Matthieu, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

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).

- **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

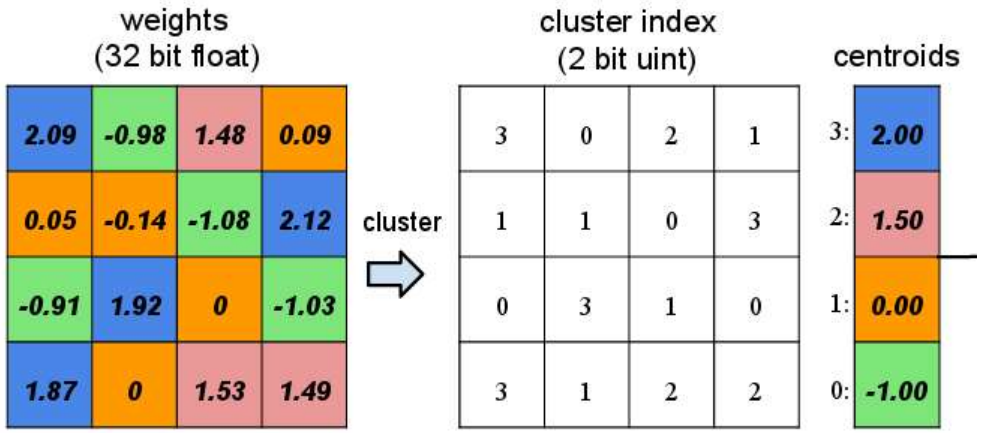
- **Original** : a man is riding a surfboard on a wave
- **Pruned 90%**: a man in a wetsuit is riding a wave on a beach

- **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

Quantization and Weight Sharing

- Quantize to fixed number of distinct values at no accuracy loss

- AlexNet conv layers quantized using 8 bits (256 16-bit weights) results in zero accuracy loss



Quantization and Weight Sharing

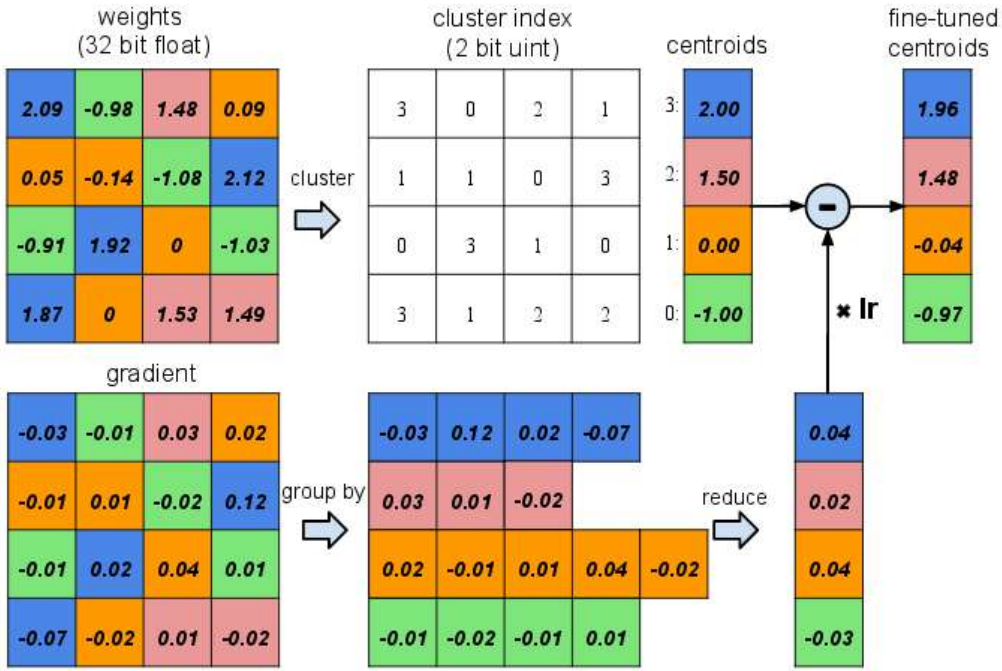


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

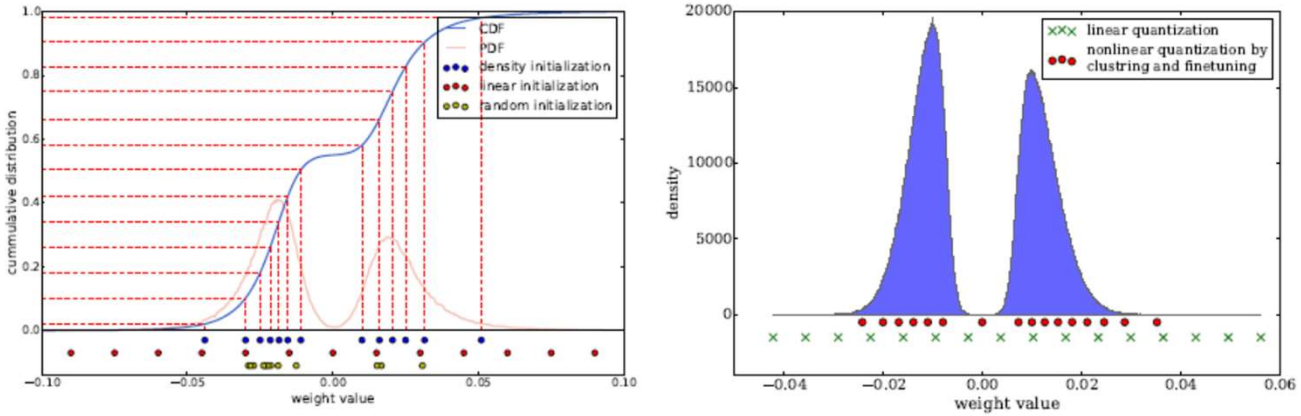
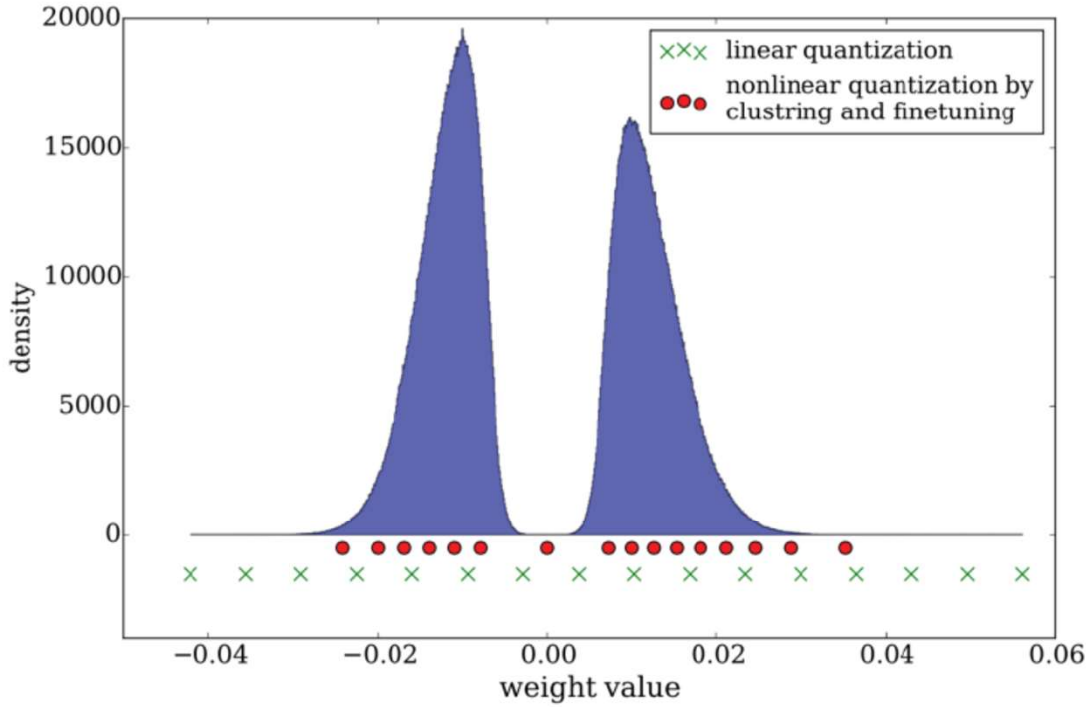


Figure 4: Left: Three different methods for centroids initialization. Right: Distribution of weights (blue) and distribution of codebook before (green cross) and after fine-tuning (red dot).

Finetune Centroids



Huffman Encoding

- General lossless compression scheme
- Encode more frequent values with less bits

AAAAAA
ABCDDD

Letter	Frequency	Encoding
A	7	0
D	3	10
B	1	110
C	1	111

19 bits vs 24 bits for 2-bit encoding

- Huffman, D. (1952) “A Method for the Construction of Minimum-Redundancy Codes”

Results

- Compression Ratios (same or better accuracy)
  - LeNet-300-100 – 40X
  - AlexNet – 35X
  - VGG16 – 49X
  - LeNet-5 – 39X

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**×)

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

1. Introduciton
2. Matrix Factorization
3. Weight Pruning
4. Quantization
5. Deep Compression: Pruning + Quantization + Encoding
6. **Design Small Architectures: 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 consisting 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

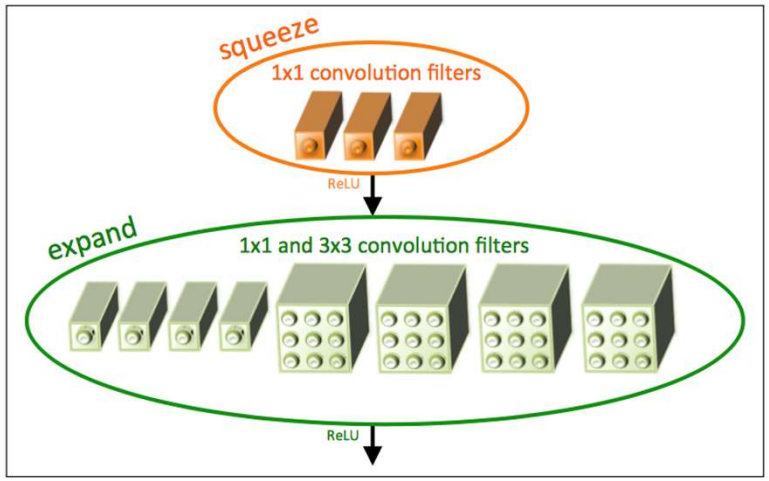


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

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

Microarchitecture – Fire Module

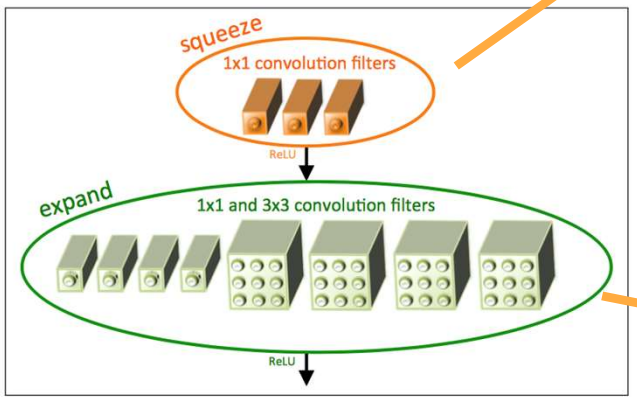


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

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)

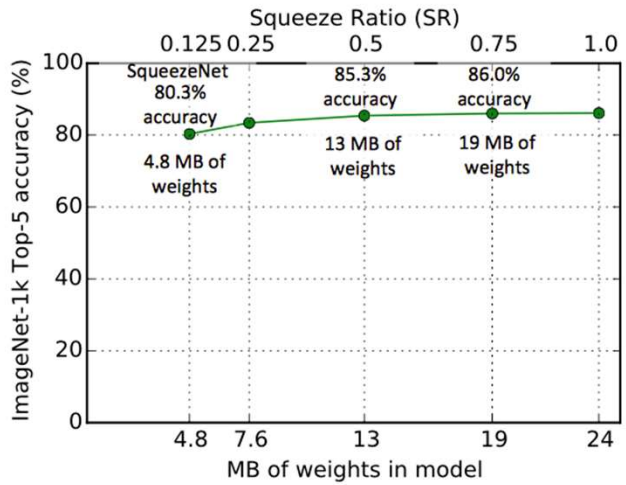
$$e_i = e_{i,1x1} + e_{i,3x3}$$

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

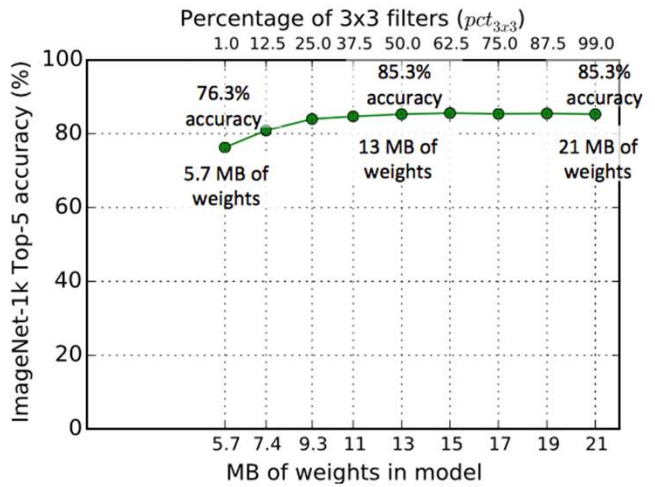
$$e_{i,3x3} = \boldsymbol{pct_{3x3}} * e_i$$

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.

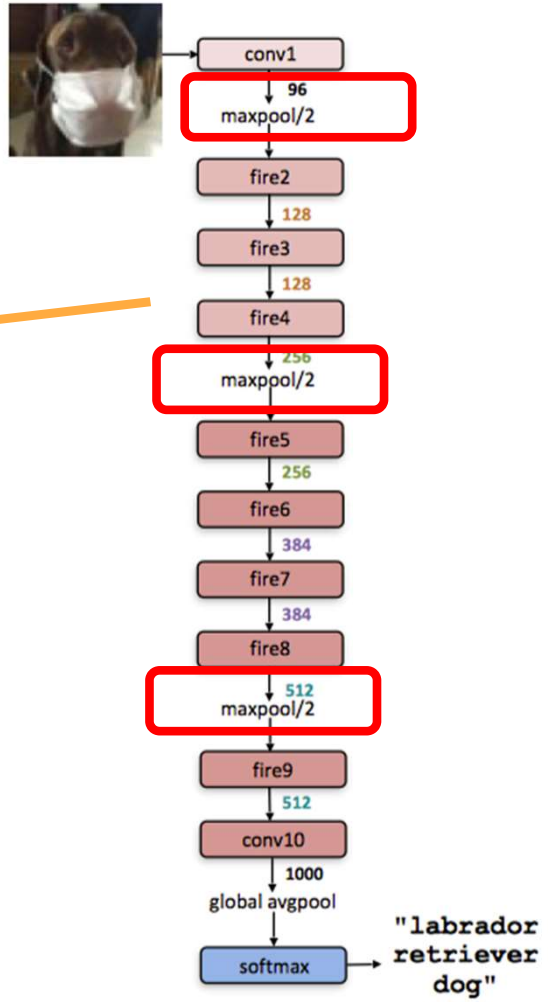
Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size."

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**

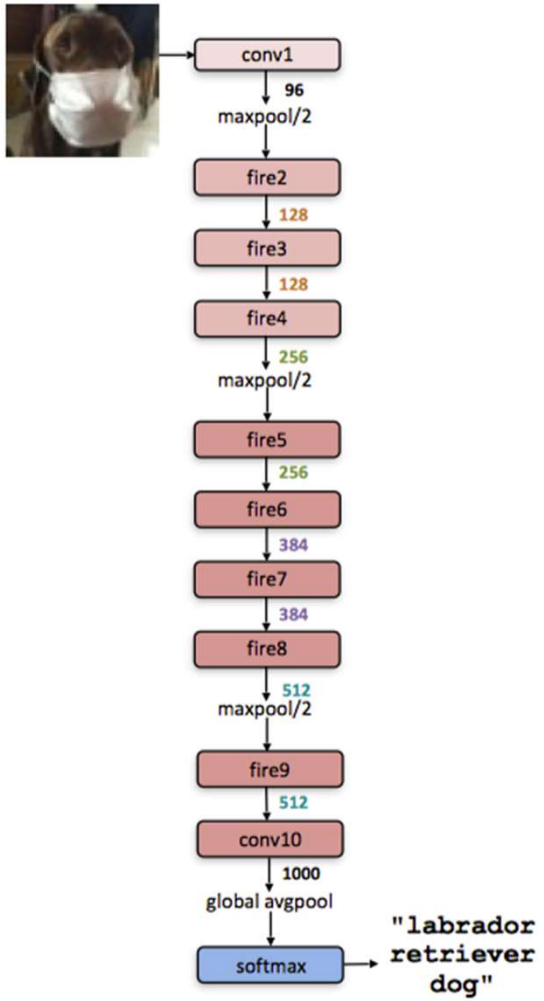


Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size."

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

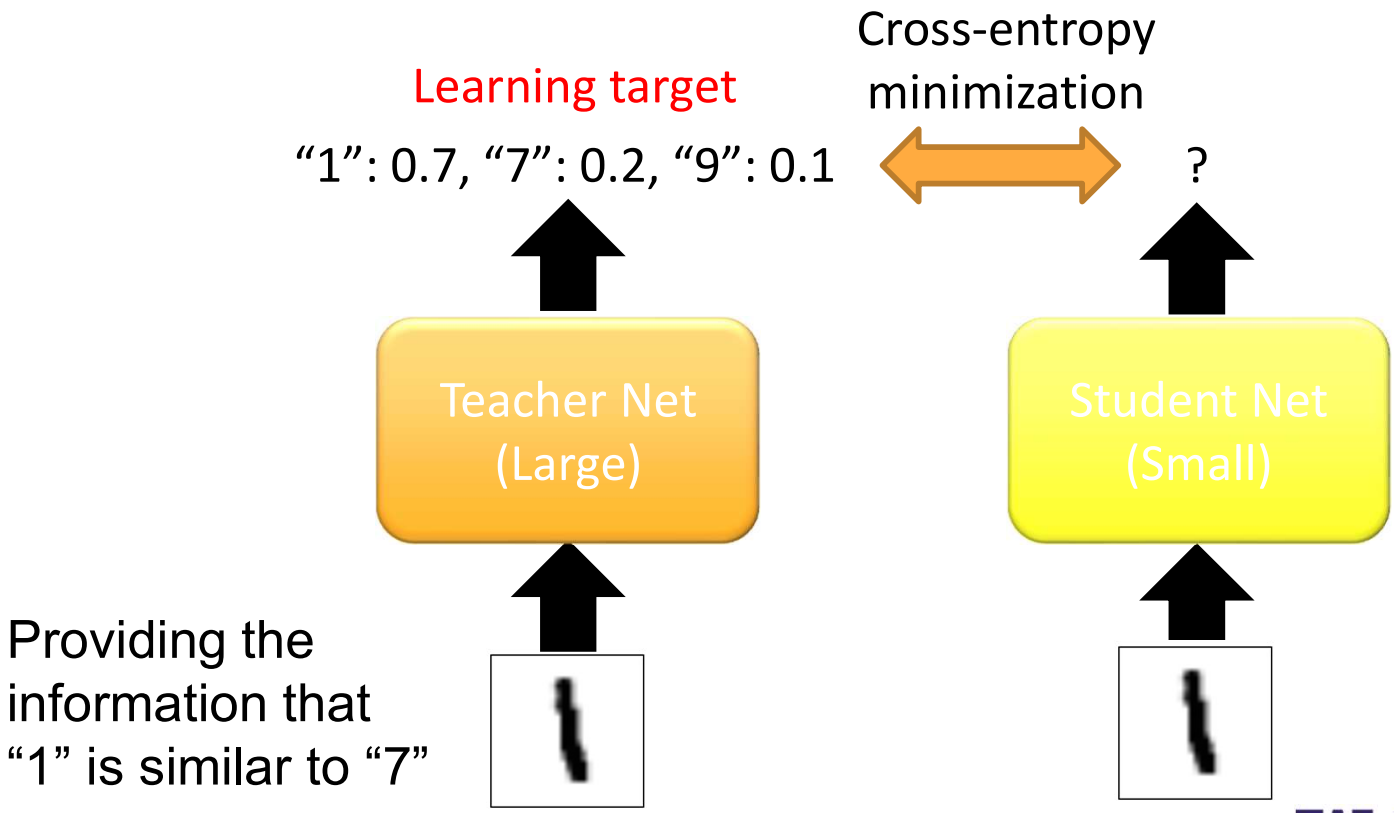
Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size."

Outline

1. Introduction
2. Matrix Factorization
3. Weight Pruning
4. Quantization
5. Deep Compression: Pruning + Quantization + Encoding
6. Design Small Architectures-- SqueezeNet
7. **Knowledge Distillation with Teacher-Student Network**

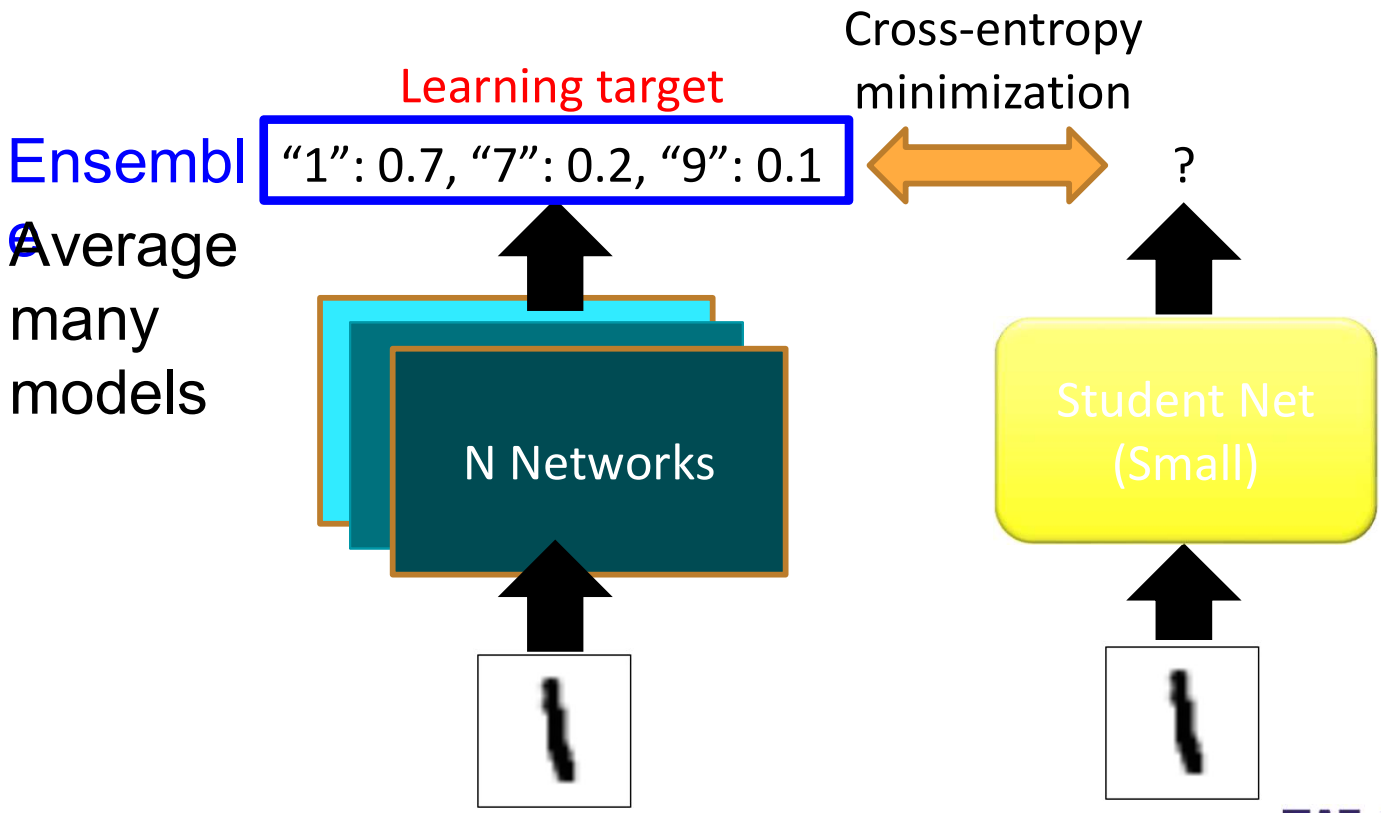
Knowledge Distillation

Knowledge Distillation
https://arxiv.org/pdf/1503.02531.pdf
Do Deep Nets Really Need to be Deep?
https://arxiv.org/pdf/1312.6184.pdf



Knowledge Distillation

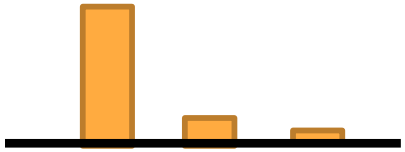
Knowledge Distillation
https://arxiv.org/pdf/1503.02531.pdf
Do Deep Nets Really Need to be Deep?
https://arxiv.org/pdf/1312.6184.pdf

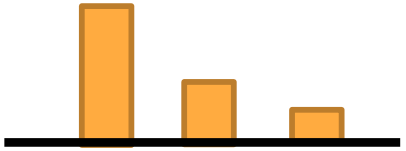


Knowledge Distillation

- Temperature for softmax

$$y_i' = \frac{exp(y_i)}{\sum_j exp(y_j)} \quad \xrightarrow{T = 100} \quad y_i' = \frac{exp(y_i/T)}{\sum_j exp(y_j/T)}$$





$y_1 = 100 \quad y_1' = 1$

$y_2 = 10 \quad y_2' \approx 0$

$y_3 = 1 \quad y_3' \approx 0$

$y_1/T = 1 \quad y_1' = 0.56$

$y_2/T = 0.1 \quad y_2' = 0.23$

$y_3/T = 0.01 \quad y_3' = 0.21$

Summary

- Techniques to Compress Pretrained Networks

  ➔ Architecture Simplification: On a single layer: SVD, Flattern Convolution, Depth-Wise Convolution

  - 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.

- Techniques to Design Smaller Networks

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