
Software Optimizations

Week 4
09/26/2022

Team



Matheus Farias

matheusfarias
@g.harvard.edu



Zergham Ahmed

zerghamahmed
@g.harvard.edu



He-Yen Hsieh

heyenhsieh
@g.harvard.edu



Xin Dong

xindong
@g.harvard.edu

Content

1. Motivation
2. Background
 - Accelerate Neural Network
 - Brief introduction of pruning and quantization
3. Quantization Functions
 - Uniform Quantization
 - Logarithmic Quantization
 - Term Quantization
 - Piecewise Linear Quantization
4. Post-Training Quantization
5. Quantization Aware Training
 - a. How to take gradients?
 - b. Mixed precision
 - c. Binary/Ternary Neural Networks
6. Pruning
 - a. The Lottery Ticket Hypothesis
 - b. Leveraging Sparsity

Motivation

Motivation

- AlexNet achieved the best accuracy (the ImageNet challenge in 2012)

Timeline	Model	#params	#ops
2012	AlexNet	61M params	724M MACs
2014	VGGNet-16	138M params	15.5G MACs
2015	ResNet-50	25.5M params	3.9G MACs
⋮			⋮
2022	CSWin-Transoformer	173M params	48.4G MACs

For example: Self-driving

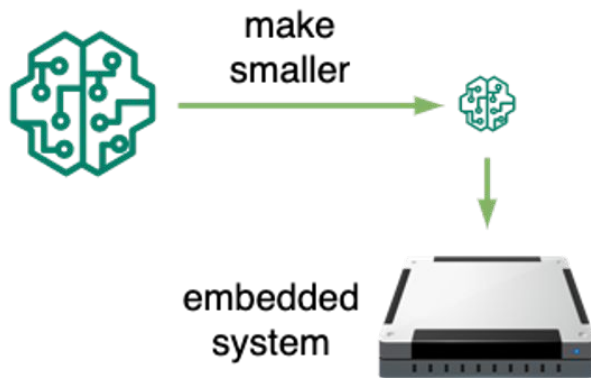
- Imagine:
- Design an algorithm for the self-driving application
- The goal is to detect whether a person on the road



Figure source: [1] Geest et al., "Online Action Detection", ECCV 2016.

For example: Self-driving

- We want:
 - Real-time: fast inference
 - Low operations: save energy (energy is limited to a battery)
- Make network smaller by reducing parameters and computations
 - Embedded system

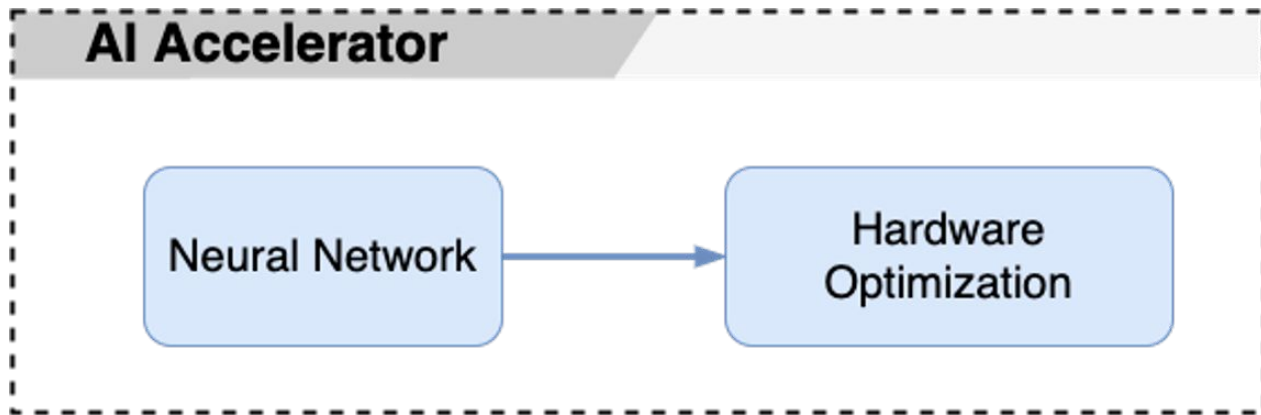


AI Accelerators Strategies

Hardware and Software Views

Accelerate Neural Network

Early work [1]:
focus on hardware
optimization in DNNs
accelerators



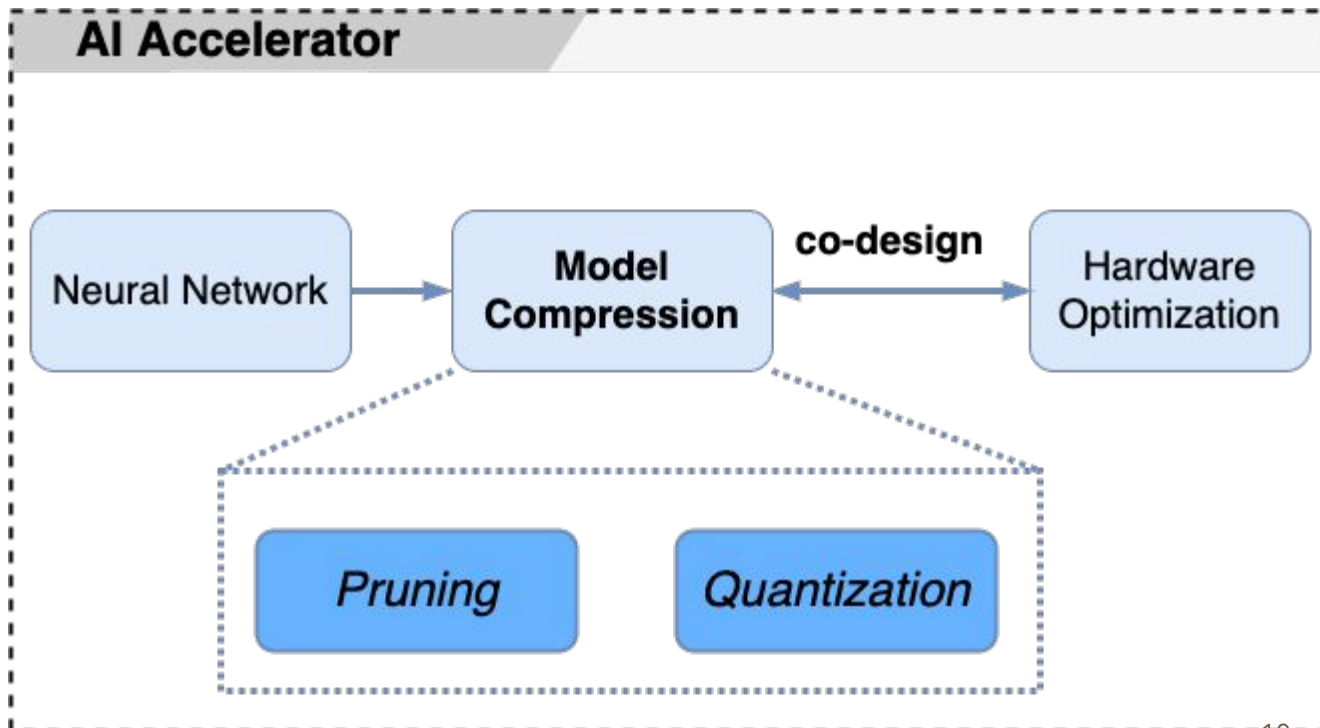
[1] Chen et al., "Dadiannao: A machine-learning supercomputer." MICRO, 2014.

Accelerate Neural Network

Recent works [1, 2]:
compression and
software optimization
of DNNs

[1] Han et al., "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." ICLR, 2016.

[2] Molchanov et al., "Pruning convolutional neural networks for resource efficient inference." ICLR, 2017.



Basic Concept of Pruning and Quantization

Brief introduction of Pruning

- remove unnecessary or less important connections
 - network becomes a sparse network (required storage and computations all reduce)
- similar to synaptic changes in humans
 - the number of synapses in early childhood is at the peak
 - Afterward synaptic connections gradually decrease

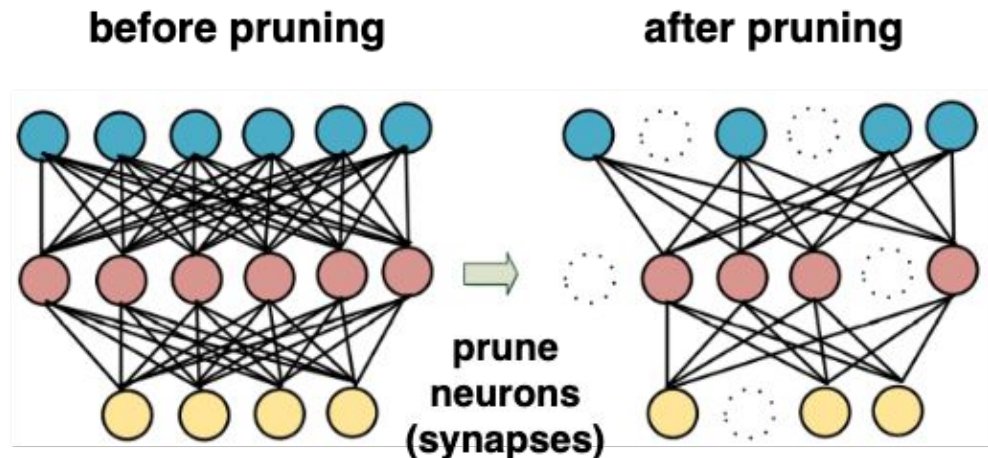
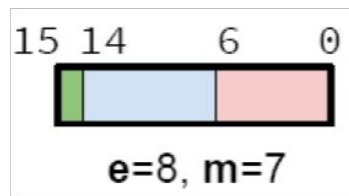
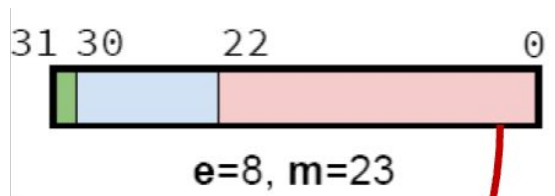


Figure source: [5] Liu et al., "Bringing AI to edge: From deep learning's perspective." Neurocomputing, 2022.¹²

Brief introduction of Quantization

- network components are approximated with low bit-width
- For example, 32-bit floating-point weights are mapped to a 16-bit fixed-point or 8-bit integer.

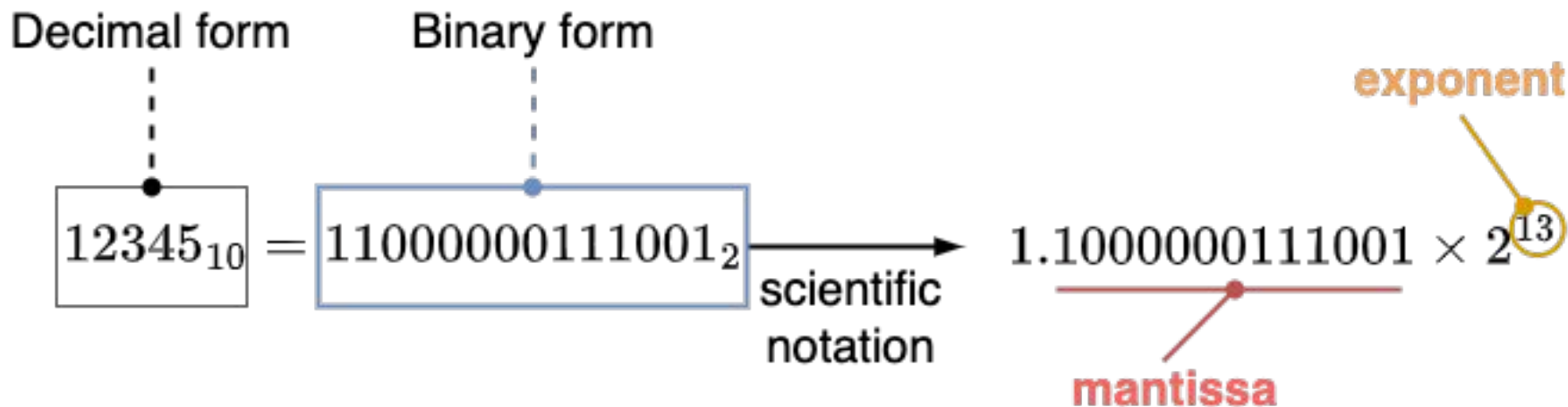


Reduce precisions

e: exponent, m: mantissa

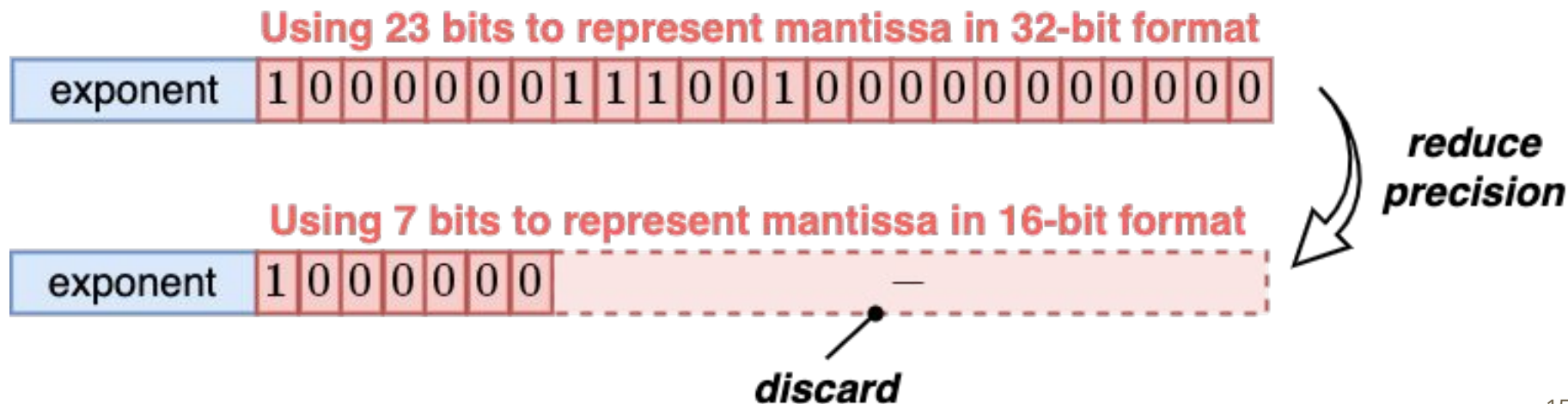
Mantissa and exponent

- network components are approximated with low bit-width
- For example, 32-bit floating-point weights are mapped to a 16-bit fixed-point or 8-bit integer.



Mantissa and exponent

- network components are approximated with low bit-width
- For example, 32-bit floating-point weights are mapped to a 16-bit fixed-point or 8-bit integer.



Benefits of Quantization

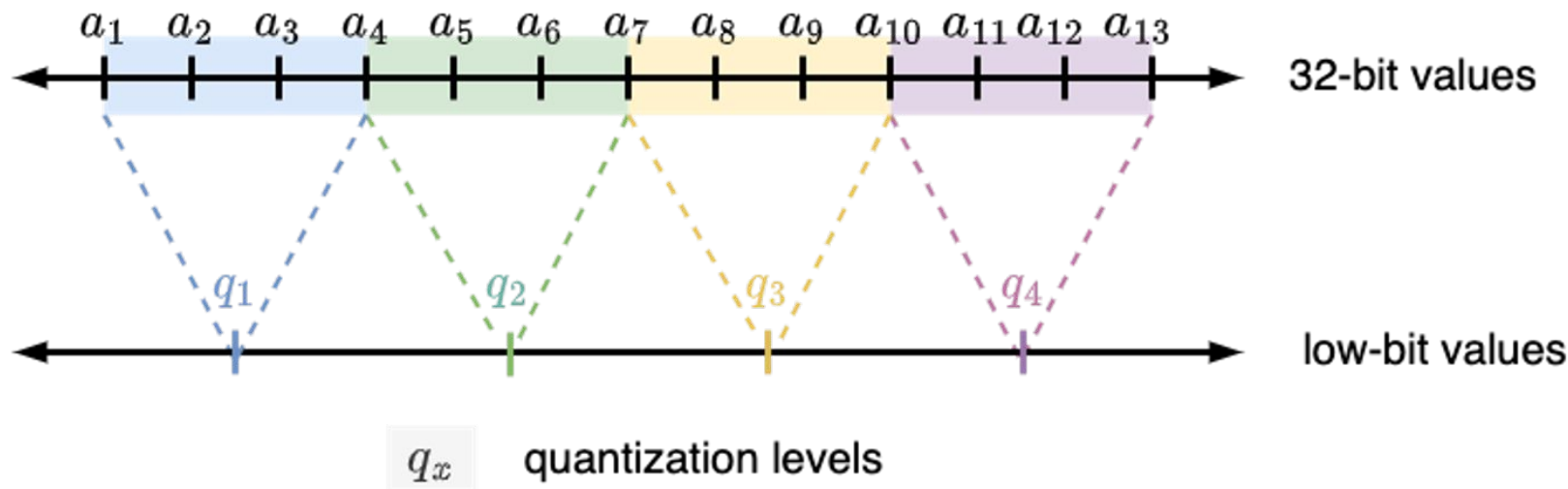
- **High compression by quantization.** Accuracy reduction is less.
- **Flexibility.** Since quantization is not dependent on the network architecture, therefore, a quantization algorithm can be used for all DNNs (CNN-based, RNN-based).
- **Cost reduction of hardware accelerator designing.** For example, in 1-bit quantization, a 32-bit floating-point multiplier can be replaced by an XNOR operator.

Quantization on Neural Networks

- Each component of the neural networks can be compressed by quantization.
 - Three types of components: weights, activations, and gradients
- Each parameter in the neural networks can be quantized
 - Weights quantization is the most common.
- Bias and other parameters (for example, batch normalization parameters) in most works are kept in full-precision.
 - Work in [1] have quantized them.
- Decrease the complexity of the neural network
 - We can talk about methods for leveraging sparsity and even see quantization as a regularization method

Concept of Quantization

- **A mapping function.** A continuous space \rightarrow a discrete space
 - full-precision values \rightarrow new values with lower bit-width.
 - These new values: *quantization levels*



Model Compression Differences across Domains

- Intuitively, some domains such as object detection will receive more harm to accuracy when using model compression. Generative models are harmed less since they are overparameterized.

Quantization approaches

Uniform Quantization and Logarithmic Quantization

Quantization Functions

- **Uniform Quantization.**
 - Quantized values have a bitwidth equal to a fixed number
 - *Partition the range of FP32 numbers into **uniform intervals***
 - *The quantization function maps FP32 numbers in these intervals to integers*
 - *The quantization error is bounded by the interval size*

Quantization Functions

- **Uniform Quantization.**
 - Quantizing x within $[-L, L]$ using 8 bits

L : max value

sf : scale factor

b : bitwidth

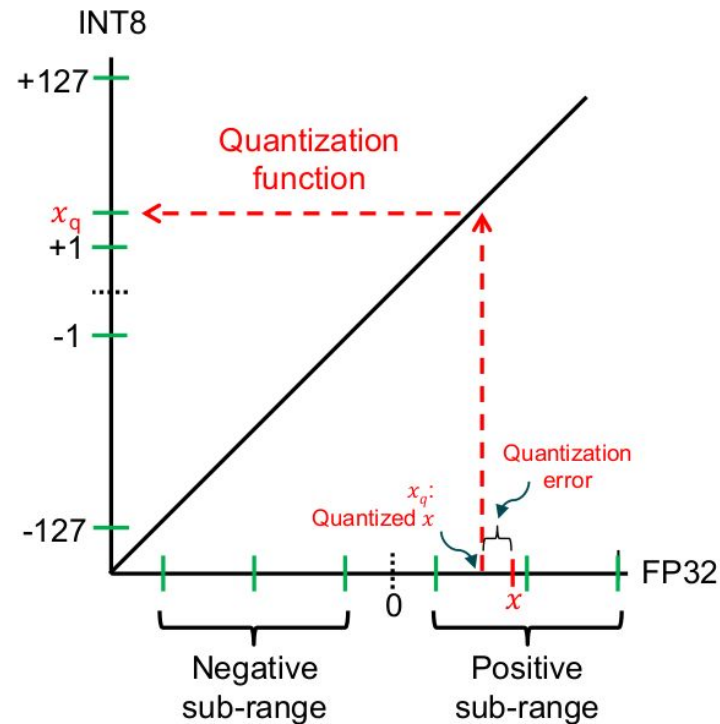
x : value (float)

$\lfloor \cdot \rfloor$: floor function

Step 1: $sf = \frac{L}{2^{b-1}-1}$

Step 2: $x_c = \text{clamp}(x, -L, L)$

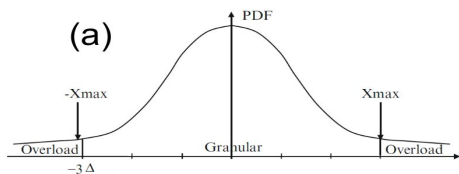
Step 3: $x_q = \left\lfloor \frac{x_c}{sf} + 0.5 \right\rfloor \times sf$



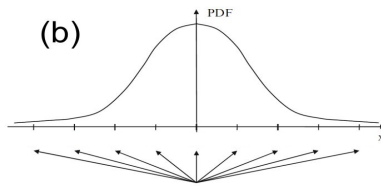
Quantization Functions

- **Nonuniform Quantization.**

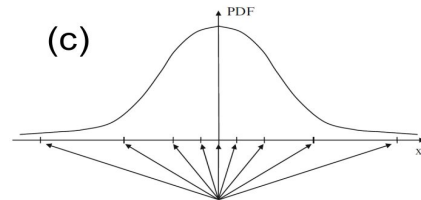
- In DNNs, **weights** and **activations** follow a **bell-shaped distribution**
- When minimizing mean quantization errors, we should weight quantization errors by their probabilities. **Nonuniform!**



Bell-shaped distribution



Quantizing into uniform intervals
(**not ideal**)



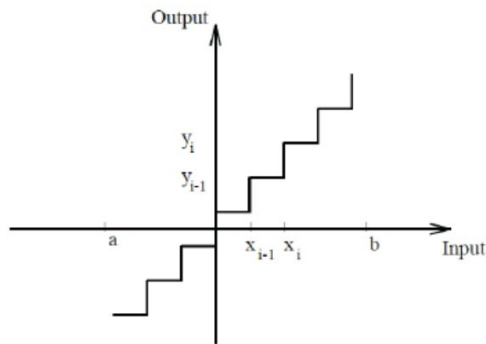
Quantizing into nonuniform intervals (**ideal but hard to define**)

Quantization Functions

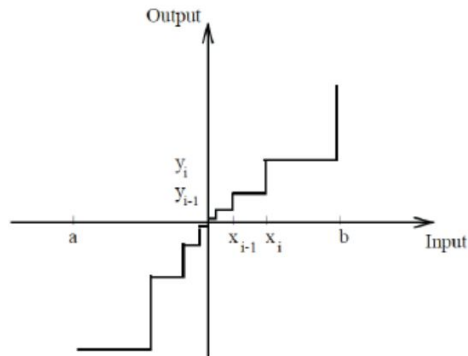
- **Logarithm Quantization.**

- Edge case of nonuniform quantization
- We just keep the MSB after performing uniform quantization

(a) Uniform quantization function



(b) Logarithmic quantization function



Quantization Functions

- **Logarithm Quantization.**
 - Very harsh (cannot be done post-training)
 - Multiplications now are super efficient, it is just a shift

$$2^3 \times 101_2 = 101000_2$$

Quantization Functions

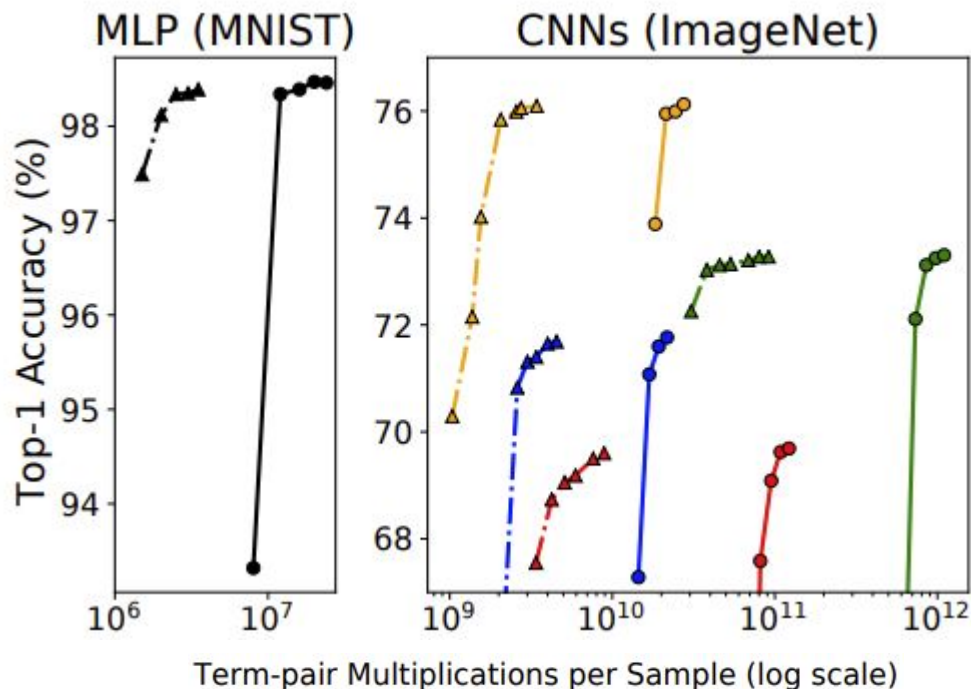
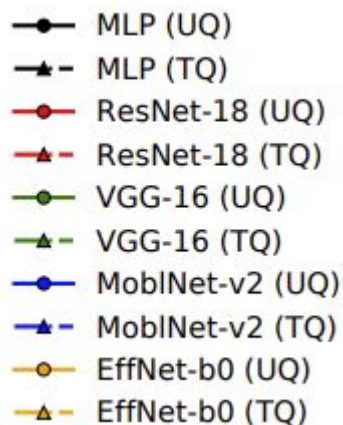
- **Term Quantization.**

- Intermediate procedure between uniform and logarithmic
- We define a group of values and how many terms we want to keep

(a) 5-bit uniform quantization						(b) Logarithmic quantization						(c) TQ($\alpha = 8, g = 4$)					
	2^4	2^3	2^2	2^1	2^0		2^4	2^3	2^2	2^1	2^0		2^4	2^3	2^2	2^1	2^0
21	1	0	1	0	1	16	1	0	1	0	1	21	1	0	1	0	1
6	0	0	1	1	0	4	0	0	1	1	0	6	0	0	1	1	0
17	1	0	0	0	1	16	1	0	0	0	1	16	1	0	0	0	1
11	0	1	0	1	1	8	0	1	0	1	1	10	0	1	0	1	1

Quantization Functions

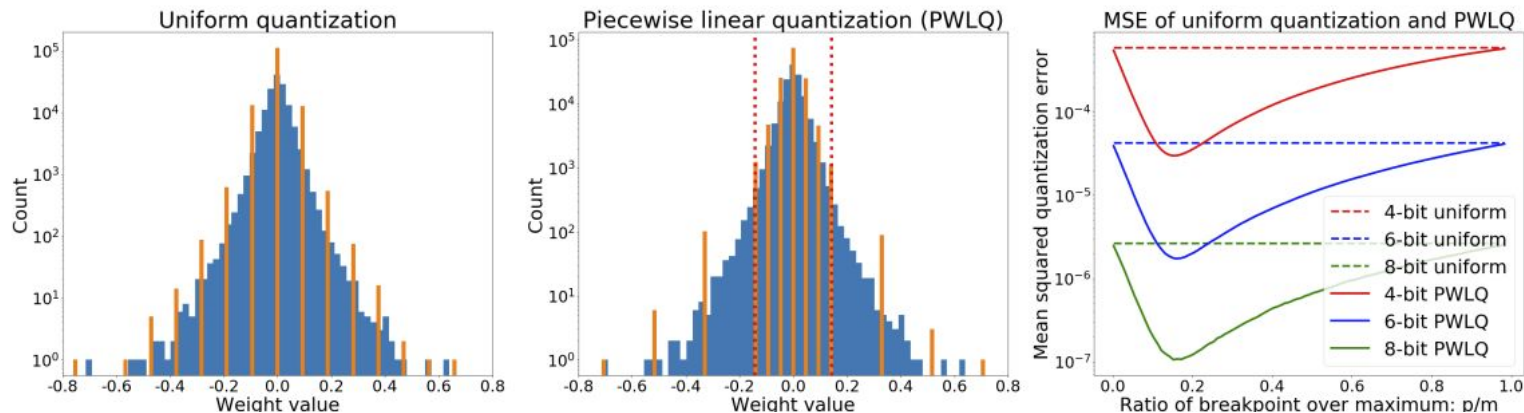
- Term Quantization.



Quantization Functions

- **Piecewise Linear Quantization.**

- Instead of trying to find the ideal nonuniform quantization, we group the weights into UQ with different number of bits
- Regions close to the mean are more frequent (larger number of bits)
- We do not care much about values in the tail (smaller number of bits)



Quantization Functions

- **Piecewise Linear Quantization.**

Table 1. Top-1 accuracy (%) and requirement of hardware accumulators for PWLQ with multiple breakpoints on weights

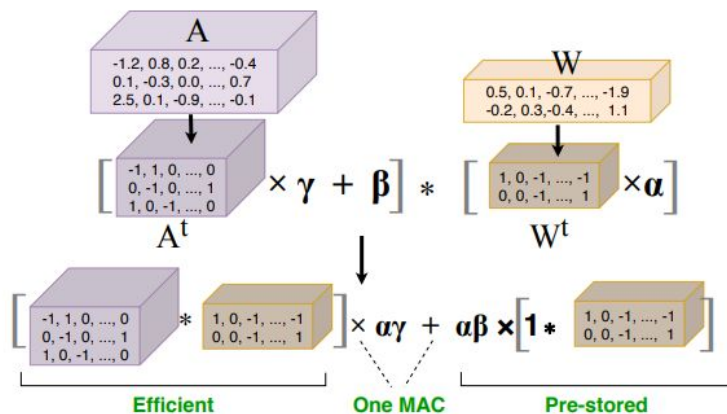
Number of Breakpoints	Hardware Accumulators	Inception-v3 (77.49)			ResNet-50 (76.13)			MobileNet-v2 (71.88)		
		5-bit	4-bit	3-bit	5-bit	4-bit	3-bit	5-bit	4-bit	3-bit
One	Three	77.28	75.72	61.76	75.62	74.28	67.30	69.05	54.34	16.77
Two	Five	77.31	76.73	71.40	75.94	75.24	73.27	70.01	65.74	36.44
Three	Seven	77.46	77.00	74.07	76.06	75.77	73.84	70.43	67.71	55.17


Quantization – Training

Post-training and quantization-aware

Post-training Quantization

- Without a full training dataset and many computation resources to perform end-to-end backpropagation. So the computation-heavy operations (e.g., conv) can be implemented efficiently.



	Network	Numerics	Operations	Memory Speed-Up	Inference Speed-Up	Accuracy
	Full-Precision DNN	W: 32 bit A: 32 bit	+, -, x	1x	1x	High
	BNN	W: 1 bit A: 1 bit	XNOR, Bitcount	~32x	(CPU) ~58x	Low
	(Ours) BENN (K ensemble)	W: 1 bit A: 1 bit	XNOR, Bitcount	~(32/K)x	(CPU) ~58x	High

Post-training Quantization

- Many trained FP32 models can be quantized to INT8 without much loss in performance, some models exhibit a significant drop in performance after quantization

Model	Full-Precision	INT8
ResNet-50	0.752	0.75
ResNet-152	0.768	0.766
MobileNetV2	0.719	0.001 ? See the next page
Nasnet-Mobile	0.74	0.722

* From Krishnamoorthi, Raghuraman. "Quantizing deep convolutional networks for efficient inference: A whitepaper." arXiv preprint arXiv:1806.08342 (2018).

Post-training Quantization

- Many trained FP32 models can be quantized to INT8 without much loss in performance, some models exhibit a significant drop in performance after quantization
- If we do per-layer post-training quantization for MobileNetV2, its accuracy will be destroyed after quantization. The root cause is that different channels in a layer of MobileNetV2 have various ranges.
- A quick solution proposed in [1] is to apply per-channel quantization for MobileNetV2.

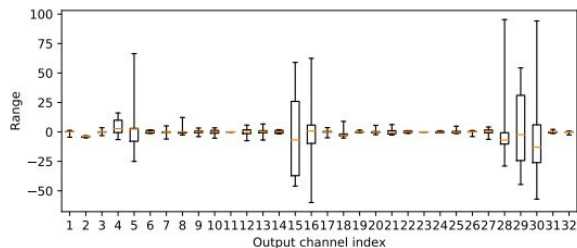


Figure 2. Per (output) channel weight ranges of the first depthwise-separable layer in MobileNetV2. In the boxplot the min and max value, the 2nd and 3rd quartile and the median are plotted for each channel. This layer exhibits strong differences between channel weight ranges.

Model	FP32	INT8
Original model	71.72%	0.12%
Replace ReLU6	71.70%	0.11%
+ equalization	71.70%	69.91%
+ absorbing bias	71.57%	70.92%
Per channel quantization	71.72%	70.65%

[1] Nagel, Markus, et al. "Data-free quantization through weight equalization and bias correction." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

Quantization-Aware Training

- Gradient mismatch issue of quantization-aware training

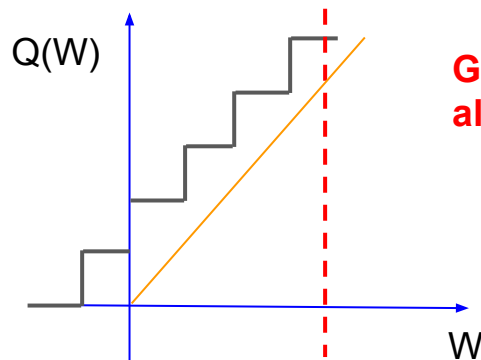
$$h = Q(W_f^T) \times x$$

$$y = P(h)$$

To get the gradient of W

$$\frac{\partial L}{\partial W_f^T} = \frac{\partial L}{\partial y} \underbrace{\frac{\partial y}{\partial h}}_1 \underbrace{\frac{\partial h}{\partial Q(W_f^T)}}_1 \frac{\partial Q(W_f^T)}{\partial W_f^T}$$

Straight-forward Estimator (STE) [1]



Gradient is zero almost everywhere.



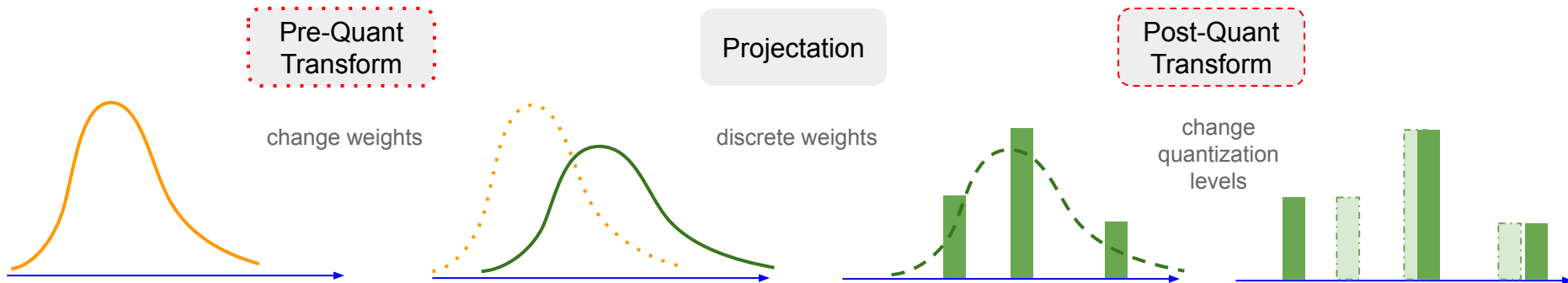
[1] Hinton, Geoffrey, et al. Neural networks for machine learning. Coursera, video lectures

[2] Cai, Zhaowei, et al. "Deep learning with low precision by half-wave gaussian quantization." CVPR, 2017.

[3] Yin, Penghang, et al. "Understanding straight-through estimator in training activation quantized neural nets." ICLR, 2019.

Quantization-Aware Training

- We can organise many papers into a single, unified framework.



Quantization-Aware Training

- DoReFa-Net [1]: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients

Weight/Activation Quantization

$$\textbf{Forward: } r_o = f_{\omega}^k(r_i) = 2 \text{quantize}_k \left(\frac{\tanh(r_i)}{2 \max(|\tanh(r_i)|)} + \frac{1}{2} \right) - 1. \quad (9)$$

$$\textbf{Backward: } \frac{\partial c}{\partial r_i} = \frac{\partial r_o}{\partial r_i} \frac{\partial c}{\partial r_o} \quad (10)$$

Gradient quantization

$$f_{\gamma}^k(dr) = 2 \max_0(|dr|) \left[\text{quantize}_k \left[\frac{dr}{2 \max_0(|dr|)} + \frac{1}{2} + N(k) \right] - \frac{1}{2} \right]. \quad (12)$$

The quantization of gradient is done on the backward pass only. Hence we apply the following STE on the output of each convolution layer:

$$\textbf{Forward: } r_o = r_i \quad (13)$$

$$\textbf{Backward: } \frac{\partial c}{\partial r_i} = f_{\gamma}^k \left(\frac{\partial c}{\partial r_o} \right). \quad (14)$$

[1] Zhou, Shuchang, et al. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients." arXiv preprint arXiv:1606.06160 (2016).

Quantization-Aware Training

- DoReFa-Net [1]: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients

```
def quantize_k(r_i, k):
    scale = (2**k - 1)
    r_o = torch.round( scale * r_i ) / scale
    return r_o

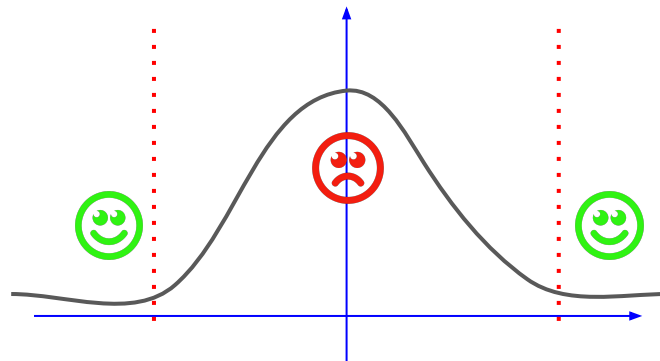
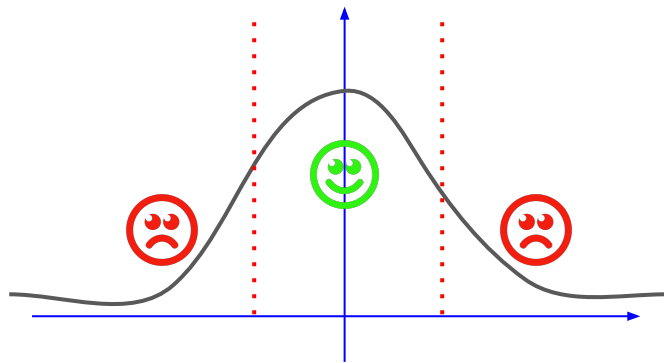
class DoReFaQuant(Function):
    @staticmethod
    def forward(ctx, r_i, k):
        tanh = torch.tanh(r_i).float()
        # scale = 2**k - 1.
        # quantize_k = torch.round( scale * (tanh / 2*torch.abs(tanh).max() + 0.5 ) ) / scale
        r_o = 2*quantize_k( tanh / (2*torch.max(torch.abs(tanh)).detach() + 0.5 , k) - 1
        # r_o = 2 * quantize_k - 1.
        return r_o

    @staticmethod
    def backward(ctx, dLdr_o):
        # due to STE, dr_o / d_r_i = 1 according to formula (5)
        return dLdr_o, None
```

[1] Zhou, Shuchang, et al. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients." arXiv preprint arXiv:1606.06160 (2016).

Quantization-Aware Training

- If we use uniform quantization, there is a trade-off between range and resolution.



It is sub-optimal to use the min-max of weights as the range of quantization.

Quantization-Aware Training

- Learnable Quantization Range

$$y = PACT(x) = 0.5(|x| - |x - \alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$

$$y_q = round(y \cdot \frac{2^k - 1}{\alpha}) \cdot \frac{\alpha}{2^k - 1}$$

**We can now compute
the gradient of the
range parameter**

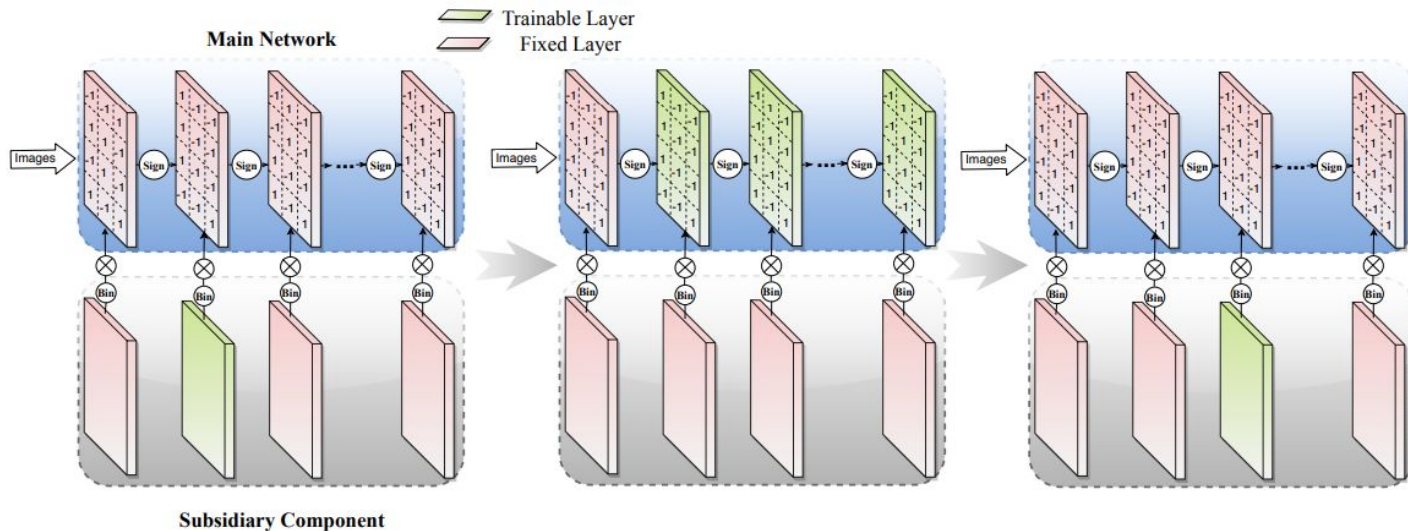
$$\frac{\partial y_q}{\partial \alpha}$$

$$= \frac{\partial y_q}{\partial y} \frac{\partial y}{\partial \alpha} = \begin{cases} 0, & x \in (-\infty, \alpha) \\ 1, & x \in [\alpha, +\infty) \end{cases}$$

inside weights (no contribution)
outliers

- [1] Choi, Jungwook, et al. "Pact: Parameterized clipping activation for quantized neural networks." arXiv, 2018.
- [2] Li, Yuhang*, Xin Dong*, and Wei Wang. "Additive Powers-of-Two Quantization: A Non-uniform Discretization for Neural Networks." arXiv, 2019.

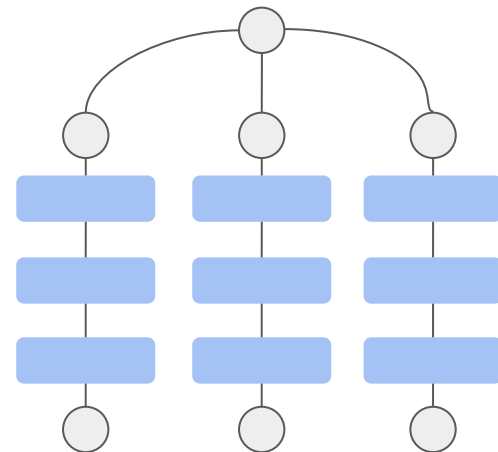
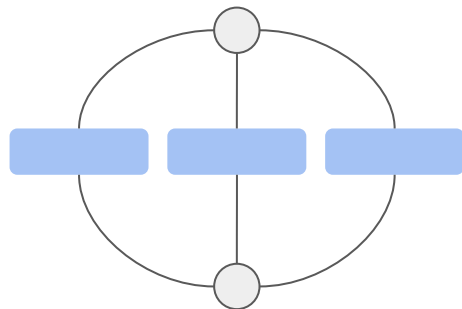
Can We Prune a Quantization Network?



[1] Yinghao Xu*, Xin Dong*, Yudian Li, Hao Su. A Main/Subsidiary Network Framework for Simplifying Binary Neural Network. CVPR, 2019.

Bridge Accuracy Gap

- Use wider neural networks [1]
- Mixed precision weights and activation [2]
- Make several 'copies' for layers [3]
- Ensemble quantized models [4]



[1] Mishra, Asit, et al. "WRPN: wide reduced-precision networks." ICLR, 2018.

[2] Wang, Kuan, et al. "HAQ: Hardware-Aware Automated Quantization with Mixed Precision." CVPR, 2019.

[3] Lin, Xiaofan, et al. "Towards accurate binary convolutional neural network." NeurIPS, 2017.

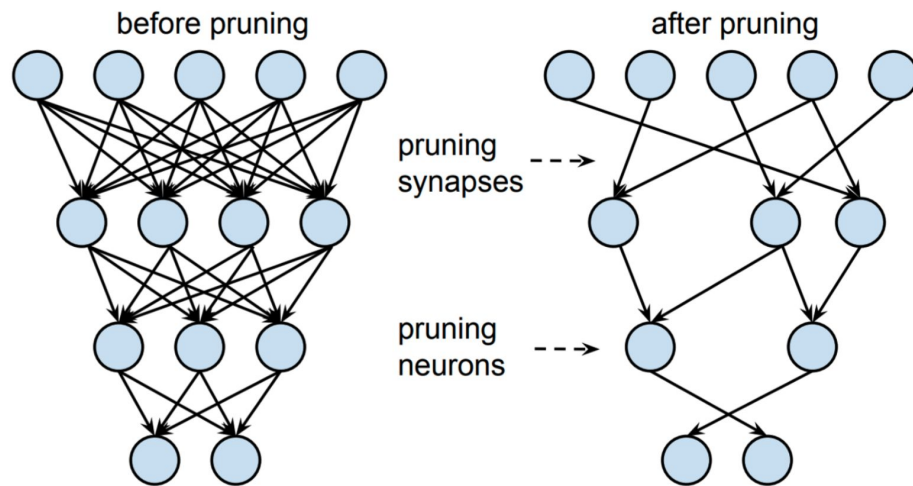
[4] Shilin Zhu, Xin Dong, Hao Su. "Binary Ensemble Neural Network: More Bits per Network or More Networks per Bit?" CVPR, 2019.

Pruning

Structured and Unstructured

Pruning

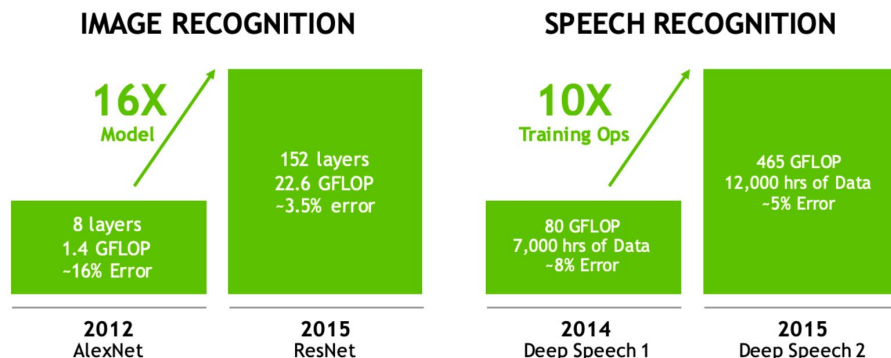
- Removing weights and connections to compress the Neural network
- Reduce the amount of matrix multiplications shortening inference time
- Take care not to prune too aggressively



(Figure 3): Han, Song, et al.
"Learning both weights and connections for efficient neural network." NeurIPS, 2015.

Why Prune?

- Many parameters in neural networks, some are redundant and do not affect output that much
- On ImageNet dataset, pruning was used to reduce VGG-16 parameters by a factor of 13x from 138 million to 10.3 million without accuracy loss[1]
- Smaller networks can fit on mobile devices and microcontrollers

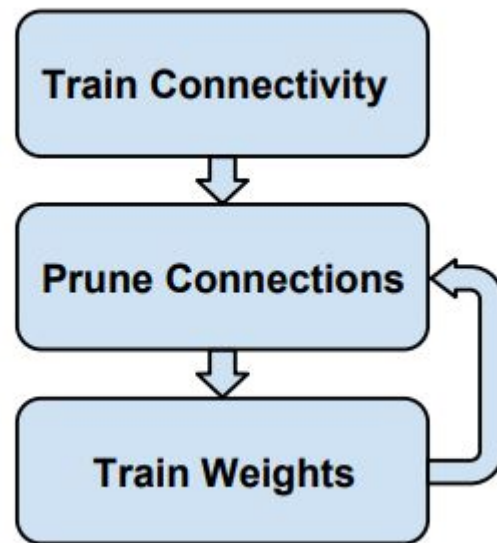


[1] Han, Song, et al. "Learning both weights and connections for efficient neural network." NeurIPS, 2015.

Standard Unstructured Pruning Process

1. Train dense network
2. Remove connections with weights below a threshold
3. Re-train remaining weights
4. Repeat 2-3 to find minimum number of connections

*learn the connections not only the weights



(Figure 2): Han, Song, et al. "Learning both weights and connections for efficient neural network." NeurIPS, 2015.

Structured vs Unstructured Pruning

- Unstructured = remove weights and connections
- Structured = remove whole filters and kernels

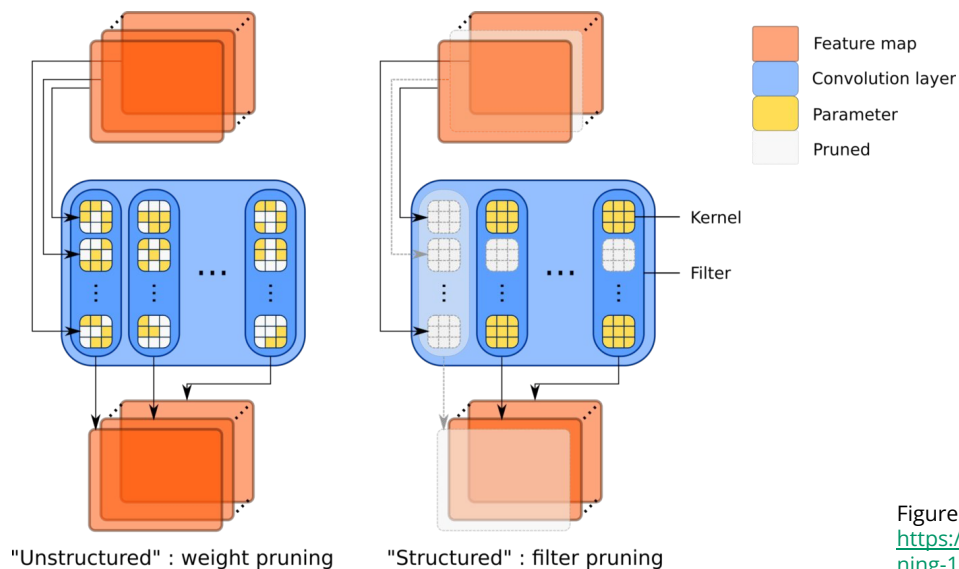


Figure source:

<https://towardsdatascience.com/neural-network-pruning-101-af816aaea61>

Unstructured Pruning

- Set weights with low magnitudes to 0, skip 0 computations
- Results in unstructured sparsity
- High accuracy but has inefficient implementation
 - Overhead from keeping track of indices for non-zeros

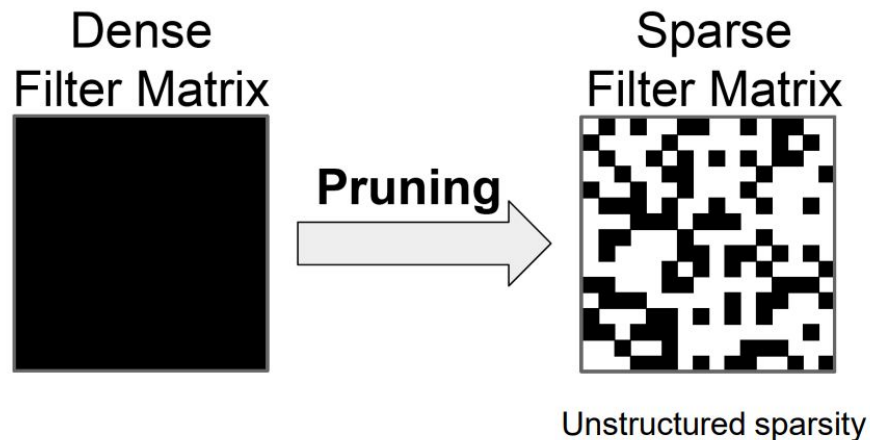
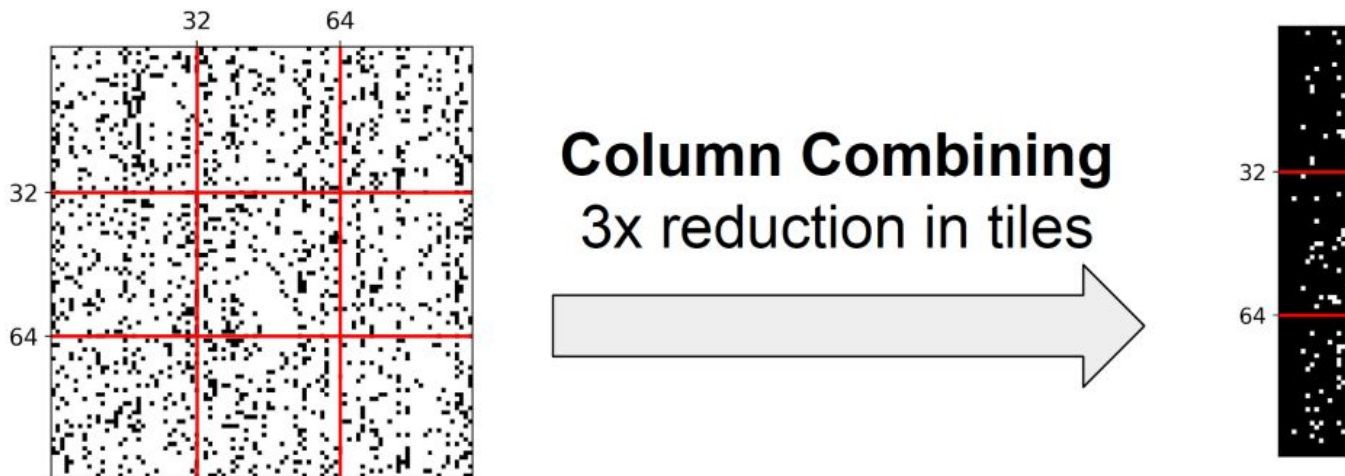


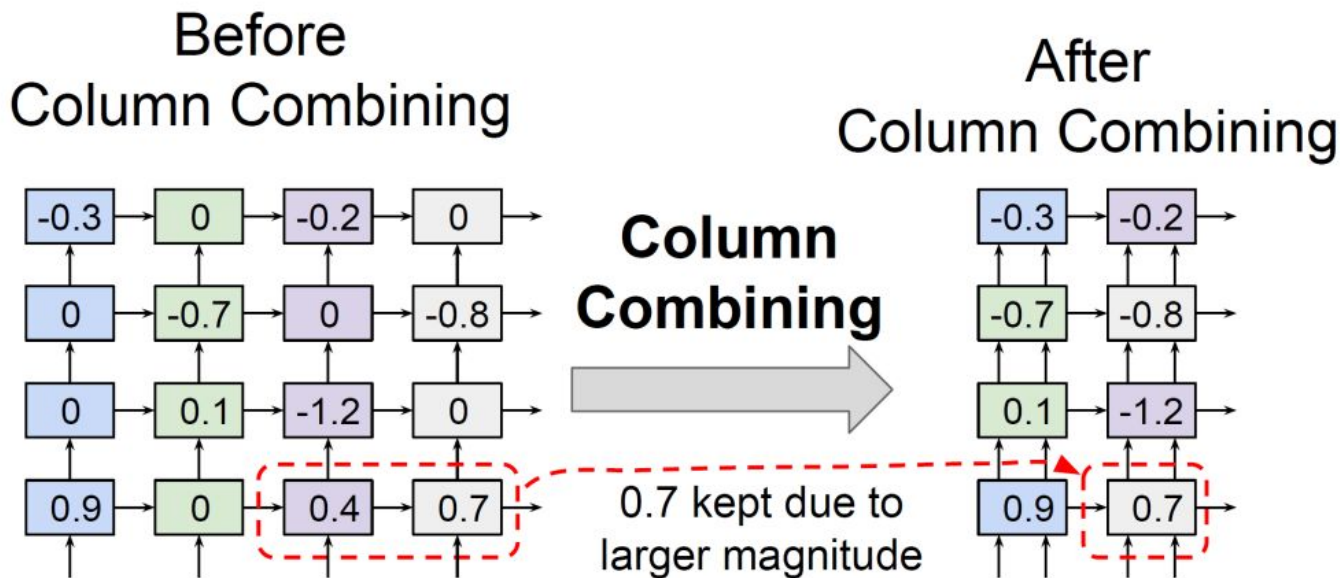
Figure source:
Professor HT
Kung's Lecture
Notes

Column Combining

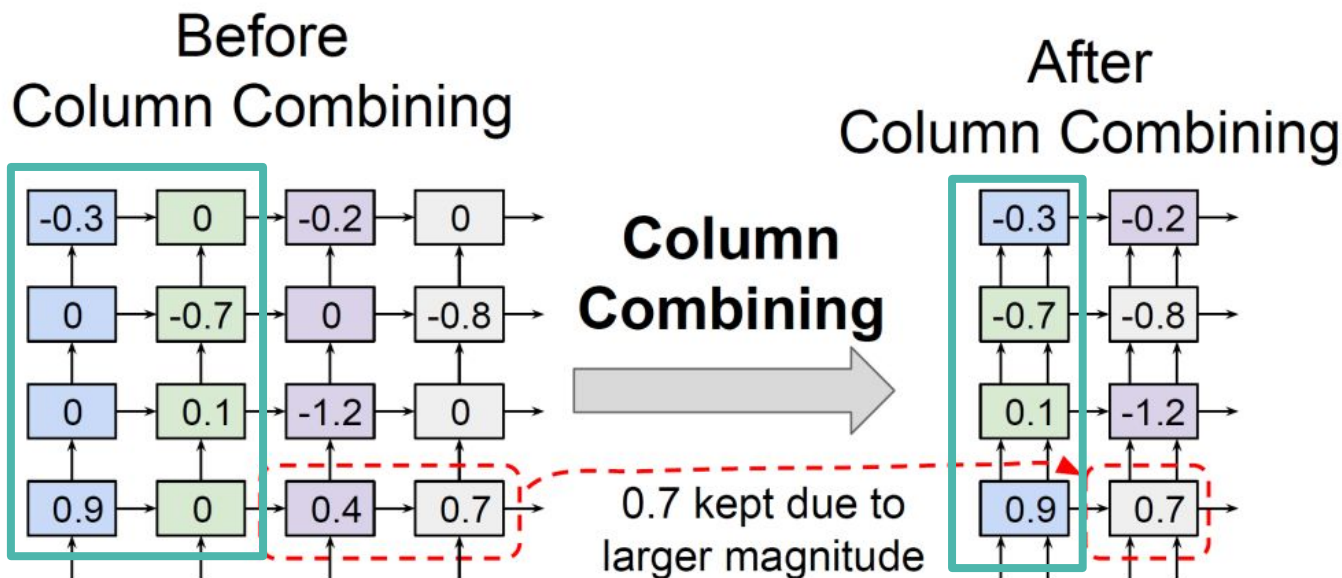
- Reduce memory usage by changing sparse matrix representation



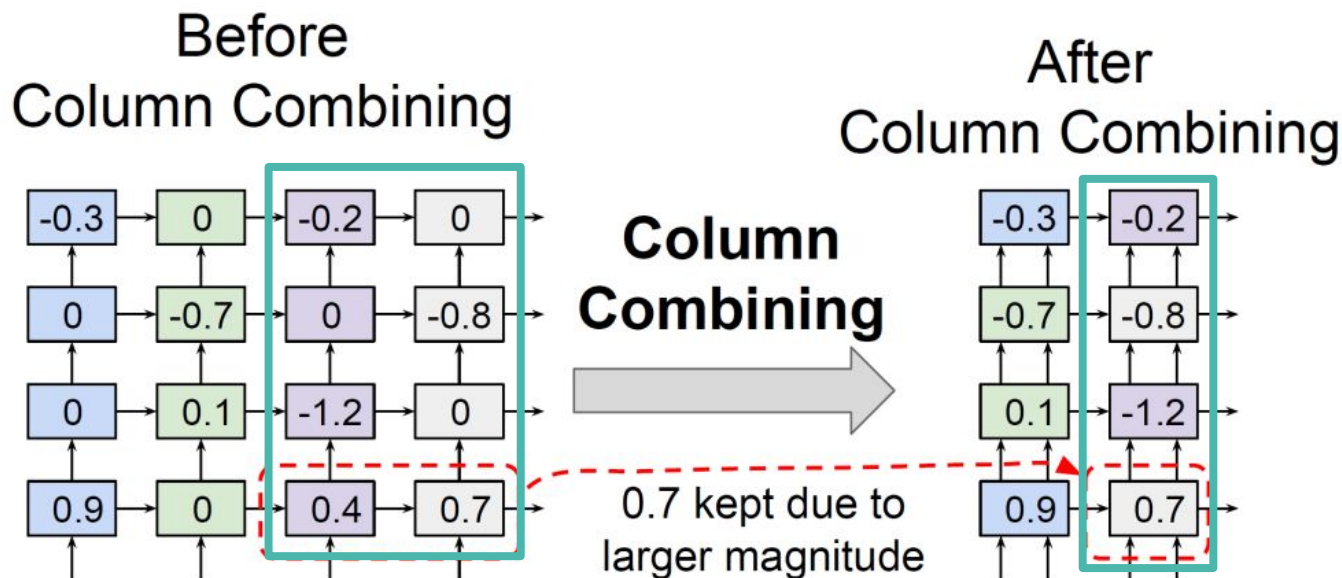
Column Combining Continued



Column Combining Continued

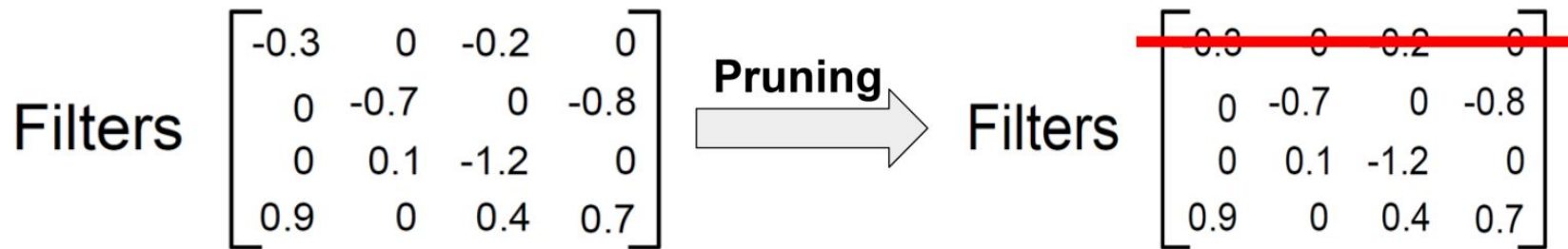


Column Combining Continued



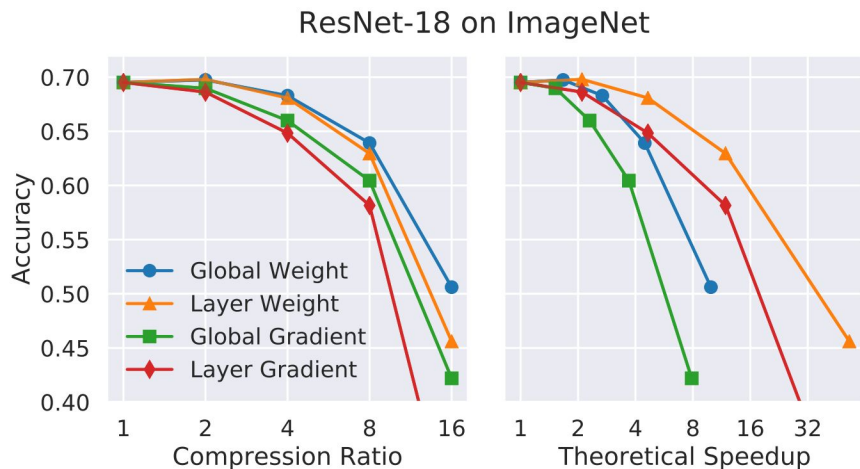
Structured Pruning Example

- Within a single layer, set filters to 0 which have small L1 or L2 Norm
- Efficient implementation on hardware, but low accuracy
 - No need to deal with indexing location of non-zeros



Comparing Structured and Unstructured Pruning

- This figure shows that Global (**unstructured pruning**) methods achieve the best **accuracies** for gradient and weights for higher compression ratios. However, Layer (**structured pruning**) achieves greater theoretical **speedups** for that same compression ratio



Methods to Maintain Accuracy

- We can be less aggressive and set a low threshold for sensitive layers of the network
- An example would be the first layer of the convolutional network?
- Why do you think so?

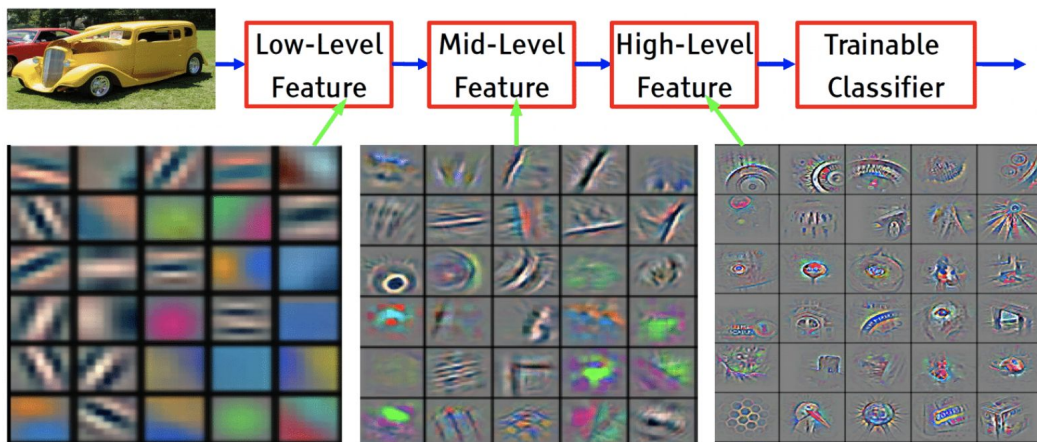


Figure source:
Zeiler, Matthew D., and
Rob Fergus. "Visualizing
and understanding
convolutional networks."
ECCV. Springer, Cham,
2014.

Methods to Maintain Accuracy

- Answer:
- If you mess up the accuracy of low-level features, you will mess up the high-level features since it is composed of low-level features
- You can test this empirically by pruning different layers with differences in pruning threshold

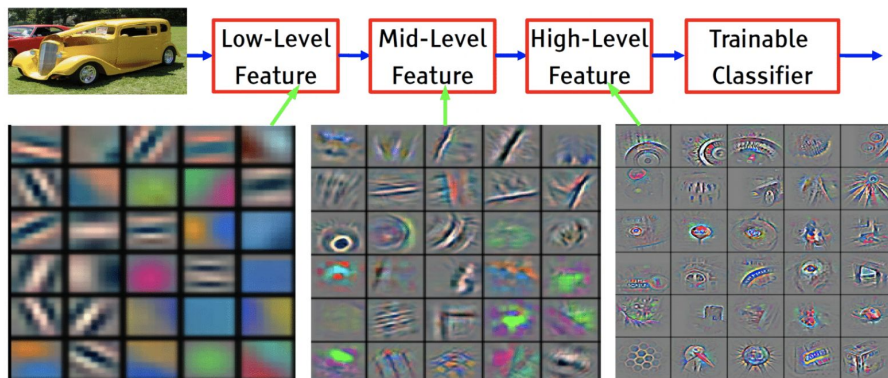


Figure source:
Zeiler, Matthew D., and
Rob Fergus. "Visualizing
and understanding
convolutional networks."
ECCV. Springer, Cham,
2014.

Problems with Pruning

- Expensive during training time since we have to re-train the pruned network
- Unstructured pruning approaches are not hardware friendly
- Accuracy loss - might be difficult to get a good balance

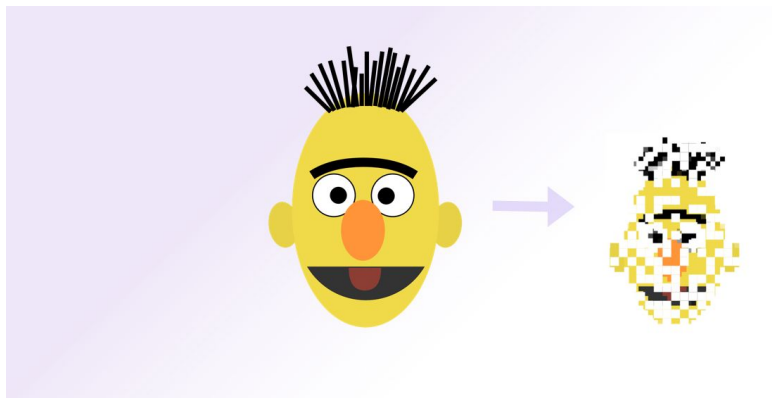


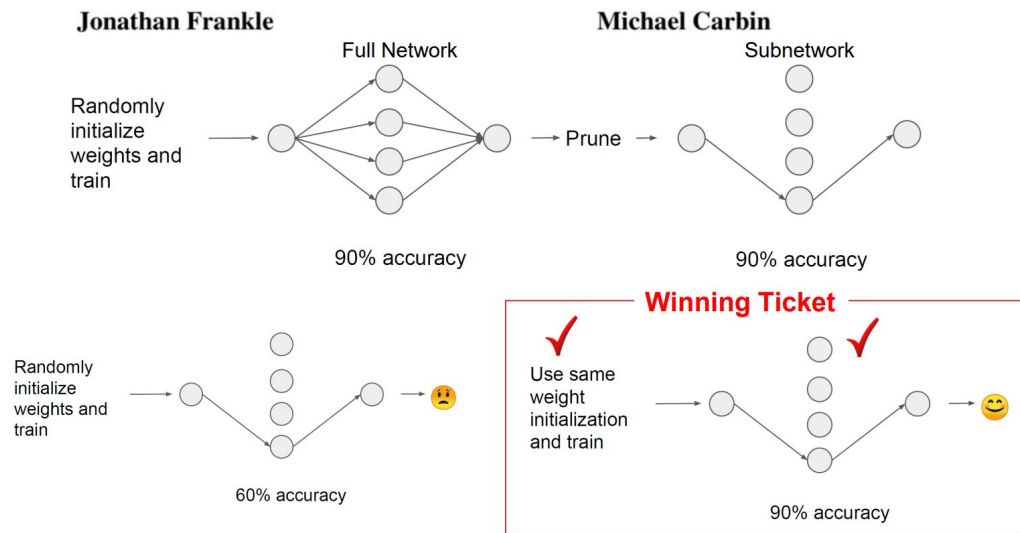
Figure source:
<https://rasa.com/blog/compressing-bert-for-faster-prediction-2/>

Lottery Ticket Hypothesis

Find a Subnetwork

Lottery Ticket Hypothesis

- Hypothesis: Can we train the pruned subnetwork directly without having to train-prune-retrain?
- Less expensive - save a lot of training time
- How can we find winning tickets?



Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis, Finding Sparse, Trainable Neural Networks." ICLR, 2019.

Figure source:
<https://towardsdatascience.com/breaking-down-the-lottery-ticket-hypothesis-ca1c053b3e58>

**ME, FINDING OUT ABOUT
THE LOTTERY TICKET HYPOTHESIS**



imgflip.com

Meme source: Figure
source:
<https://blog.kensho.com/rays-r-d-run-down-a-neurips-dive-542ece303a6459>

Leveraging Sparsity

- We can skip zeros during inference
- Libraries for optimized Sparse Matrix Multiplication
 - ARMPL Sparse Basic Linear Algebra Subprograms (BLAS)
- Specialized Hardware: Nvidia A100 GPU Sparse Tensor Core
- **Open area of research: can we build specialized hardware in the TinyML space so that structured sparsity can be exploited?**

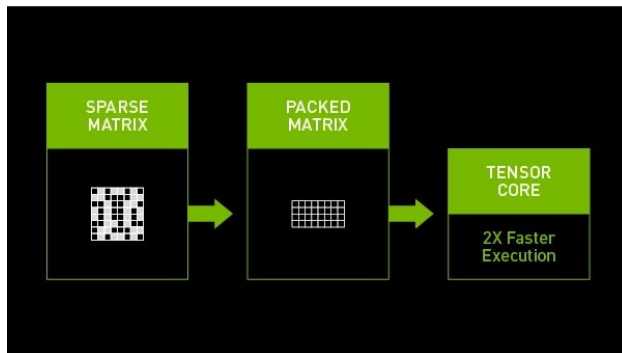


Figure source:
<https://blogs.nvidia.com/blog/2020/05/14/sparsity-ai-inference/>

Example of Pruning in Code

- Tensorflow has built-in pruning
- `initial_sparsity` = % of 0 weights
- `Final_sparsity` is targeted %
- Here we pruned 30% of weights
 - $0.80 - 0.50 = 0.30$

```
In [0]:

import tensorflow_model_optimization as tfmot

prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

# Compute end step to finish pruning after 2 epochs.
batch_size = 128
epochs = 2
validation_split = 0.1 # 10% of training set will be used for validation set.

num_images = train_images.shape[0] * (1 - validation_split)
end_step = np.ceil(num_images / batch_size).astype(np.int32) * epochs

# Define model for pruning.
pruning_params = {
    'pruning_schedule': tfmot.sparsity.keras PolynomialDecay(initial_sparsity=0.50,
                                                             final_sparsity=0.80,
                                                             begin_step=0,
                                                             end_step=end_step)
}

model_for_pruning = prune_low_magnitude(model, **pruning_params)

# `prune_low_magnitude` requires a recompile.
model_for_pruning.compile(optimizer='adam',
                          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                          metrics=['accuracy'])

model_for_pruning.summary()
```

Pruning + Quantization

Unified Pruning and Quantization

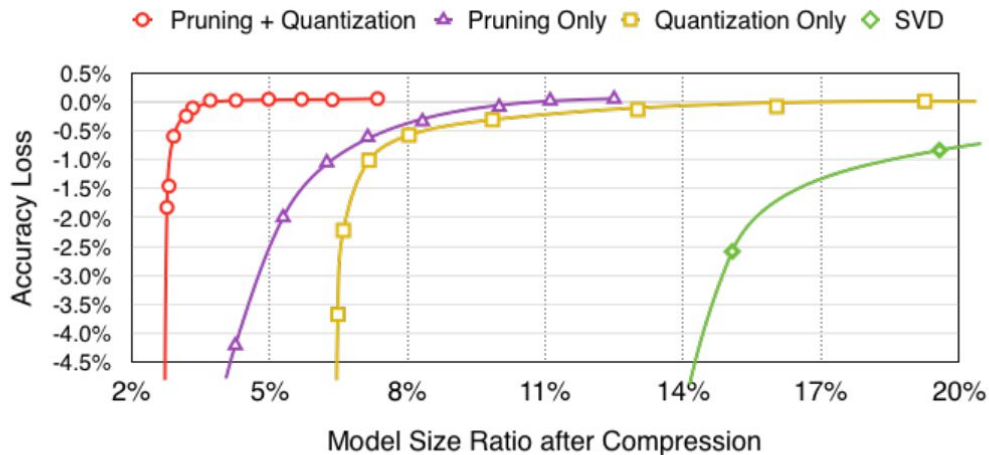


Figure 6: Accuracy v.s. compression rate under different compression methods. Pruning and quantization works best when combined.

Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." ICLR, 2015.

How we improved the slides

- Add accuracy differences between structured and unstructured pruning
- Include answer to my pruning question in the slides
- Include pruning code
- Include quantization code
- Include the acceleration effect of quantized neural network
- Add quantitative comparison among quantization methods
- Before starting to a new topic, reminding the agenda of the talk is helpful for better contextualizing the work.
- Define mantissa and exponent more
- Add model compression differences among domains ex: image classification, generation etc.
- Quantization is a type of regularization (slide about quantization on neural networks)
- Slide numbering

Key papers

- [Piecewise Linear Quantization](#)
- [Term Quantization](#)
- [Column Combining](#)
- [DoReFa-Net](#)
- [PACT](#)
- [Lottery Ticket Hypothesis](#)
- [Pruning + Quantization \(Deep Compression\)](#)
- [What is the State of Neural Network Pruning?](#)