



Neural Network Optimization for Efficient Inference

Alexander Suslov

Neural Network Applications

Self-Driving Car



This image is in the public domain

Robots



This image is in the public domain

Image processing



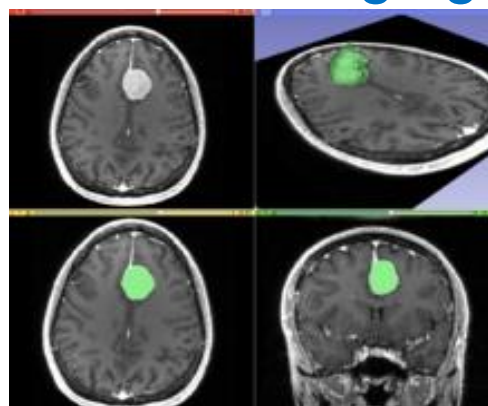
This image is in the public domain

Machine Translation



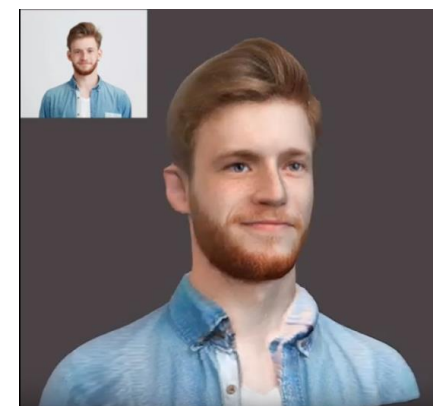
This image is in the public domain

Medical imaging



This image is in the public domain

3D scanning



This image is in the public domain

Where does Inference of Neural Networks Compute?

Standalone



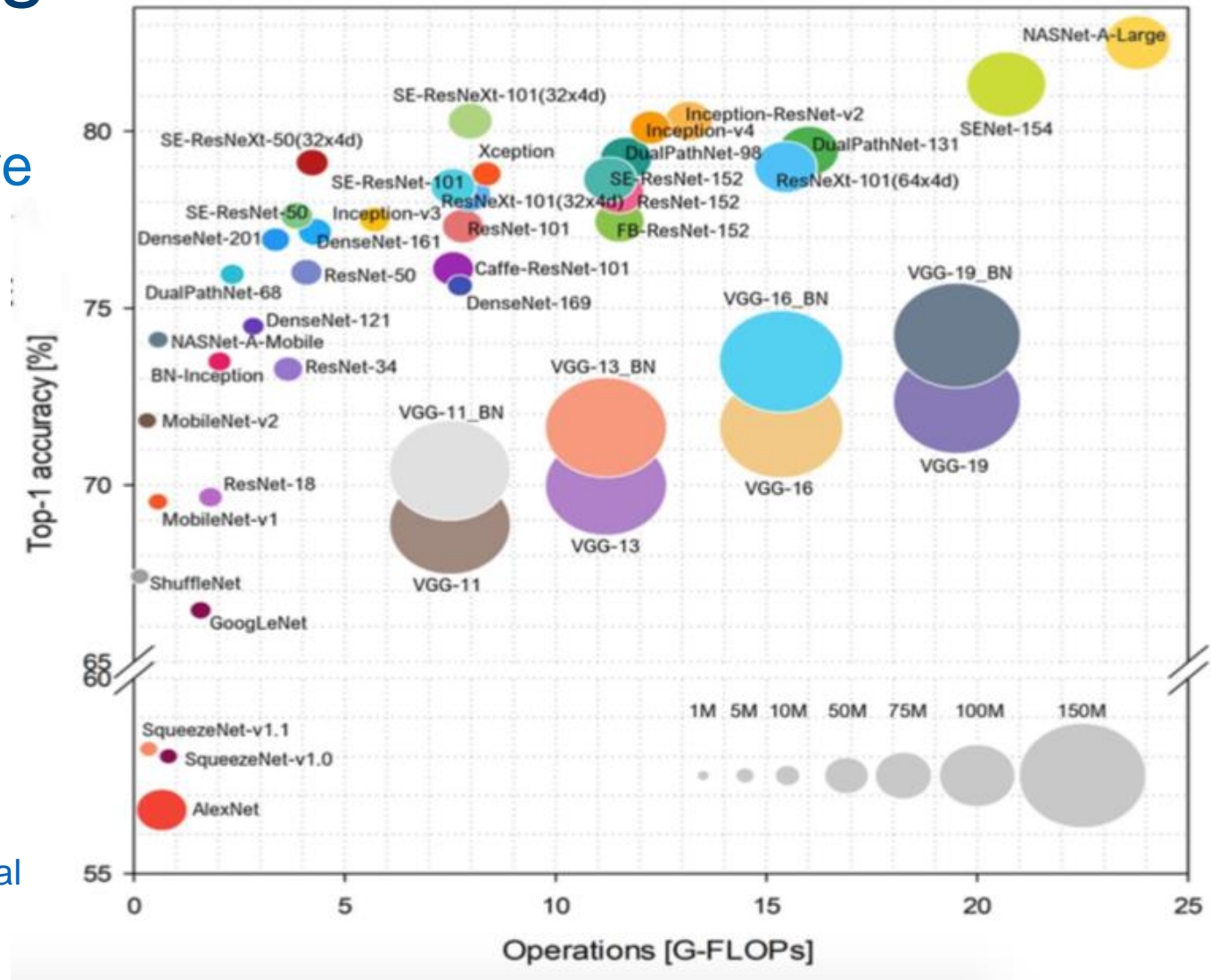
Client-Server



All images are in the public domain

Models are Getting Larger

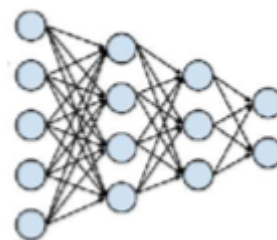
- While models are becoming more efficient, high accuracy still implies high complexity



From: [Benchmark Analysis of Representative Deep Neural Network Architectures](#), Simone Bianco et al,

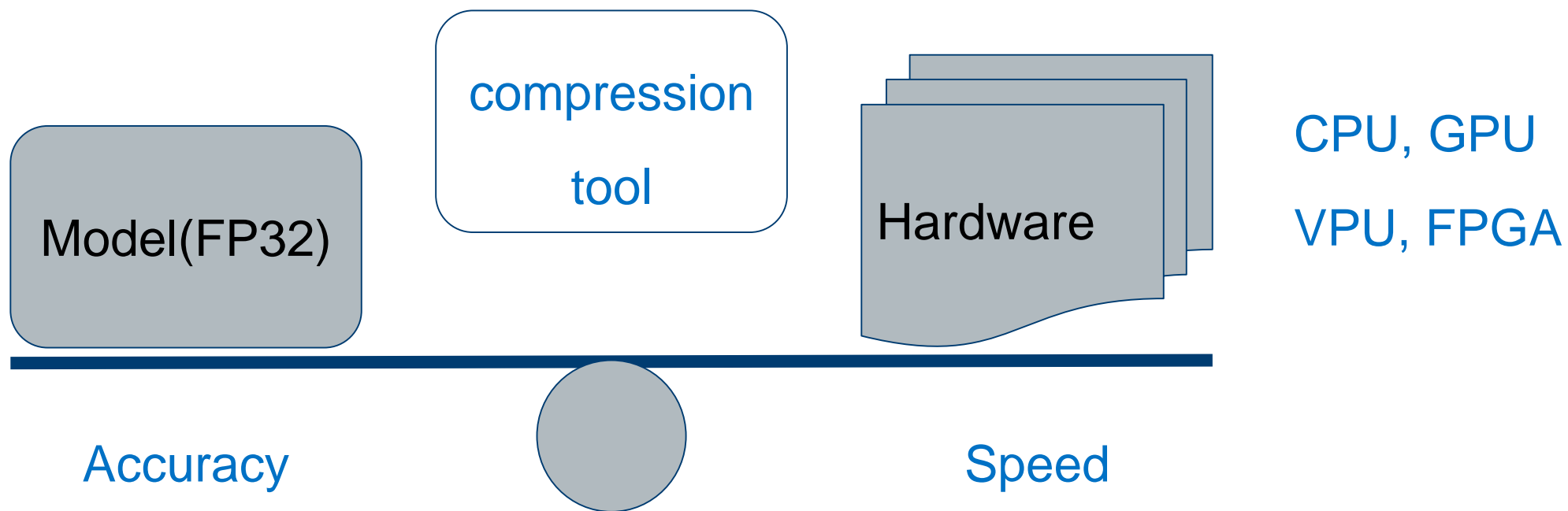
The First Challenge: Model Size

- Hard to distribute large models through over-the-air update
- The first run is slow due to loading weights.



All images are in the public domain

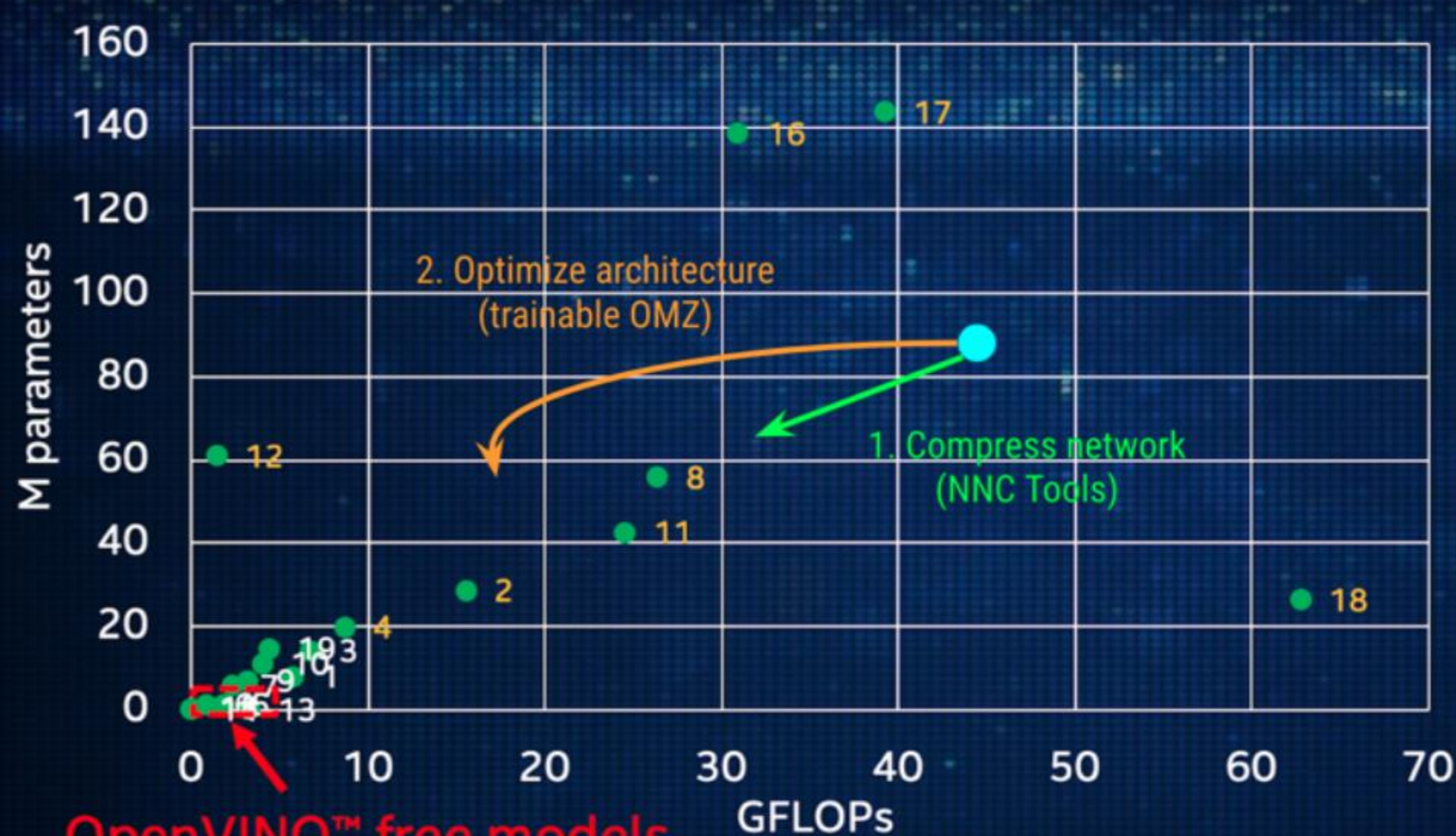
The Second Challenge: Speed



Trade off between accuracy and performance

Algorithms for Efficient Inference

1. Pruning
2. Weight Sharing
3. Quantization
4. Binary / Ternary Net
5. Distillation
6. Low Rank Approximation
7. Winograd Transformation



- 1 DenseNet-121
- 2 DenseNet-161
- 3 DenseNet-169
- 4 DenseNet-201
- 5 SqueezeNet1.0
- 6 SqueezeNet1.1
- 7 MobileNet-SSD
- 8 Inception-ResNet-v2
- 9 GoogLeNet-v1
- 10 GoogLeNet-v2
- 11 GoogLeNet-v4
- 12 AlexNet
- 13 MTCNN-p
- 14 MTCNN-r
- 15 MTCNN-o
- 16 VGG16
- 17 VGG19
- 18 SSD300
- 19 MobileNetv2-SSD

OpenVINO™ free models

Algorithms for Efficient Inference

1. Pruning

2. Weight Sharing

3. Quantization

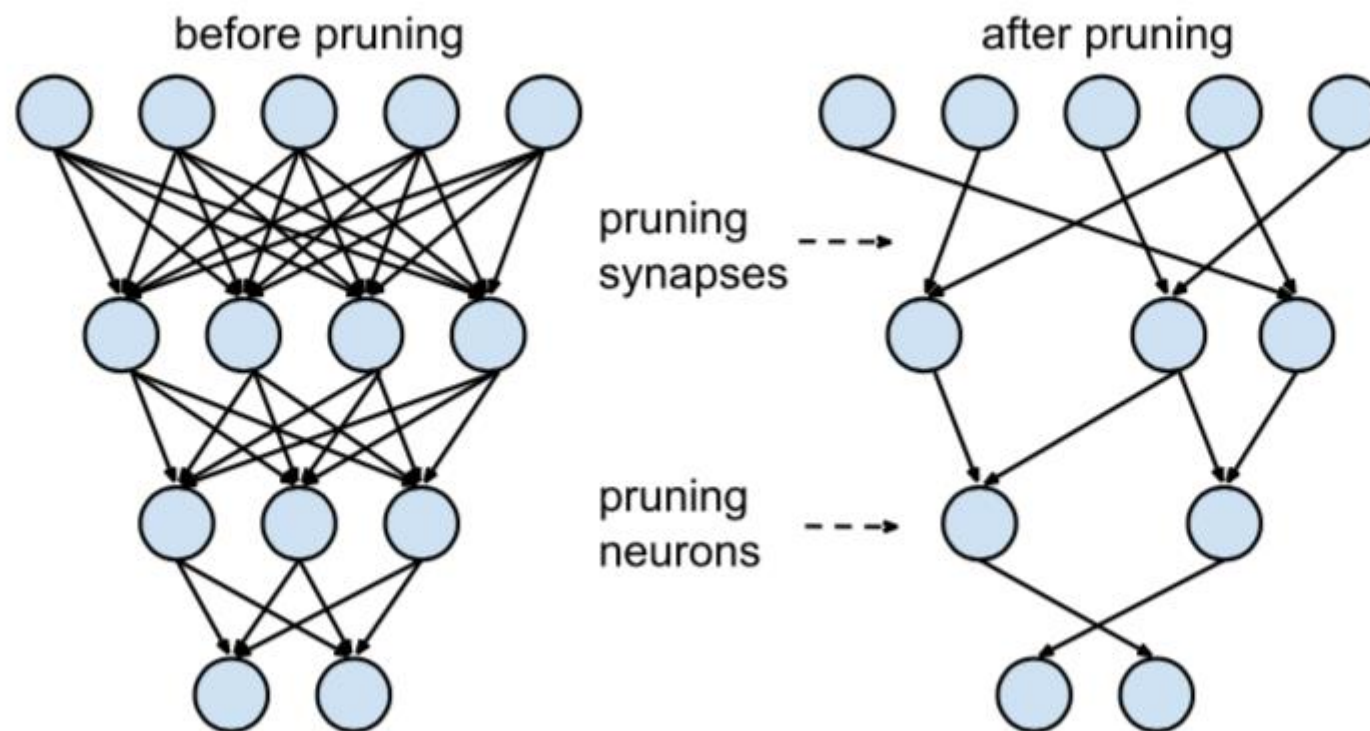
4. Binary / Ternary Net

5. Distillation

6. Low Rank Approximation

7. Winograd Transformation

Pruning Neural Networks

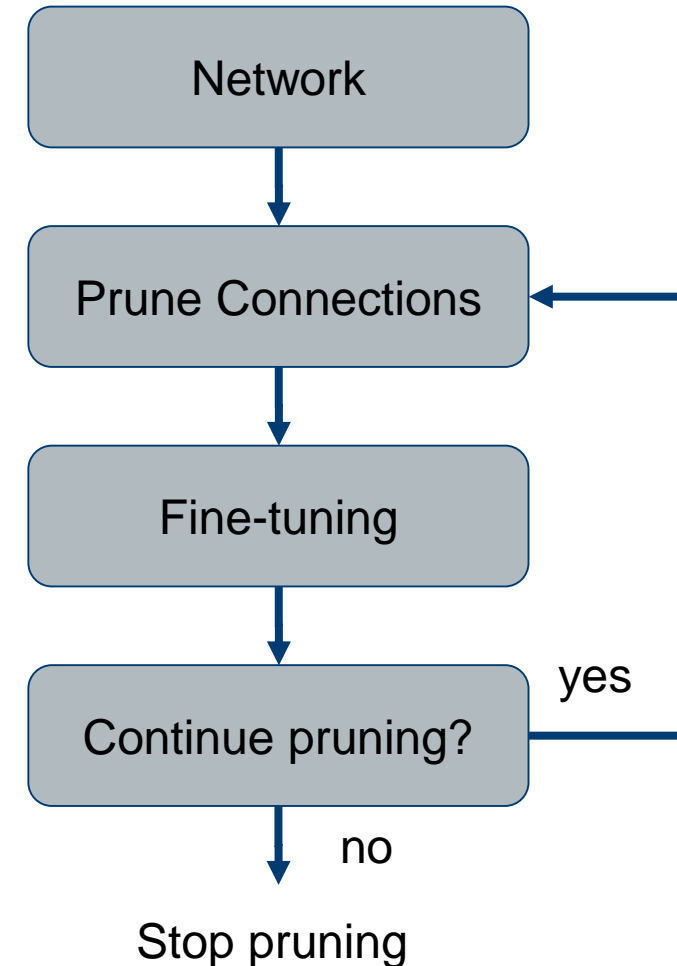


Pruning Neural Networks

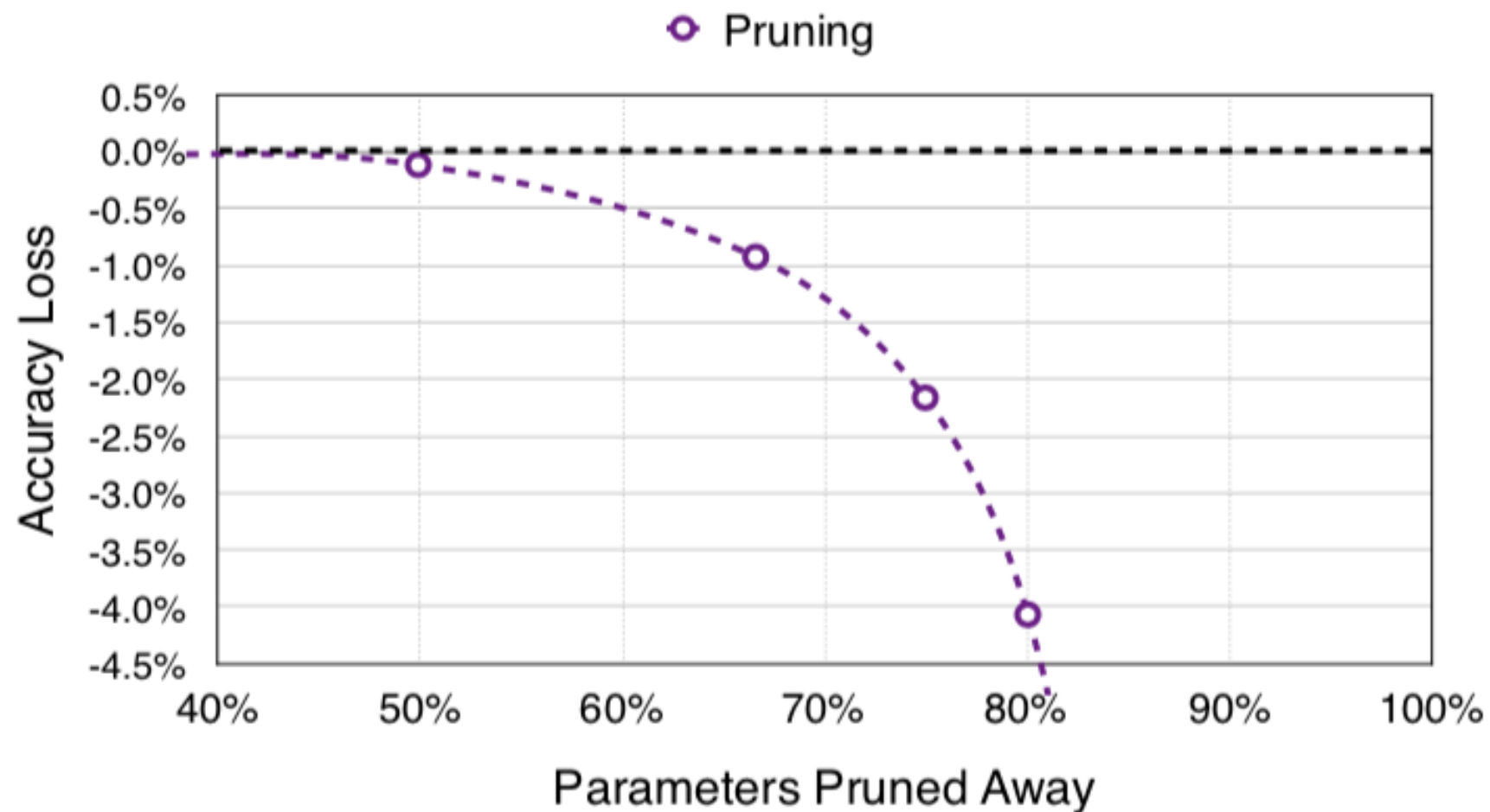
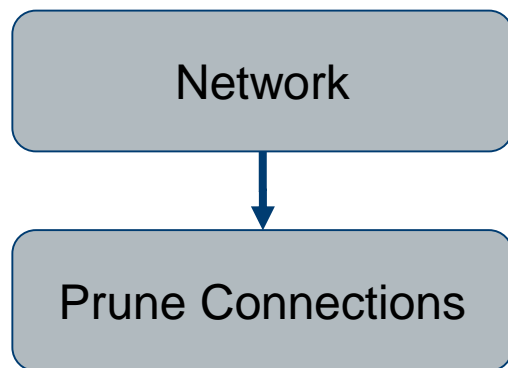
- Neural network pruning benefits
 - Reducing the binary model size
 - Smaller models reduce memory bandwidth bottlenecks
 - Faster kernels (depends on hardware support)

Pruning Neural Networks

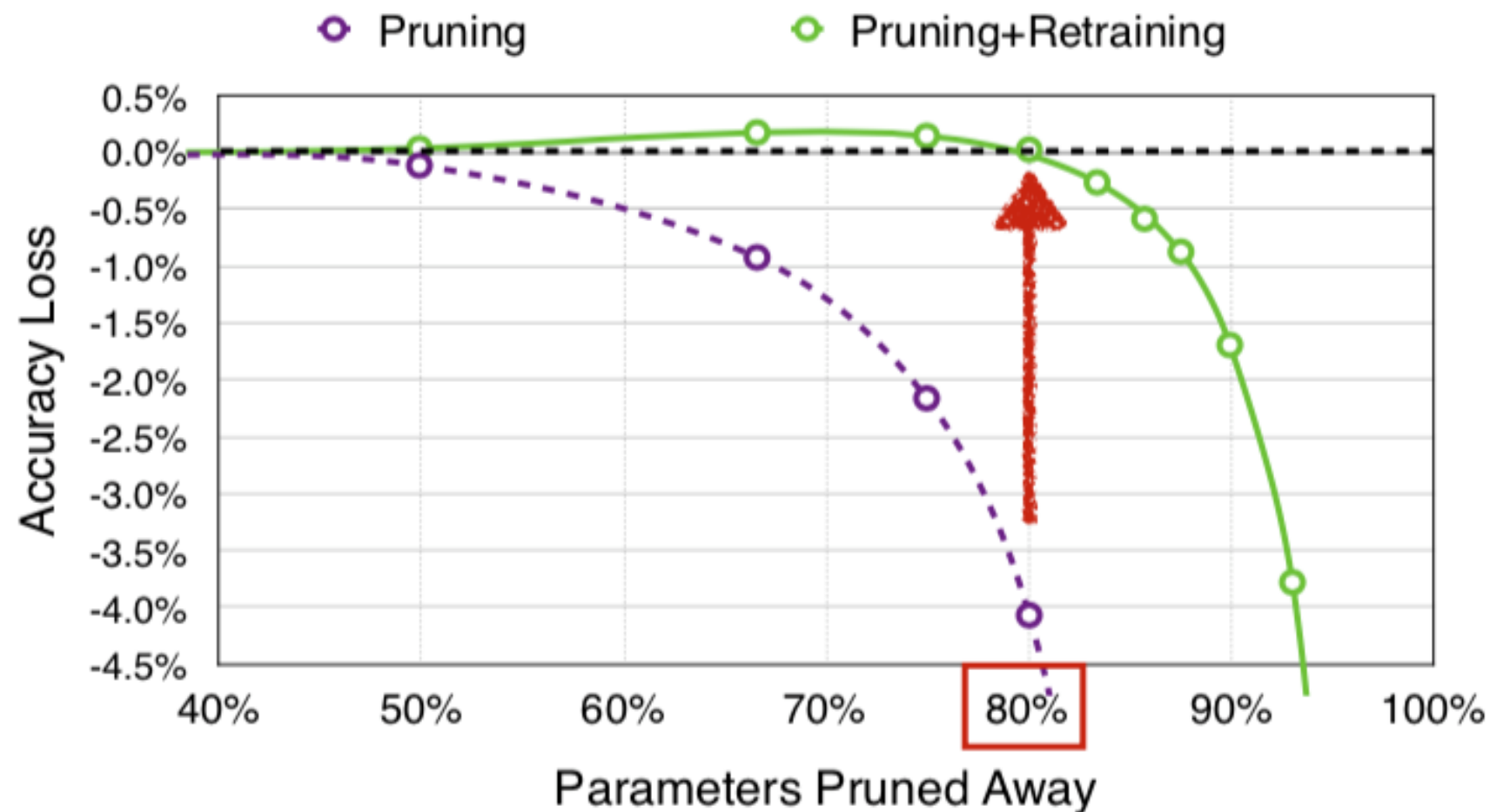
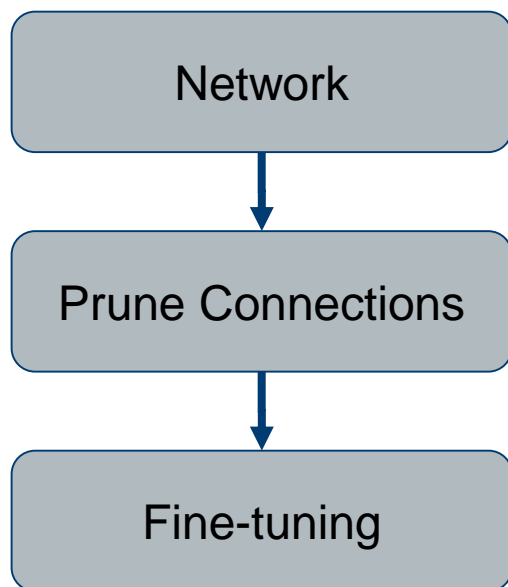
- Criteria for pruning
 - Connections with low weights
 - Neurons or filters with low impact
- External limitations
 - Spatial structure of pruned weights



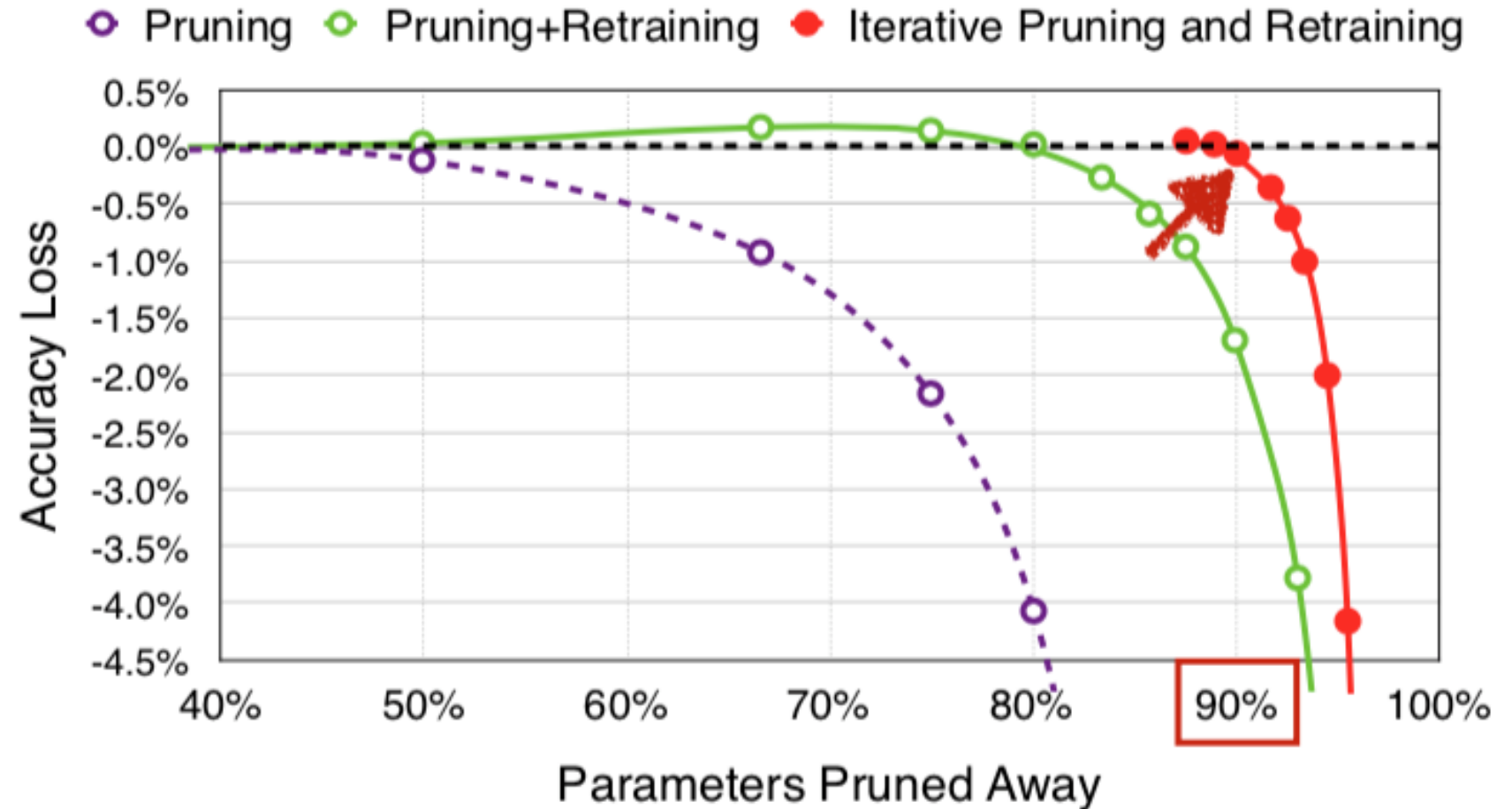
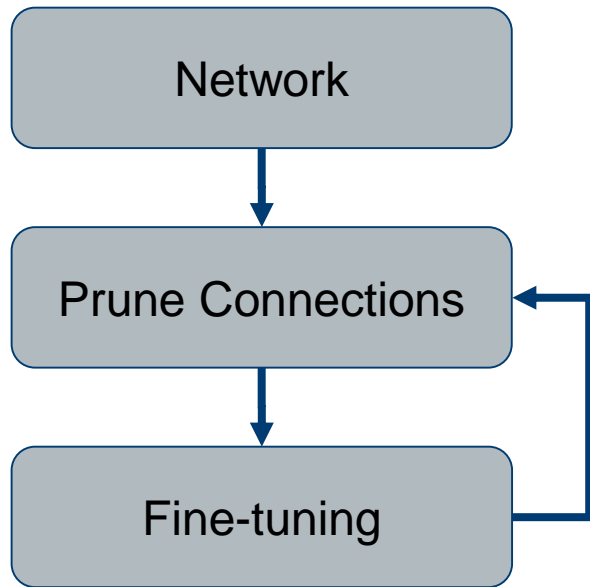
Pruning Neural Networks



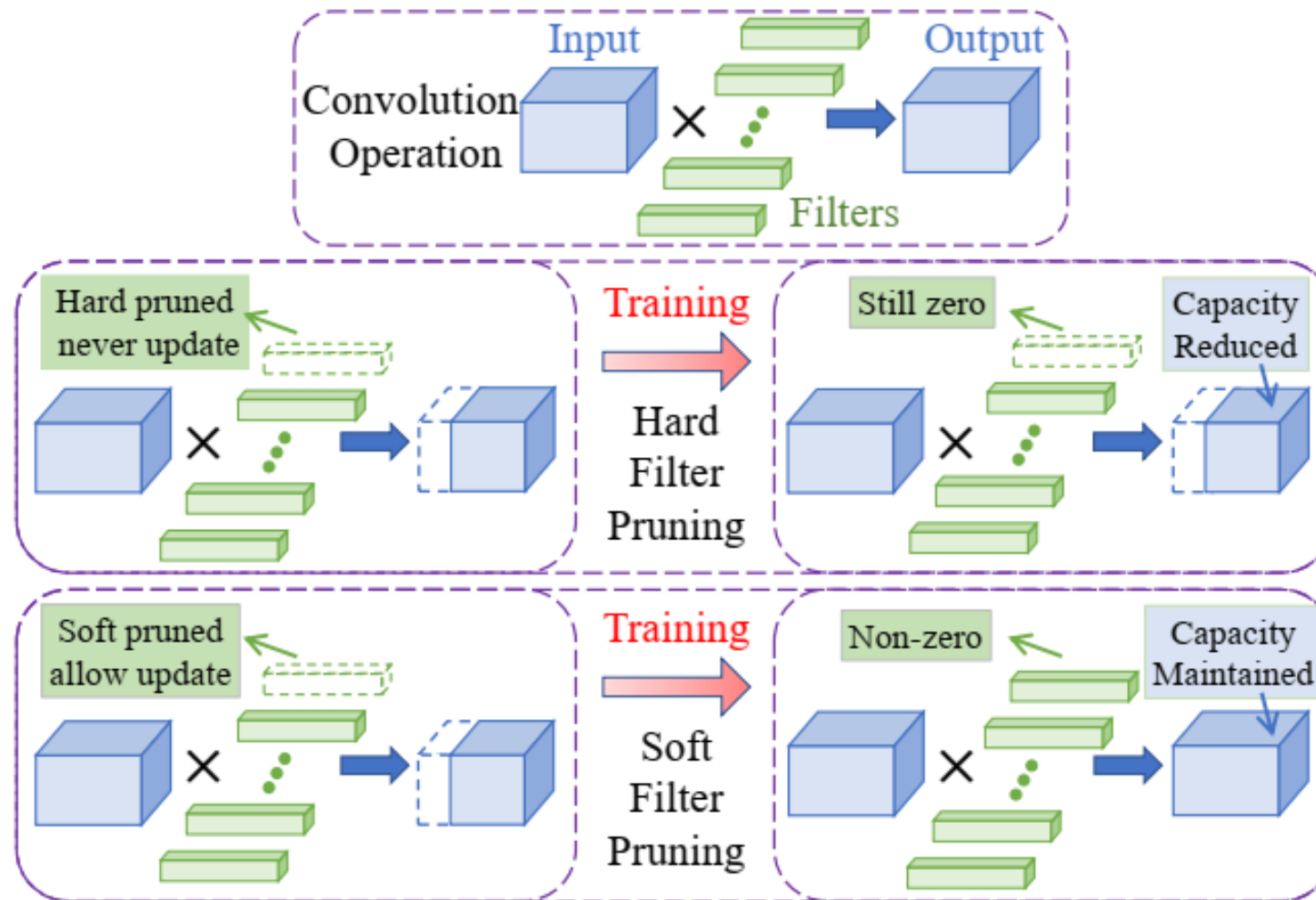
Retrain to Recover Accuracy



Iteratively Retrain to Recover Accuracy



Filter Pruning: Main Approaches



Filter Pruning: Filter Importance Criteria

- L1, L2

$$||F||_p = \sqrt[p]{\sum_{c,k_1,k_2=1}^{C,K,K} |F(c, k_1, k_2)|^p}$$

- “Geometric Median”

$$G(F_i) = \sum_{F_j \in \{F_1, \dots, F_m\}, j \neq i} ||F_i - F_j||_2$$

Filter Pruning: Results

| Models | Compression algorithm | Dataset | Top-1 Accuracy FP32 model (%) | Top-1 Accuracy Pruned model (%) |
|-----------|--|----------|----------------------------------|------------------------------------|
| ResNet-18 | Filter pruning, 30%, magnitude criterion | ImageNet | 69.76 | 68.69 |
| ResNet-18 | Filter pruning, 30%, geometric median criterion | ImageNet | 69.76 | 68.97 |
| ResNet-34 | Filter pruning, 30%, magnitude criterion | ImageNet | 73.31 | 72.54 |
| ResNet-34 | Filter pruning, 30%, geometric median criterion | ImageNet | 73.31 | 72.60 |
| ResNet-50 | Filter pruning, 30%, magnitude criterion | ImageNet | 76.13 | 75.7 |
| ResNet-50 | Filter pruning, 30%, geometric median criterion | ImageNet | 76.13 | 75.7 |

https://github.com/openvinotoolkit/nncf_pytorch

Sparsification: Main Approaches

- **Magnitude Sparsity**
 - After each training epoch the method calculates a threshold based on the current sparsity ratio and uses it to zero weights which are lower than this threshold.
- **Regularization-Based (RB) Sparsity**
 - The sparsification algorithm based on probabilistic approach and loss regularization.

Ordinary convolution

$$output = conv(x, w)$$

Sparsifying weights we reparametrize weights as:

$$\bar{w} = w * z$$

Where:

w – weights, $w \in R$

z – binary mask, $z \in [0, 1]$

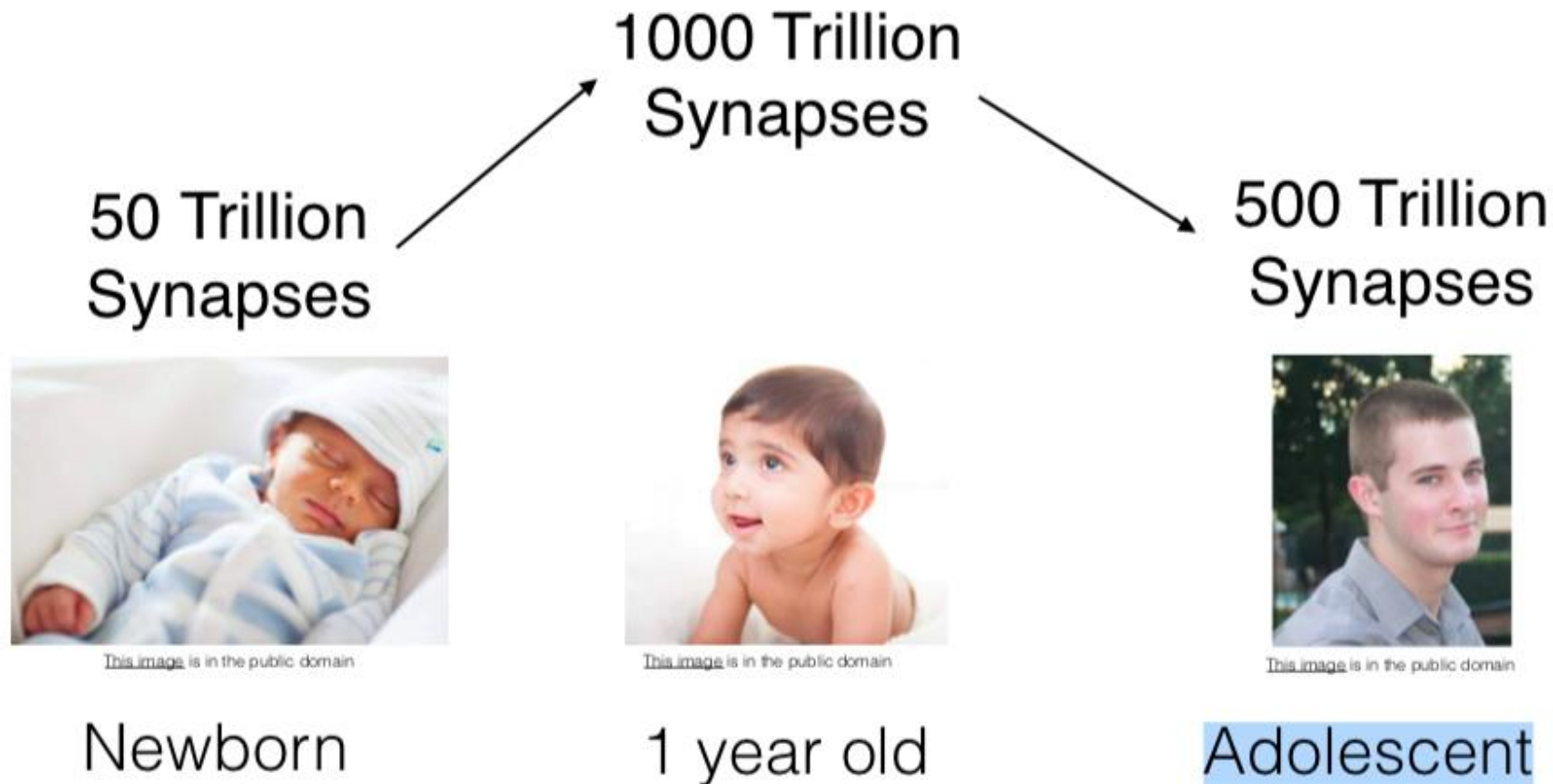
We train these masks using modified loss

$$Loss = Loss_{task} + \alpha \left(\sum_l^{Layers} ||z|| - target \right)^2$$

Sparsification: Results

| Models | Compression algorithm | Dataset | Top-1 Accuracy FP32 model (%) | Top-1 Accuracy Pruned model (%) |
|--------------|---------------------------------------|----------|----------------------------------|------------------------------------|
| inception-v3 | RB-sparsity, 50% sparsity rate | ImageNet | 77.46 | 77.25 |
| inception-v3 | Magnitude sparsity, 50% sparsity rate | ImageNet | 77.46 | 77.24 |
| inception-v3 | RB-sparsity, 92% sparsity rate | ImageNet | 77.46 | 76.6 |
| mobilenet-v2 | RB-sparsity, 50% sparsity rate | ImageNet | 71.8 | 71.2 |
| mobilenet-v2 | Magnitude sparsity, 50% sparsity rate | ImageNet | 71.8 | 70.8 |
| mobilenet-v2 | RB-sparsity, 78% sparsity rate | ImageNet | 71.8 | 69.98 |

Pruning Happens in Human Brain



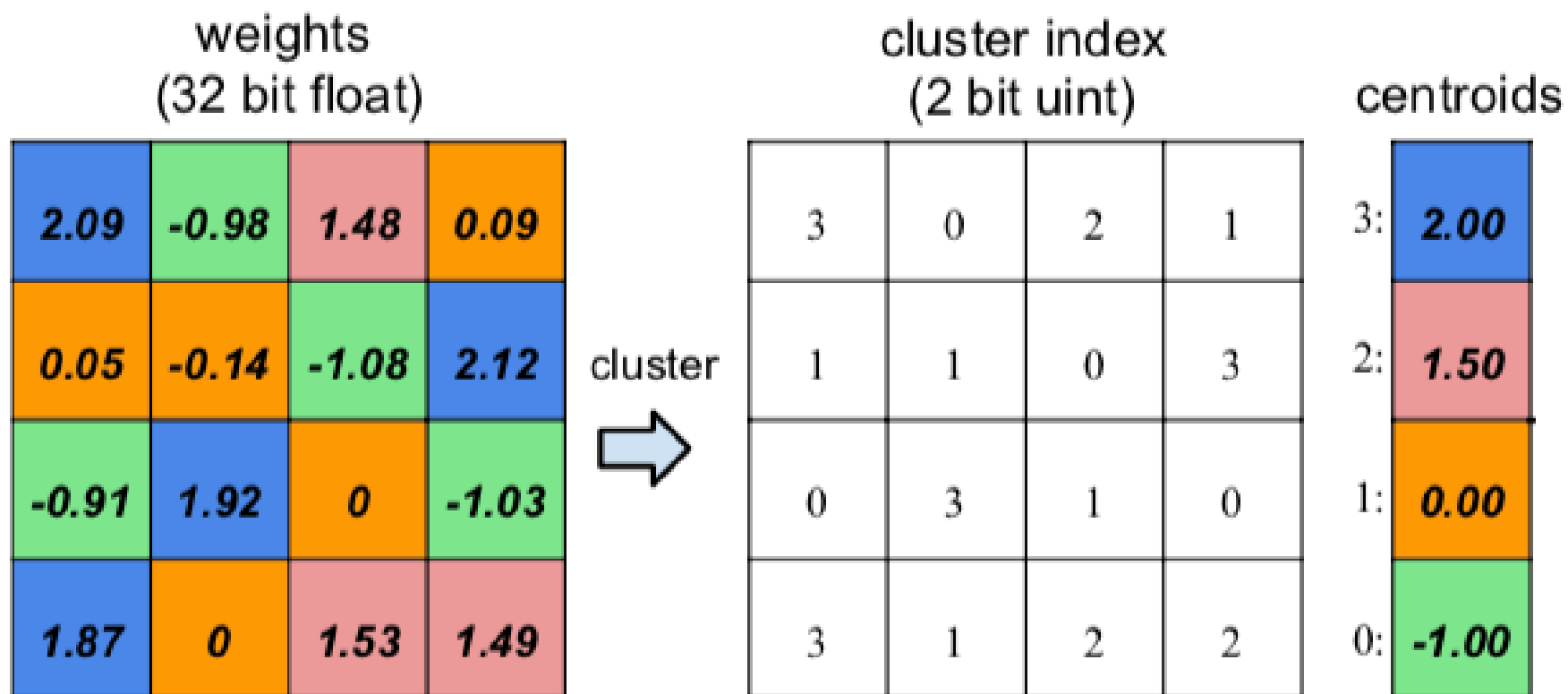
Christopher A Walsh. Peter Huttenlocher (1931-2013). Nature, 502(7470):172–172, 2013.

Slide credits by Song Han

Algorithms for Efficient Inference

1. Pruning
2. Weight Sharing
3. Quantization
4. Binary / Ternary Net
5. Distillation
6. Low Rank Approximation
7. Winograd Transformation

Weight sharing



Pruning + Weight sharing + Huffman Encoding

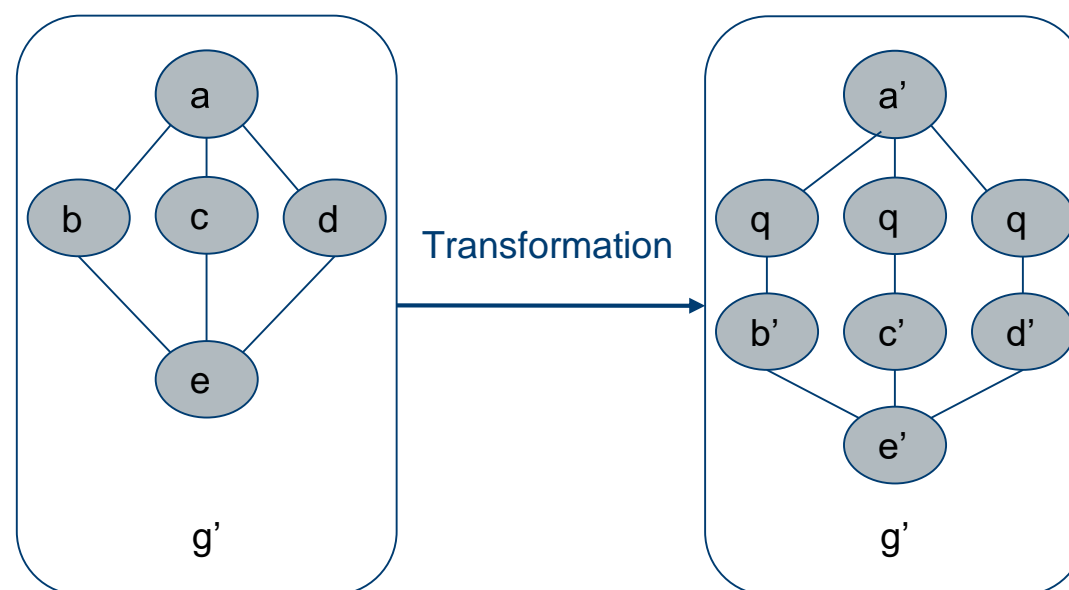
| Network | Original Size | Compressed Size | Compression Ratio | Original Accuracy | Compressed Accuracy |
|-----------|----------------|-----------------|-------------------|-------------------|---------------------|
| LeNet-300 | 1070KB → 27KB | | 40x | 98.36% | → 98.42% |
| LeNet-5 | 1720KB → 44KB | | 39x | 99.20% | → 99.26% |
| AlexNet | 240MB → 6.9MB | | 35x | 80.27% | → 80.30% |
| VGGNet | 550MB → 11.3MB | | 49x | 88.68% | → 89.09% |
| GoogleNet | 28MB → 2.8MB | | 10x | 88.90% | → 88.92% |
| ResNet-18 | 44.6MB → 4.0MB | | 11x | 89.24% | → 89.28% |

Algorithms for Efficient Inference

1. Pruning
2. Weight Sharing
- 3. Quantization**
4. Binary / Ternary Net
5. Distillation
6. Low Rank Approximation
7. Winograd Transformation

Quantization: Overview

This is the process of transforming a neural network such that it can be represented and executed at a lower precision by discretizing the original neural network weights and activations.

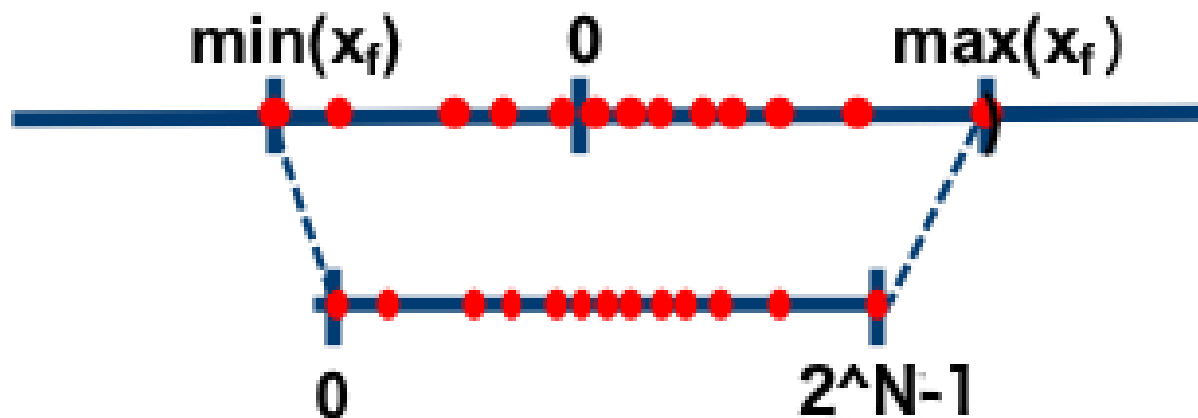


Quantization: Overview

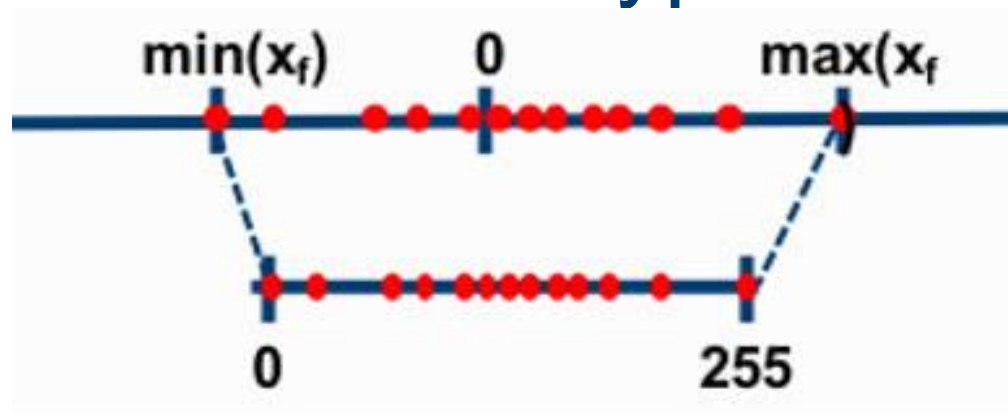
- **Quantization function** is a mapping of values from high to low precision
- **Neural network transformation** is the process of getting g' from g
- **Quantization algorithm** computes quantization parameters required by new neural network g' and optimizes g' via fine-tuning or post-training algorithms.
 - Post-training quantization without dataset
 - Post-training quantization with dataset
 - Quantization aware training

Quantization: Quantization function

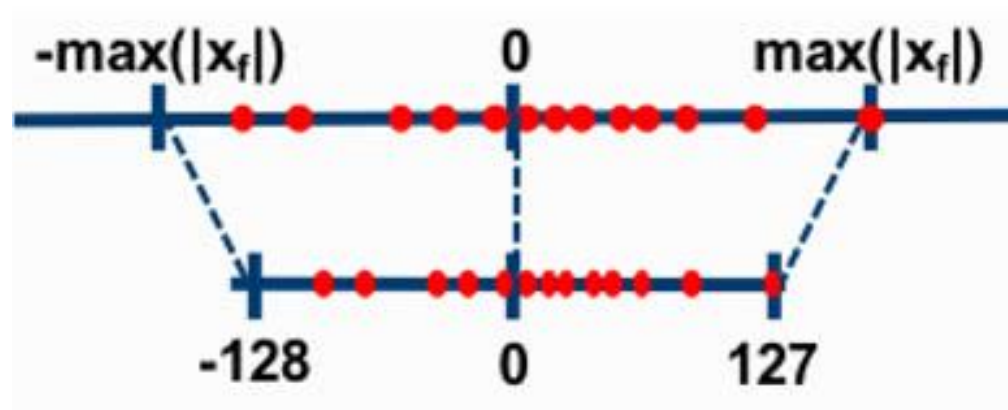
- Quantization refers to mapping values from fp32 to a lower precision format with specified parameters:
 - Precision
 - Quantization type
 - Granularity



Quantization: Quantization types



Asymmetric Mode



Symmetric Mode

Asymmetric Quantization

- Quantization:

$$x_{int} = round\left(\frac{x}{\Delta}\right) + z$$

$$x_Q = clamp(0, N_{levels} - 1, x_{int})$$

- Δ specifies the step size of the quantizer and floating point zero maps to zero-point.
- z - zero-point.
- $N_{levels} = 256$ for 8-bits of precision

- De-quantization:

$$x_{float} = (x_Q - z)\Delta$$

Asymmetric Quantization

- 2D convolution:

$$y(k, l, n) = \Delta_w \Delta_x \text{conv}(w_Q(k, l, m; n) - z_w, x_Q(k, l, m) - z_x)$$

$$y(k, l, n) = \text{conv}(w_Q(k, l, m; n), x_Q(k, l, m)) - z_w \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} \sum_{m=0}^{N-1} x_Q(k, l, m) \\ - z_x \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} \sum_{m=0}^{N-1} w_Q(k, l, m; n) + z_x z_w$$

Symmetric Quantization

- Quantization, zero-point = 0

- Activations:

$$x_{int} = \text{round}\left(\frac{x}{\Delta}\right)$$

$$x_Q = \text{clamp}(-N_{levels}/2, N_{levels}/2 - 1, x_{int}) \quad \text{if signed}$$

$$x_Q = \text{clamp}(0, N_{levels} - 1, x_{int}) \quad \text{if un-signed}$$

- Weights

$$x_Q = \text{clamp}(-(N_{levels}/2 - 1), N_{levels}/2 - 1, x_{int}) \quad \text{if signed}$$

$$x_Q = \text{clamp}(0, N_{levels} - 2, x_{int}) \quad \text{if un-signed}$$

Symmetric Quantization

- MKL DNN Int8 Workflow:

$$X_{s32} = W_{s8} \times cru8 + b_{s32} \approx Q_{\alpha} Q_w X_{f32}$$

$$\text{where } X_{f32} = W_{f32} \times \alpha_{f32} + b_{f32}$$

$Q_{\alpha} = \frac{255}{R_{\alpha}}$ is the quantization factor for activations with non-negative values.

$Q_w = \frac{127}{R_w}$ is the quantization factor for weights.

$$\alpha_{u8} = \lceil Q_{\alpha} \alpha_{f32} \rceil \in [0, 255]$$

$$W_{s8} = \lceil Q_w W_{f32} \rceil \in [-127, 127]$$

$$b_{s32} = \lceil Q_{\alpha} Q_w b_{f32} \rceil \in [-2^{31}, 2^{31} - 1]$$

Quantization granularity

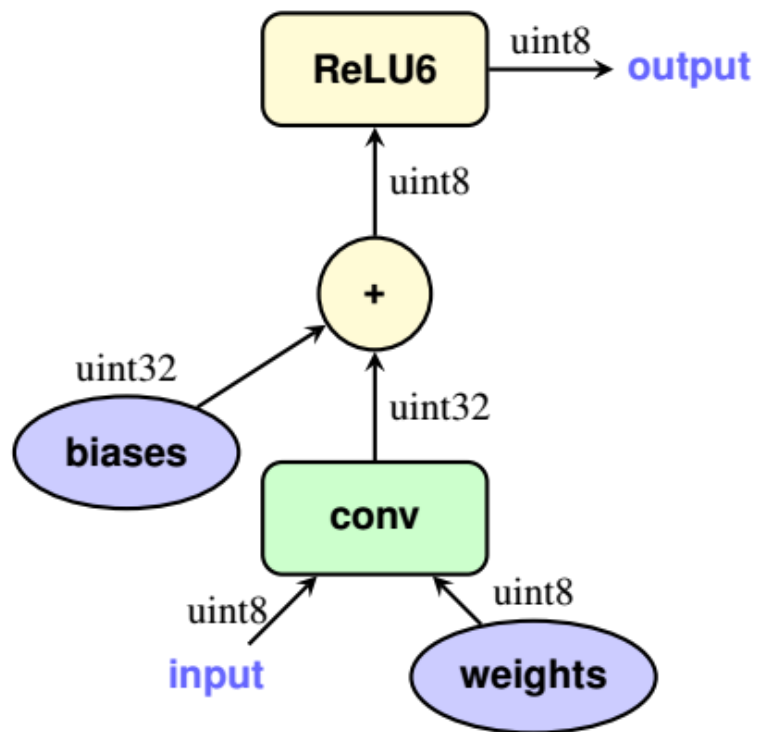
- We consider different granularities of quantization:
 - Per-layer quantization
 - Same mapping for all elements in a layer.
 - Per-channel quantization:
 - Choose quantizer parameters independently for each row (fc layers) or for each conv kernel (conv layers)

Fake-quantization

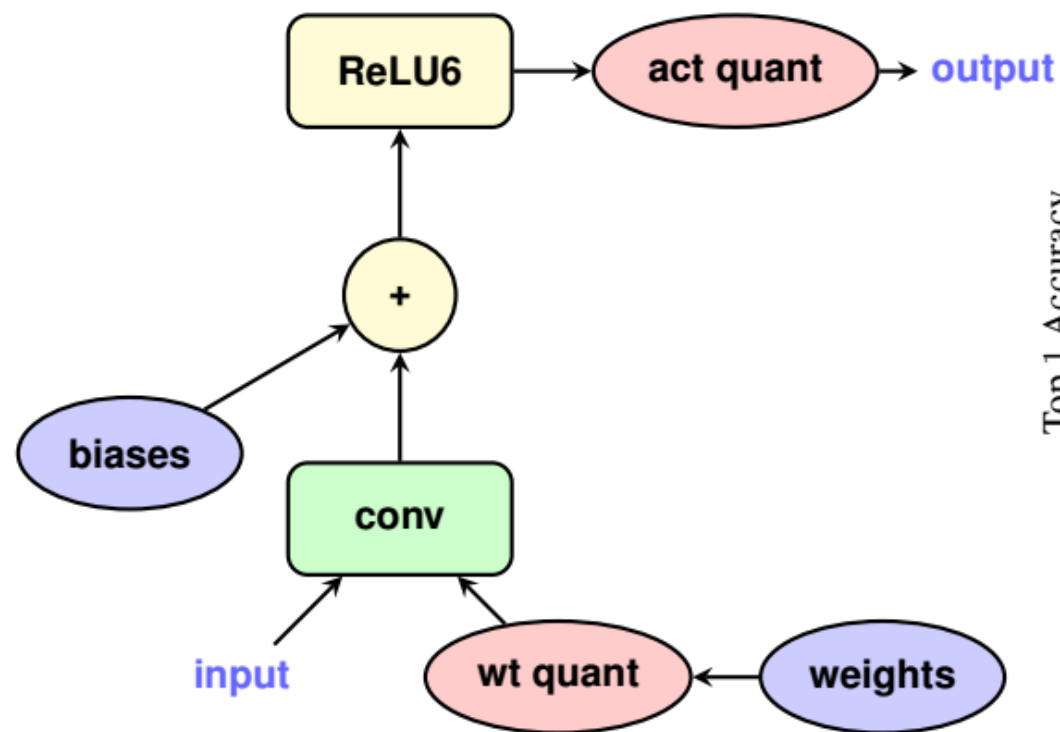
- Emulate quantization by quantizing and de-quantizing:
 - Values are still in floating point, but with reduced precision

$$\begin{aligned}\text{clamp}(r; a, b) &:= \min(\max(x, a), b) \\ s(a, b, n) &:= \frac{b - a}{n - 1} \\ q(r; a, b, n) &:= \left\lfloor \frac{\text{clamp}(r; a, b) - a}{s(a, b, n)} \right\rfloor s(a, b, n) + a, \\ &\quad (12)\end{aligned}$$

Quantization: Neural Network Transformation



(a) Integer-arithmetic-only inference



(b) Training with simulated quantization

Top 1 Accuracy

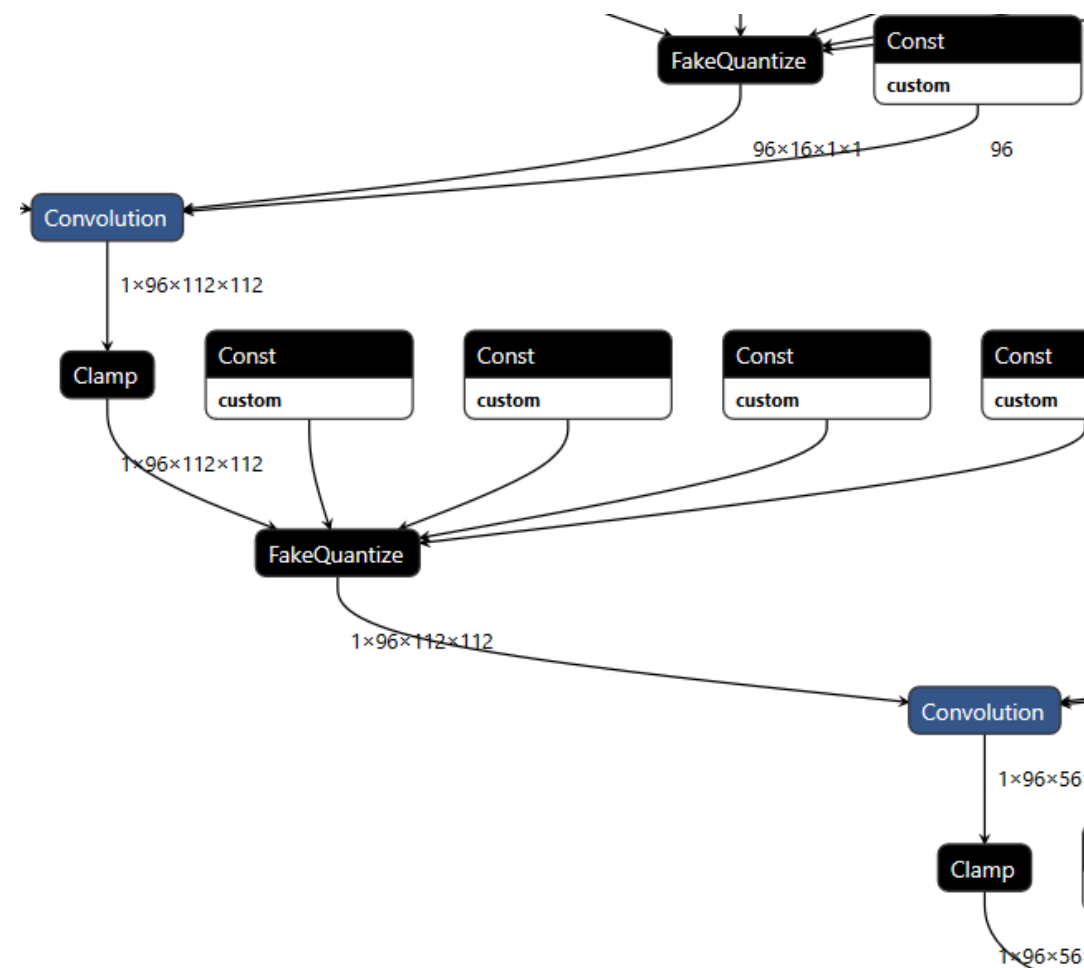
Quantization: Workflow

Step1: Insert Fake Quantization Layers

Step2: Choose optimal quantization parameters (precision, type, granularity)

Step3: Initialize fake quantization layer parameters

Step4: Optimization

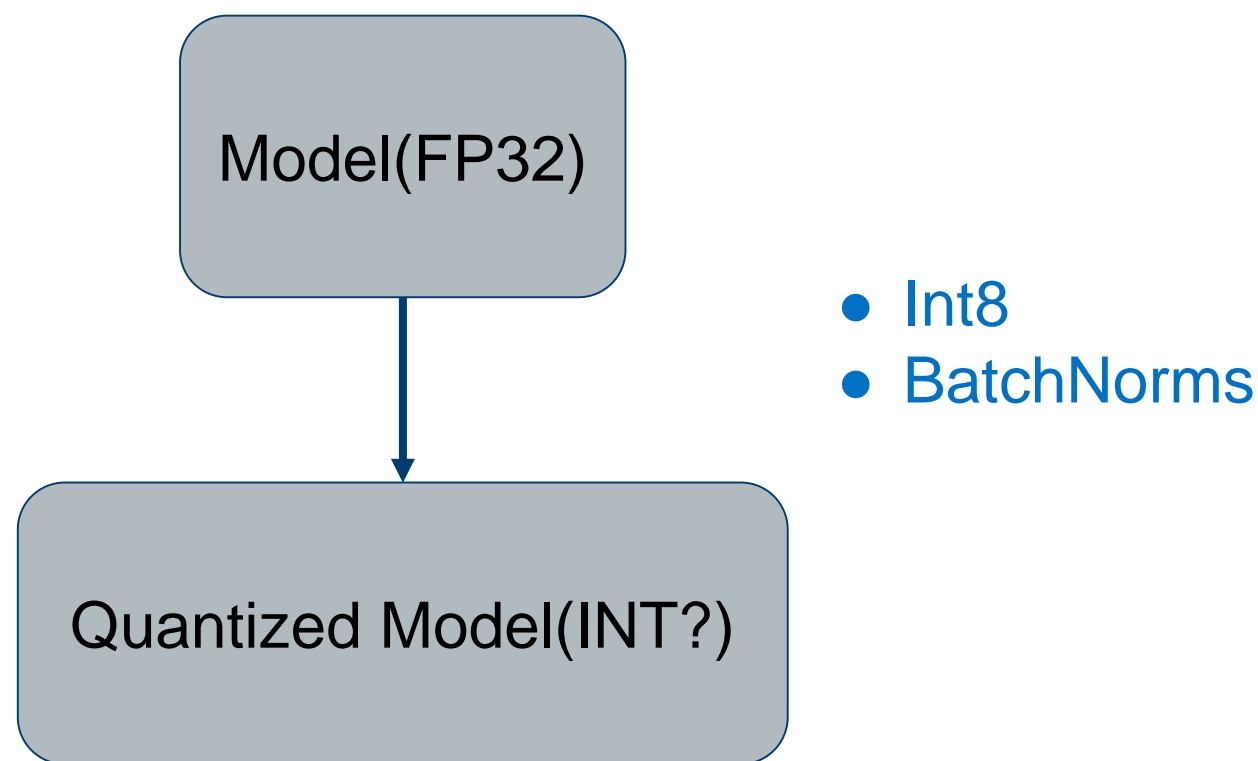


Quantization algorithms

- Post-Training Quantization without dataset (Data Free Quantization)
- Post-Training Quantization with dataset
- Quantization Aware Training

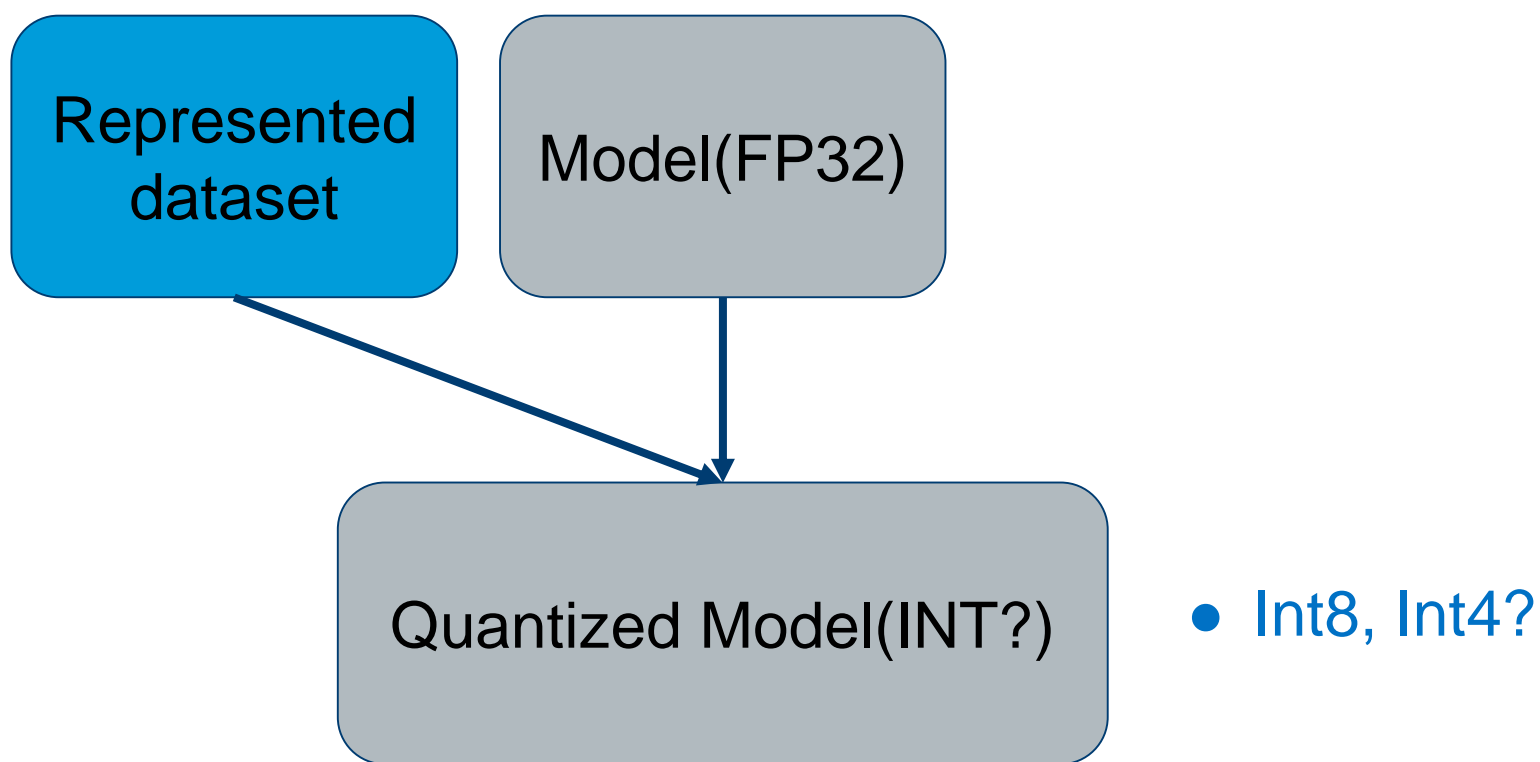
Quantization algorithms: Usage scenario

- Level 1: Post-Training Quantization w/o dataset



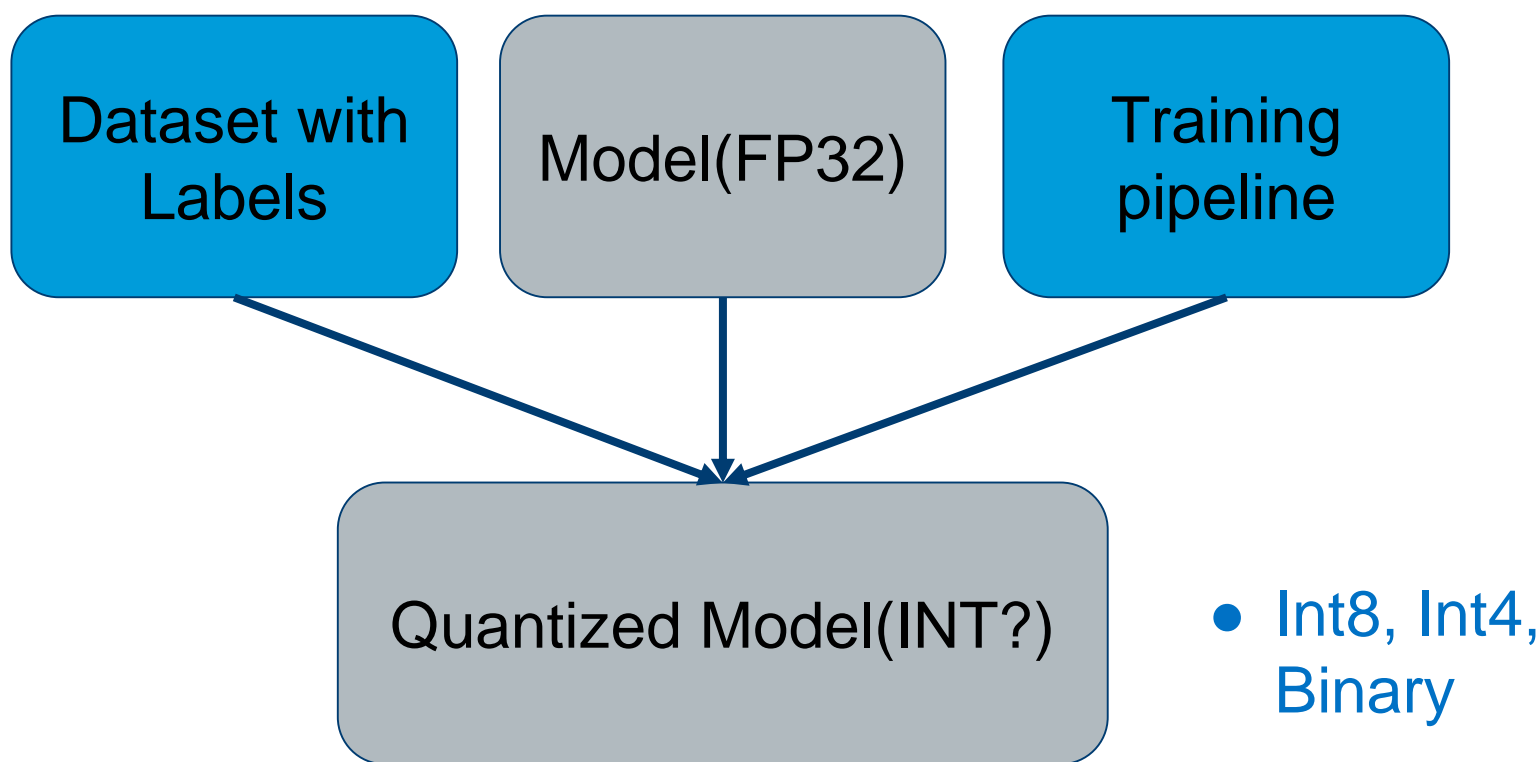
Quantization algorithms: Usage scenario

- Level 2: Post-Training Quantization w dataset



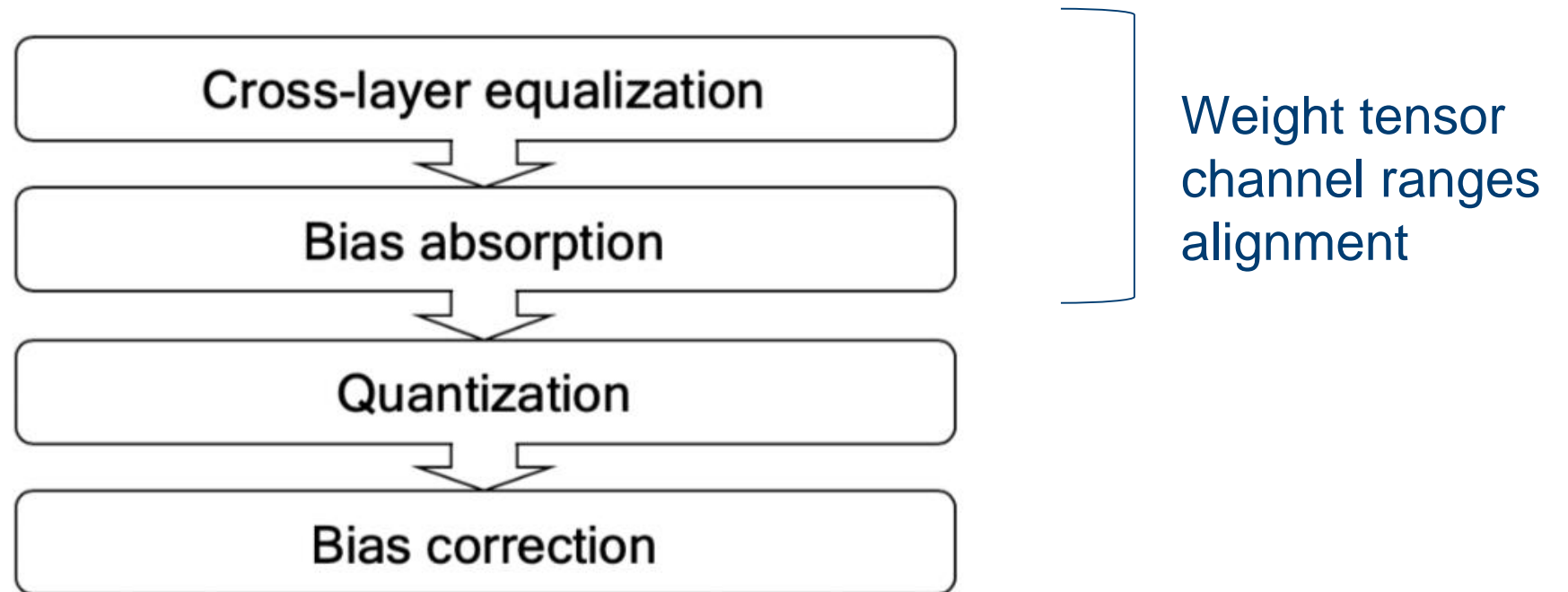
Quantization algorithms: Usage scenario

- Level 3: Quantization aware training



Data-Free Quantization (Level 1)

- The basic idea is to use BatchNorm statistics to estimate the range of activations.
- Algorithm Flow:



Quantization with represented dataset (Level 2)

- Weights:
 - $\text{input_low} = \min(W)$
 - $\text{Input_high} = \max(W)$
- Activations:
 - Run N examples through the FP32 model and collect for each layer the per-channel pre-activation statistics:
 - moving average of the minimum and maximum values across batches to determine the quantizer parameters for activations.
 - Our observations show that the use of robust statistic boosts accuracy metric (Hodges-Lehmann estimator)

Bias correction

It is iterative procedure. This procedure is ran on a network with quantized weights only:

1. Run N examples through the FP32 model and collect for each layer the per-channel pre-activation means $E[y]$.
2. For each layer L in the quantized model:
 - Collect the per-channel pre-activation means $E[\bar{y}]$ of layer L for the same N examples as in step 1.
 - Compute the per-channel biased quantization error $E[\epsilon] = E[\bar{y}] - E[y]$
 - Subtract $E[\epsilon]$ from the layer's bias parameter

Quantization via training (Level 3)

Algorithm:

1. Create a training graph of the floating-point model.
2. Insert fake quantization operations in locations where tensors will be downcasted to fewer bits during inference.
3. Train in simulated quantized mode until convergence.
4. Create and optimize the inference graph for running in a low bit inference engine.
5. Run inference using the quantized inference graph.

Summary: Quantization approaches

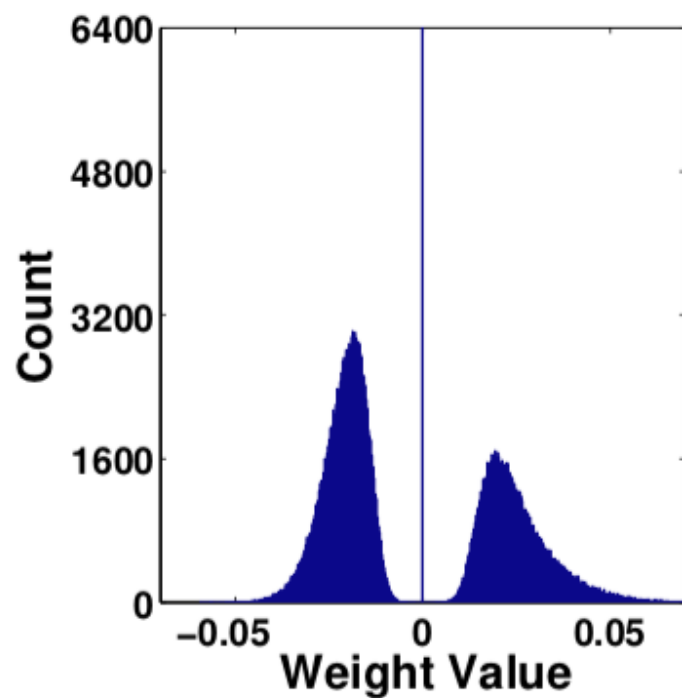
| | ~D | ~BP | ~AC | MobileNetV2 | | MobileNetV1 | | ResNet18 | | |
|--------------------|----|-----|-----|-------------|--------------|-------------|--------------|----------|---------------------------|--------------------|
| | | | | FP32 | INT8 | FP32 | INT8 | FP32 | INT8 | INT6 |
| DFQ (ours) | ✓ | ✓ | ✓ | 71.7% | 71.2% | 70.8% | 70.5% | 69.7% | 69.7% | 66.3% |
| Per-layer [18] | ✓ | ✓ | ✓ | 71.9% | 0.1% | 70.9% | 0.1% | 69.7% | 69.2%* | 63.8%* |
| Per-channel [18] | ✓ | ✓ | ✓ | 71.9% | 69.7% | 70.9% | 70.3% | 69.7% | 69.6%* | 67.5%* |
| QT [16] ^ | ✗ | ✗ | ✓ | 71.9% | 70.9% | 70.9% | 70.0% | - | 70.3% [†] | 67.3% [†] |
| SR+DR [†] | ✗ | ✗ | ✓ | - | - | - | 71.3% | - | 68.2% | 59.3% |
| QMN [31] | ✗ | ✗ | ✗ | - | - | 70.8% | 68.0% | - | - | - |
| RQ [21] | ✗ | ✗ | ✗ | - | - | - | 70.4% | - | 69.9% | 68.6% |

Table 5. Top1 ImageNet validation results for different models and quantization approaches. The top half compares level 1 approaches (~D: data free, ~BP: backpropagation-free, ~AC: Architecture change free) whereas in the second half we also compare to higher level approaches in literature. Results with * indicates our own implementation since results are not provided, ^ results provided by [18] and [†] results from table 2 in [21].

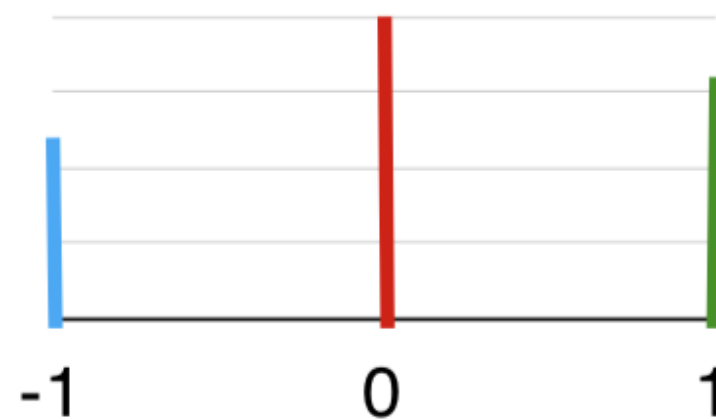
Algorithms for Efficient Inference

1. Pruning
2. Weight Sharing
3. Quantization
- 4. Binary / Ternary Net**
5. Distillation
6. Low Rank Approximation
7. Winograd Transformation

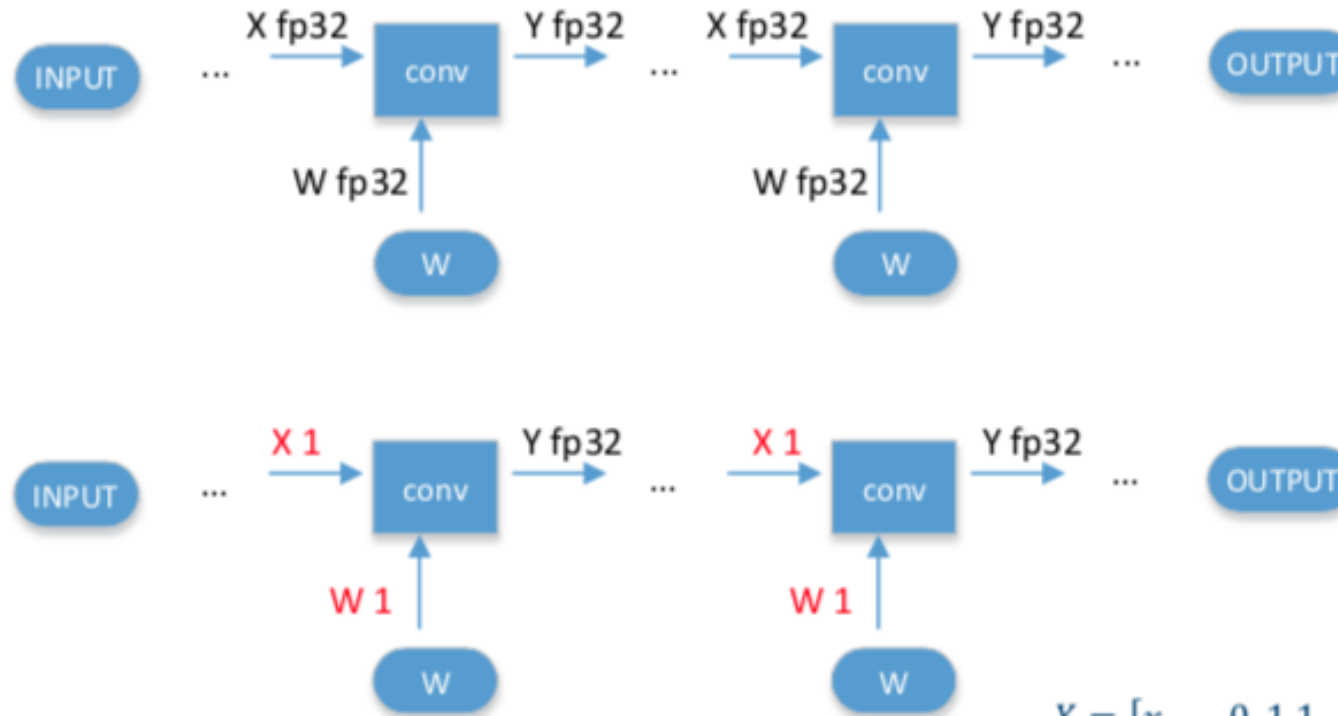
Binary / Ternary Net



\Rightarrow



Binary Net



1 value inference in conv is dot product

$$y_1 = w_1 * x_1 + w_2 * x_2 + \dots + w_{32} * x_{32}$$

32 * and 32+ operations

$$y_1 = \{-1, +1\} * \{-1, +1\} + w_2 * x_2 + \dots$$

$$x_i * w_i = \begin{matrix} x \backslash w & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & +1 \end{matrix} \quad \{-1, +1\},$$

$$a \text{ XNOR } b = \begin{matrix} a \backslash b & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix} \quad \{0, 1\},$$

$X = [x_1, \dots, 0, 1, 1, \dots, x_{32}], W = [w_1, \dots, w_{32}] \rightarrow \text{int32 number}$

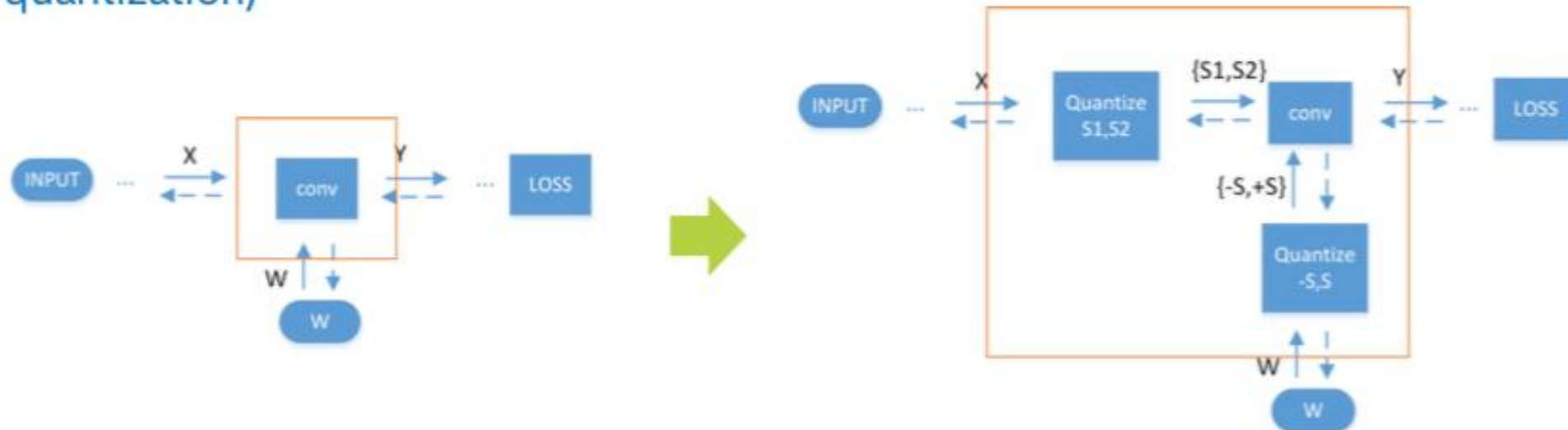
Calc $y = 2 * \text{popcount}(W \text{ XNOR } X) - 32$

We have only 1 XNOR and 1 POPCOUNT operations

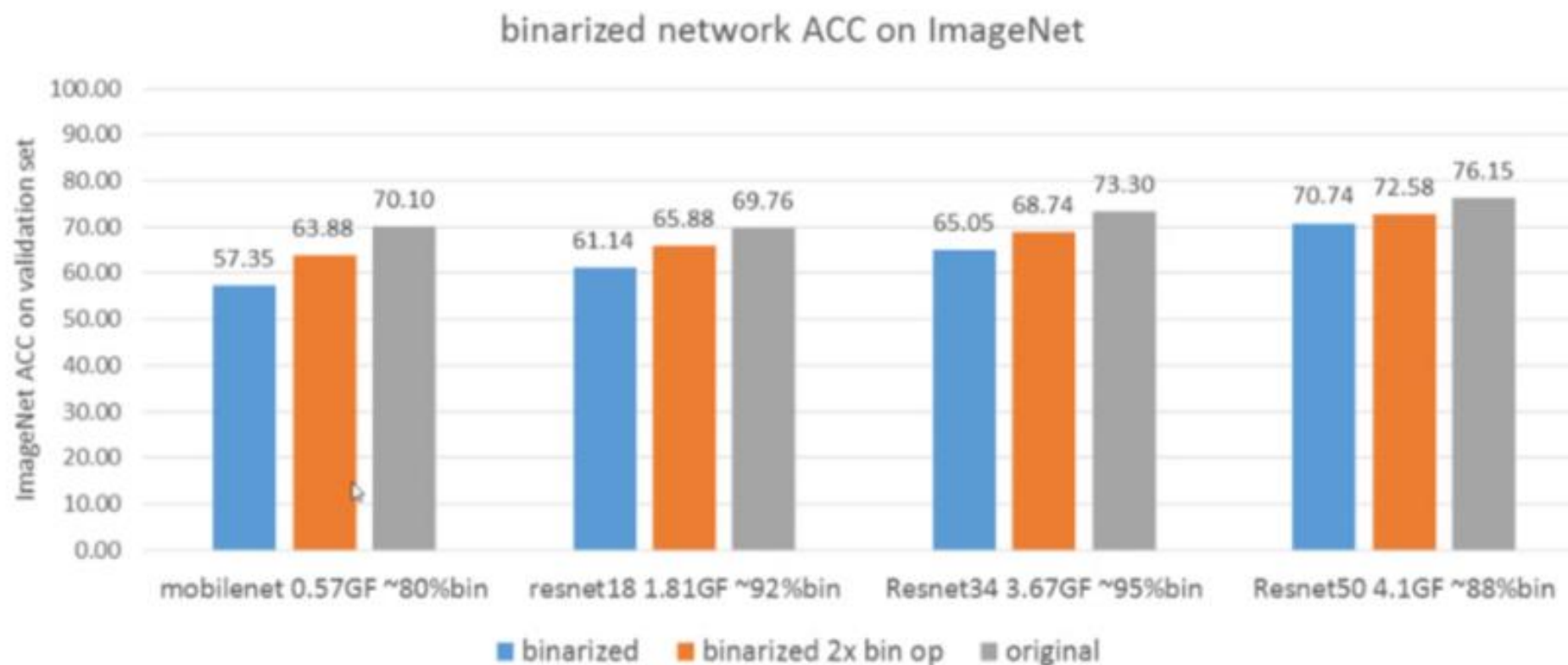
Binary Net

During binarization process selected convolutional layers of the original CNN are replaced with binary convolution alternatives.

Network is modified by inserting special quantization layer for input activations and weights that converts any full-precision value into two pretrained values (so-called “fake” quantization)



Binary Net



Algorithms for Efficient Inference

1. Pruning
2. Weight Sharing
3. Quantization
4. Binary / Ternary Net
- 5. Distillation**
6. Low Rank Approximation
7. Winograd Transformation

Distillation

- **L2 – Ba [14]**
 - L2 loss between teacher and student logits
 - No labels required

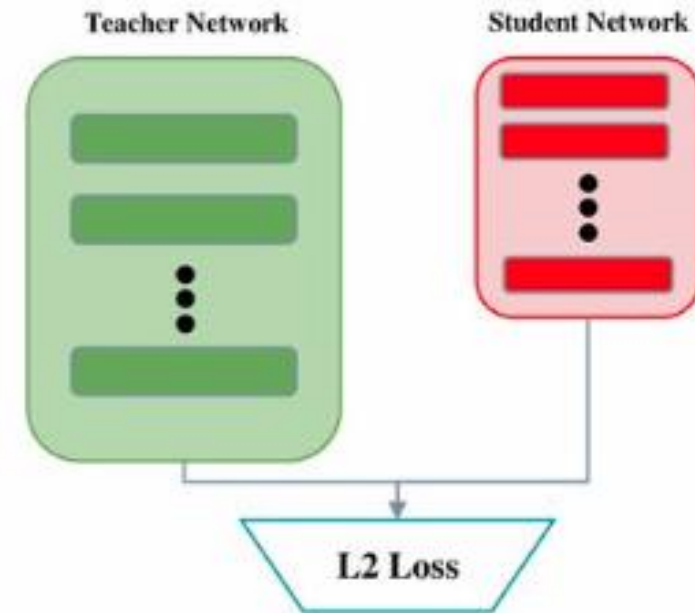


Figure 6. Teacher-Student model, L2

- [14] J. Ba and R. Caruana, "Do deep nets really need to be deep?" In Advances in neural information processing systems, 2014, pp. 2654–2662.
- [15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," ArXiv preprint arXiv:1503.02531, 2015.
- [16] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," ArXiv preprint arXiv:1412.6550, 2015.

Distillation

- L2 – Ba [14]
 - L2 loss between teacher and student logits
 - No labels required
- **Knowledge Distillation [15]**
 - Soft target: softmax cross entropy with teacher logits
 - Hard target: softmax cross entropy with correct labels

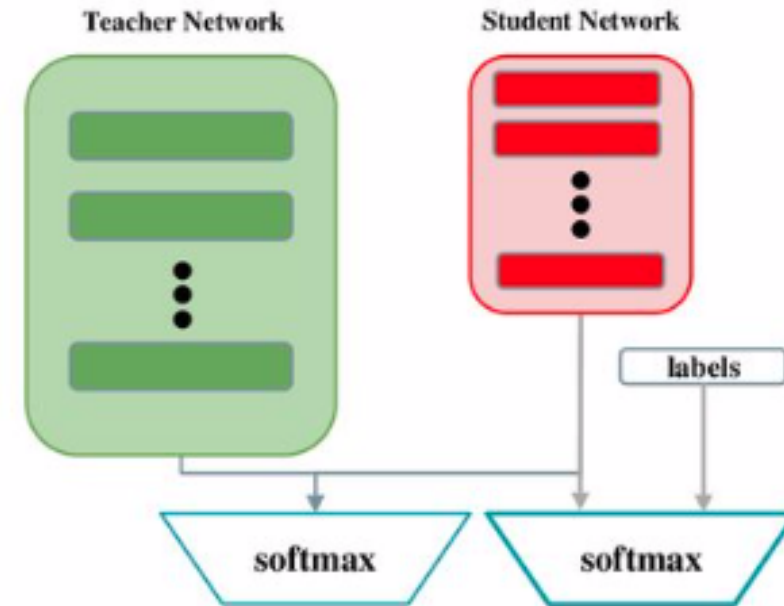


Figure 7. Teacher-Student model, KD

[14] J. Ba and R. Caruana, "Do deep nets really need to be deep?" In Advances in neural information processing systems, 2014, pp. 2654–2662.

[15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," ArXiv preprint arXiv:1503.02531, 2015.

[16] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," ArXiv preprint arXiv:1412.6550, 2015.

Distillation

- L2 – Ba [14]
 - L2 loss between teacher and student logits
 - No labels required
- Knowledge Distillation [15]
 - Soft target: softmax cross entropy with teacher logits
 - Hard target: softmax cross entropy with correct labels
- **FitNets [16]**
 - Knowledge Distillation with hints in the middle points of the network
 - Student is deeper than the teacher

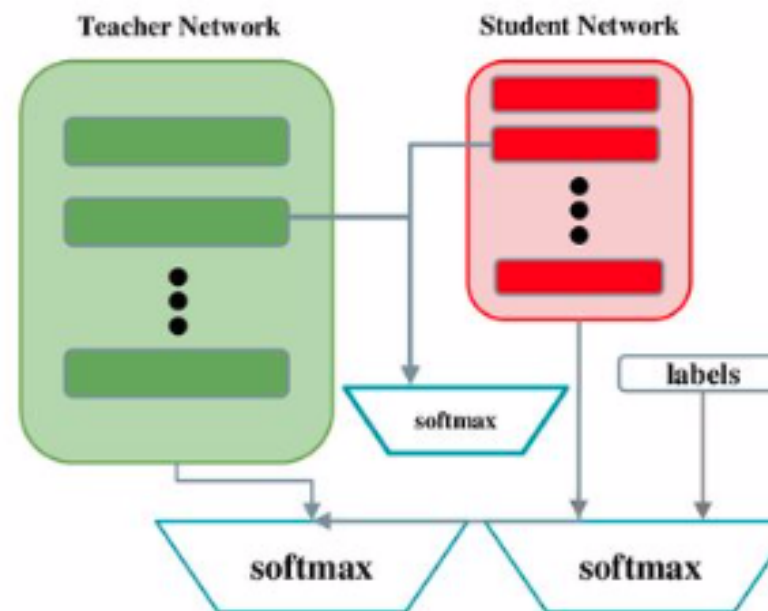


Figure 8. Teacher-Student model, FitNets

- [14] J. Ba and R. Caruana, "Do deep nets really need to be deep?" In *Advances in neural information processing systems*, 2014, pp. 2654–2662.
- [15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *ArXiv preprint arXiv:1503.02531*, 2015.
- [16] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *ArXiv preprint arXiv:1412.6550*, 2015.

Algorithms for Efficient Inference

1. Pruning
2. Weight Sharing
3. Quantization
4. Binary / Ternary Net
5. Distillation
- 6. Low Rank Approximation**
7. Winograd Transformation

Low Rank Approximation

- Basis filter set => Basis feature maps
- Final feature map = linear combination of basis feature maps
- Rank-1 basis filter => decomposed into a sequence of horizontal and vertical filters
- ~2.4x speedup, no performance drop

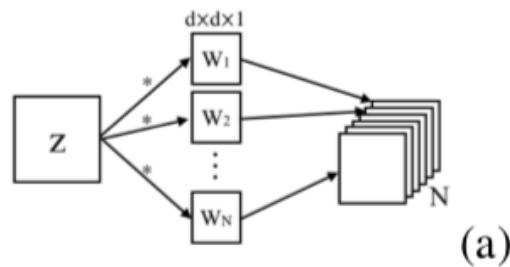
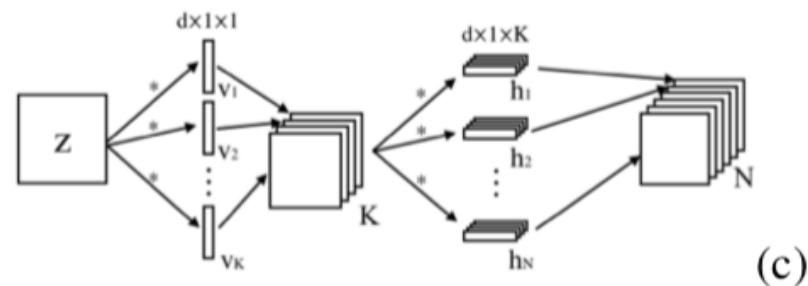
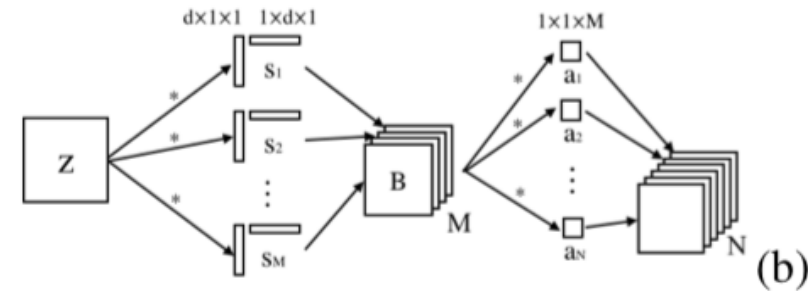


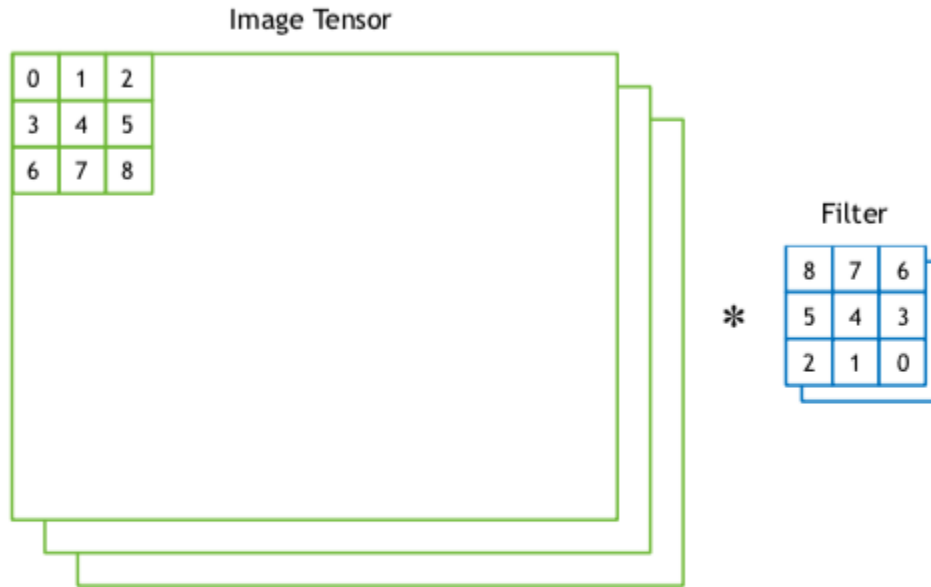
Figure 1: (a) The original convolutional layer acting on a single-channel input *i.e.* $C=1$. (b) The approximation to that layer using the method of Scheme 1. (c) The approximation to that layer using the method of Scheme 2. Individual filter dimensions are given above the filter layers.



Algorithms for Efficient Inference

1. Pruning
2. Weight Sharing
3. Quantization
4. Binary / Ternary Net
5. Distillation
6. Low Rank Approximation
7. Winograd Transformation

Winograd Transformation



9xC FMAs/Output: Math Intensive

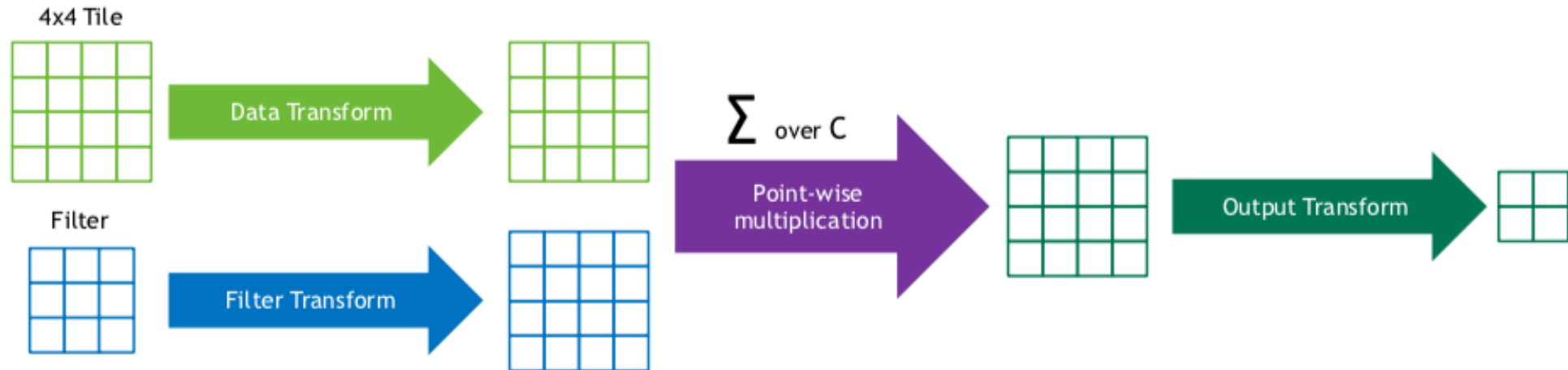
$$\sum \begin{aligned} & 0 \times 8 + 1 \times 7 + 2 \times 6 + \\ & 3 \times 5 + 4 \times 4 + 5 \times 3 + \\ & 6 \times 2 + 7 \times 1 + 8 \times 0 \end{aligned}$$

9xK FMAs/Input: Good Data Reuse

| | | |
|--------------|--------------|--------------|
| 4×0 | 4×1 | 4×2 |
| 4×3 | 4×4 | 4×5 |
| 4×6 | 4×7 | 4×8 |

Direct convolution: we need $9 \times C \times 4 = 36 \times C$ FMAs for 4 outputs

Winograd Transformation



Direct convolution: we need $9 \times C \times 4 = 36 \times C$ FMAs for 4 outputs

Winograd convolution: we need $16 \times C$ FMAs for 4 outputs: **2.25x** fewer FMAs

Summary

| Method | Advantages | Disadvantages |
|-----------------------------|--|--|
| Binarization & Quantization | Low latency and memory usage | High loss of accuracy |
| Pruning | Prevents overfitting, the accuracy can increase | Slow |
| Factorization | Can achieve state of the art results while decreasing the computation cost | Dependent on framework |
| Distillation | Applicable to all architectures Doesn't change the network | only applicable to classification task |

Thank you for your
attention!

Cross-Layer Equalization

Given two layers, $\mathbf{x}_1 = f(\mathbf{W}_1 \mathbf{x}_0 + \mathbf{b}_1)$ and $\mathbf{x}_2 = f(\mathbf{W}_2 \mathbf{x}_1 + \mathbf{b}_2)$ through scaling invariance we have that:

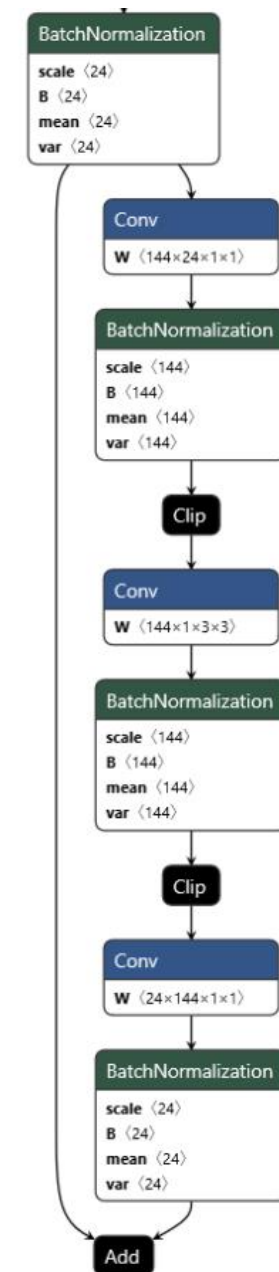
$$\begin{aligned}\mathbf{x}_2 &= f(\mathbf{W}_2 f(\mathbf{W}_1 \mathbf{x}_0 + \mathbf{b}_1) + \mathbf{b}_2) \\ &= f(\mathbf{W}_2 \mathbf{S} \hat{f}(\mathbf{S}^{-1} \mathbf{W}_1 \mathbf{x}_0 + \mathbf{S}^{-1} \mathbf{b}_1) + \mathbf{b}_2) \\ &= f(\widehat{\mathbf{W}}_2 \hat{f}(\widehat{\mathbf{W}}_1 \mathbf{x}_0 + \widehat{\mathbf{b}}_1) + \mathbf{b}_2)\end{aligned}$$

where:

$$s_i = \frac{1}{r_i^{(1)}} \sqrt{r_i^{(1)} r_i^{(2)}} \quad ; \quad r_i^{(1)} = \underline{2} \cdot \max_j |\widehat{\mathbf{W}}_{ij}^{(1)}|$$

Cross-Layer Equalization

Cross-Layer Equalization procedure is applied to a convolution sequence without branching



Bias Absorption

Running the equalization procedure on the weights potentially increases the biases of the layers.

$$\mathbf{x}_i = \mathbf{W}_i \mathbf{x}_{i-1} + \mathbf{b}_i \quad (9)$$

$$= \mathbf{W}_i (r(\mathbf{W}_{i-1} \mathbf{x}_{i-2} + \mathbf{b}_{i-1}) + \mathbf{c} - \mathbf{c}) + \mathbf{b}_i \quad (10)$$

$$= \mathbf{W}_i (r(\mathbf{W}_{i-1} \mathbf{x}_{i-2} + \hat{\mathbf{b}}_{i-1}) + \mathbf{c}) + \mathbf{b}_i \quad (11)$$

$$= \mathbf{W}_i \hat{\mathbf{x}}_{i-1} + \hat{\mathbf{b}}_i \quad (12)$$

where $\hat{\mathbf{b}}_i = \mathbf{W}_i \mathbf{c} + \mathbf{b}_i$, $\hat{\mathbf{x}}_{i-1} = \mathbf{x}_{i-1} - \mathbf{c}$, and $\hat{\mathbf{b}}_{i-1} = \mathbf{b}_{i-1} - \mathbf{c}$.

Quantization

- Asymmetric per-channel quantization is used for weights and per-tensor for activations
- Weights:
 - $\text{input_low} = \min(W)$
 - $\text{input_high} = \max(W)$
- Activations:
 - $\text{input_low} = \text{mean} - 6 * \text{var}$
 - $\text{input_high} = \text{mean} + 6 * \text{var}$
 - relu, add, pooling, concat and etc propagate distribution parameters

Bias correction

$$\begin{aligned}\mathbb{E}[\mathbf{y}] &= \mathbb{E}[\mathbf{y}] + \mathbb{E}[\boldsymbol{\epsilon}\mathbf{X}] - \mathbb{E}[\boldsymbol{\epsilon}\mathbf{X}] \\ &= \mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\boldsymbol{\epsilon}\mathbf{X}]\end{aligned}$$

$$\boldsymbol{\epsilon} = \tilde{\mathbf{W}} - \mathbf{W}$$

$$\tilde{\mathbf{y}} = \mathbf{y} + \boldsymbol{\epsilon}\mathbf{X}$$

- \mathbf{x} - input FP32
- \mathbf{W} - weights FP32
- $\tilde{\mathbf{W}}$ - quantized weights

Bias correction

- Computing the expected input:

$$\begin{aligned}\mathbb{E}[\mathbf{x}_c] &= \mathbb{E}[\text{ReLU}(\mathbf{x}_c^{pre})] \\ &= \gamma_c \mathcal{N}\left(\frac{-\beta_c}{\gamma_c}\right) + \beta_c \left[1 - \Phi\left(\frac{-\beta_c}{\gamma_c}\right)\right]\end{aligned}$$

- Quantization bias:

$$\begin{aligned}[\boldsymbol{\epsilon} * \mathbb{E}[\mathbf{x}]]_{c_o i j} &= \sum_{c_i m n} \mathbb{E}[\mathbf{x}_{c_i, i-m, j-n}] \boldsymbol{\epsilon}_{c_o c_i m n} \\ &= \sum_{c_i} \left[\mathbb{E}[\mathbf{x}_{c_i}] \sum_{m n} \boldsymbol{\epsilon}_{c_o c_i m n} \right]\end{aligned}$$