

EE P 595 A: TinyML

Lecture 2: Model Compression for TinyML

Dept. of Electrical and Computer Engineering
University of Washington

Instructor: Dinuka Sahabandu (sdinuka@uw.edu)



ELECTRICAL & COMPUTER
ENGINEERING

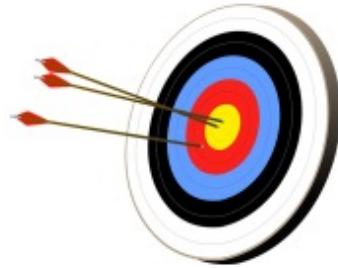
UNIVERSITY of WASHINGTON

3-Apr-24, EE P 595 A, Dinuka Sahabandu

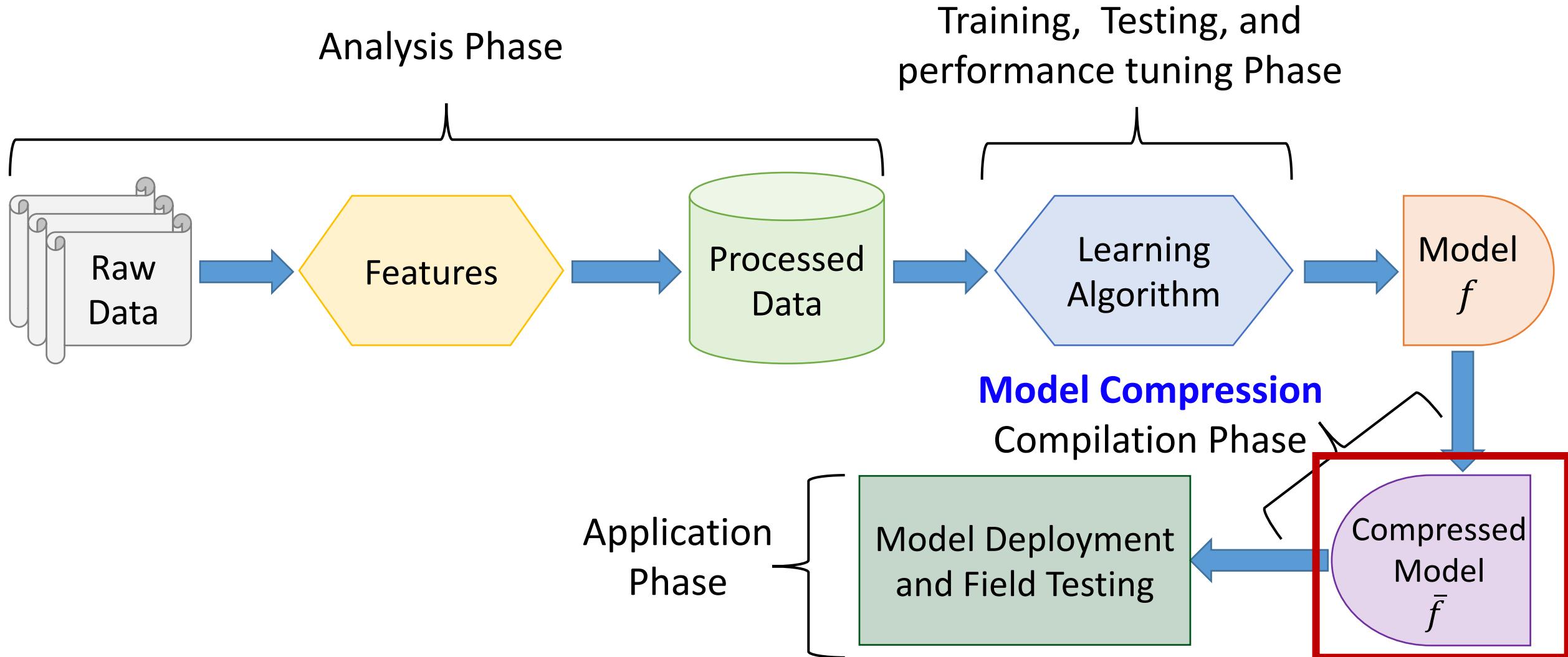


Topics Covered

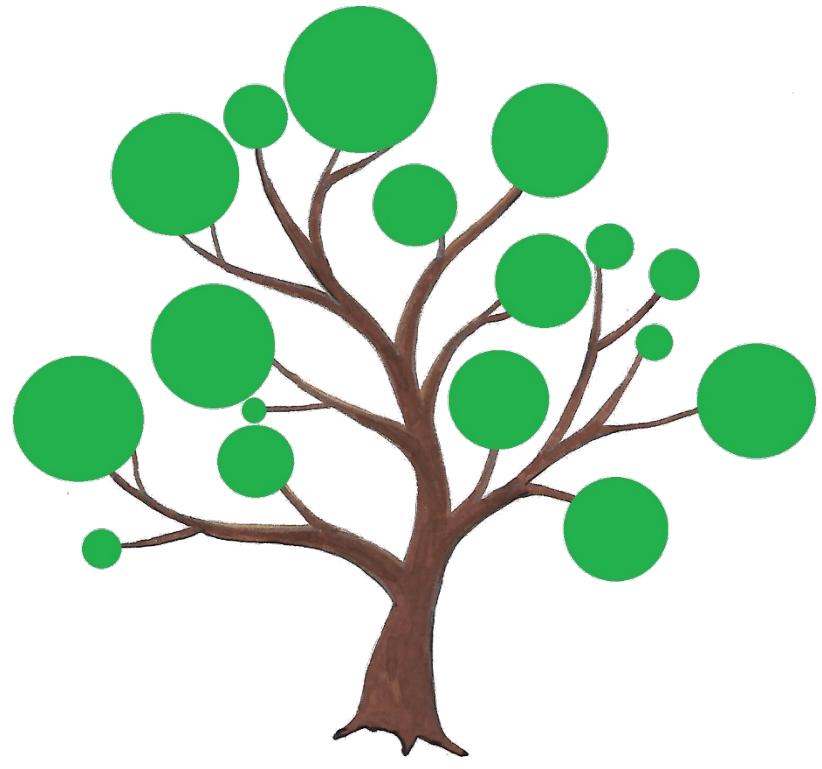
- Introduction of Different Model Compression Methods
- Model Quantization Techniques
- Model Pruning Techniques
- Knowledge Distillation for Model Compression
- Introduction to TensorFlow Lite
- Lab 2: Quantization and Pruning using TensorFlow / TensorFlow Lite



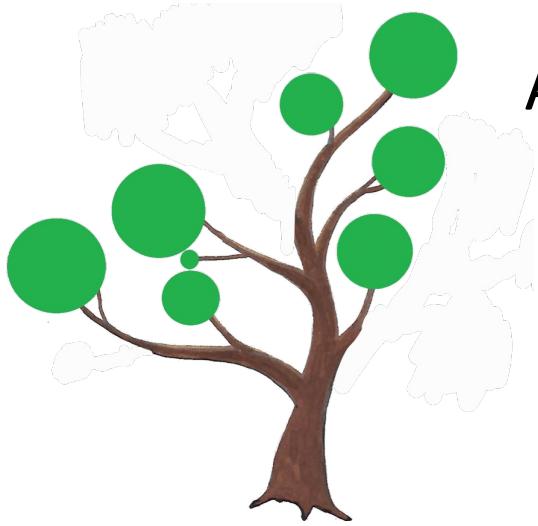
TinyML Model Compression Phase



Quantization and Pruning



A Tree



A Pruned Tree



A Quantized Tree



A Pruned and
Quantized Tree

Quantizing Image and Signal Inputs

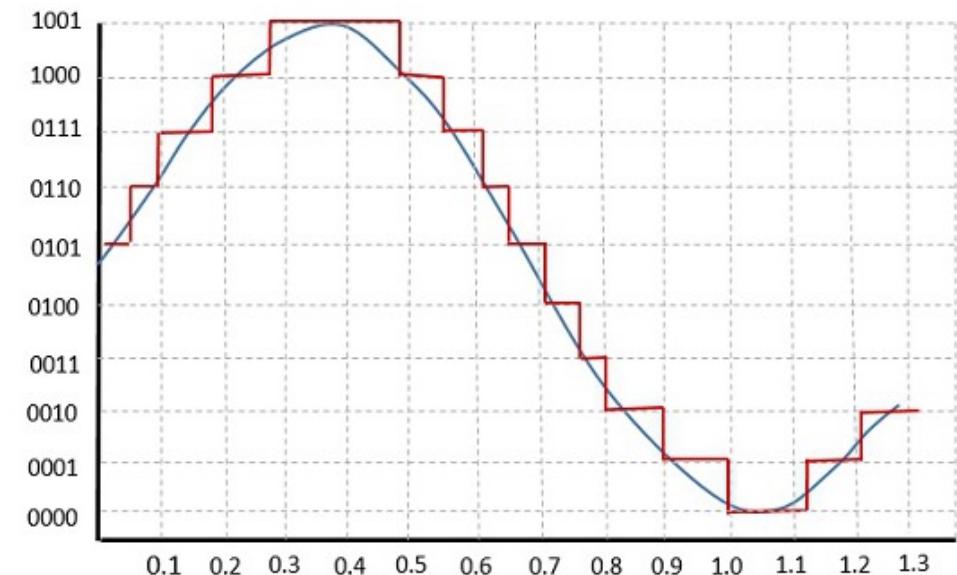
Original Image
Of Billion Colors



Quantized image with
four color Image

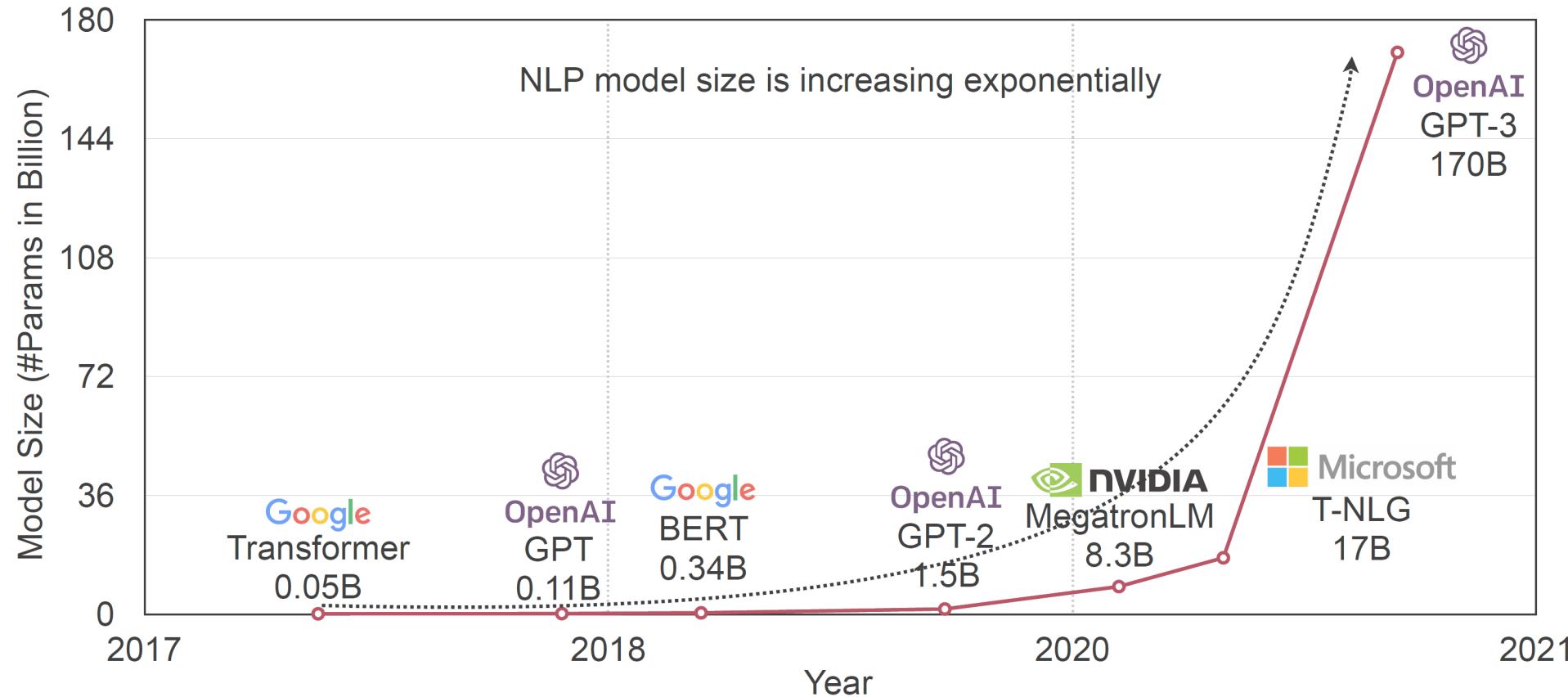


RGB (24 bits, 2^{24} colors) 2 bits, $2^2 = 4$ colors



Quantization in AI

- Model size and parameters of AI continue to growing fast!



Source: https://hanlab.mit.edu/projects/efficientnlp_old/

Quantization in AI

The Unreasonable Ineffectiveness of the Deeper Layers

Andrey Gromov*
Meta FAIR & UMD

Kushal Tirumala*
Meta FAIR

Hassan Shapourian
Cisco

Paolo Glorioso
Zyphra

Daniel A. Roberts
MIT & Sequoia Capital

Abstract

We empirically study a simple layer-pruning strategy for popular families of open-weight pretrained LLMs, finding minimal degradation of performance on different question-answering benchmarks until after a large fraction (up to half) of the layers are removed. To prune these models, we identify the optimal block of layers to prune by considering similarity across layers; then, to “heal” the damage, we perform a small amount of finetuning. In particular, we use parameter-efficient finetuning (PEFT) methods, specifically quantization and Low Rank Adapters (QLoRA), such that each of our experiments can be performed on a single A100 GPU. From a practical perspective, these results suggest that layer pruning methods can complement other PEFT strategies to further reduce computational resources of finetuning on the one hand, and can improve the memory and latency of inference on the other hand. From a scientific perspective, the robustness of these LLMs to the deletion of layers implies either that current pretraining methods are not properly leveraging the parameters in the deeper layers of the network or that the shallow layers play a critical role in storing knowledge.

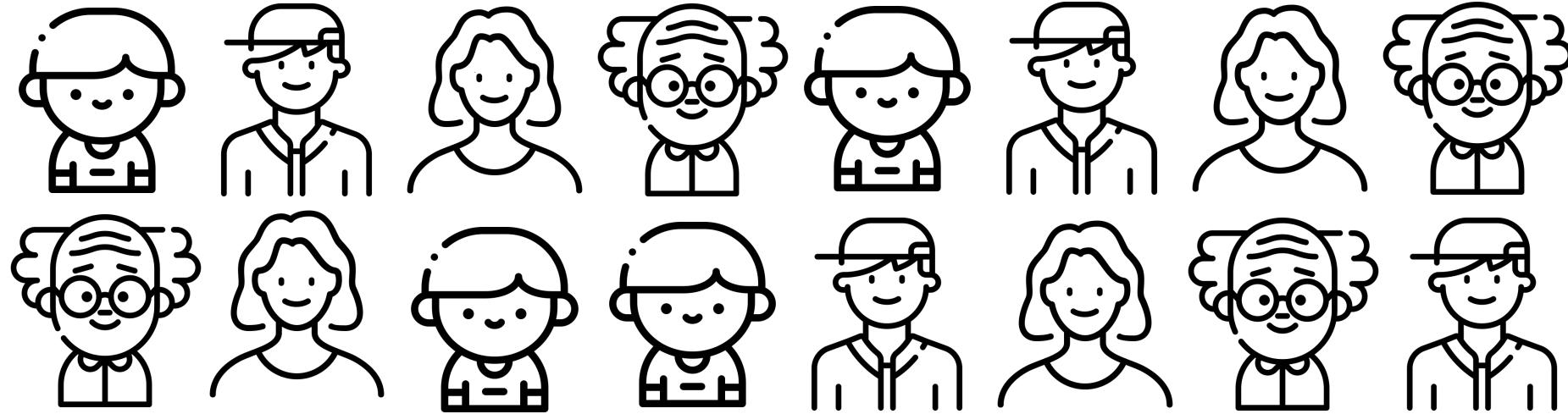
Quantization in ML

Quantization of ML offers significant benefits to TinyML!

Where and how?

- Memory usage:
8-bit versus 32-bit weights and activations stored in memory
- Power consumption:
Significant reduction in energy for both computations and memory access
- Latency:
With less memory access and simpler computations, latency can be reduced
- Silicon area:
Integer math or fewer bits requires less silicon area compared to floating point math and more bits

Quantization helps?



1820

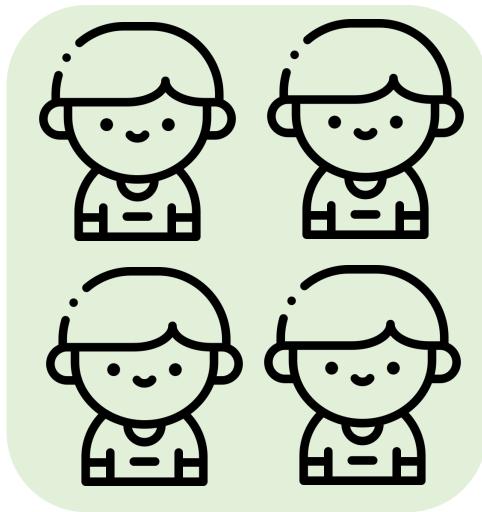


1920

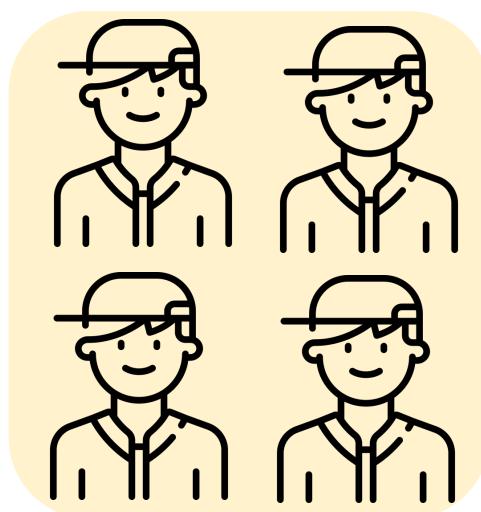


2020

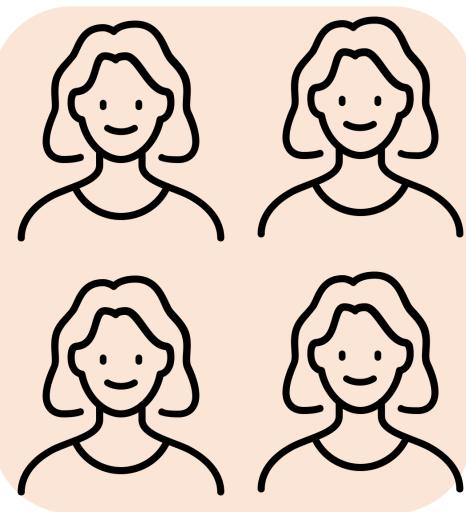
Quantization can help!



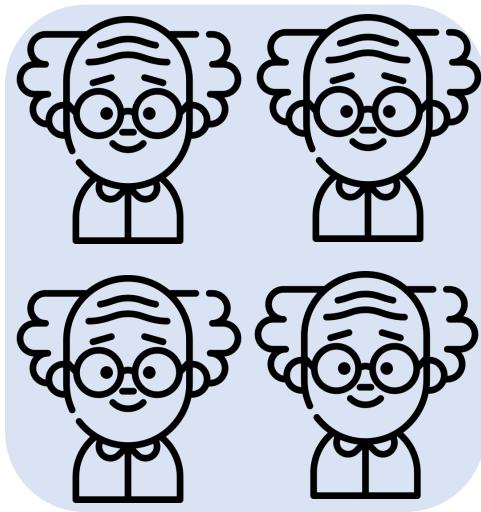
Children



Teenagers



Adults



Seniors



1820



1920



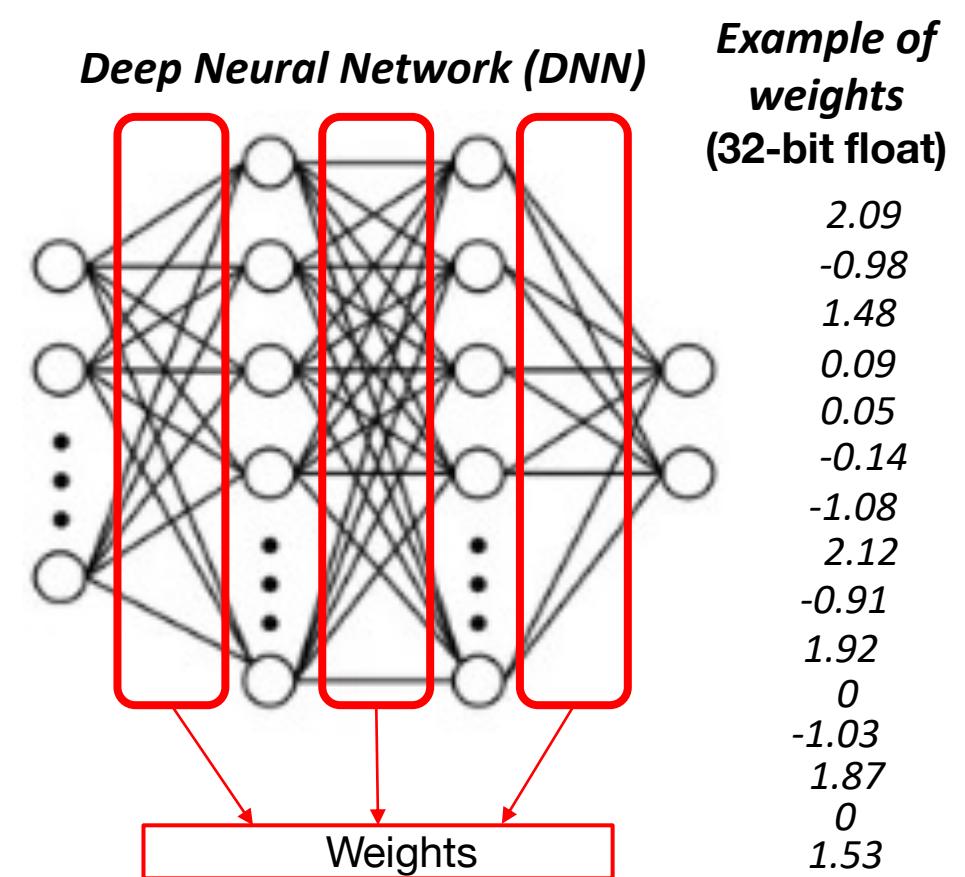
2020

Viewing Grouping or Clustering as Quantization

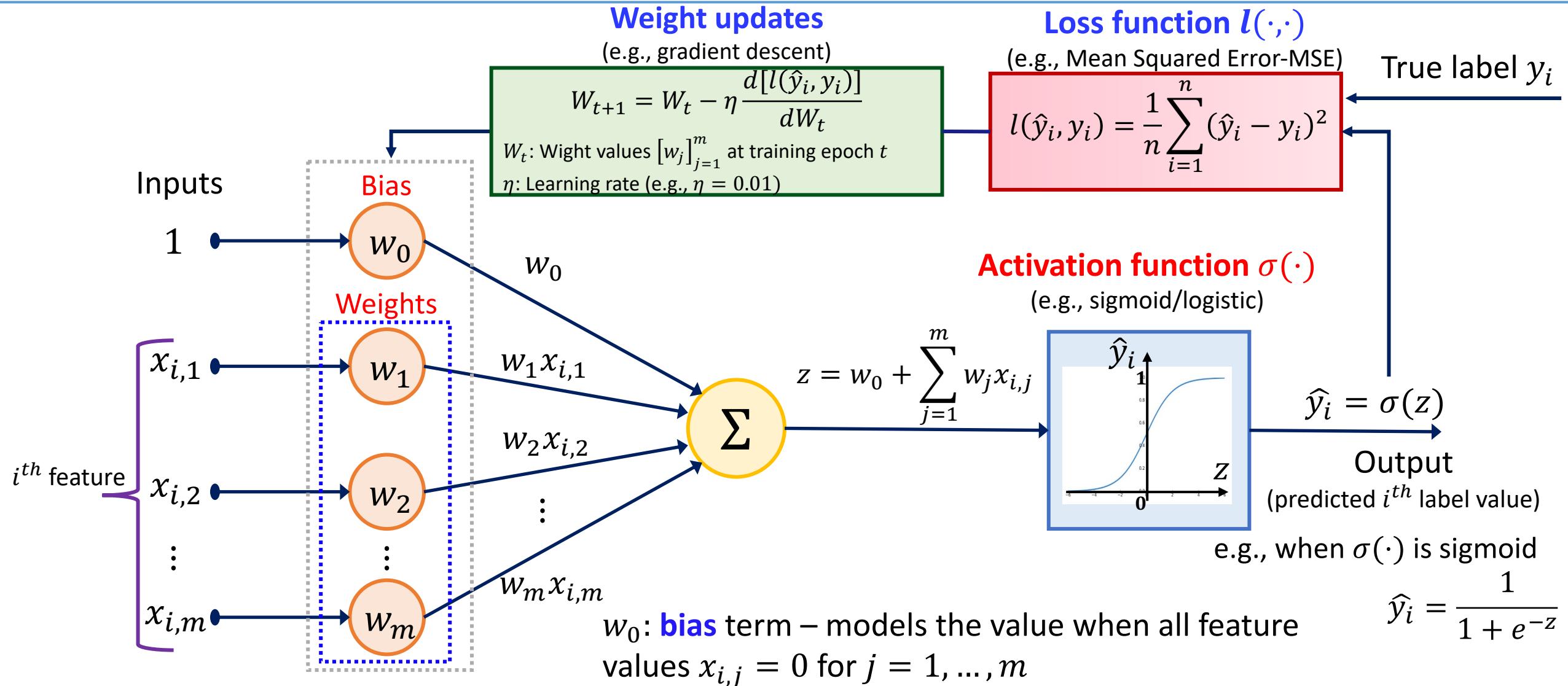


Viewing Grouping or Clustering as Quantization

- **Why do we want to quantize a DNN?**
To reduce the model size by reducing the complexity of representing model parameters
- **What parameters are we quantizing in DNN?**
Model weights (bias , activation function)

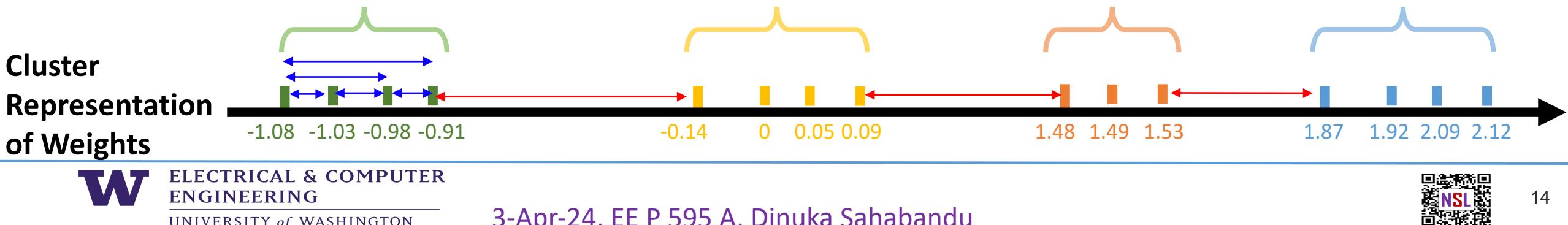


Structure of a Single Layer of Neural Network



Viewing Grouping or Clustering as Quantization

- **How do we quantize DNN model weights?**
Assign a single weight value to a cluster/group of weights **that are closer to each other**. You are doing approximation here (e.g., assign 1.5 for all three values 1.4, 1.5, 1.6)
- **How to decide which weights are closer to each other and form a cluster?** Relative distance between each pair of model weights can be used as a metric/measure to form clusters.
Relative distance between two weights w_1 and w_2 is computed using $| w_1 - w_2 |$
e.g., relative distant between -1.08 and -1.03 is $|-1.08 - -1.03| = |-1.08 + 1.03| = 0.05$
- **How is relative distance used to find different clusters?**
Relatively closer data points stay in one cluster (**intra cluster distance**) while relatively distant data points stays in another cluster (**inter cluster distance**)



Viewing Grouping or Clustering as Quantization

- How is the number of clusters is determined?

Quantization specifications determine the number of clusters

e.g., 2-bit integer quantization requires 4 clusters

- What value is used to represent each weight inside a cluster?

Distinct index for each cluster

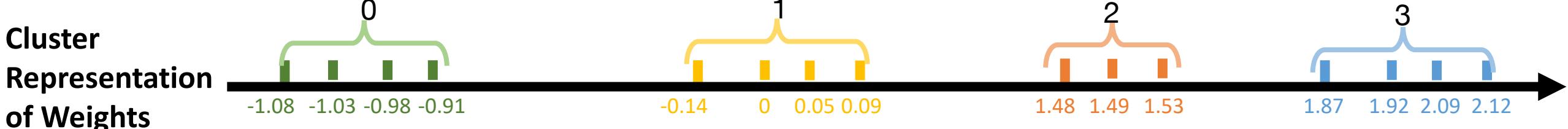
e.g., 2-bit integer quantization will have indices 0, 1, 2, 3

- What is being transferred to Tiny Hardware after quantization?

Code book that maps cluster index to cluster center

e.g., green cluster has mean value of $(-1.08 + -1.03 + -0.98 + -0.91)/4 = -1.00$

-1.08, -1.03, -0.98, -0.91, -0.41, 0, ... will be transferred as 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3



Cluster index (2-bit int)

Binary	Decimal
00	0
01	1
10	2
11	3

Cluster centers

0:	-1.00
1:	0.00
2:	1.50
3:	2.00

Codebook



Quantization

Quantization in Neural Networks

1. Theory in Quantization

- K-mean based quantization
- Linear quantization

2. Post-Training Quantization (PTQ)

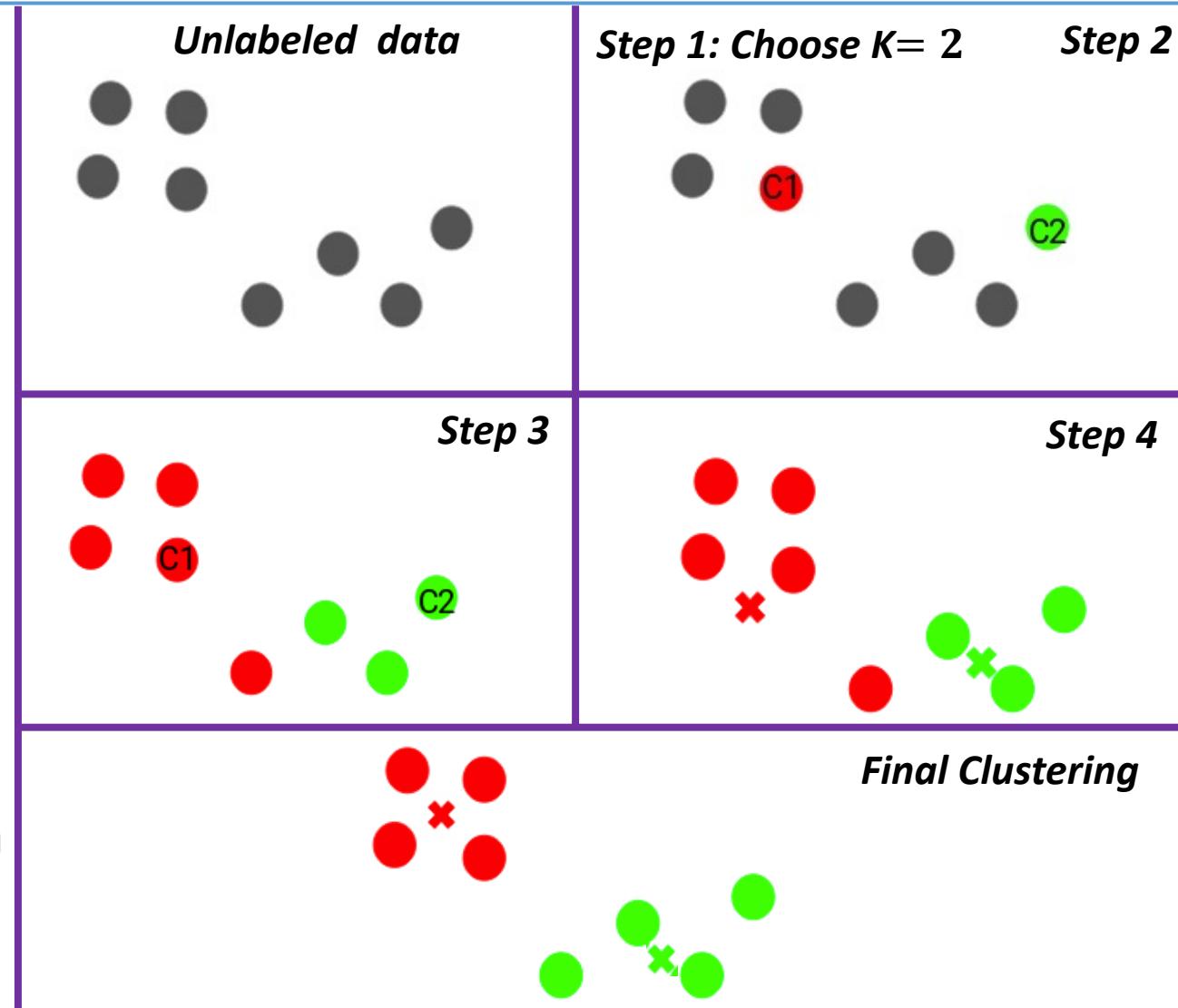
- Weight quantization
- Activation quantization
- Bias quantization

3. Quantization-Aware Training

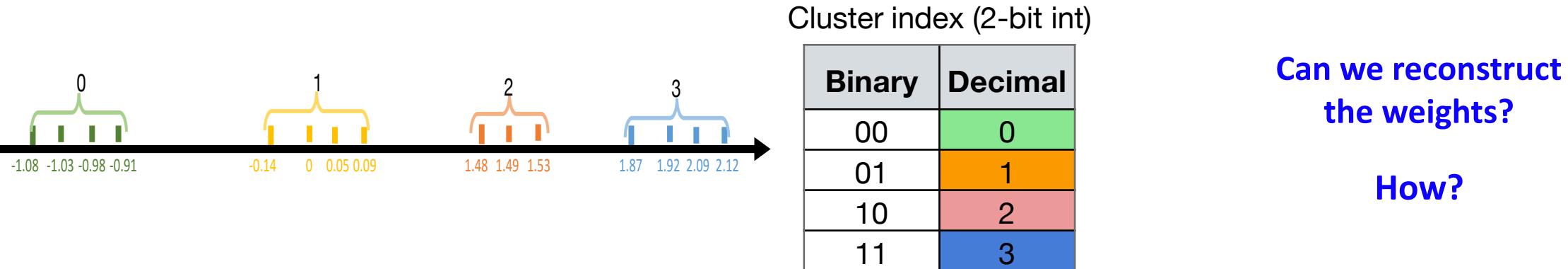
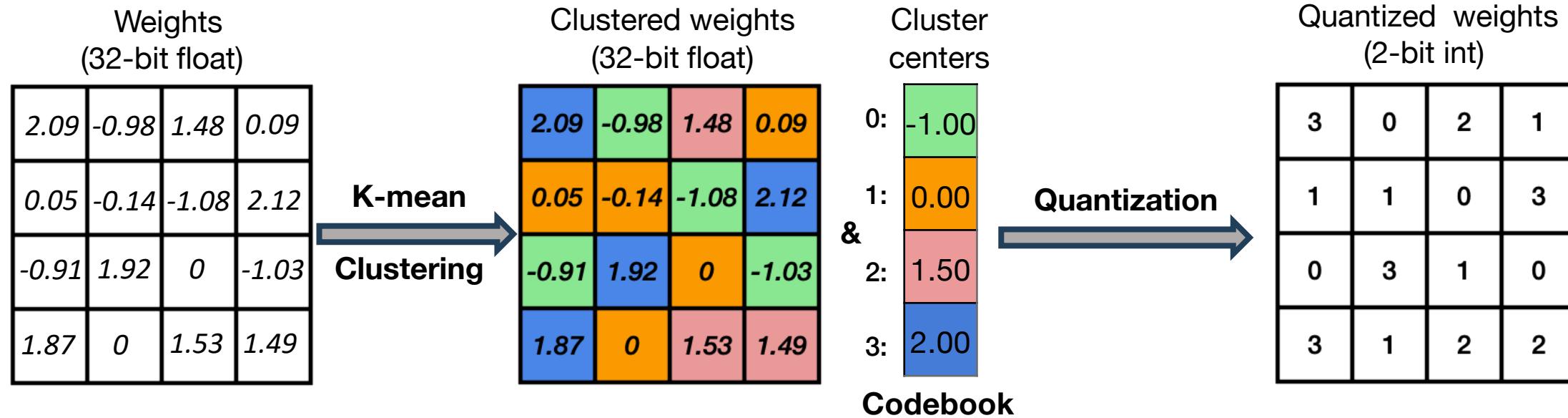


K-means Algorithm Steps Summary

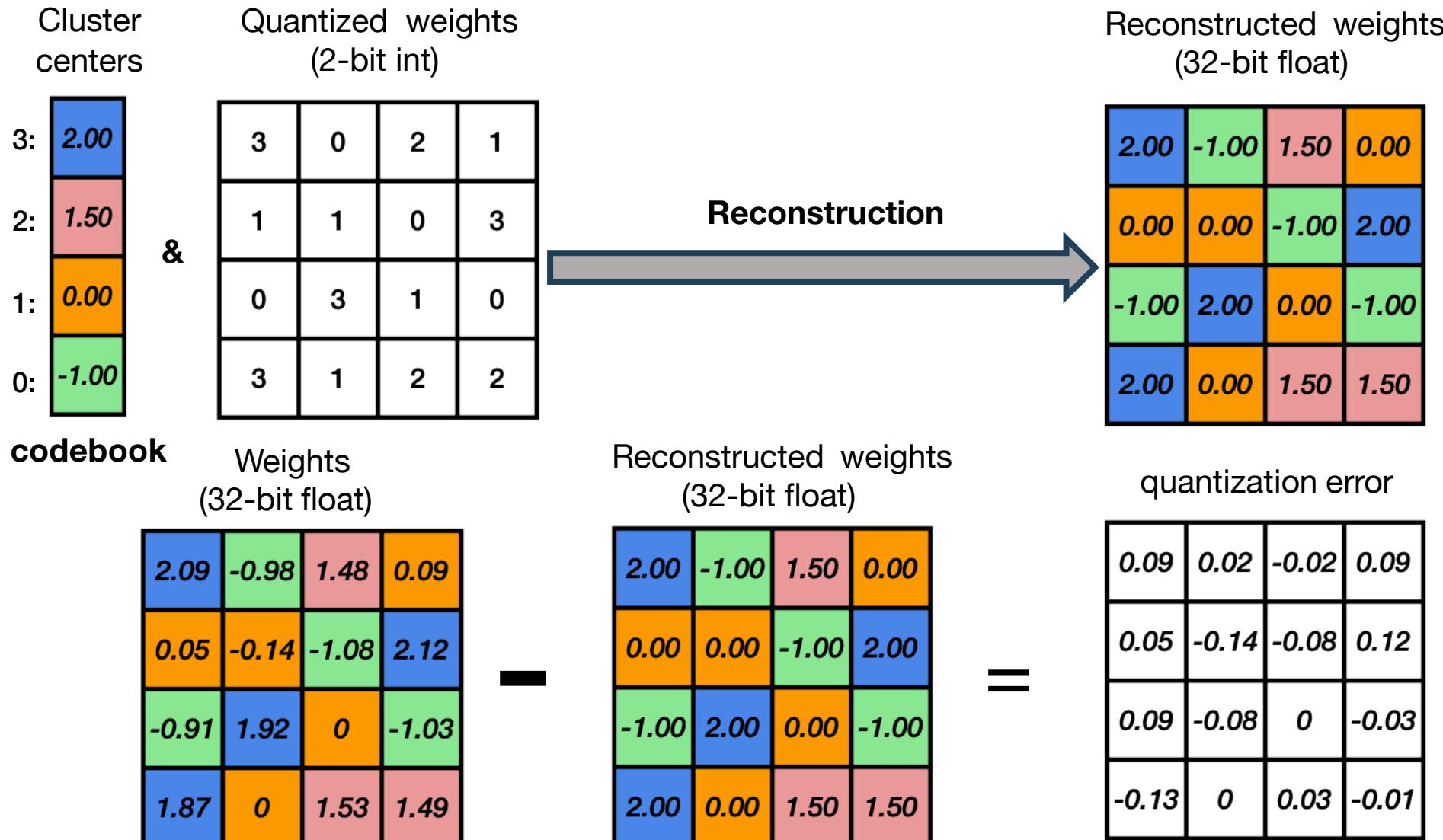
- **Step 1:** Specify number of clusters K
- **Step 2:** Initially randomly choose K data points and set them as centroids of K clusters
- **Step 3:** For each data point
 - Compute its Euclidean (l_2) distance from every centroid
 - Assign each data point to the cluster with nearest centroid
- **Step 4:** Within each cluster, take the average of all the data points and assign it as new centroid value
- **Step 5:** Repeat Step 3 and Step 4 until there is no change to the centroids/assignment of data points to clusters isn't changing



K-mean based Quantization



K-mean based Quantization: Reconstruction



K-mean based Quantization: Storage

Weights (32-bit float)			
2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Cluster centers	
0:	-1.00
1:	0.00
2:	1.50
3:	2.00

cluster index
(2-bit int)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

Codebook

$$\begin{aligned} & 32 \text{ bit} \times 16 \\ & = 512 \text{ bit} = 64 \text{ B} \end{aligned}$$

$$\begin{aligned} & 32 \text{ bit} \times 4 \\ & = 128 \text{ bit} = 16 \text{ B} \end{aligned} + \begin{aligned} & 2 \text{ bit} \times 16 \\ & = 32 \text{ bit} = 4 \text{ B} \end{aligned} = 20 \text{ B}$$

3.2 × smaller

Assume N -bit quantization, #parameters = $M \gg 2^N$.

$$\begin{aligned} & 32 \text{ bit} \times M \\ & = 32M \text{ bit} \end{aligned}$$

$$\begin{aligned} & N \text{ bit} \times M \\ & = NM \text{ bit} \end{aligned}$$

$32/N \times$ smaller



Quantization

Quantization in Neural Networks

1. Theory in Quantization

- K-mean based quantization
- Linear quantization

2. Post-Training Quantization (PTQ)

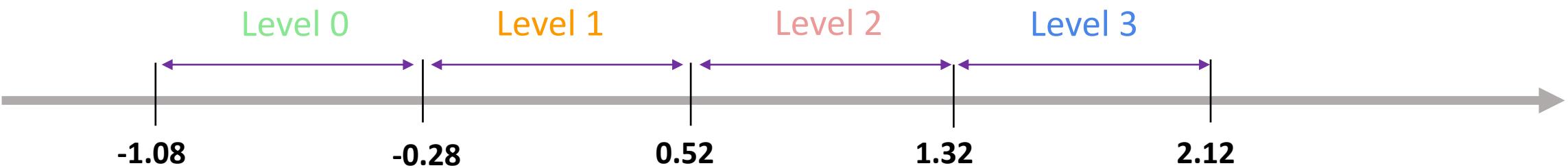
- Weight quantization
- Activation quantization
- Bias quantization

3. Quantization-Aware Training



Linear Quantization: Example

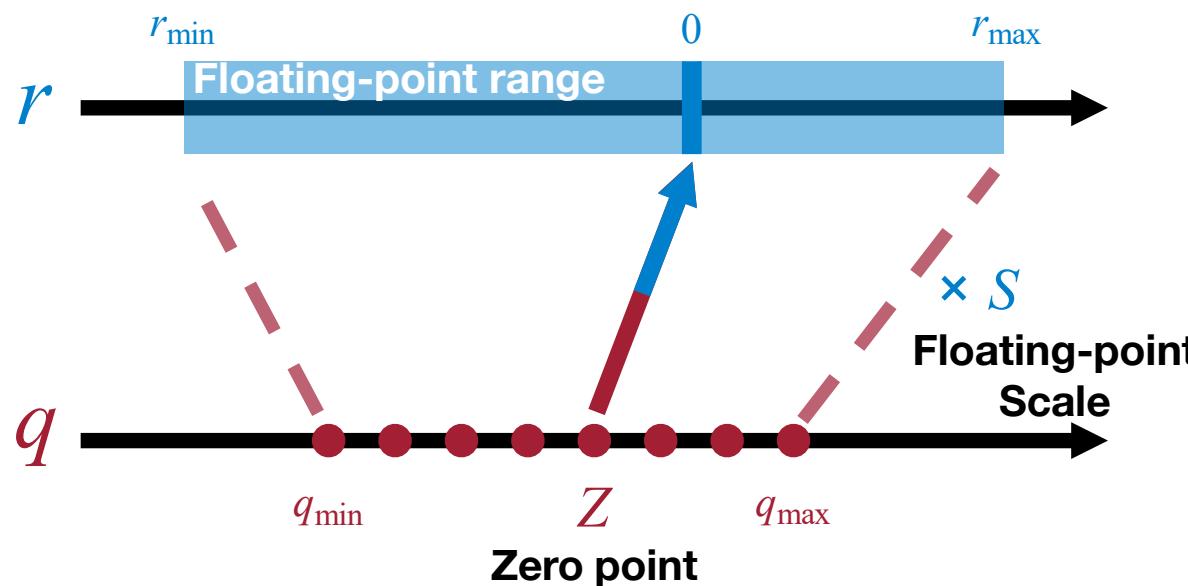
- **-1.08, -1.03, -0.98, -0.91, -0.41, 0, 0, 0.05, 0.09, 1.48, 1.49, 1.53, 1.87, 1.92, 2.09, 2.12**
- **Find the range:** Determine the minimum and maximum values. In this case, the minimum is -1.08 and the maximum is 2.12.
 $2.12 - (-1.08) = 3.2$
- **Determine the scale:** With 4 quantization levels (e.g., 2-bit), the range (-1.08 to 2.12) is divided into 4 intervals.
The interval size would be $(2.12 - (-1.08))/4 = 3.2/4 = 0.8$



- **Quantize:** Assign each number to the nearest quantization level based on these intervals.
-1.08, -1.03, -0.98, -0.91, -0.41, 0, 0, 0.05, 0.09, 1.48, 1.49, 1.53, 1.87, 1.92, 2.09, 2.12

Linear Quantization: Formal Definition

An affine mapping of integers to real numbers $r = S(q - Z)$

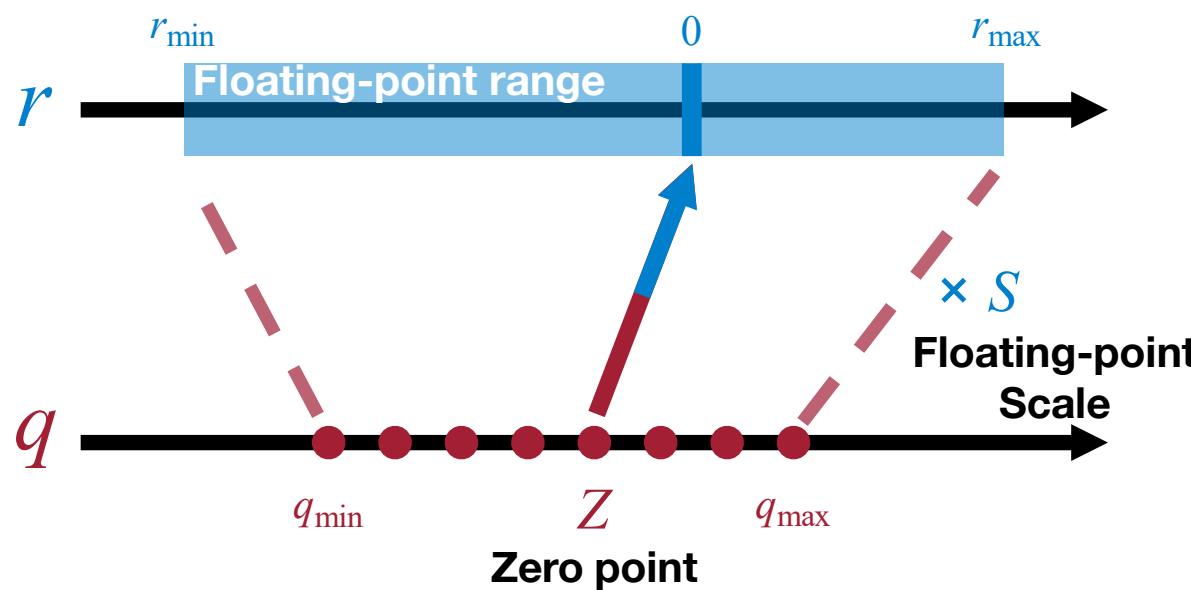


$$\begin{aligned} r_{\max} &= S(q_{\max} - Z) \\ r_{\min} &= S(q_{\min} - Z) \\ r_{\max} - r_{\min} &= S(q_{\max} - q_{\min}) \end{aligned}$$
$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Linear Quantization : Formal Definition

An affine mapping of integers to real numbers $r = S(q - Z)$



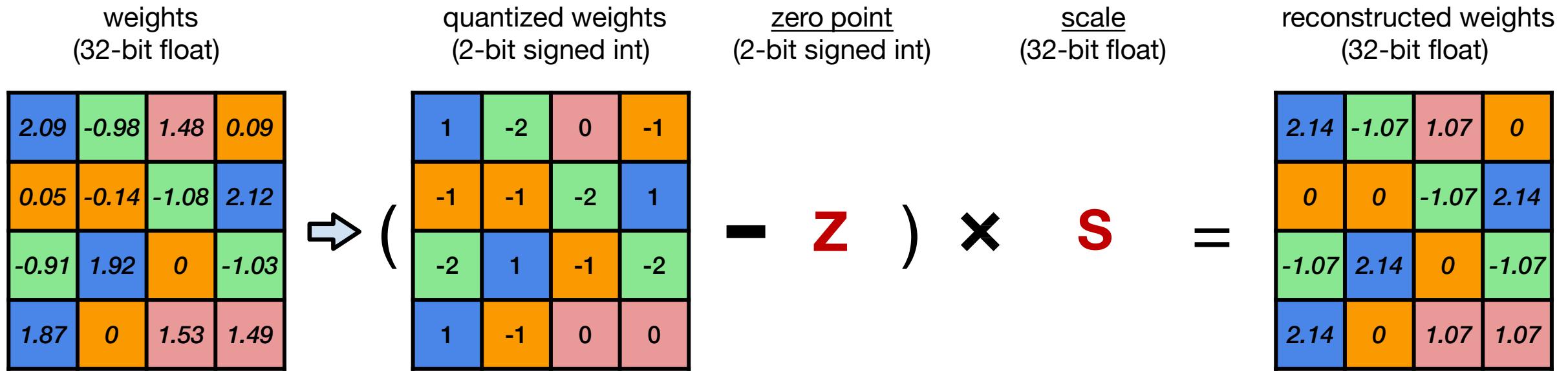
$$r_{\min} = S(q_{\min} - Z)$$

$$Z = q_{\min} - \frac{r_{\min}}{S}$$

$$Z = \text{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Linear Quantization



2-bit signed int to quantization weight mapping

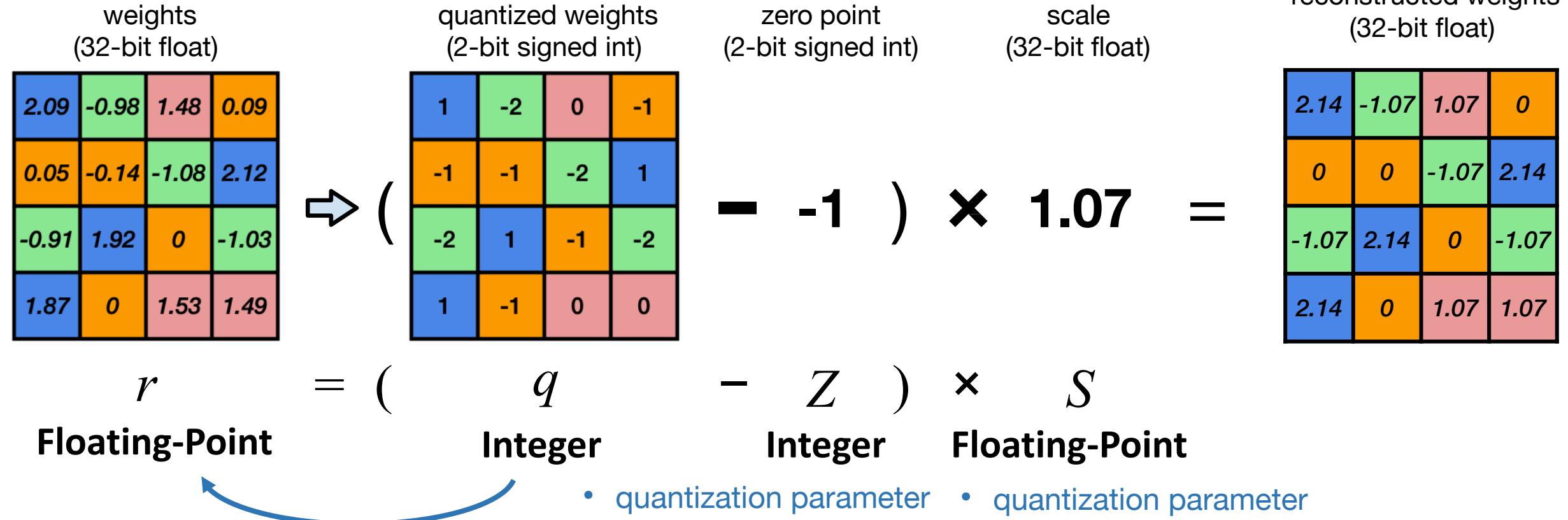
Binary	Decimal
01	1
00	0
11	-1
10	-2

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]



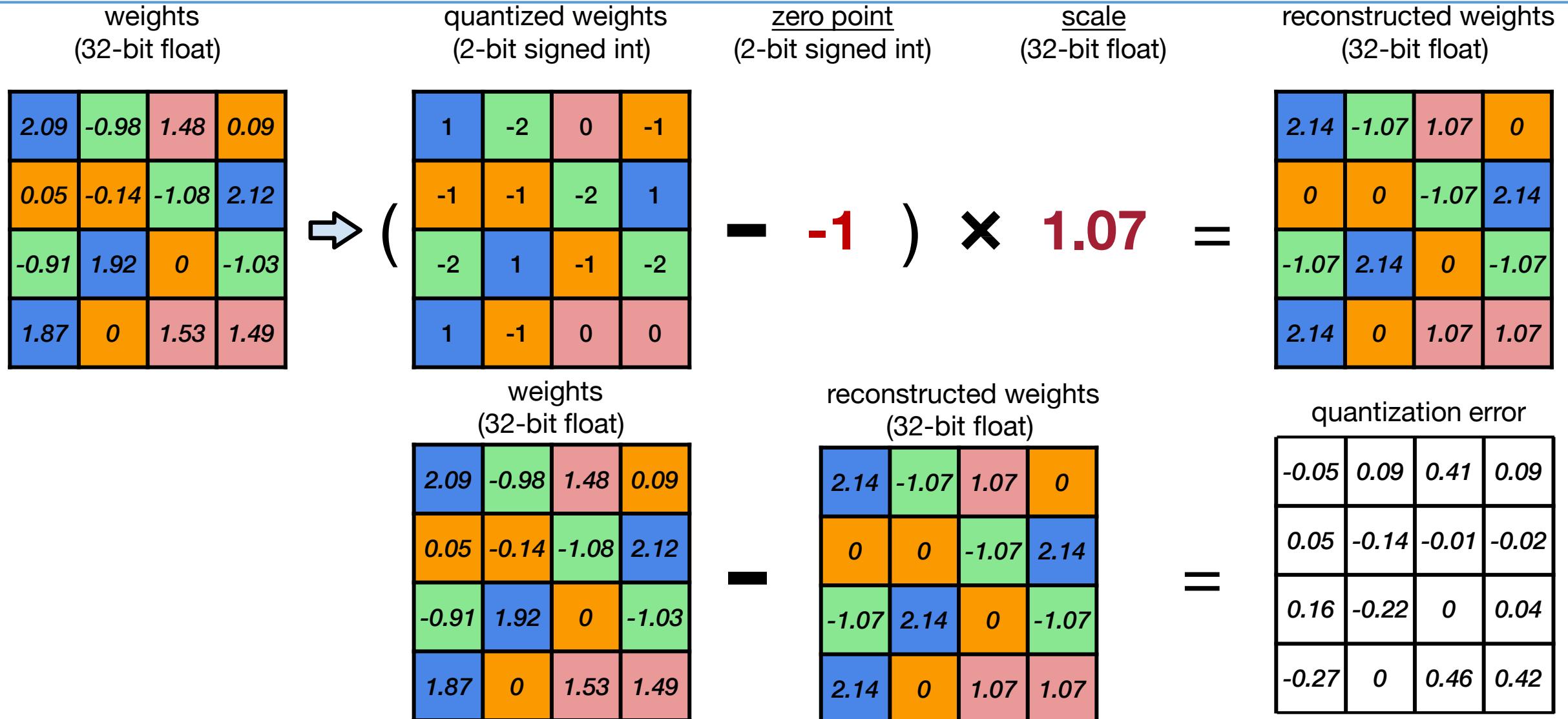
Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$



- quantization parameter
- allow real number $r=0$ be exactly representable by a quantized integer
- quantization parameter

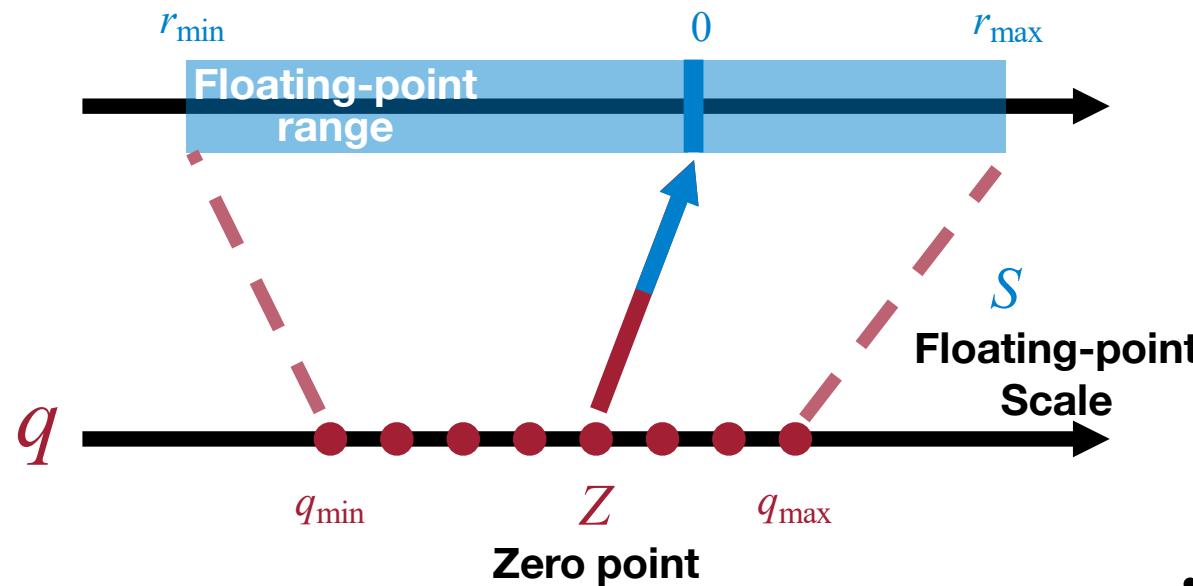
Linear Quantization



Linear Quantization: Asymmetric vs. Symmetric

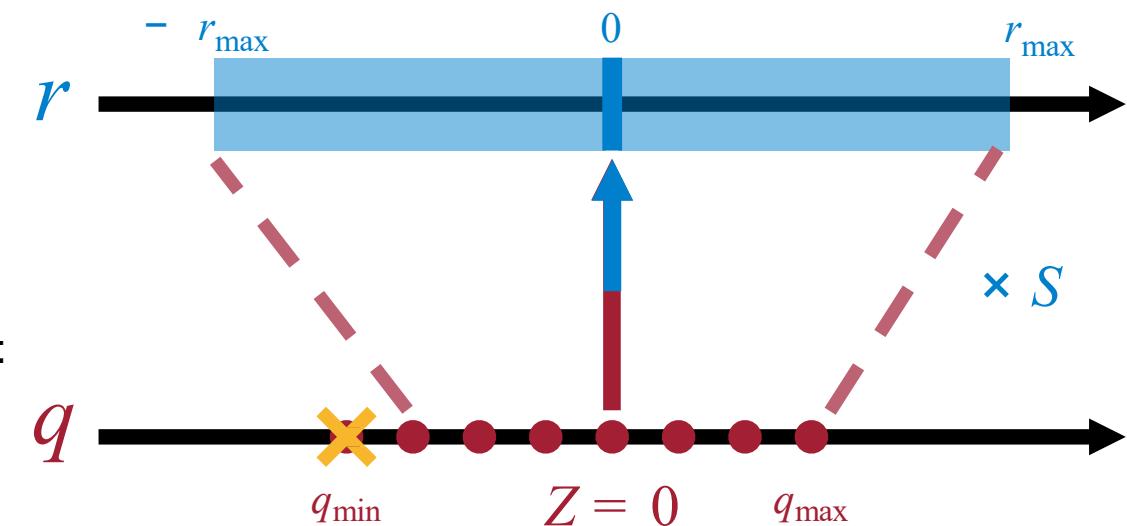
An affine mapping of integers to real numbers $r = S(q - Z)$

Asymmetric Linear Quantization



- The quantized range is fully used.
- The implementation is more complex and may require additional logic in hardware.

Symmetric Linear Quantization



- The quantized range will be wasted for biased float range.
- The implementation is much simpler.

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Quantization

Quantization in Neural Networks

1. Theory in Quantization

- K-mean based quantization

2. Post-Training Quantization (PTQ)

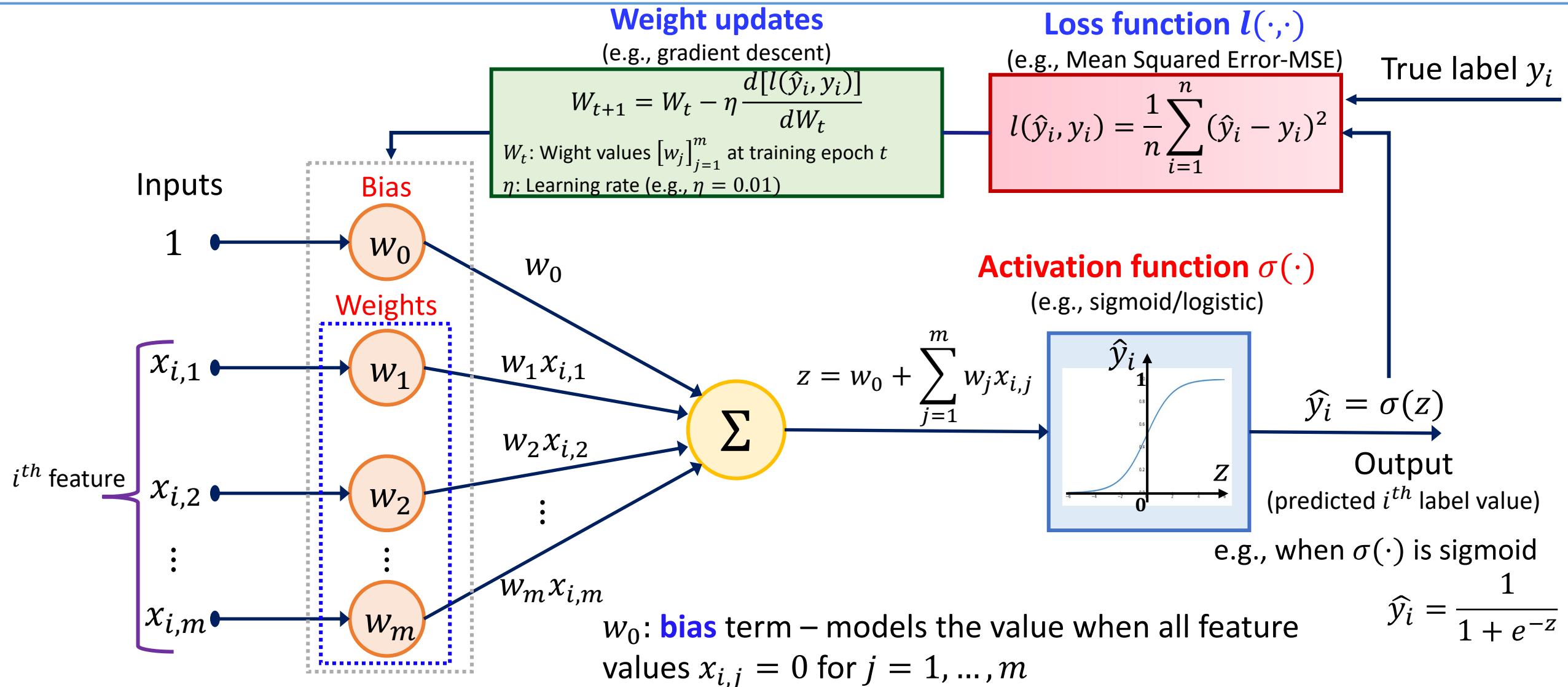
– How can we get optimal linear quantization parameters (S , Z)?

- Weight quantization
- Activation quantization
- Bias quantization

3. Quantization-Aware Training



Post Training Quantization (PTQ)



Quantization

Quantization in Neural Networks

1. Theory in Quantization

- K-mean based quantization

2. Post-Training Quantization (PTQ)

- Weight quantization
- Activation quantization
- Bias quantization

3. Quantization-Aware Training

– How should we improve the performance of quantized models?

Quantization

What algorithms to choose to improve accuracy?

Post-Training Quantization vs Quantization-Aware Training

Post-Training Quantization (PTQ)

- ✓ Takes a pre-trained network and converts it to a fixed-point network without access to the training pipeline
- ✓ Data-free or small calibration set needed
- ✓ Use though single API call
- ✗ Lower accuracy at lower bit-widths

Quantization-Aware Training (QAT)

- ✗ Requires access to training pipeline and labelled data
- ✗ Longer training times
- ✗ Hyper-parameter tuning
- ✓ Achieves higher accuracy

Source: <https://www.tinyml.org/event/tinyml-talks-a-practical-guide-to-neural-network-quantization/>

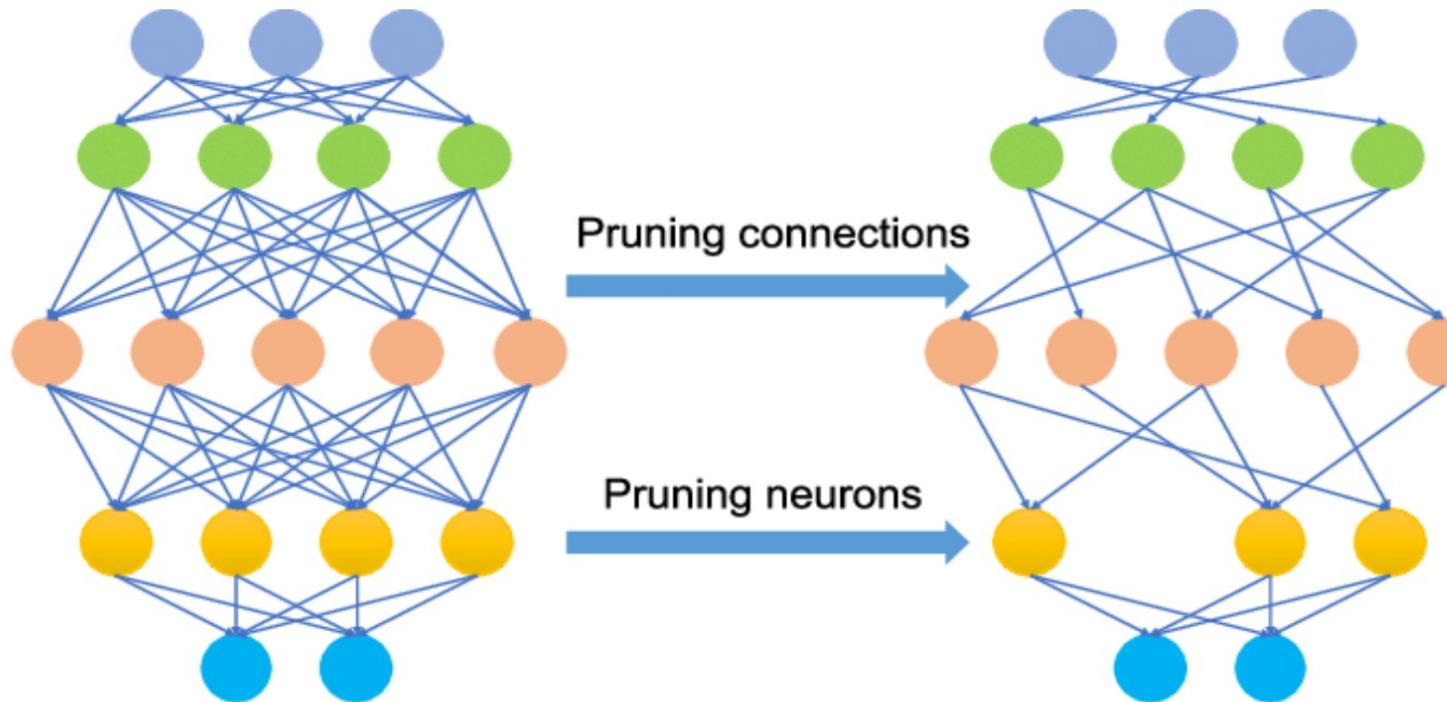




5 min Break

Model Compression: Pruning

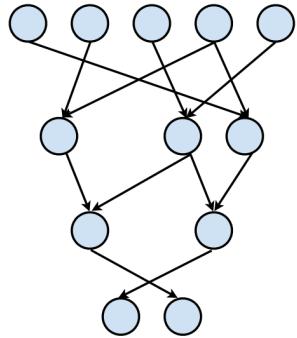
- First creates a **large model** that achieves the **required accuracy**
- Then try and **decrease the size** (prune it) **while trying to maintain the accuracy**
- Once the model has been pruned, this **new pruned model** is **trained again** on the dataset and this is known as **iterative pruning**



Source: <https://heartbeat.comet.ml/neural-network-pruning-research-review-2020-bc21a77f0295>

Pruning Method

Pruning makes neural network smaller by removing synapses and neurons.



Train Connectivity



Prune Connections

Baseline: a man is riding a surfboard on a wave.

Train Weights

Pruned 90%: a man in a wetsuit is riding a wave on a beach

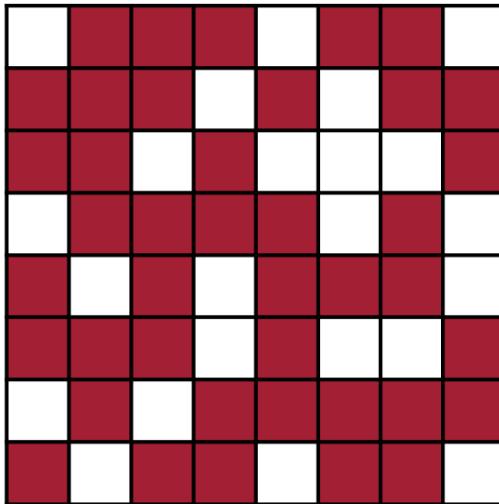


Baseline: a soccer player in red is running in the field.

Pruned 95%: a man in red shirt and black and white black shirt is running through a field.

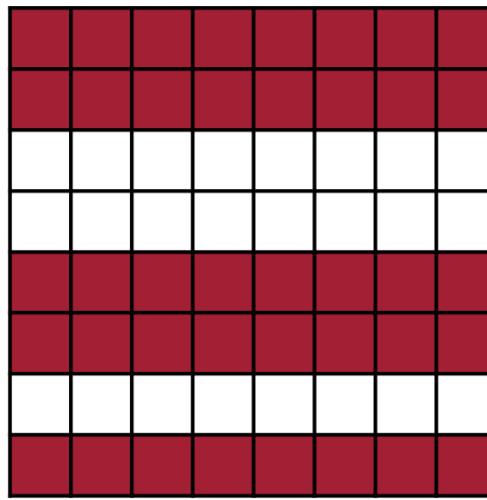
Pruning Techniques

Pruning can be classified as structured and unstructured based on the granularity.



Fine Grained/ Unstructured Pruning

- More flexible pruning index choice
- Hard to accelerate (irregular data expression)

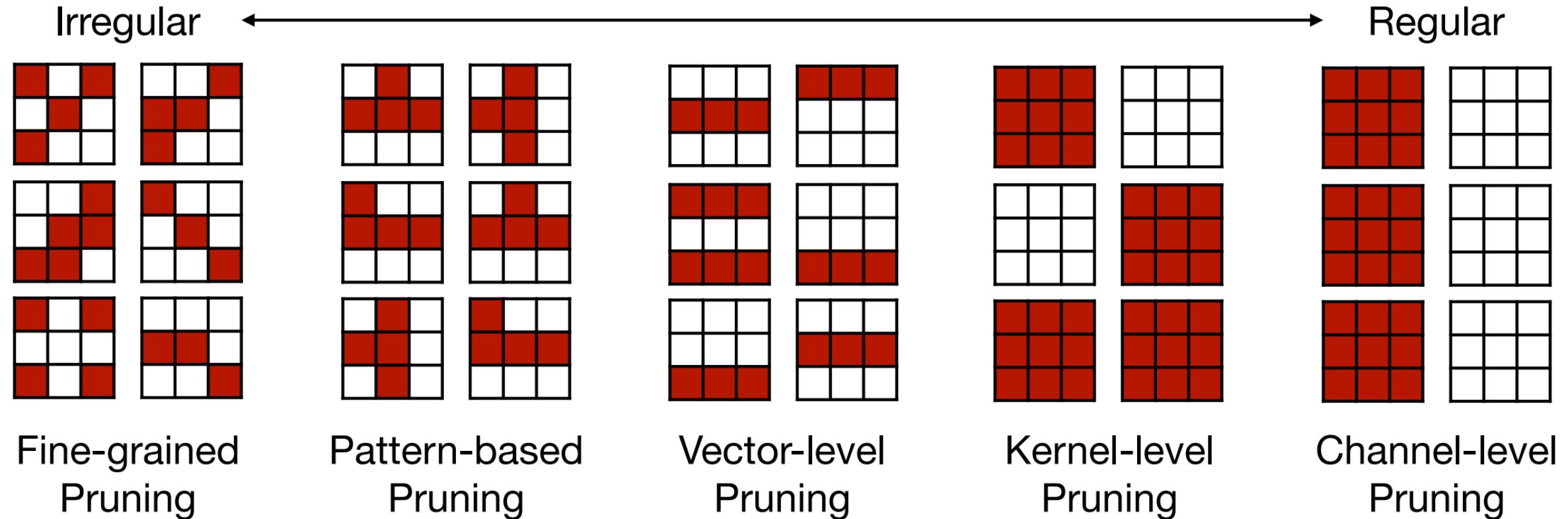


Coarse Grained/ Structured Pruning

- Less flexible pruning index choice.
- Easy to accelerate (since it is a smaller matrix)

Source: <https://hanlab.mit.edu/files/course/slides/MIT-TinyML-Lec03-Pruning-I.pdf>

Pruning Types: Granularity Based

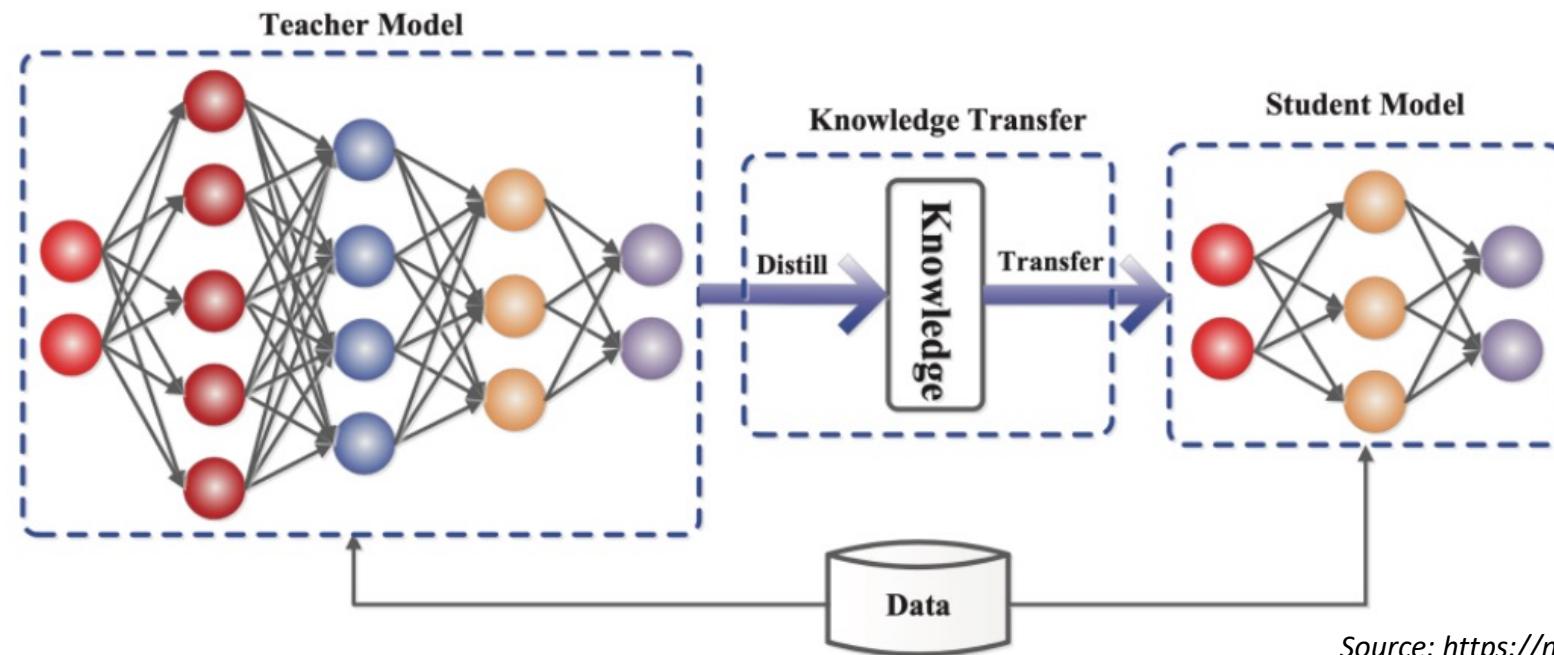


Source: <https://hanlab.mit.edu/files/course/slides/MIT-TinyML-Lec03-Pruning-I.pdf>



Model Compression: Knowledge Distillation

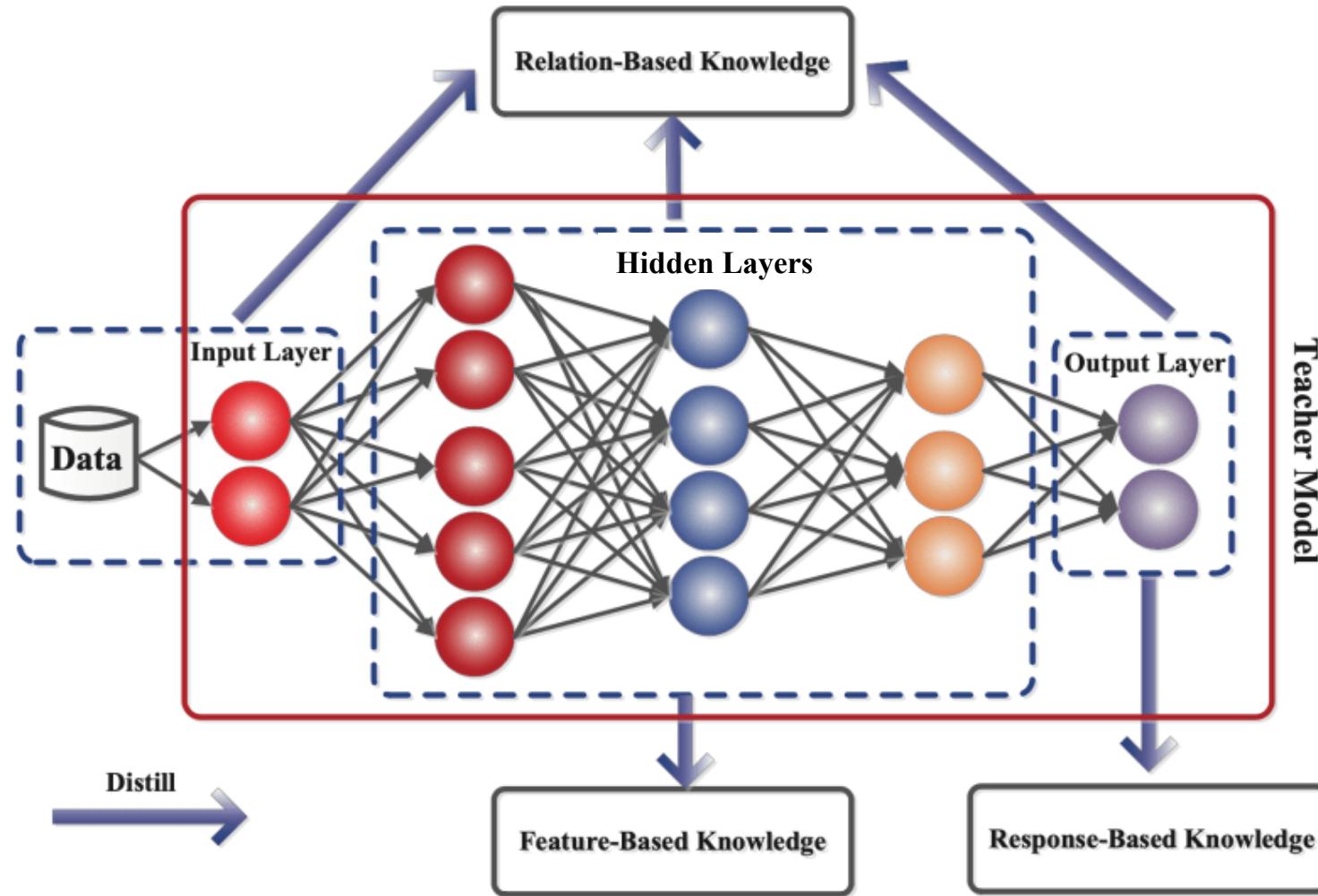
- Effectively training a **bigger ML model**, or the **TEACHER**, with a given dataset.
- The model would generate predictions based on the given data.
- A **new dataset** would be constructed which would **combine the old data and the information generated from the bigger model**.
- This new dataset would be then fed to the **smaller ML model**, or the **STUDENT**.



Source: <https://neptune.ai/blog/knowledge-distillation>

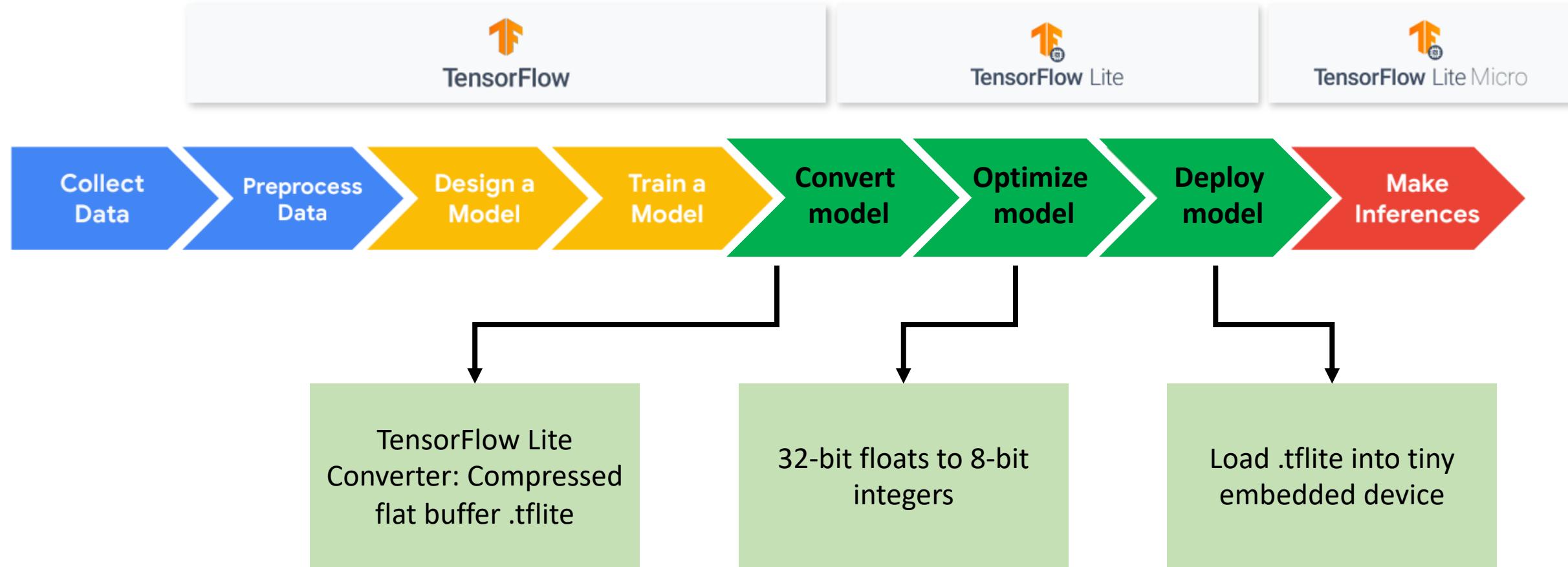
Knowledge Distillation Types

- **Feature-Based Knowledge:**
Matching intermediate representations/features
- **Response-Based Knowledge:**
Matching output logits or probabilities
- **Relation-Based Knowledge:**
Matching the relational structure between data points or features.
 - i.e., if two inputs are deemed similar by the teacher model, the student model should also learn to recognize this similarity
 - **Applications:** Computer Vision, Natural Language Processing

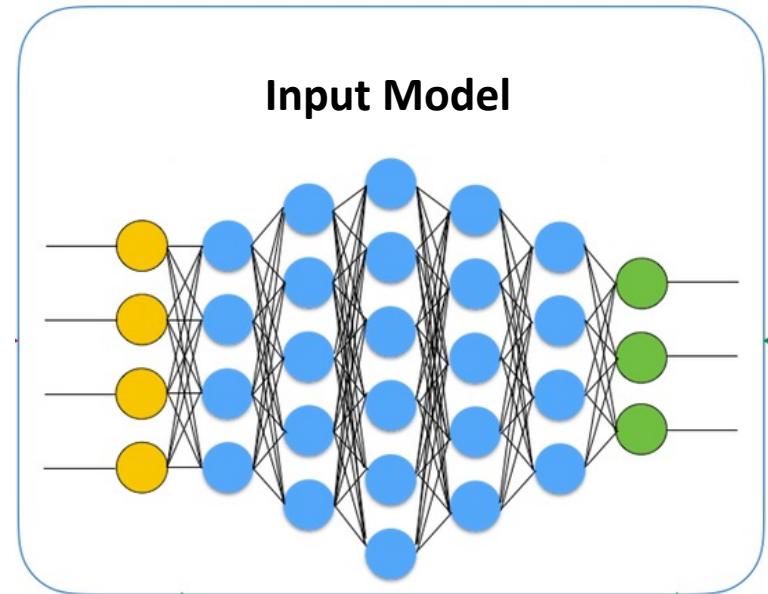


Source: <https://neptune.ai/blog/knowledge-distillation>

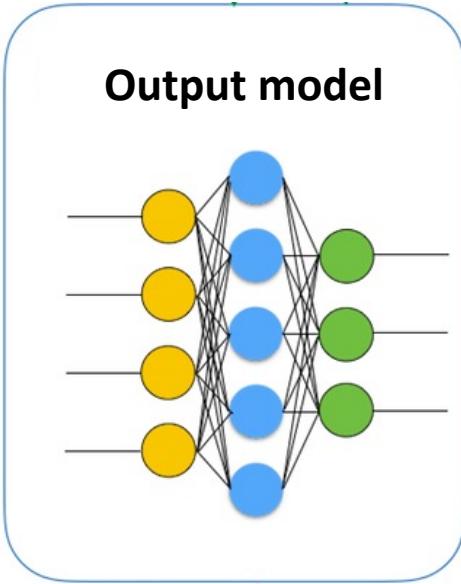
Putting it all together – TF → TF Lite for TinyML



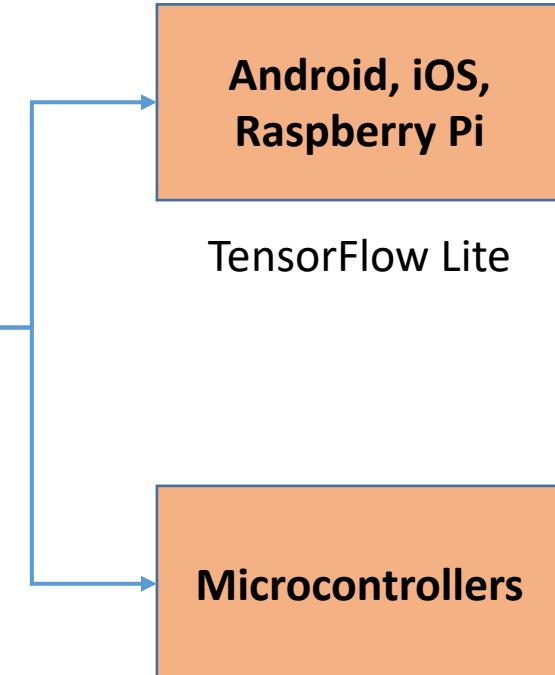
TFLite: Model Compression and Deployment



Model
Compression



Model Deployment on Tiny Devices

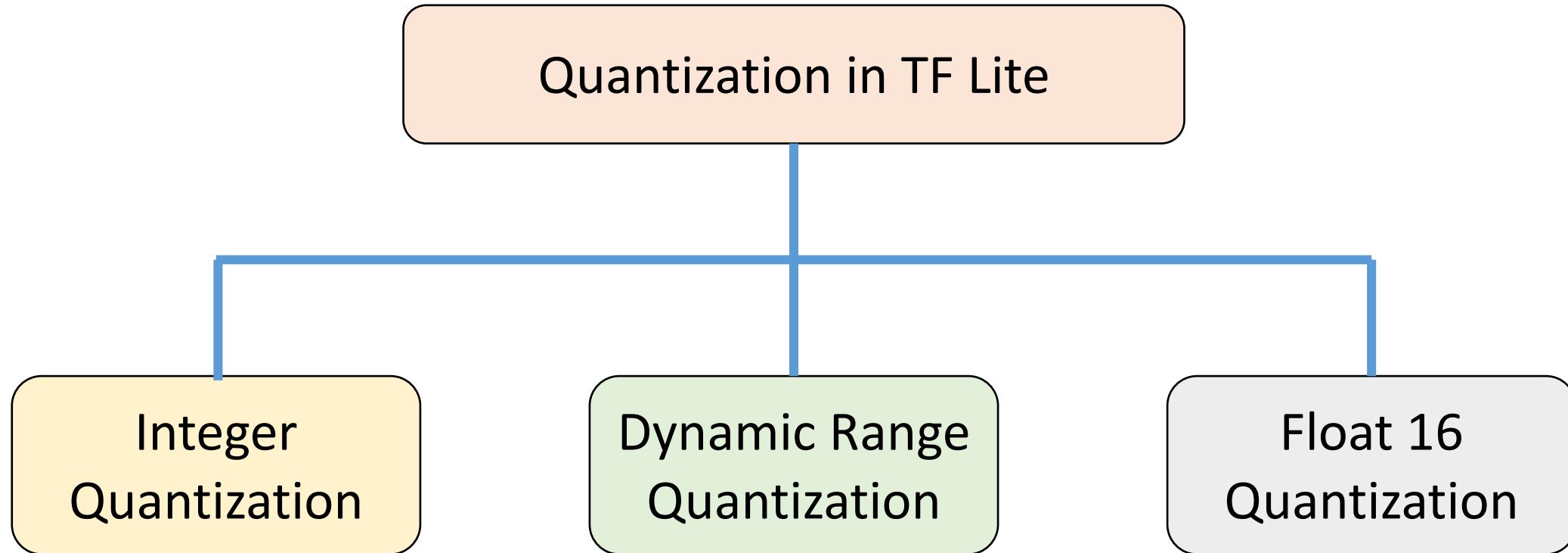


TF vs. TFLite – Summary of Key Differences

	Parameter	TensorFlow	TensorFlow Lite
Model	Training	Yes	No
	Inference	Yes (Inefficient on Edge)	Yes (Efficient on Edge)
	Number of Operators	~1400	~130
	Native Quantization Tooling	No	Yes
Hardware	Operating System	Yes	Yes
	Memory Mapping of Models	No	Yes
	Delegation to accelerators	Yes	Yes
Software	Base Binary Size	> 3 MB	100 KB
	Base Memory Footprint	~ 5 MB	300 KB
	Optimized Architecture	X86, TPUs, GPUs	Arm Cortex A, x86



Quantization in TensorFlow Lite



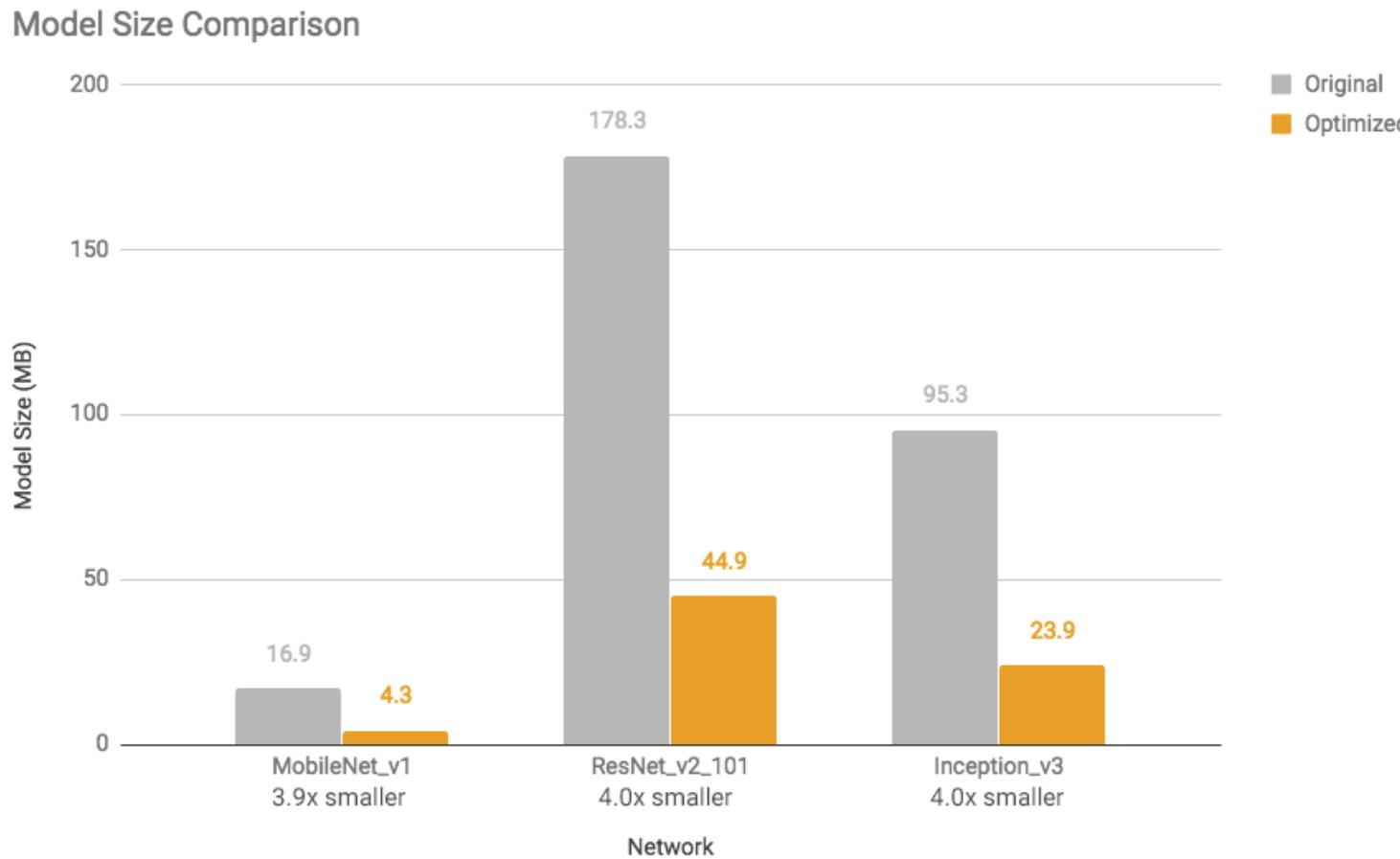
Integer Quantization

- Integer quantization is an optimization strategy that converts 32-bit floating-point numbers (such as weights and activation outputs) to the nearest 8-bit fixed-point numbers.
- Results in a smaller model and increased inferencing speed, which is valuable for low-power devices such as microcontrollers.
- Full integer quantization means convert all weights and activation outputs into 8-bit integer data—whereas other strategies may leave some amount of data in floating-point.



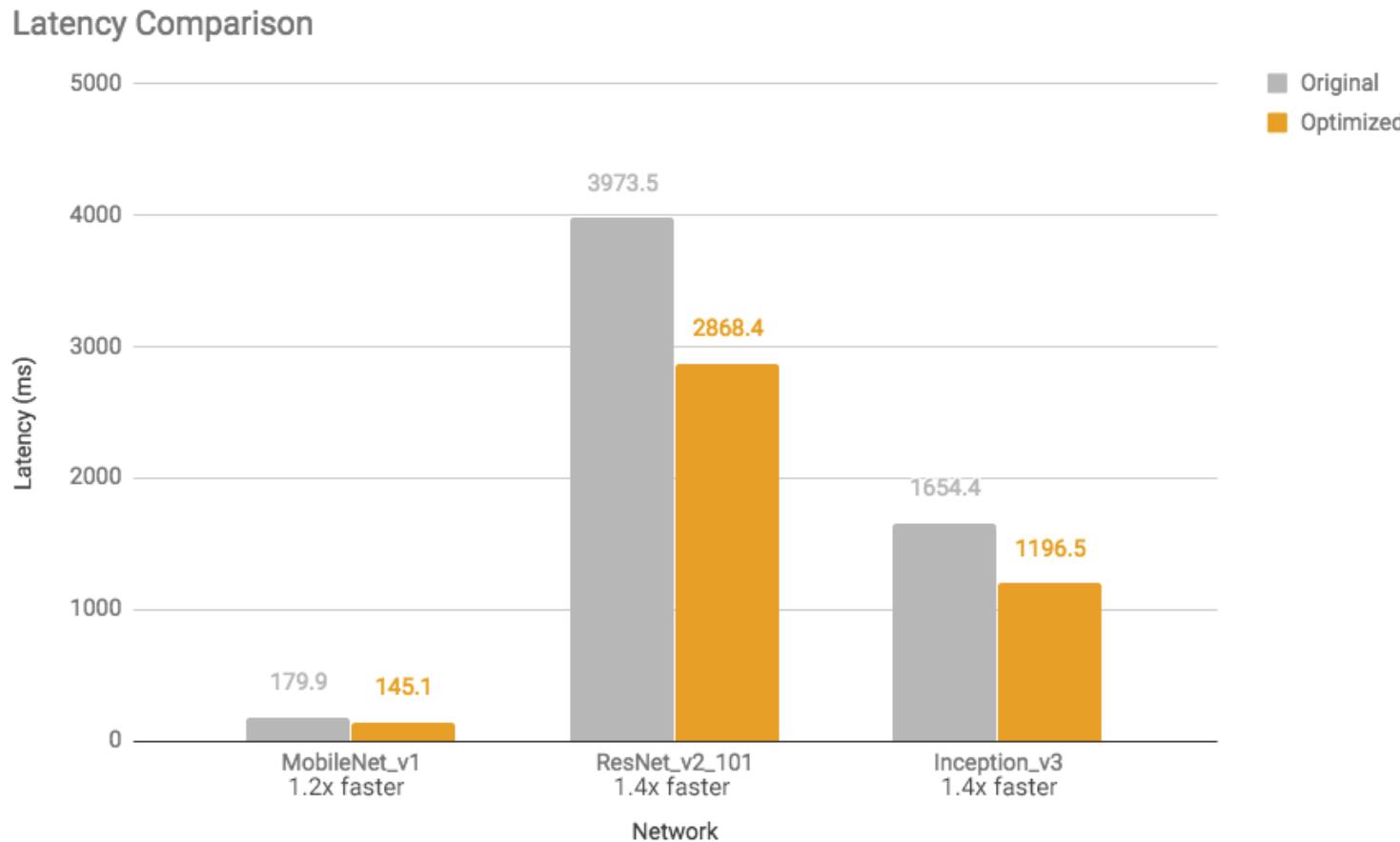
Integer Quantization

Performance: Model Size



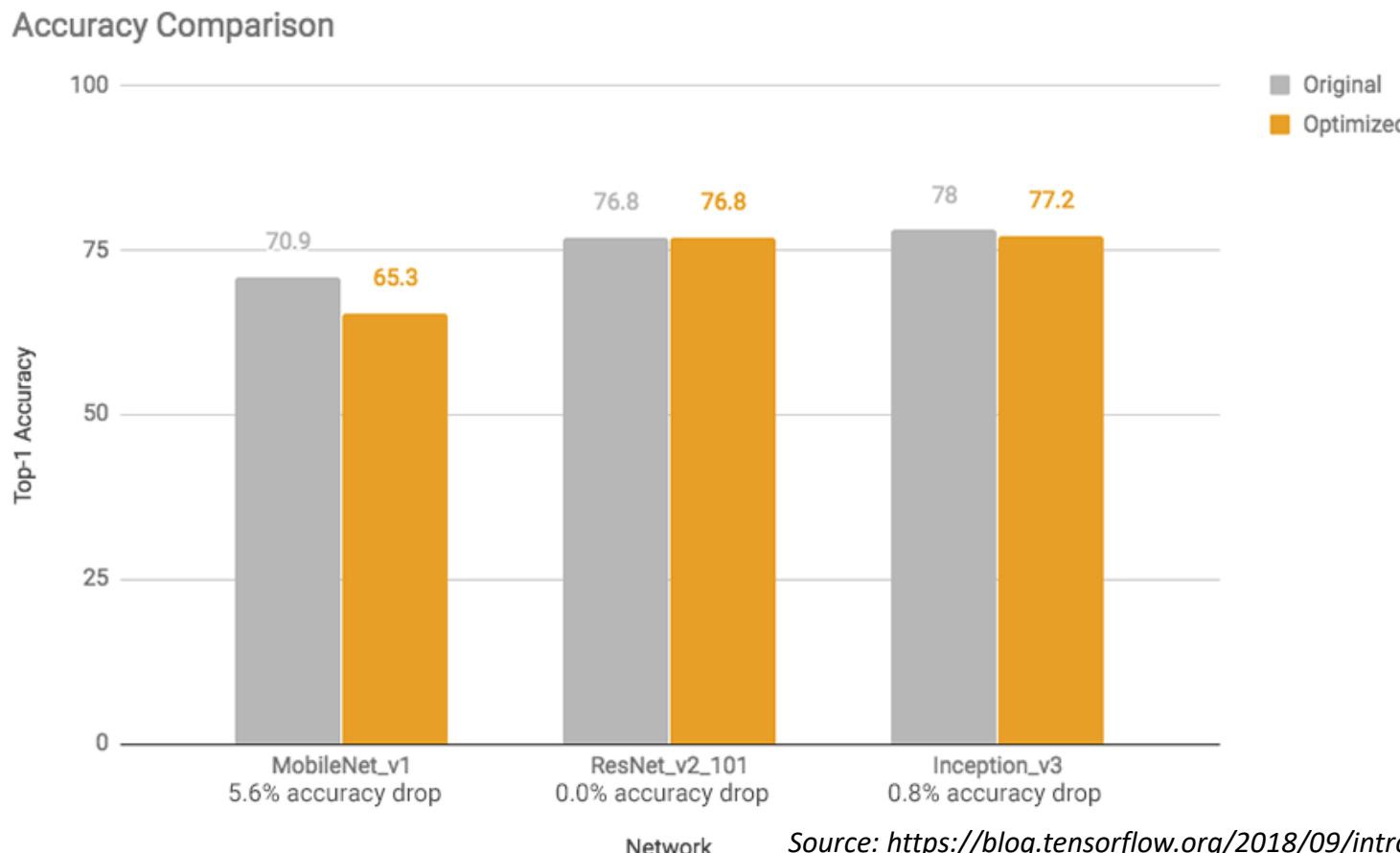
Integer Quantization

Performance: Latency



Integer Quantization

Performance: Accuracy



Dynamic Range Quantization

- Convert **weights to 8-bit precision** as part of model conversion from TensorFlow's graphdefs to TF Lite's flat buffer format.
- **Determine Activation Range at Runtime:** The **range of the activations** is computed during the model's execution, not beforehand. This means that for each batch of input data, the activation values are analyzed, and their range is determined.
- **Scale and Quantize:** Based on this range, the activations are then scaled and quantized into the desired format (e.g., 8-bit) for computation. After the computations are complete, they are converted back to floating-point numbers for the next layer's input.
- Achieves a 4x reduction in the model size.

Float-16 Quantization

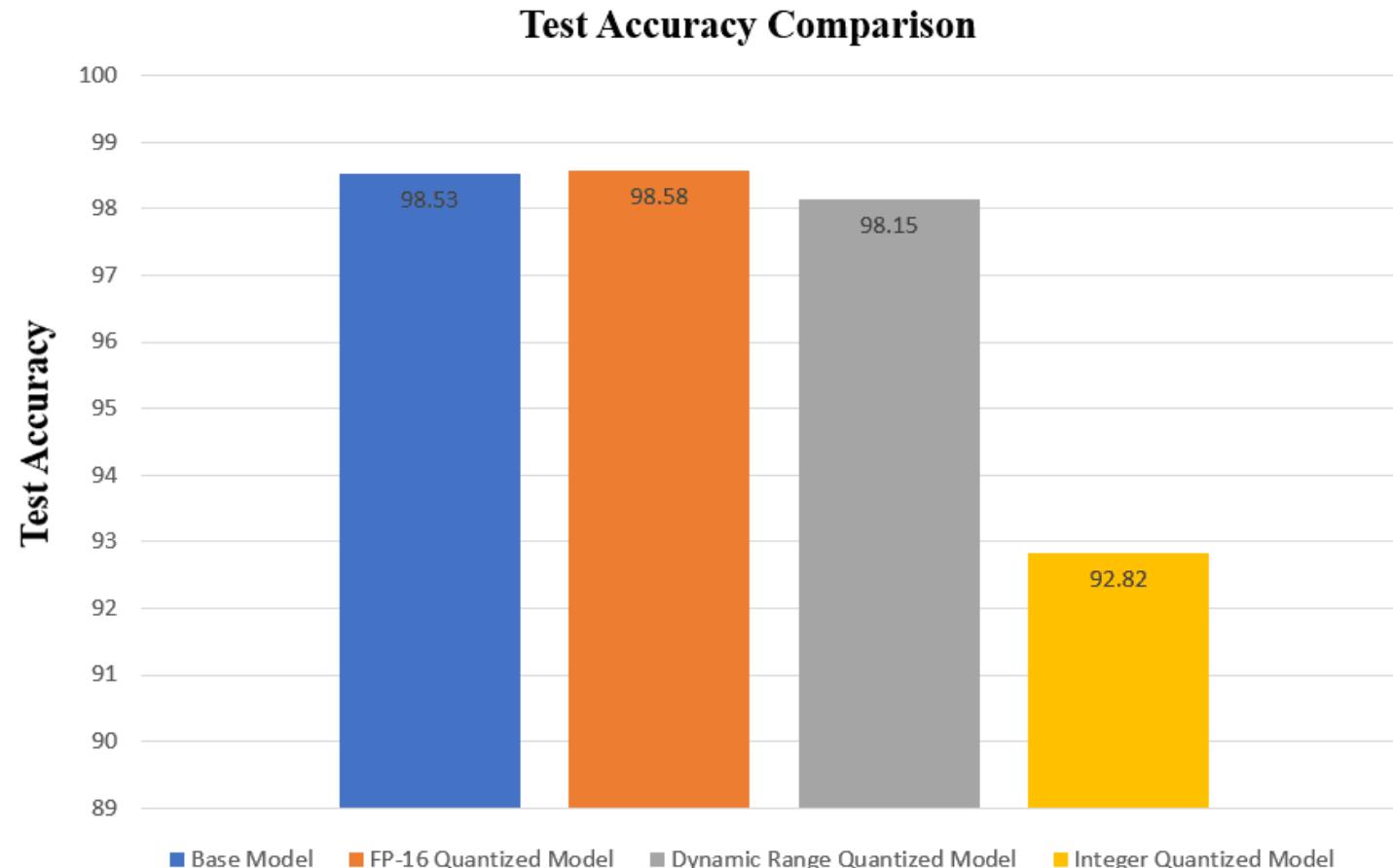
- Convert weights to 16-bit floating point values during model conversion from TensorFlow to TF Lite's flat buffer format.



- Results in a 2x reduction in model size for a minimal impacts on latency and accuracy.
- GPUs, can compute natively in this reduced precision arithmetic, realizing a speedup over traditional floating point execution.

Quantization in TinyML: Comparison

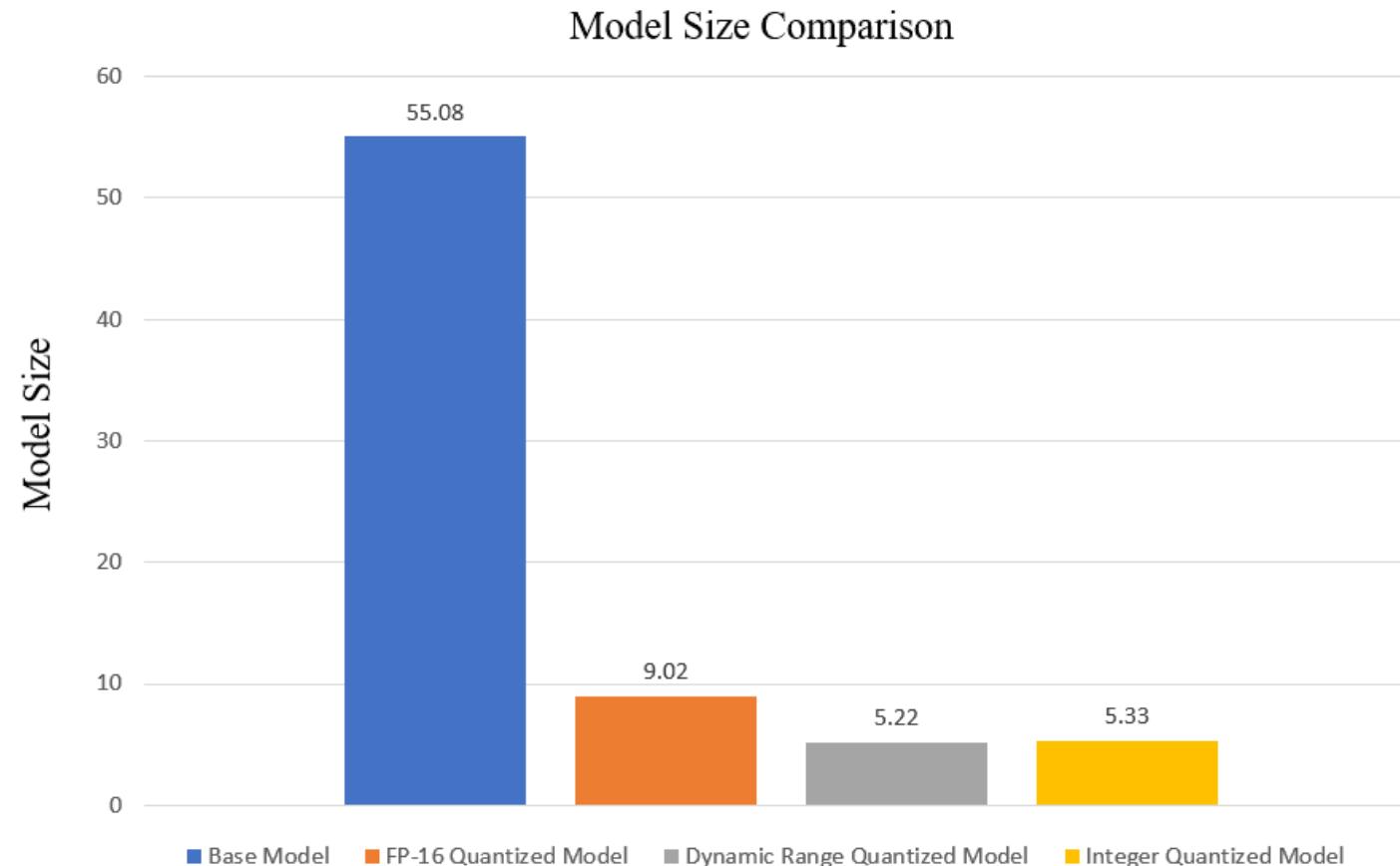
Among Integer & Dynamic Range & Float 16 Quantization



Source: <https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/>

Quantization in TinyML: Comparison

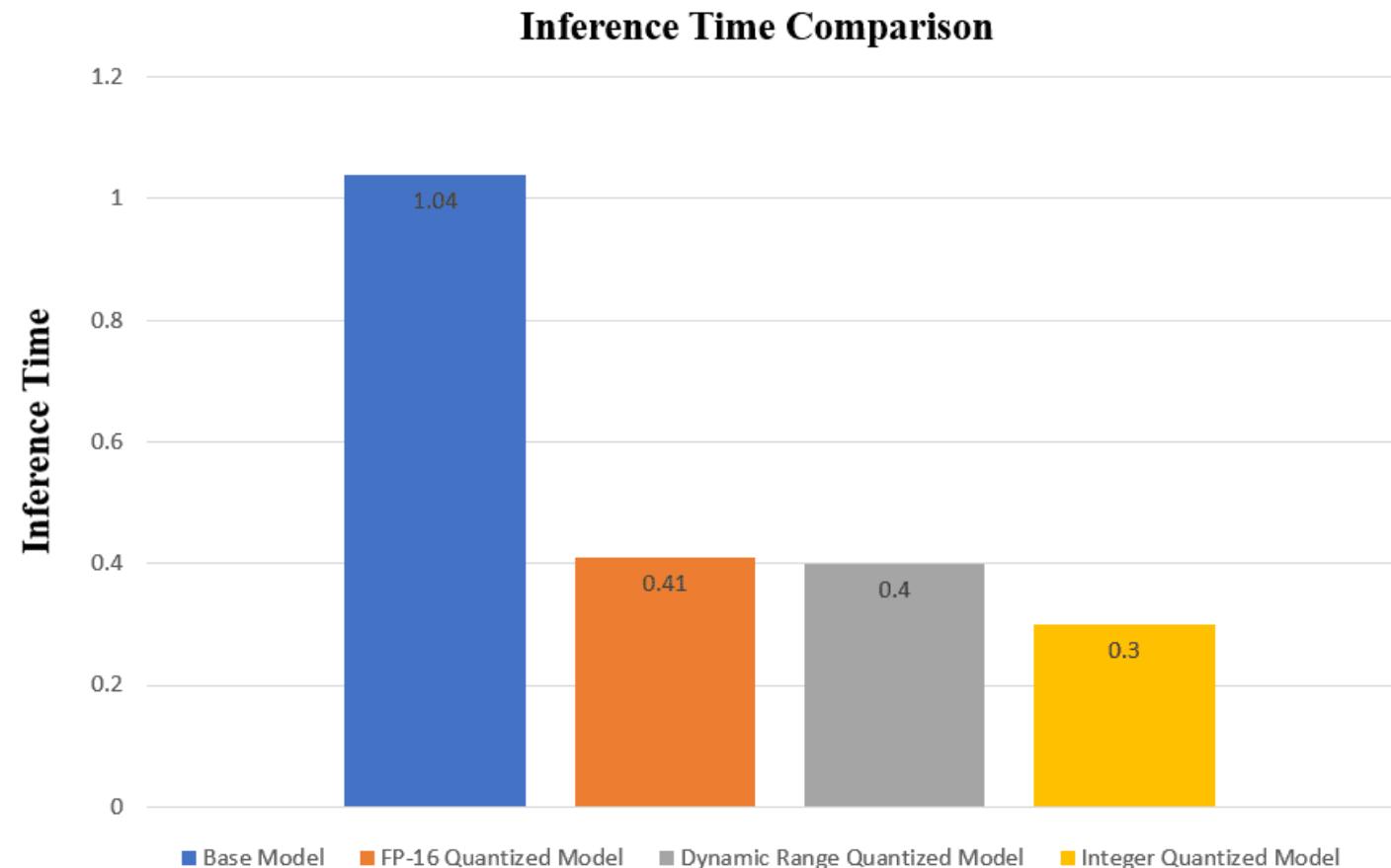
Among Integer & Dynamic Range & Float 16 Quantization



Source: <https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/>

Quantization in TinyML: Comparison

Among Integer & Dynamic Range & Float 16 Quantization



Source: <https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/>

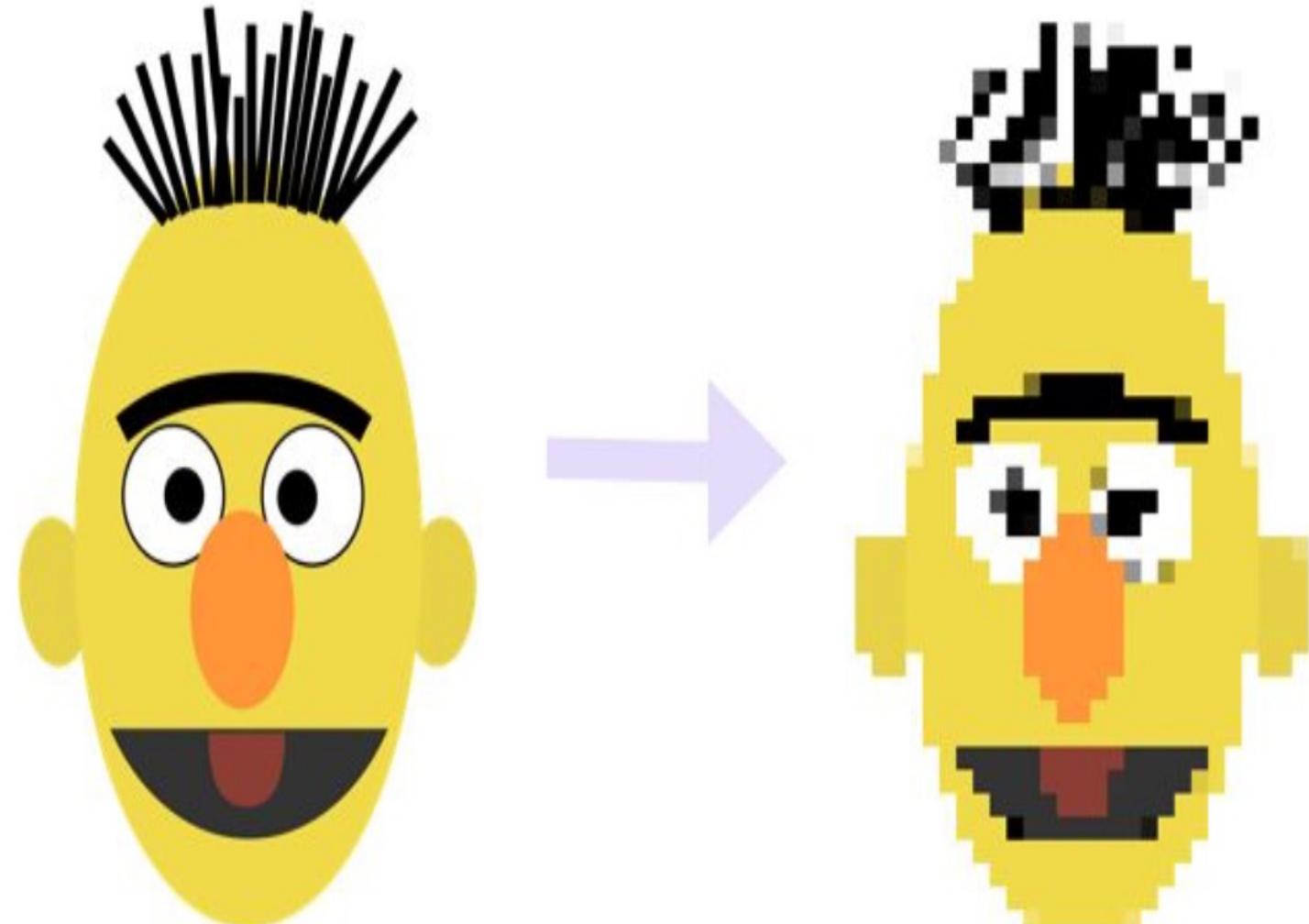
Quantization in TinyML: Summary

Summary table of the choices and the benefits:

Technique	Benefits	Hardware
Dynamic range quantization	4x smaller, 2x-3x speedup	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, Microcontrollers
Float16 quantization	2x smaller, GPU acceleration	CPU, GPU

Week #2 Lab

- Training ML model using Cats and Dogs Dataset
- Quantizing ML model
- Pruning ML model
- Evaluating the performance of compressed ML Models



Lab 2 Road Map

1. Installing and Configuring Needed Python Libraries
2. Preparing Cats and Dogs Dataset
3. Loading EFNet Machine Learning Model
4. Compiling the EFNet Model
5. Training the EFNet Model
6. Evaluating the Trained EFNet Model



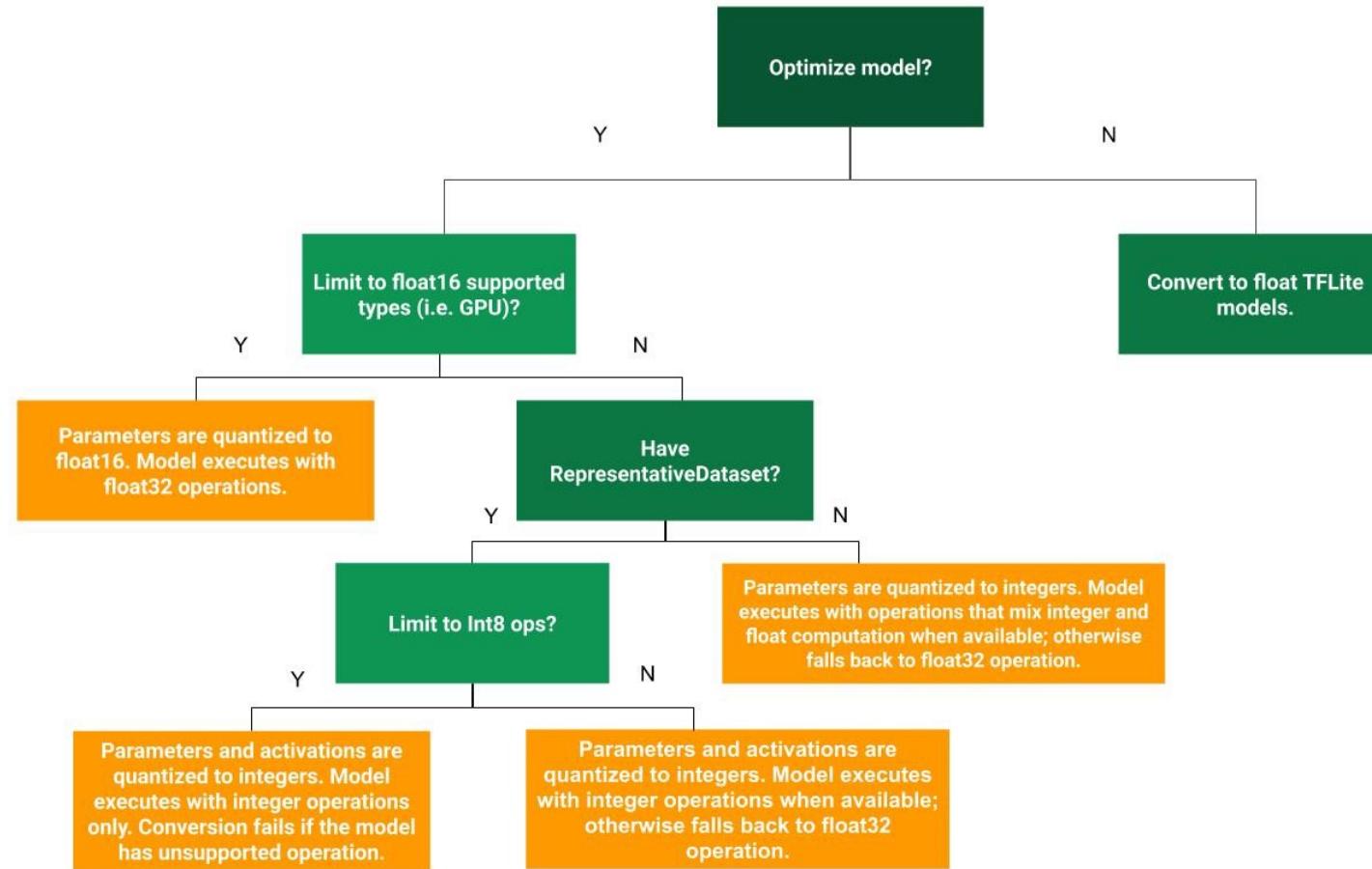
Lab 2 Road Map

7. Float-16 Quantization on EFNet Model
8. Dynamic Range Quantization on EFNet Model
9. Integer Quantization on EFNet Model
10. Weight Pruning on EFNet Model



Quantization in TensorFlow Lite

Determine which post-training quantization method is best





10 min Break