MERCon 2025
MORATUWA ENGINEERING RESEARCH CONFERENCE

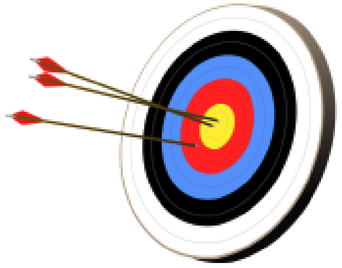
TinyML: A Compact Revolution in Engineering AI

Session 2: Model Compression Techniques

Dr. Dinuka Sahabandu
Assistant Teaching Professor
Department of Electrical and Computer Engineering
University of Washington

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY of WASHINGTON

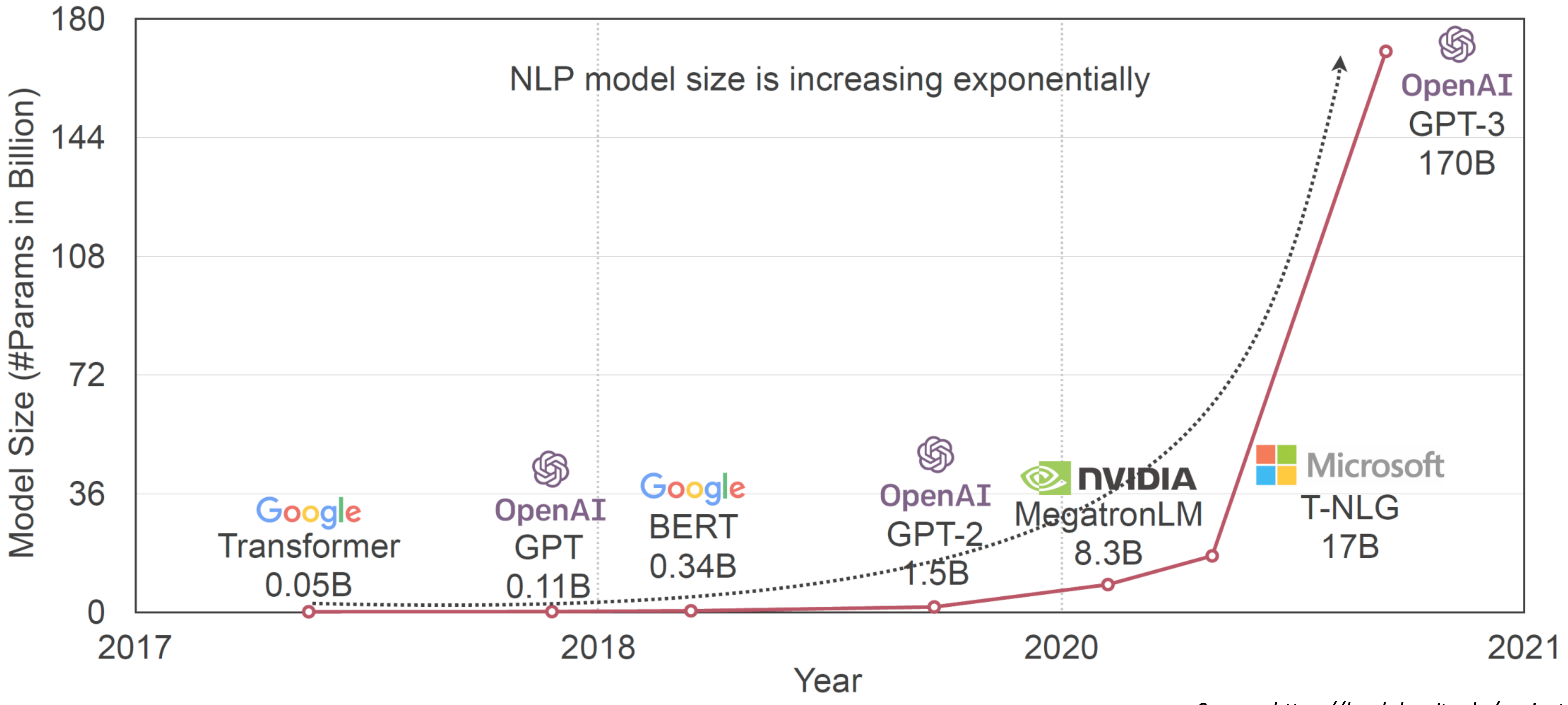
Topics Covered



- Introduction of Different Model Compression Methods
- Model Quantization Techniques
- Introduction to TensorFlow Lite
- Model Pruning Techniques
- Knowledge Distillation (KD) for Model Compression
- Coding session: Quantization, Pruning, and KD using TensorFlow / TensorFlow Lite

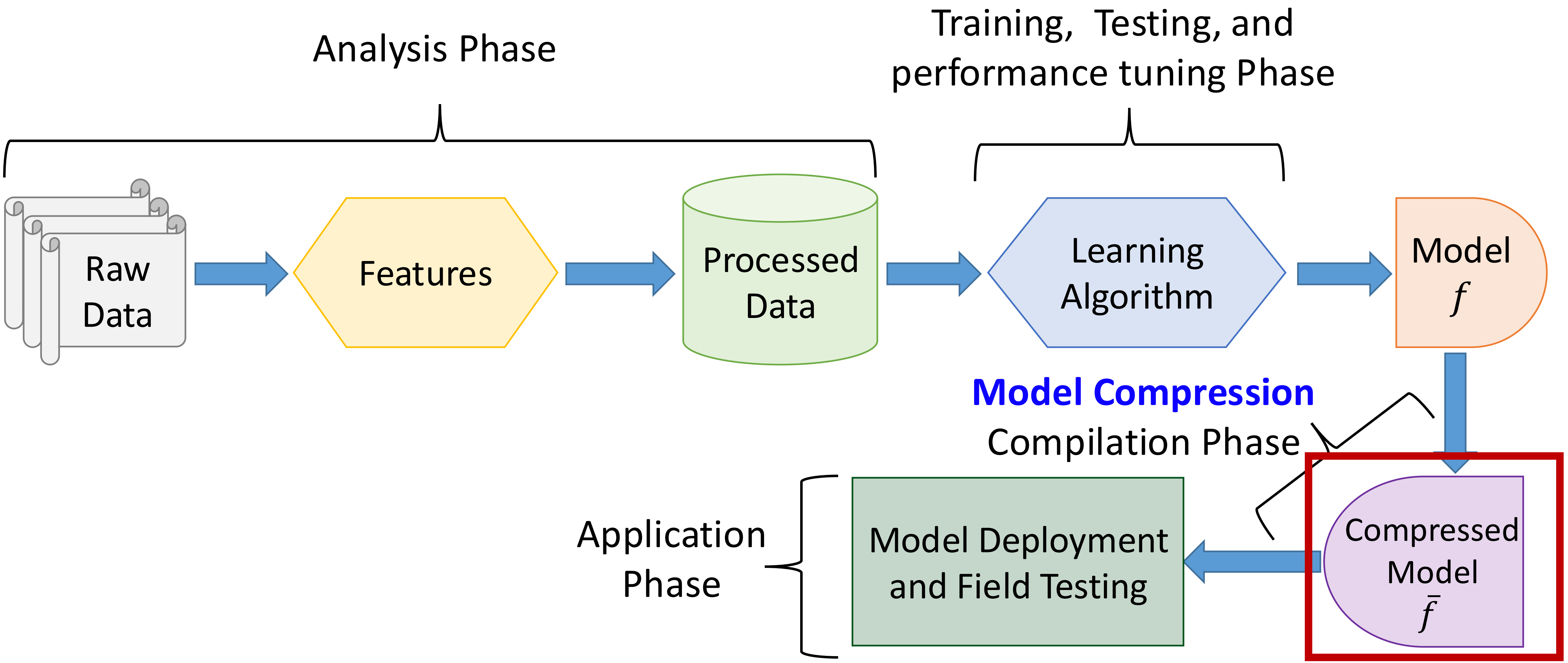
Trend of AI Model Size

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

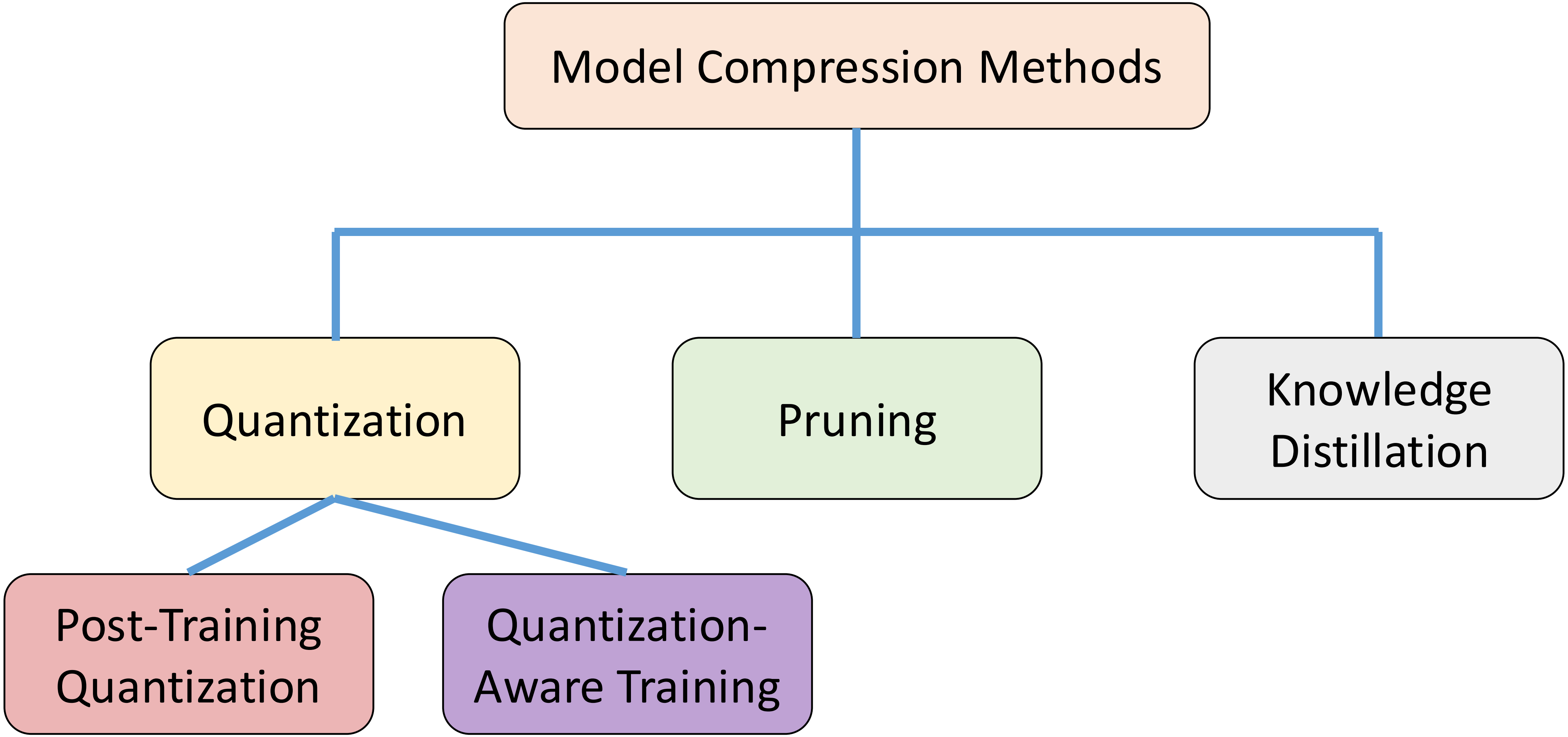


Source: https://hanlab.mit.edu/projects/efficientnlp_old/

TinyML Model Compression Phase



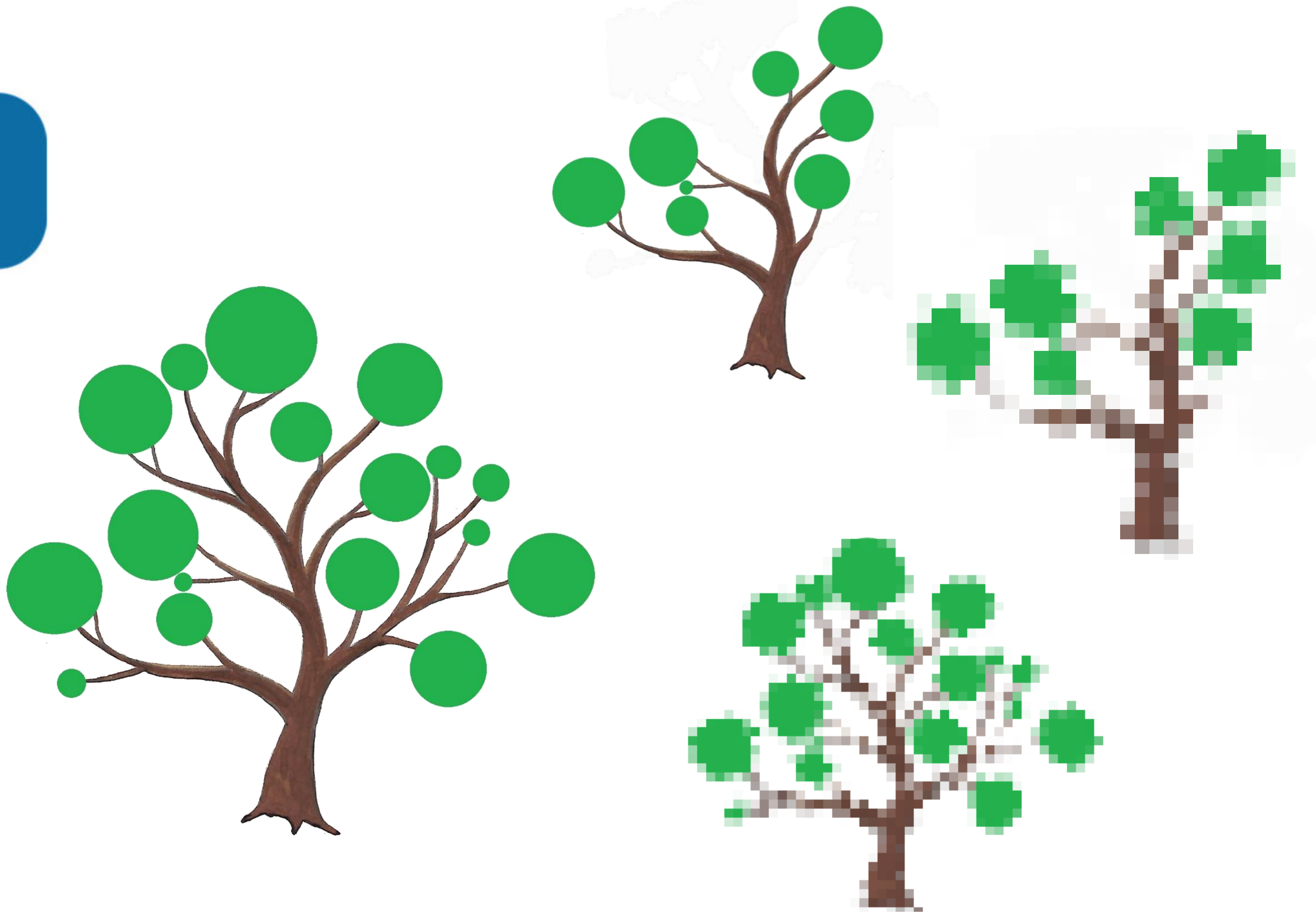
Key Model Compression Techniques



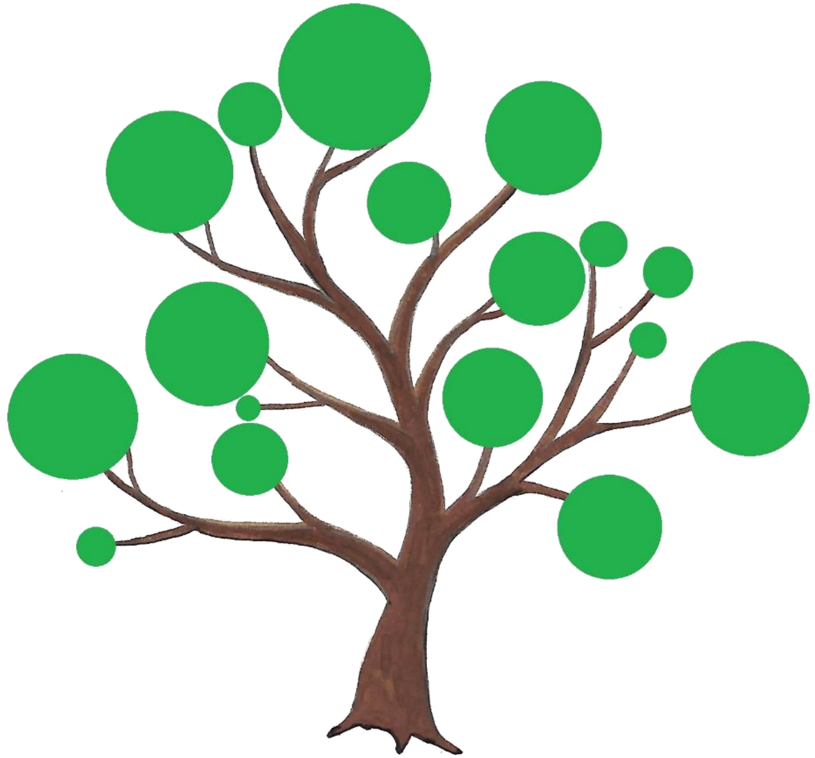
Quantization Vs. Pruning Discussion

1. What is quantization?

2. What is pruning?



Quantization and Pruning



A Tree

A Pruned Tree

A Quantized Tree

A Pruned and
Quantized Tree

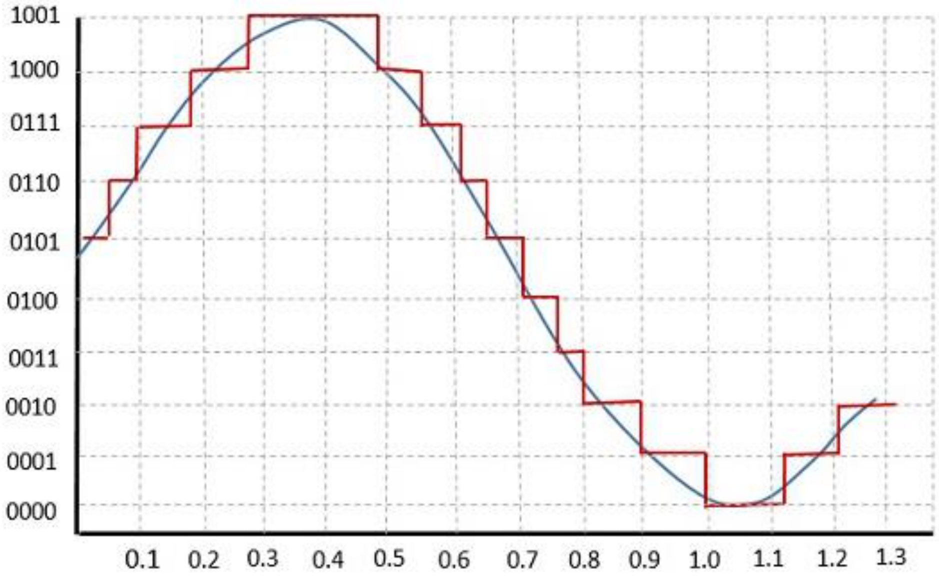
Quantizing Image and Signal Inputs

Original Image Of Billion Colors

RGB (24 bits, 2^{24} colors)

Quantized image with four color Image

2 bits, $2^2 = 4$ colors



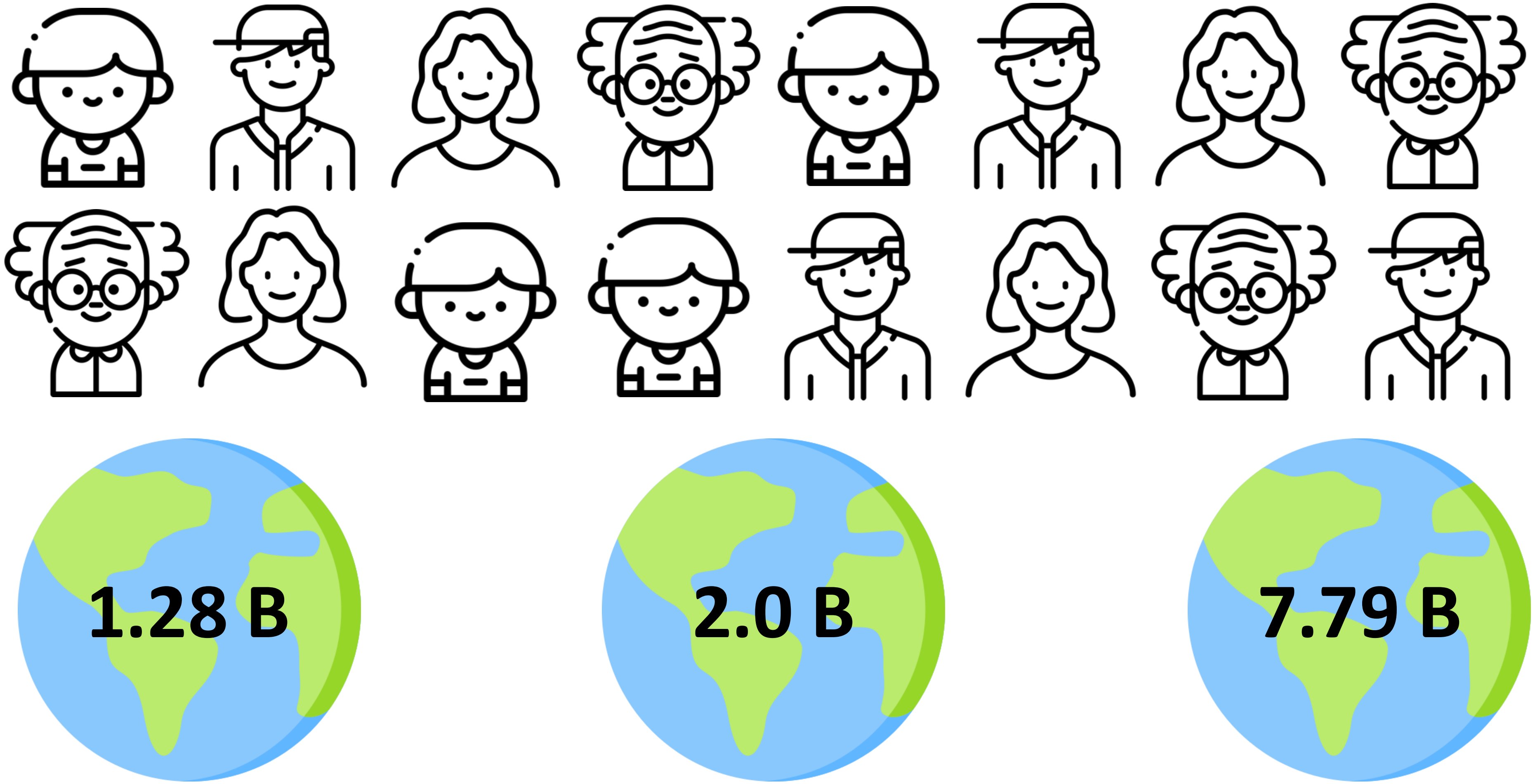
Quantization in ML

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

How Does Quantization Help?



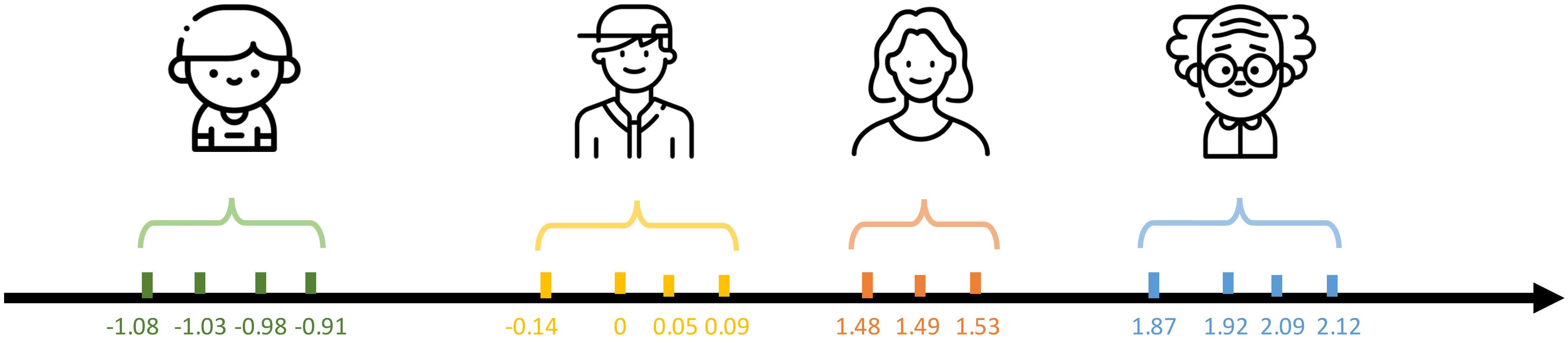
1820 1920 2020

Quantization can help!

Children Teenagers Adults Seniors

1820 1920 2020

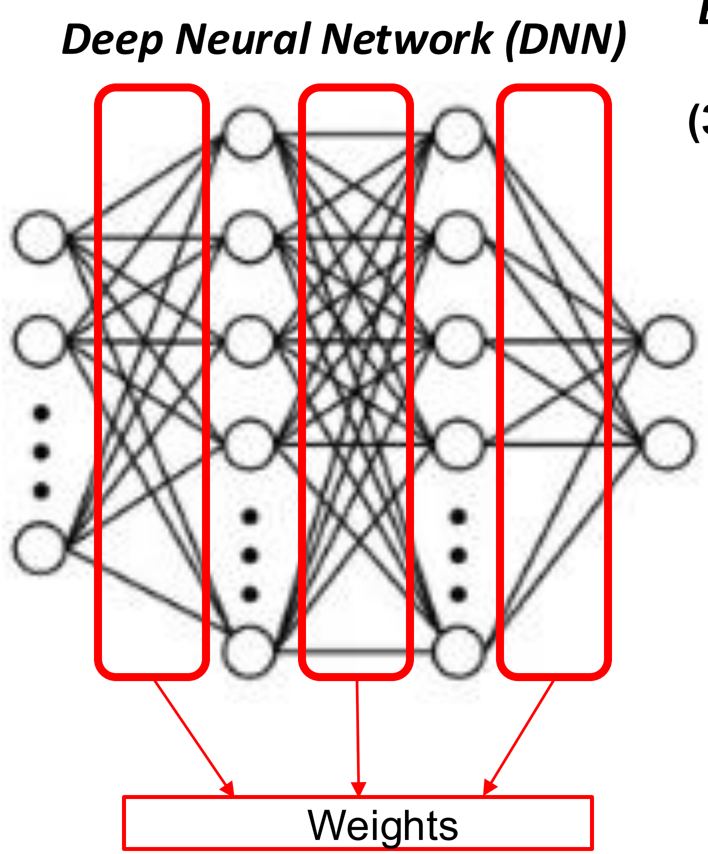
Viewing Grouping or Clustering as Quantization



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

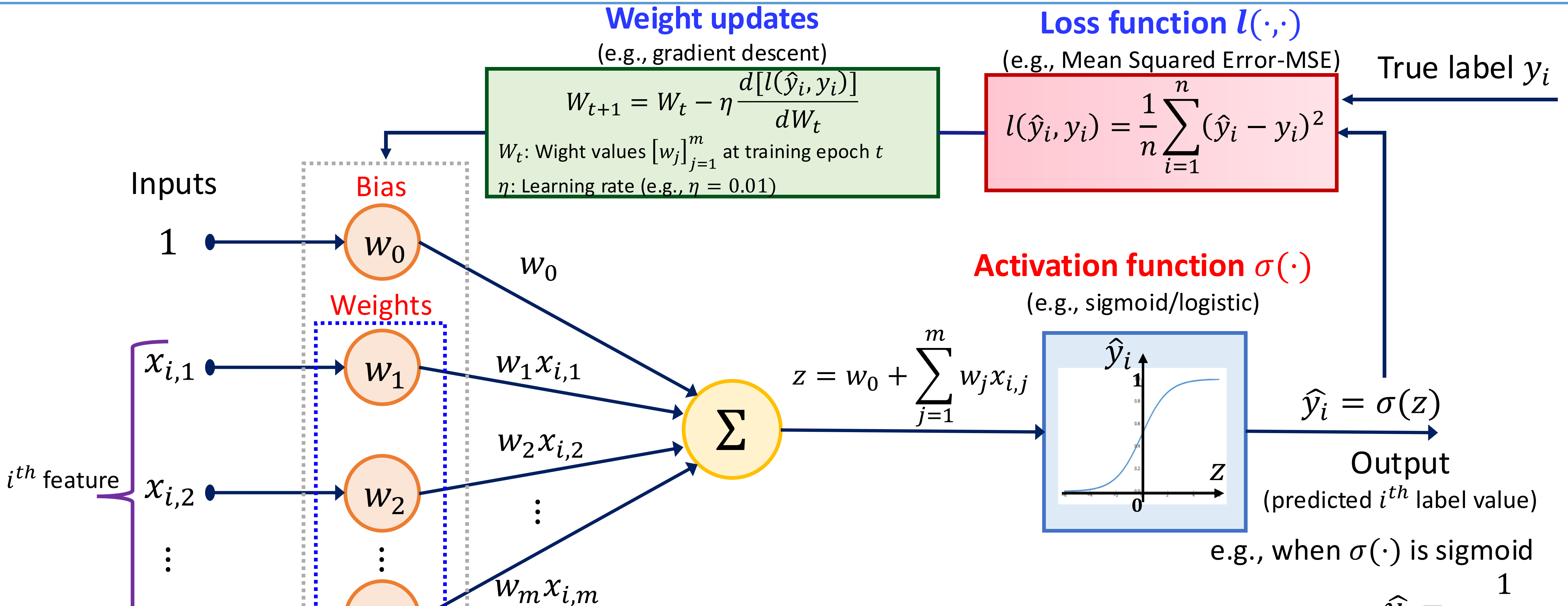
- **What parameters are we quantizing in DNN?**
  Model weights (bias , activation function)



***Example of 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

Structure of a Single Layer of Neural Network



w_0: **bias** term – models the value when all feature values $x_{i,j} = 0$ for $j = 1, \ldots, m$

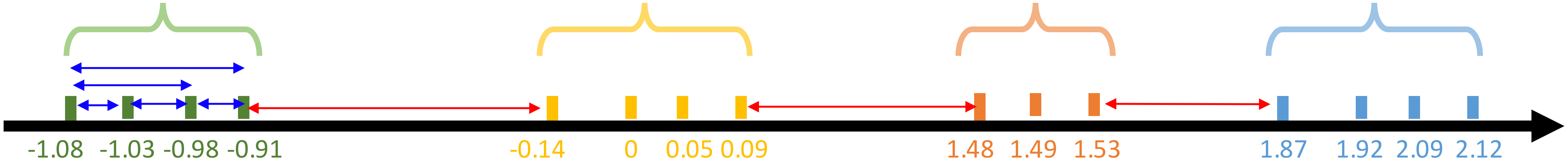
Viewing Grouping or Clustering as Quantization

- **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)

Cluster Representation of Weights



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 $2^2 = 4$ 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 **and N-bit quantized values**
  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

Cluster Representation of Weights

Quantization

Quantization in Neural Networks

1. Theory in Quantization

- K-mean based quantization
- Linear quantization

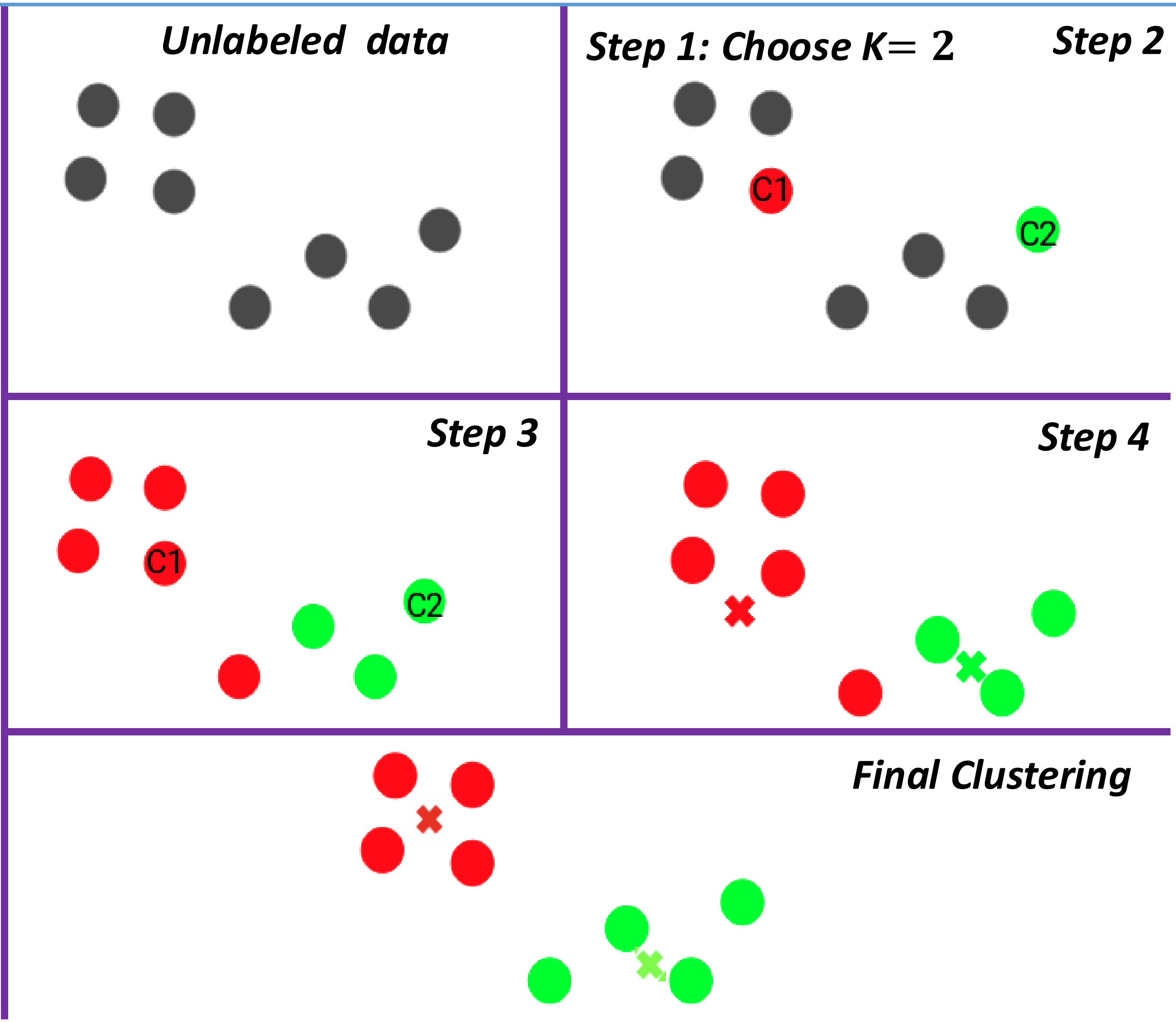
2. Post-Training Quantization (PTQ)

- Weight quantization
- Activation quantization
- Bias quantization

3. Quantization-Aware Training (QAT)

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

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

K-mean
Clustering

Clustered 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

Codebook

Quantization

Quantized weights
(2-bit int)

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

Cluster index (2-bit int)

Binary	Decimal
00	0
01	1
10	2
11	3

Can we reconstruct the weights?

How?

K-mean based Quantization: Reconstruction

Cluster centers — **codebook**

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

&

Quantized weights (2-bit int)

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

Reconstruction →

Reconstructed weights (32-bit float)

2.00	-1.00	1.50	0.00
0.00	0.00	-1.00	2.00
-1.00	2.00	0.00	-1.00
2.00	0.00	1.50	1.50

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

−

Reconstructed weights (32-bit float)

2.00	-1.00	1.50	0.00
0.00	0.00	-1.00	2.00
-1.00	2.00	0.00	-1.00
2.00	0.00	1.50	1.50

=

quantization error

0.09	0.02	-0.02	0.09
0.05	-0.14	-0.08	0.12
0.09	-0.08	0	-0.03
-0.13	0	0.03	-0.01

Source: Deep Compression [Han et al., ICLR 2016]

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

32 bit × 16
= 512 bit = 64 B

Cluster centers

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

Codebook

cluster index (2-bit int)

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

32 bit × 4
= 128 bit = 16 B **+** 2 bit × 16
= 32 bit = 4 B **=** 20 B

3.2 × smaller

Assume *N*-bit quantization, #parameters = *M* >> 2^N.

32 bit × *M*
= *32M* bit

N bit × *M*
= *NM* bit

32/*N* × smaller

Source: Deep Compression [Han et al., ICLR 2016]

Quantization

Quantization in Neural Networks

1. Theory in Quantization

- K-mean based quantization
- Linear quantization

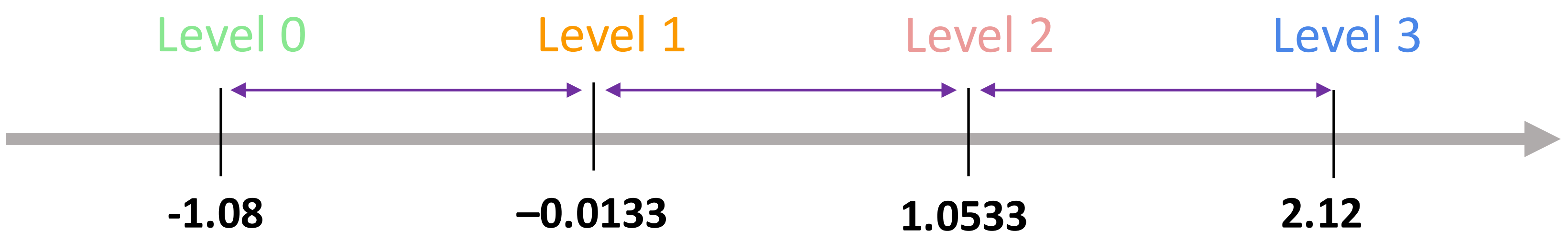
2. Post-Training Quantization (PTQ)

- Weight quantization
- Activation quantization
- Bias quantization

3. Quantization-Aware Training (QAT)

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.
  Range = 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 3 $(2^4 - 1)$ intervals.
  The interval size would be (2.12−(−1.08))/3 = 3.2/3 = 1.0667

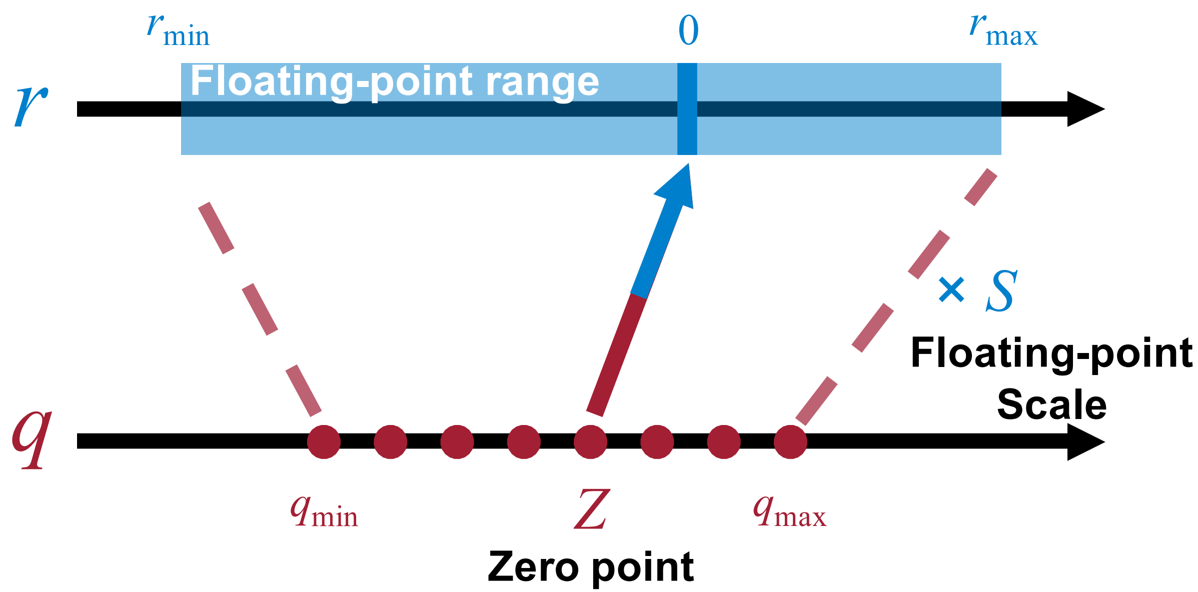


- **Quantize:** Assign each number to the nearest quantization level based on these intervals.

-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 $r = S(q - Z)$



$$r_{max} = S(q_{max} - Z)$$

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

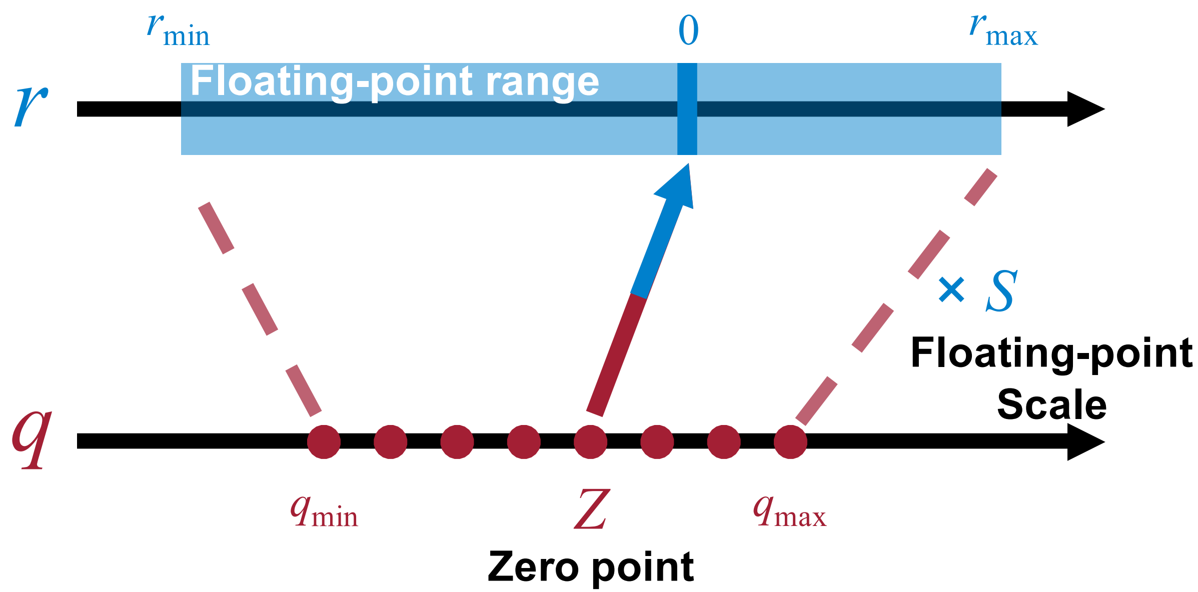
$$r_{max} - r_{min} = S(q_{max} - q_{min})$$

$$\boxed{S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}}$$

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

Linear Quantization : Formal Definition

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



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

$$\downarrow$$

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

$$\downarrow$$

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

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

Linear Quantization

$$
\underset{\text{weights (32-bit float)}}{\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix}}
\Rightarrow
\left(\underset{\text{quantized weights (2-bit signed int)}}{\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix}} - \underset{\text{zero point (2-bit signed int)}}{Z} \right) \times \underset{\text{scale (32-bit float)}}{S} = \underset{\text{reconstructed weights (32-bit float)}}{\begin{bmatrix} 2.14 & -1.07 & 1.07 & 0 \\ 0 & 0 & -1.07 & 2.14 \\ -1.07 & 2.14 & 0 & -1.07 \\ 2.14 & 0 & 1.07 & 1.07 \end{bmatrix}}
$$

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 $r = S(q - Z)$

weights (32-bit float) | quantized weights (2-bit signed int) | zero point (2-bit signed int) | scale (32-bit float) | reconstructed weights (32-bit float)

$$\begin{bmatrix} 2.09 & -0.98 & 1.48 & 0.09 \\ 0.05 & -0.14 & -1.08 & 2.12 \\ -0.91 & 1.92 & 0 & -1.03 \\ 1.87 & 0 & 1.53 & 1.49 \end{bmatrix} \Rightarrow \left(\begin{bmatrix} 1 & -2 & 0 & -1 \\ -1 & -1 & -2 & 1 \\ -2 & 1 & -1 & -2 \\ 1 & -1 & 0 & 0 \end{bmatrix} - \mathbf{-1} \right) \times \mathbf{1.07} = \begin{bmatrix} 2.14 & -1.07 & 1.07 & 0 \\ 0 & 0 & -1.07 & 2.14 \\ -1.07 & 2.14 & 0 & -1.07 \\ 2.14 & 0 & 1.07 & 1.07 \end{bmatrix}$$

$$r = (q - Z) \times S$$

Floating-Point (r) | **Integer** (q) | **Integer** (Z) | **Floating-Point** (S)

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

S:
- quantization parameter

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

Linear Quantization

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

⇨ (

quantized weights (2-bit signed int)

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

− **-1**) × **1.07** =

zero point (2-bit signed int): -1

scale (32-bit float): 1.07

reconstructed weights (32-bit float)

2.14	-1.07	1.07	0
0	0	-1.07	2.14
-1.07	2.14	0	-1.07
2.14	0	1.07	1.07

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

−

reconstructed weights (32-bit float)

2.14	-1.07	1.07	0
0	0	-1.07	2.14
-1.07	2.14	0	-1.07
2.14	0	1.07	1.07

=

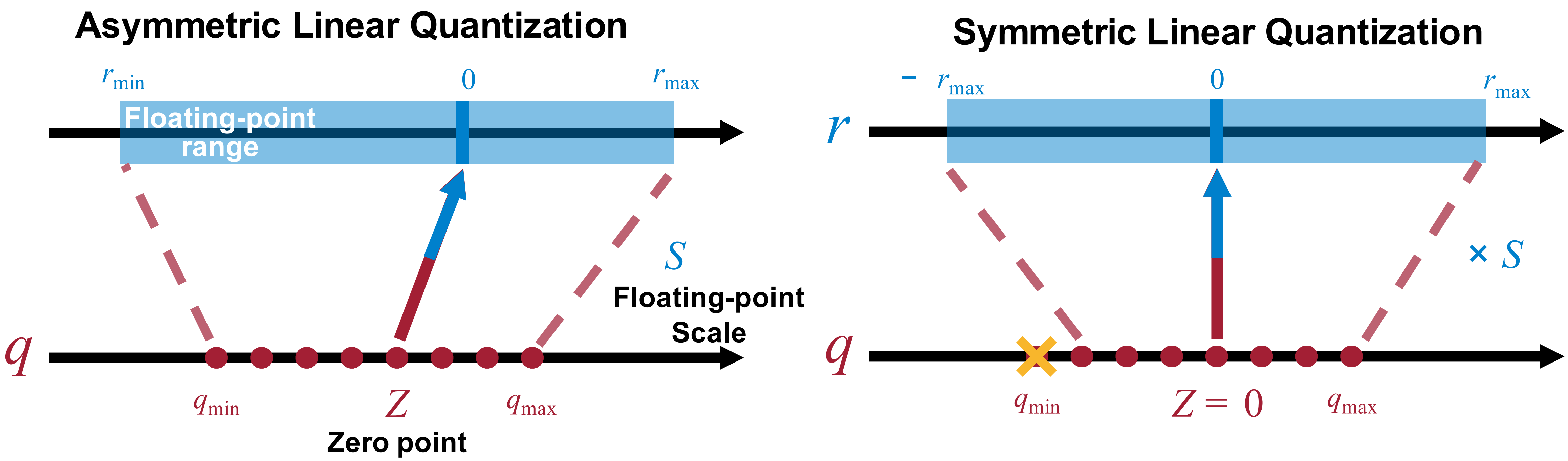
quantization error

-0.05	0.09	0.41	0.09
0.05	-0.14	-0.01	-0.02
0.16	-0.22	0	0.04
-0.27	0	0.46	0.42

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

Linear Quantization: Asymmetric vs. Symmetric

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



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

- 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

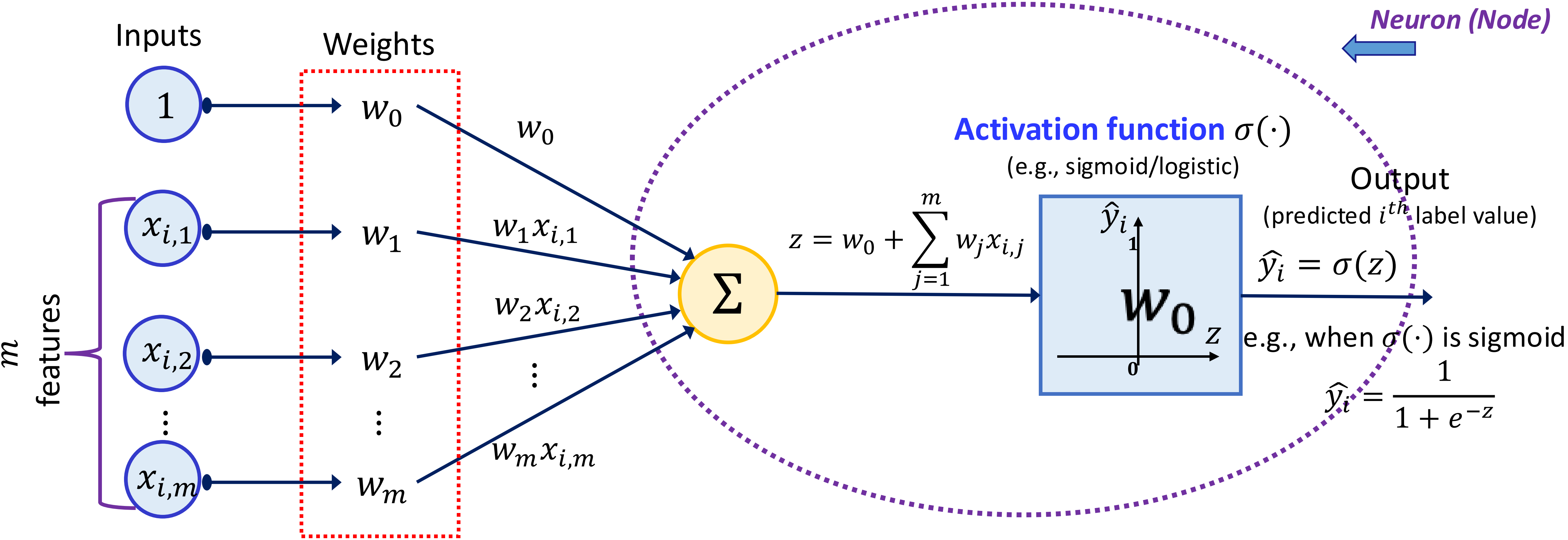
Quantization in Neural Networks

1. Theory in Quantization
   - K-mean based quantization
   - Linear 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)



w_0: **bias term** – models the value when all feature values $x_{i,j} = 0$ for $j = 1, \ldots, m$

Quantization

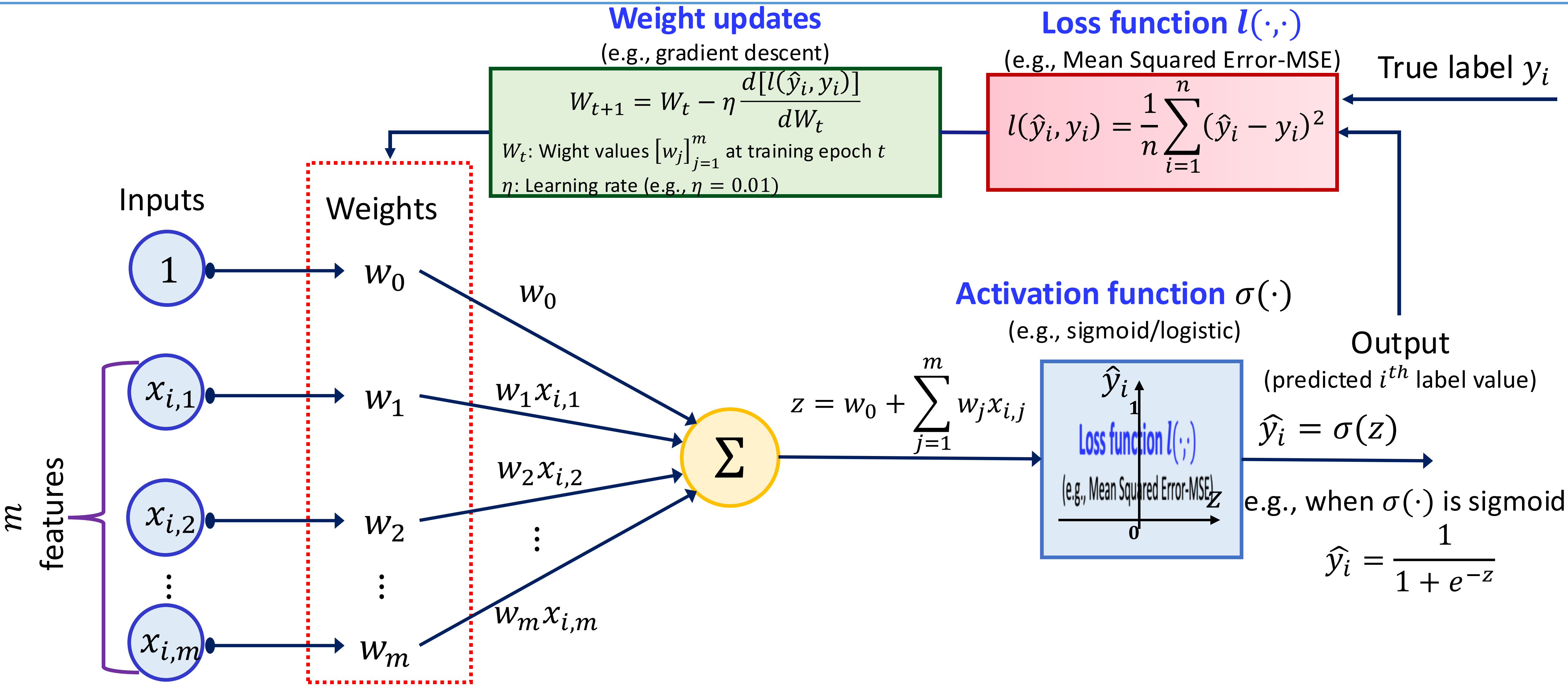
Quantization in Neural Networks

1. Theory in Quantization
    - K-mean based quantization

2. Post-Training Quantization (PTQ)
    - Weight quantization
    - Activation quantization
    - Bias quantization

3. Quantization-Aware Training (QAT)
– How should we improve the performance of quantized models?

Quantization-Aware Training (QAT)



Quantization-Aware Training (QAT)

- Quantization-Aware Training = **simulate int8 quantization during training**
- Model **learns to adapt to quantization (rounding/clipping) errors**
- Leads to high-accuracy int8 models

1. Start with float32 model
2. Insert fake quant ops
3. Train with QAT
4. Convert to real int8 model for deployment

QAT Workflow

Quantization-Aware Training

1. **Forward Pass:**
    - Float32 values are **fake quantized**:
        - Quantized: $q = \text{round}((x - z)/s)$
        - Dequantized: $\hat{x} = q \cdot s + z$
    - The model uses $\hat{x}$ (the **reconstructed value**) for computations.
2. **Backward Pass:**
    - Gradients are computed **with respect to** $\hat{x}$.
    - This allows smooth gradient flow even though quantization involves rounding and clipping (non-differentiable operations).

Quantization

What algorithms to choose to improve accuracy?

Post-Training Quantization vs Quantization-Aware Training

✓ : Pros

✗ :Cons

Post-Training Quantization (PTQ)

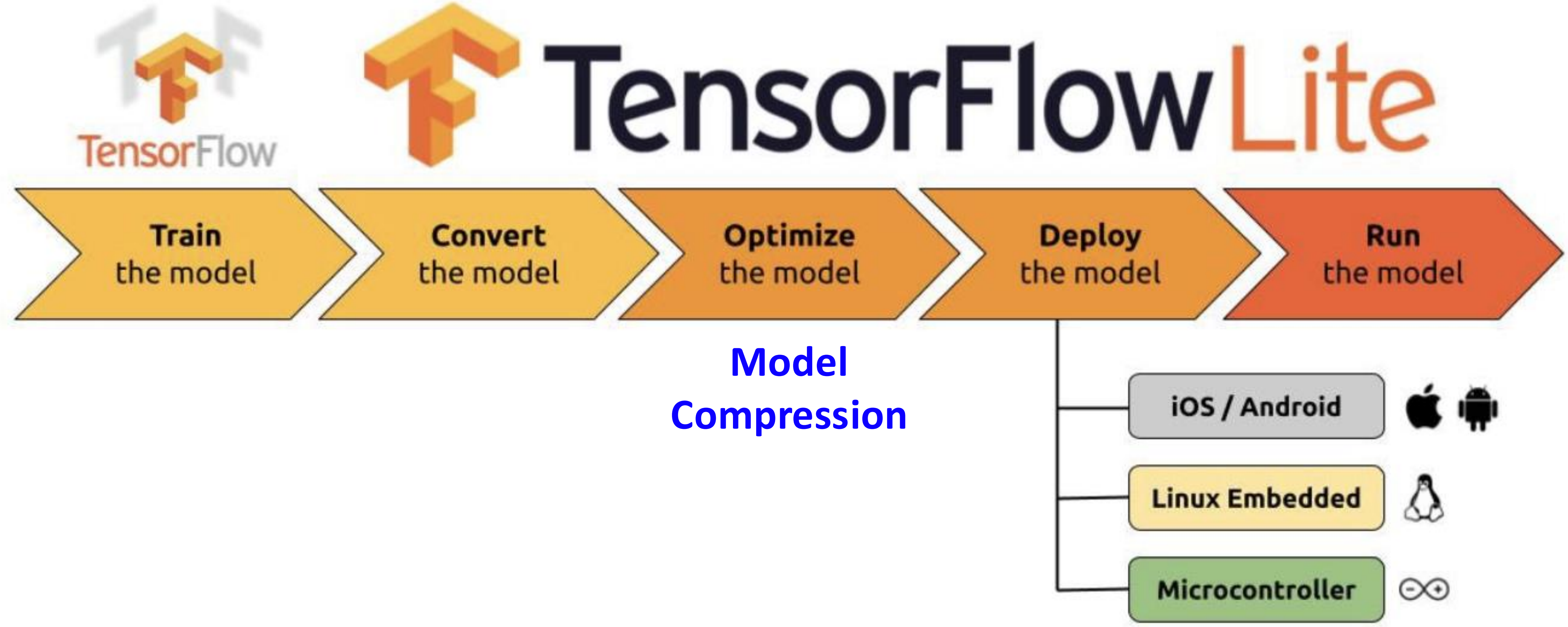
- ✓ Takes a pre-trained network and converts it to a fixed-point network without access to the training pipeline
- ✓ Data-free or small calibration set needed
- ✓ Use though single API call
- ✗ Lower accuracy at lower bit-widths

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/

TensorFlow (TF) and TFLite Workflow for TinyML



Source: https://leonardocavagnis.medium.com/tinyml-machine-learning-for-embedded-system-part-i-92a34529e899

Quantization in TensorFlow Lite

- Quantization in TF Lite
  - Integer Quantization
  - Dynamic Range Quantization
  - Float 16 Quantization

Integer Quantization

- 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.

- **Requires a calibration dataset**: A small, representative dataset is used to determine the range of activations for accurate quantization.

- Results in a **smaller model and increased inferencing speed**, which is valuable for low-power devices such as microcontrollers.

Inference Under Integer Quantization

1. **Input (int8)**
2. **Weights (int8)**
3. **Computation (e.g., Conv or MatMul) → int32 output**
    - Because `int8 × int8 = int32`

Example

$$-128 \times -128 = 16384$$

$$127 \times 127 = 16129$$

These results **exceed the maximum range of int8,** which is only **–128 to +127**.

So if you stored the result in int8, you'd get **overflow** and wrong results.

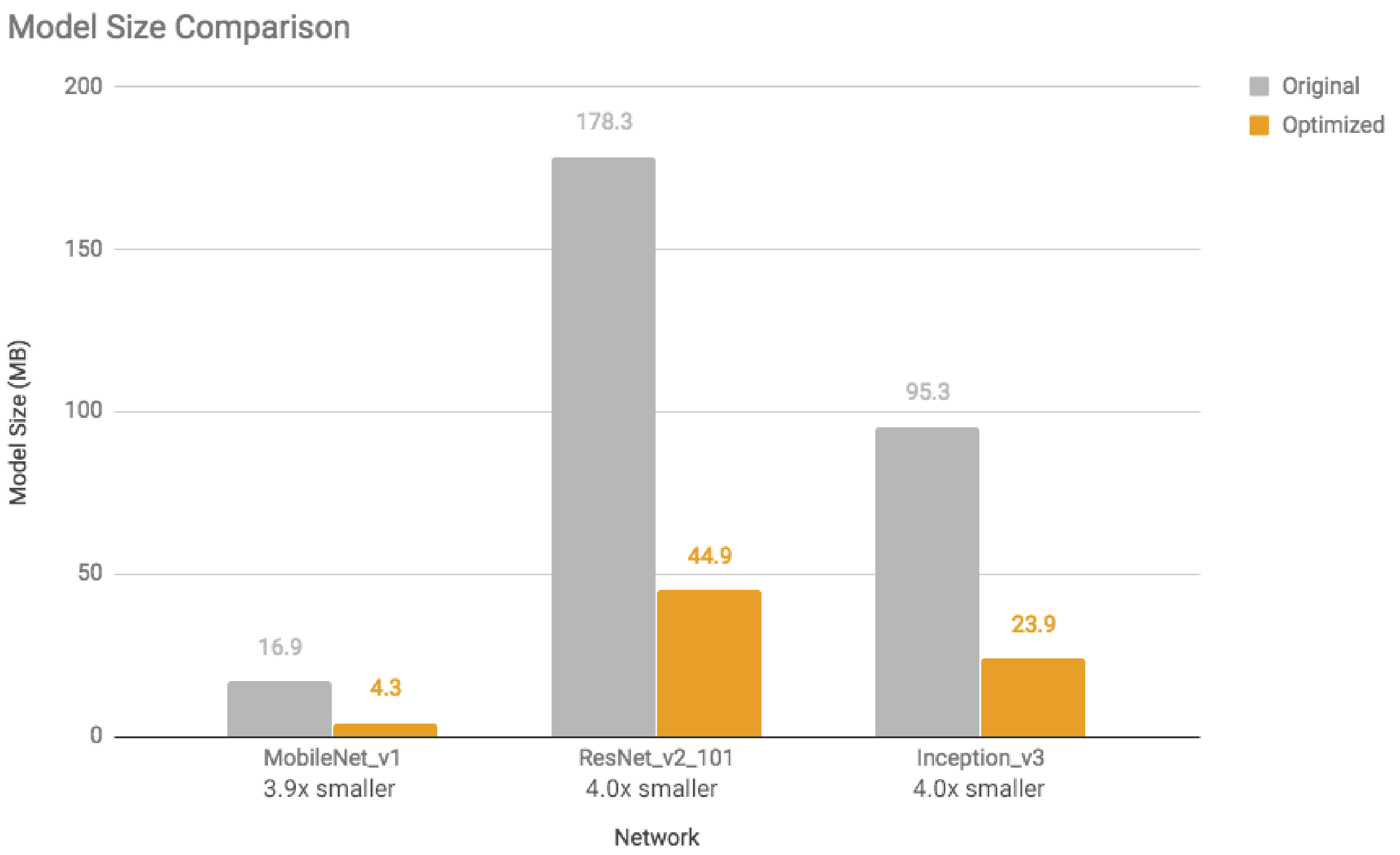
4. **Bias (int32 or float32)** is added
5. **Requantization:**
    - The int32 result is scaled and shifted to **int8**
    - This is a **single fused integer operation** using precomputed scale and zero-point

$$y_{\text{int8}} = \text{round}\left(y_{\text{int32}} \times \text{scale}_{\text{new}}\right) + \text{zero\_point}$$

6. **Output (int8)** → passed directly to the next layer

Integer Quantization

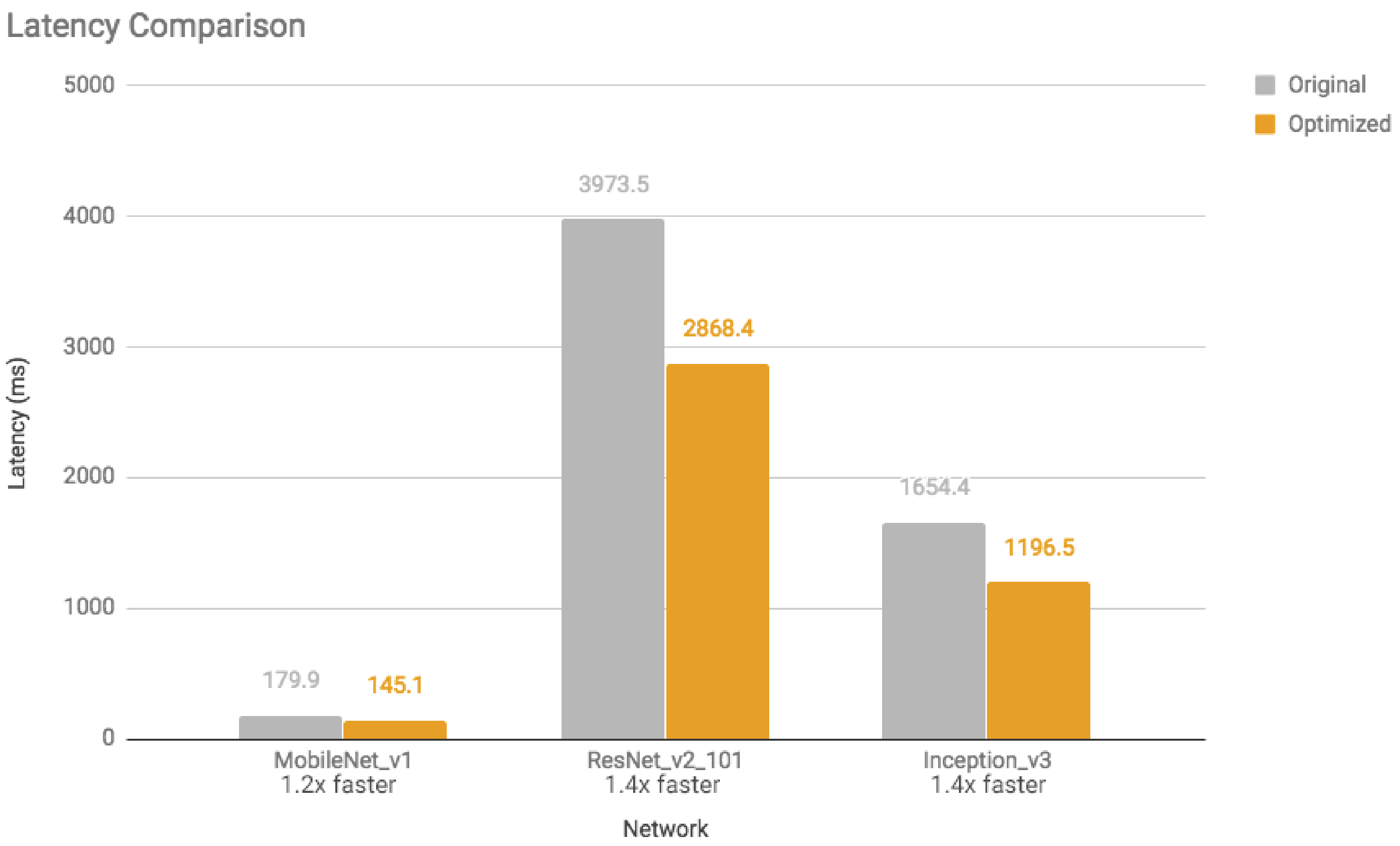
Performance: Model Size



Source: https://blog.tensorflow.org/2018/09/introducing-model-optimization-toolkit.html

Integer Quantization

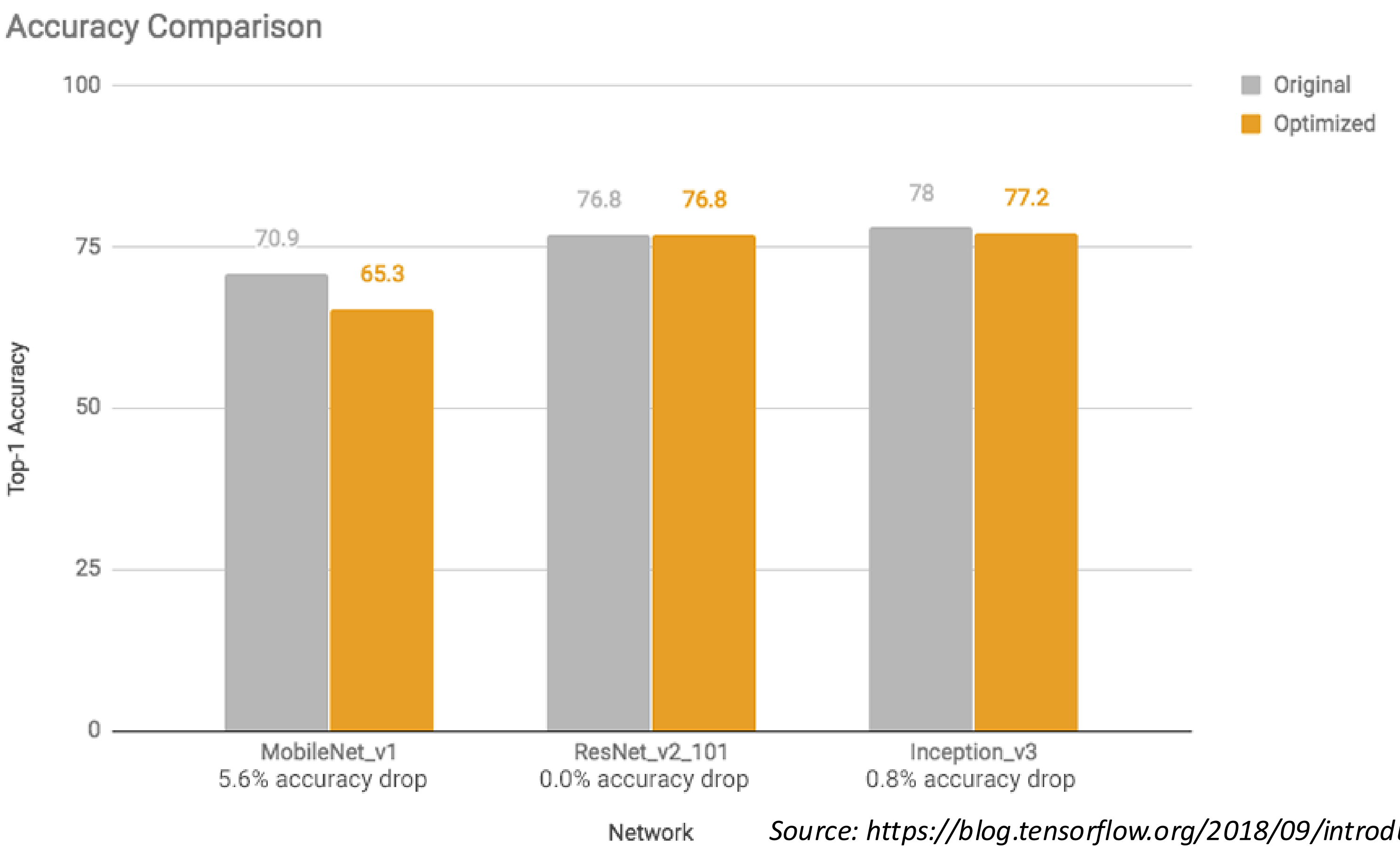
Performance: Latency



Source: https://blog.tensorflow.org/2018/09/introducing-model-optimization-toolkit.html

Integer Quantization

Performance: Accuracy



Source: https://blog.tensorflow.org/2018/09/introducing-model-optimization-toolkit.html

Dynamic Range Quantization

- **Converts only weights to 8-bit precision**.
- **Keep activations in float32 during inference**. Unlike full int8 quantization, activations are **not quantized.**
- **No calibration data needed**: Since activations are not quantized, there's **no need to determine activation ranges or scale them**.
- **Inference computation uses float32 activations and dequantized weights**: Quantized weights are **converted back to float32** before computations.
- **Achieves a 4× reduction in model size** (for the weights only), while maintaining nearly the same accuracy as the original float32 model.

Inference Under Dynamic Range Quantization

1. **Input (float32)** — Input remains in float32; no quantization is applied.
2. **Weights (int8)**
    - Weights are stored in int8 format.
    - They are dequantized before use:

$$w_{\text{float}} = (w_{\text{int8}} - \text{zero\_point}) \times \text{scale}$$

3. **Computation (e.g., Conv or MatMul) → float32 output**
    - Performed in float32 using dequantized weights and float32 inputs:

$$y = x_{\text{float}} \cdot w_{\text{float}} + b_{\text{float}}$$

4. **Bias (float32)** — Bias is kept and used directly in float32.
5. **Output (float32)** — Output remains in float32 and is passed to the next layer.

Float-16 Quantization

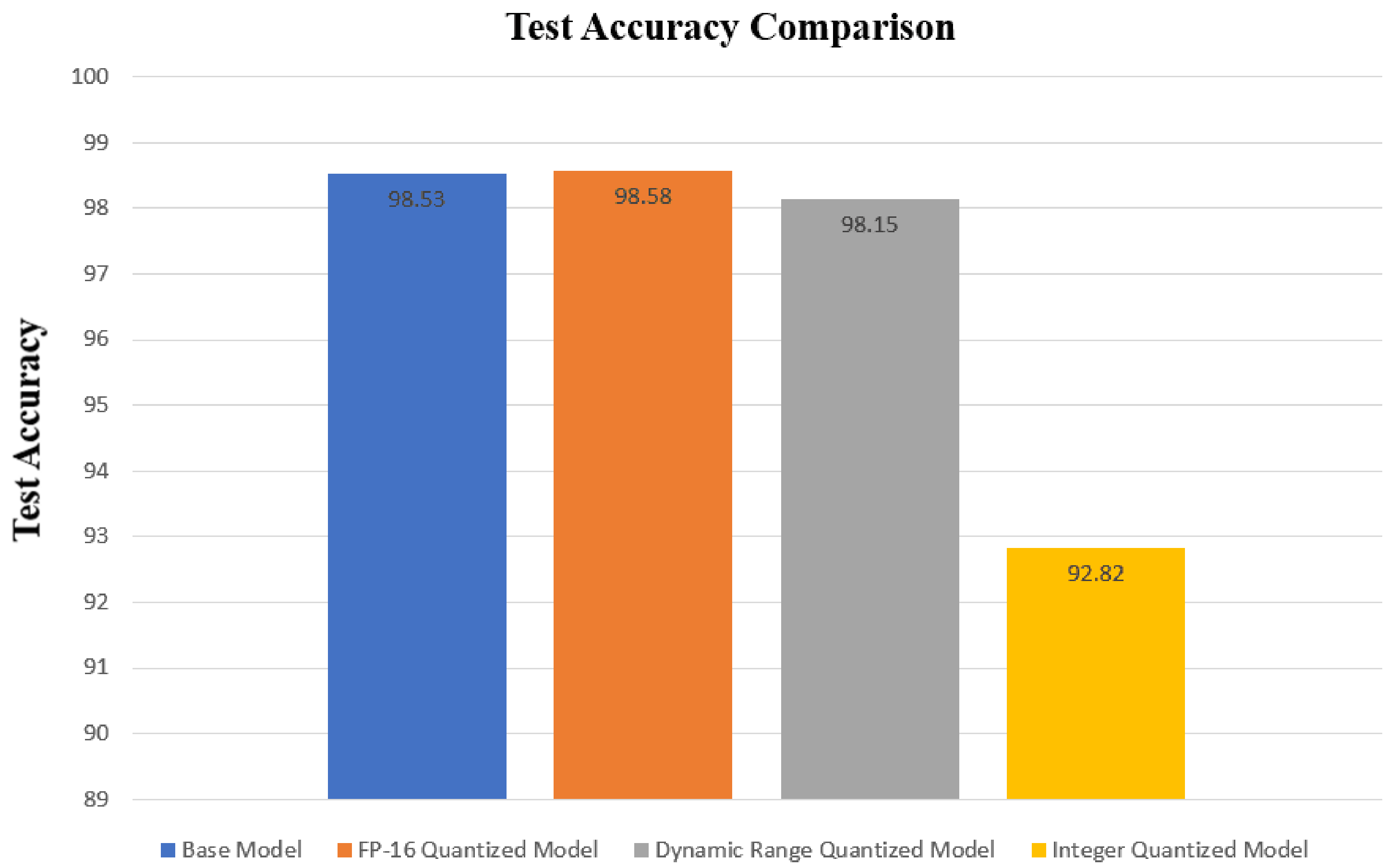
- **Convert weights to 16-bit floating point values** during model conversion from TensorFlow to TF Lite's flat buffer format.

Original float32 Weights		**Applying Float16 Quantization**
[0.25890732, -1.4732046, 0.9910725]	→	[0.2589, -1.473, 0.9911]

- 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

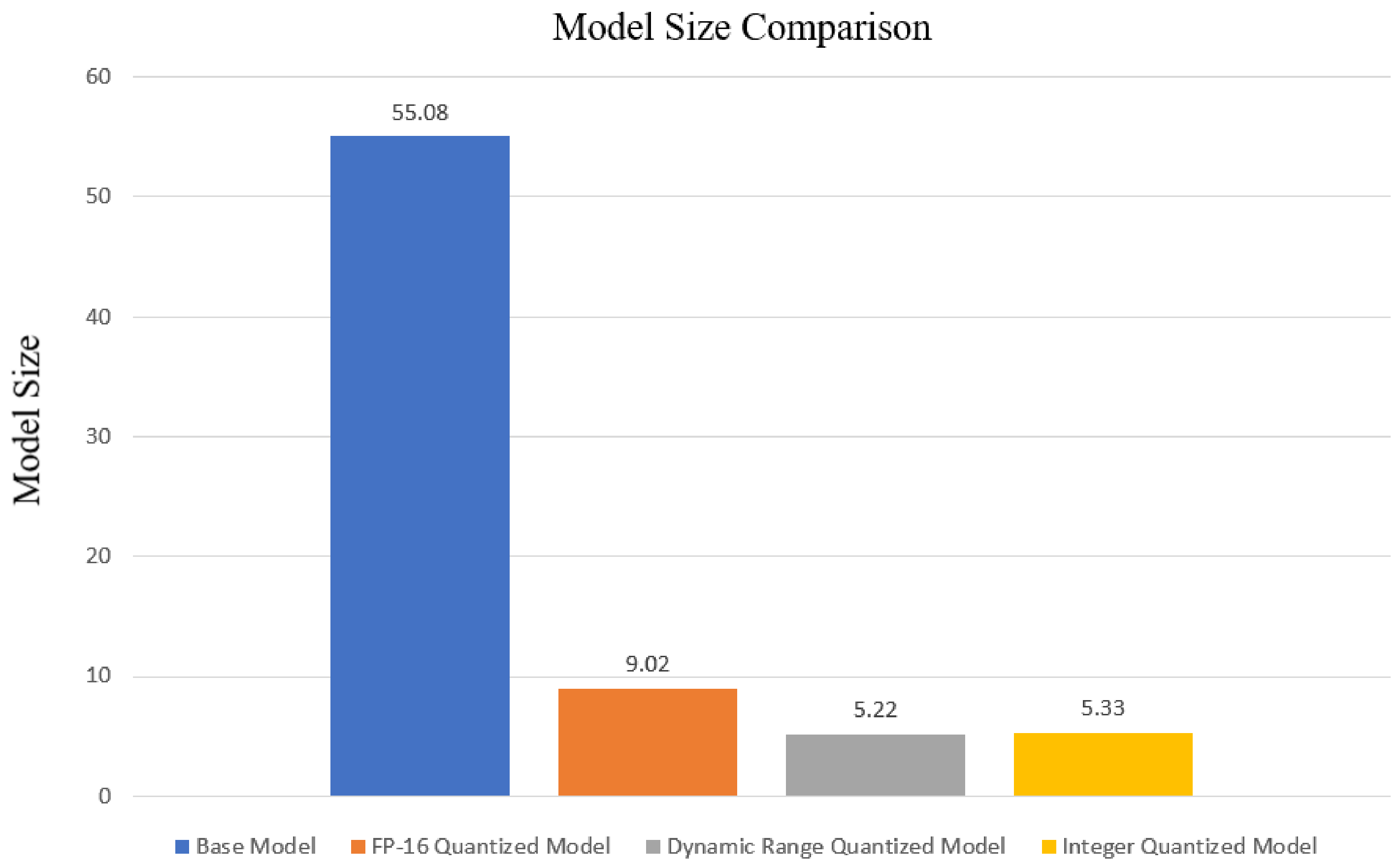
Among Integer & Dynamic Range & Float 16 Quantization



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

Quantization in TinyML: Comparison

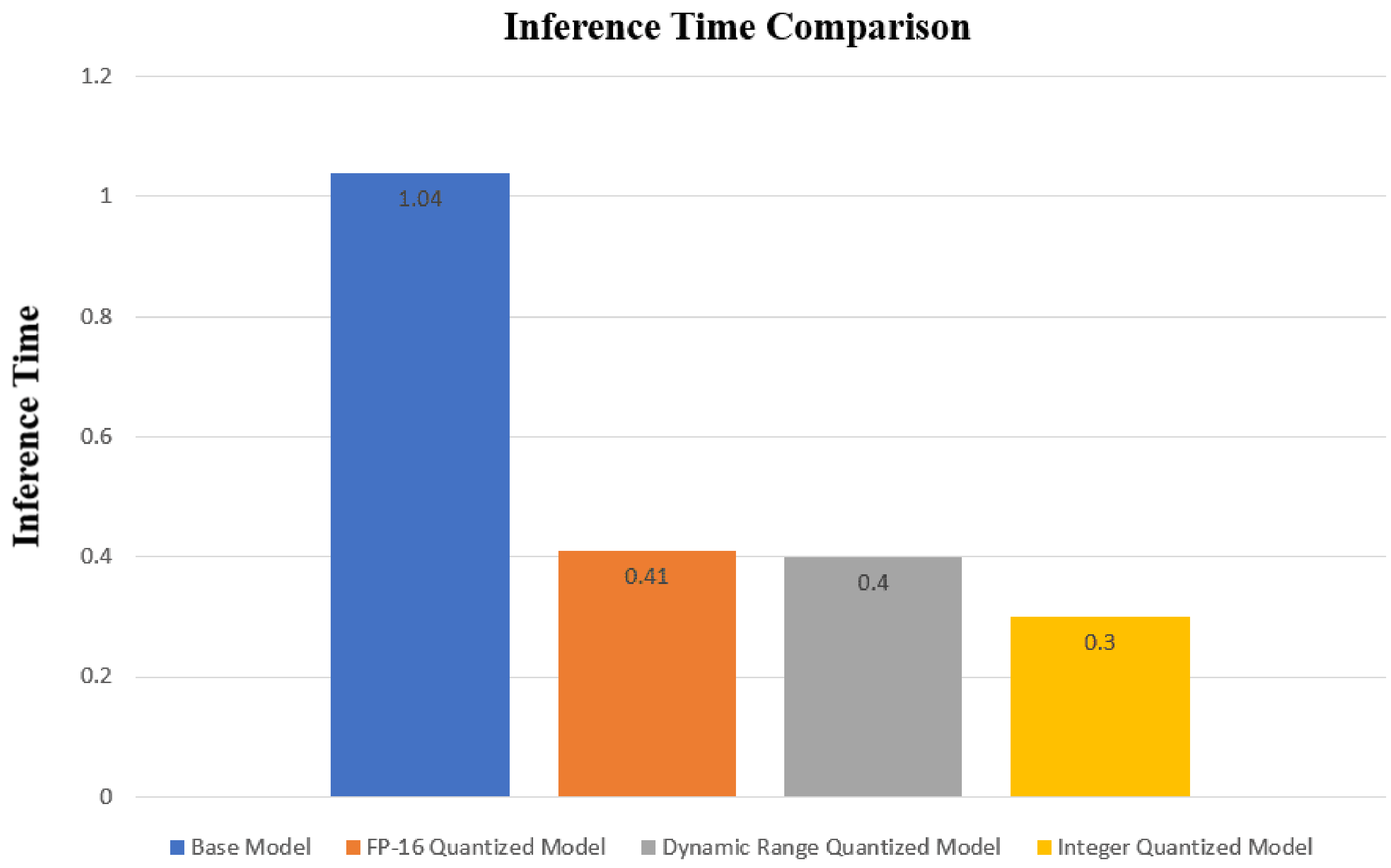
Among Integer & Dynamic Range & Float 16 Quantization



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/

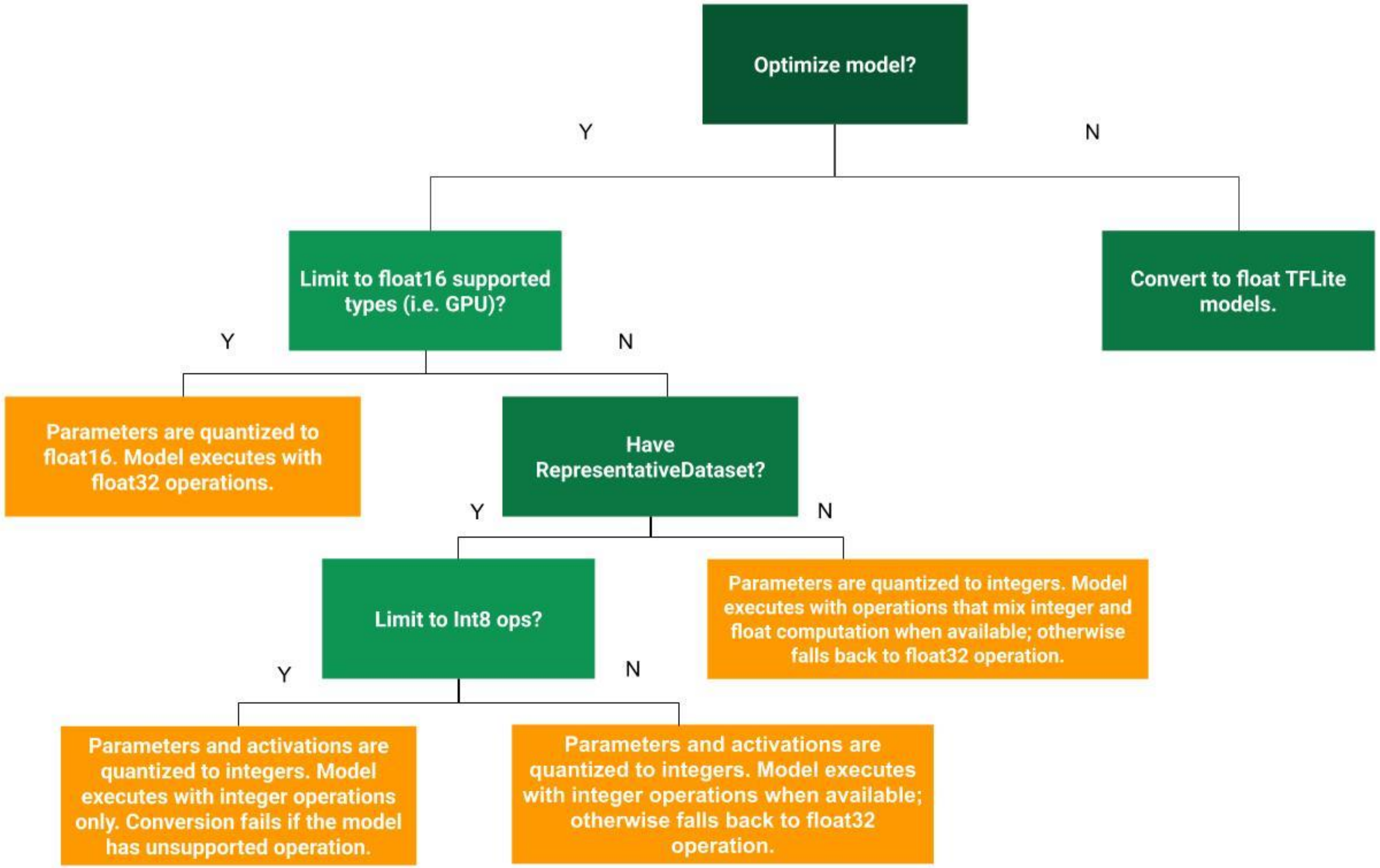
Quantization in TinyML: Summary

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

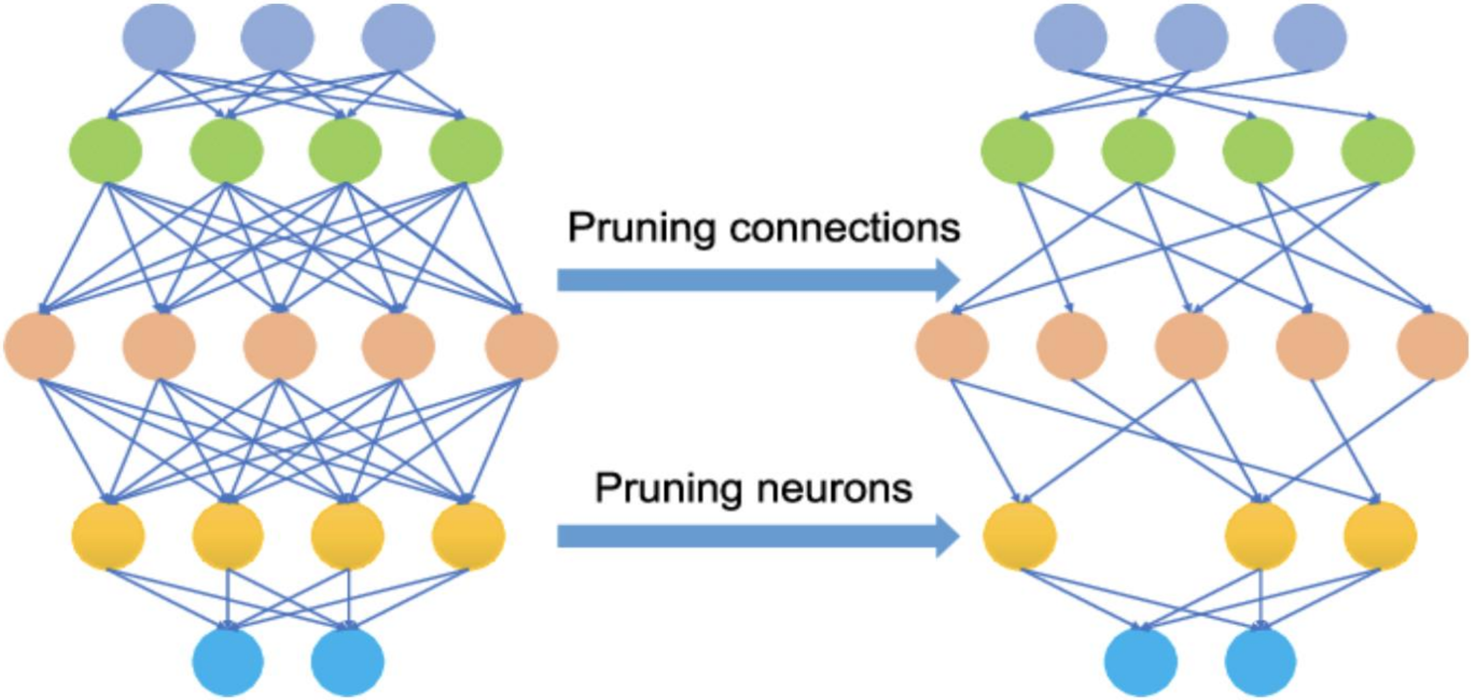
Quantization in TensorFlow Lite

Determine which post-training quantization method is best



Model Compression: Pruning

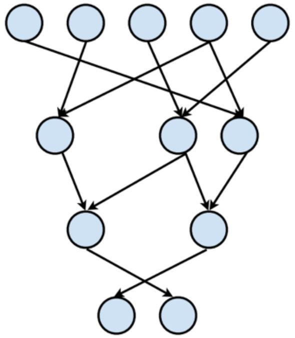
- First creates a **large model** that achieves the **required accuracy**
- Then try and **decrease the size** (prune it) **while trying to maintain the accuracy**
- Once the model has been pruned, this **new pruned model** is **trained again** on the dataset and this is known as **iterative pruning**



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

Train Weights

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

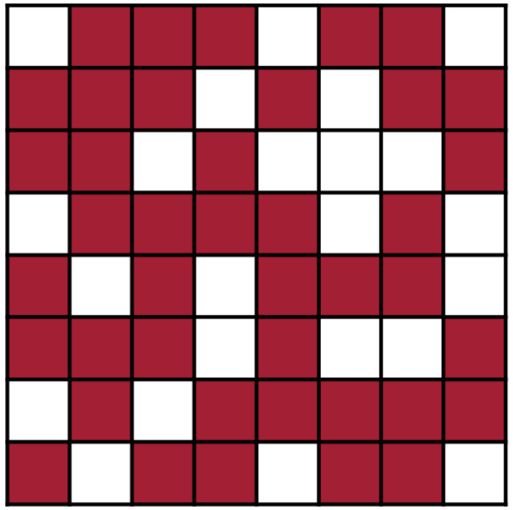
Pruned 90%: a man in a wetsuit is riding a wave on a beach

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.

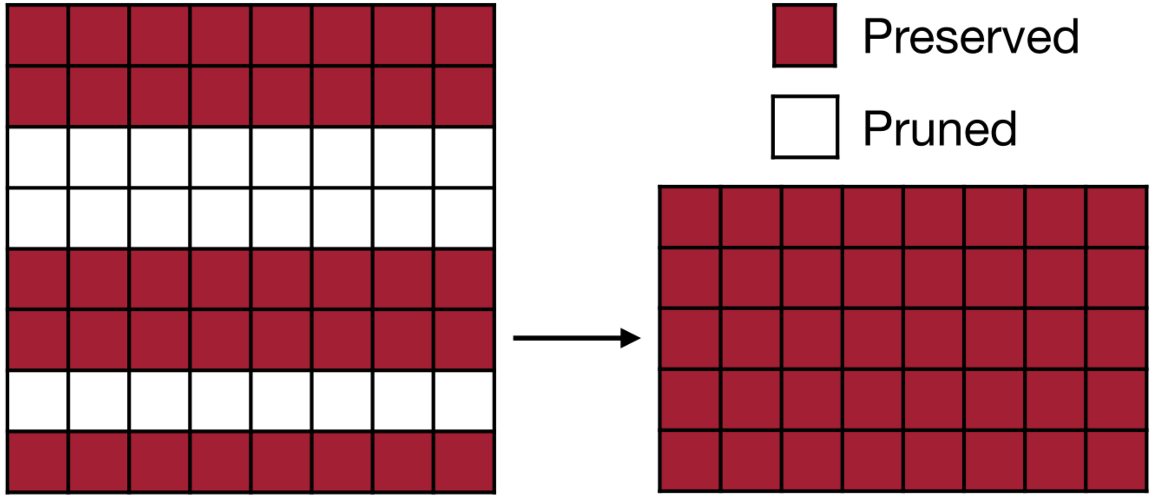
Pruning Techniques

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)

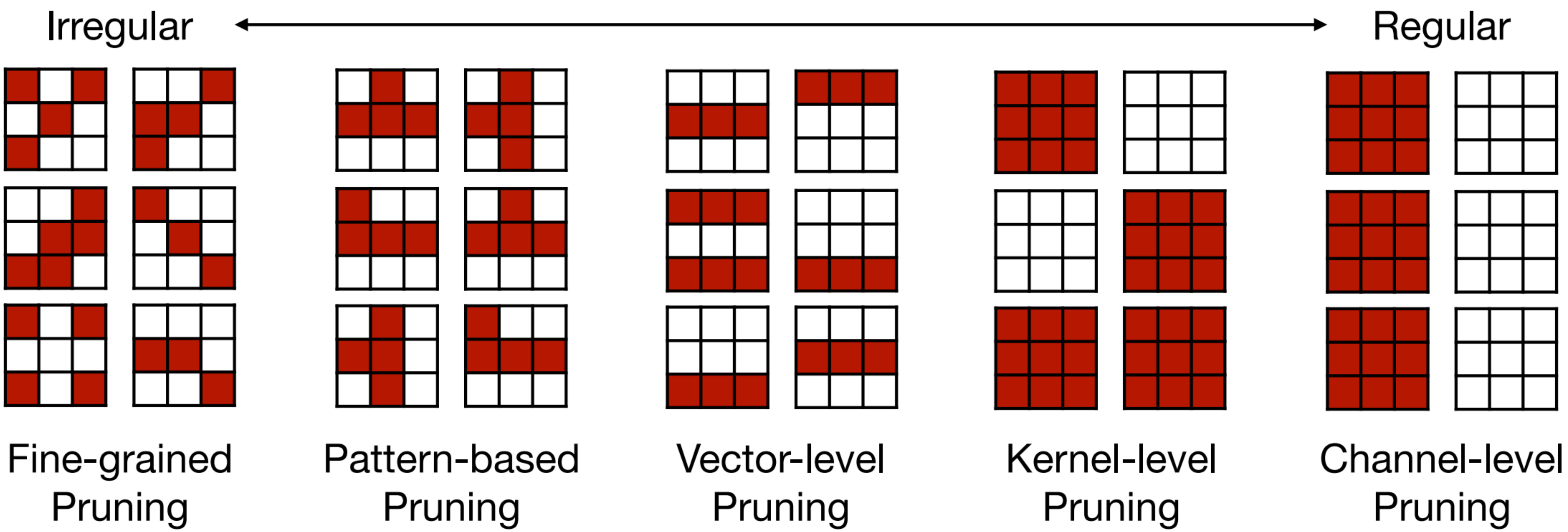


Coarse Grained/ Structured Pruning

- 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

Pruning in LLMs

The Unreasonable Ineffectiveness of the Deeper Layers

Andrey Gromov* Meta FAIR & UMD
Kushal Tirumala* Meta FAIR
Hassan Shapourian Cisco
Paolo Glorioso Zyphra

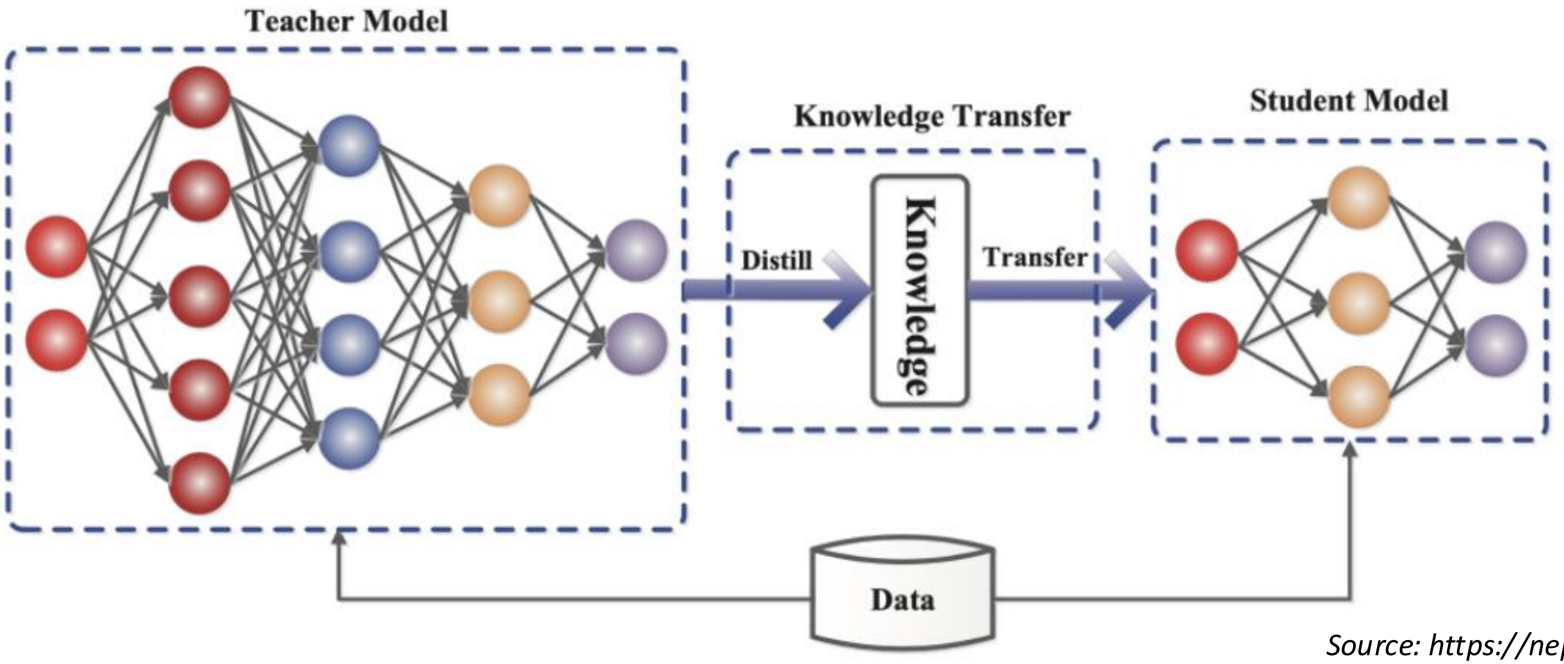
Daniel A. Roberts MIT & Sequoia Capital

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.

Model Compression: Knowledge Distillation

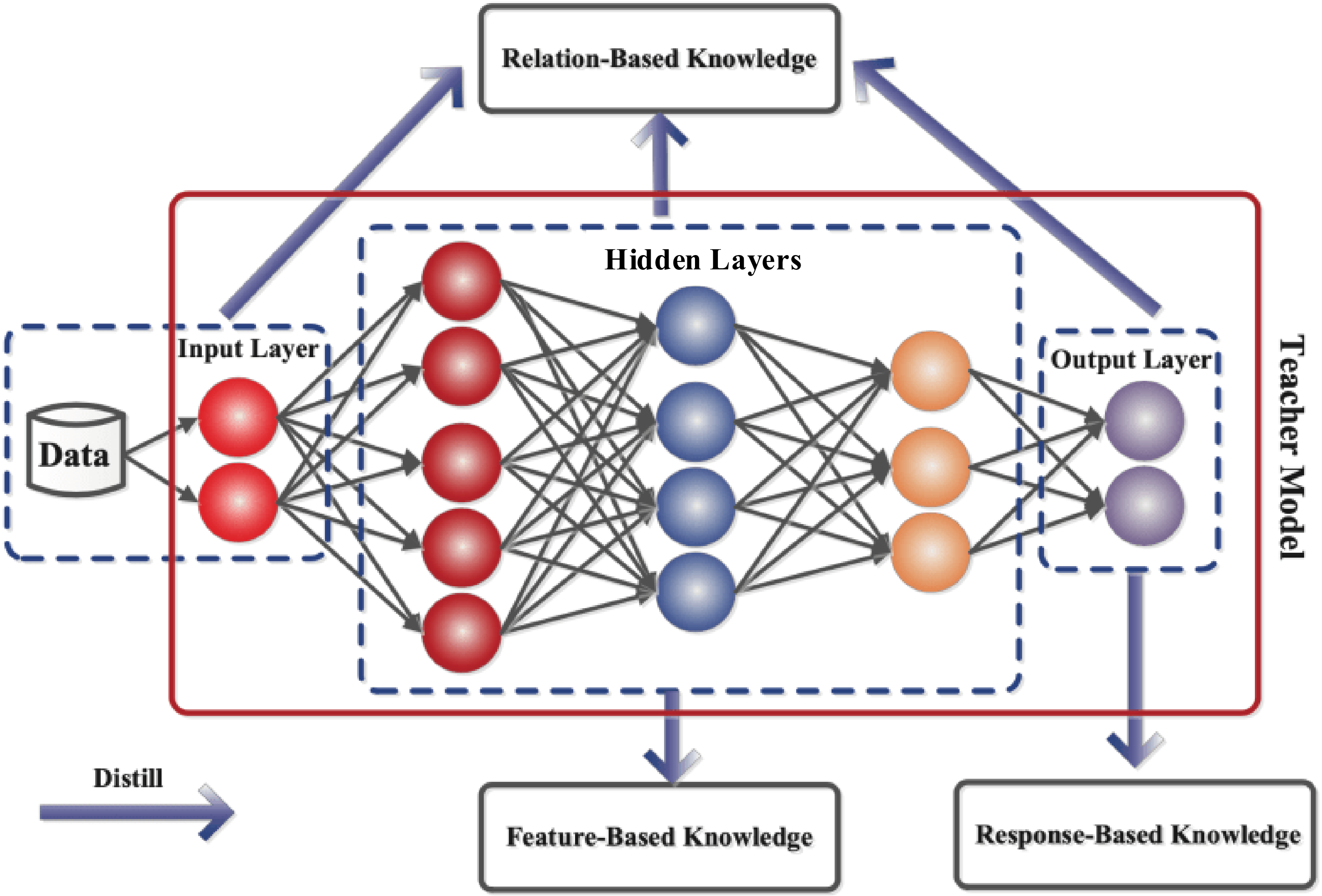
- Effectively training a **bigger ML model**, or the **TEACHER**, with a given dataset.
- The model would generate predictions based on the given data.
- A **new dataset** would be constructed which would **combine the old data and the information generated from the bigger model**.
- This new dataset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



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

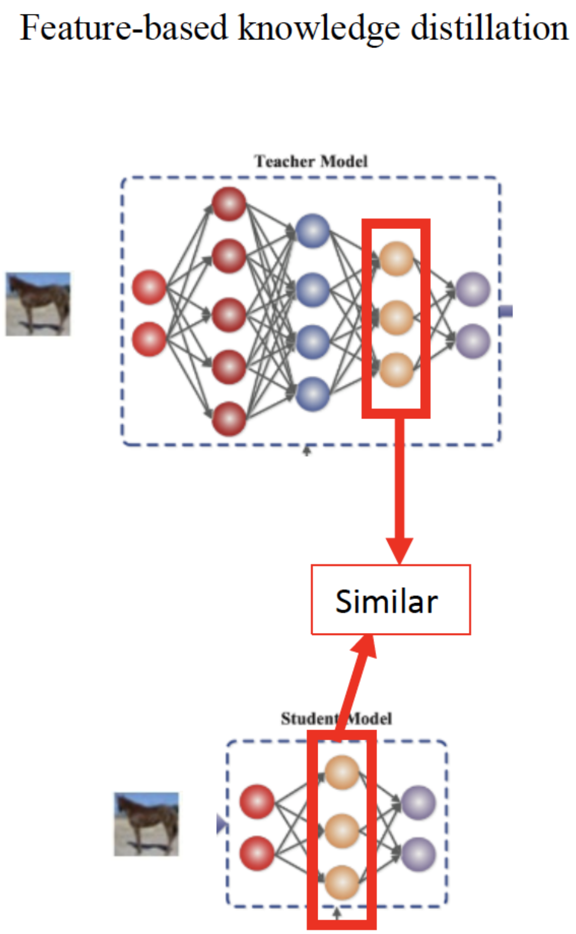
Knowledge Distillation Types

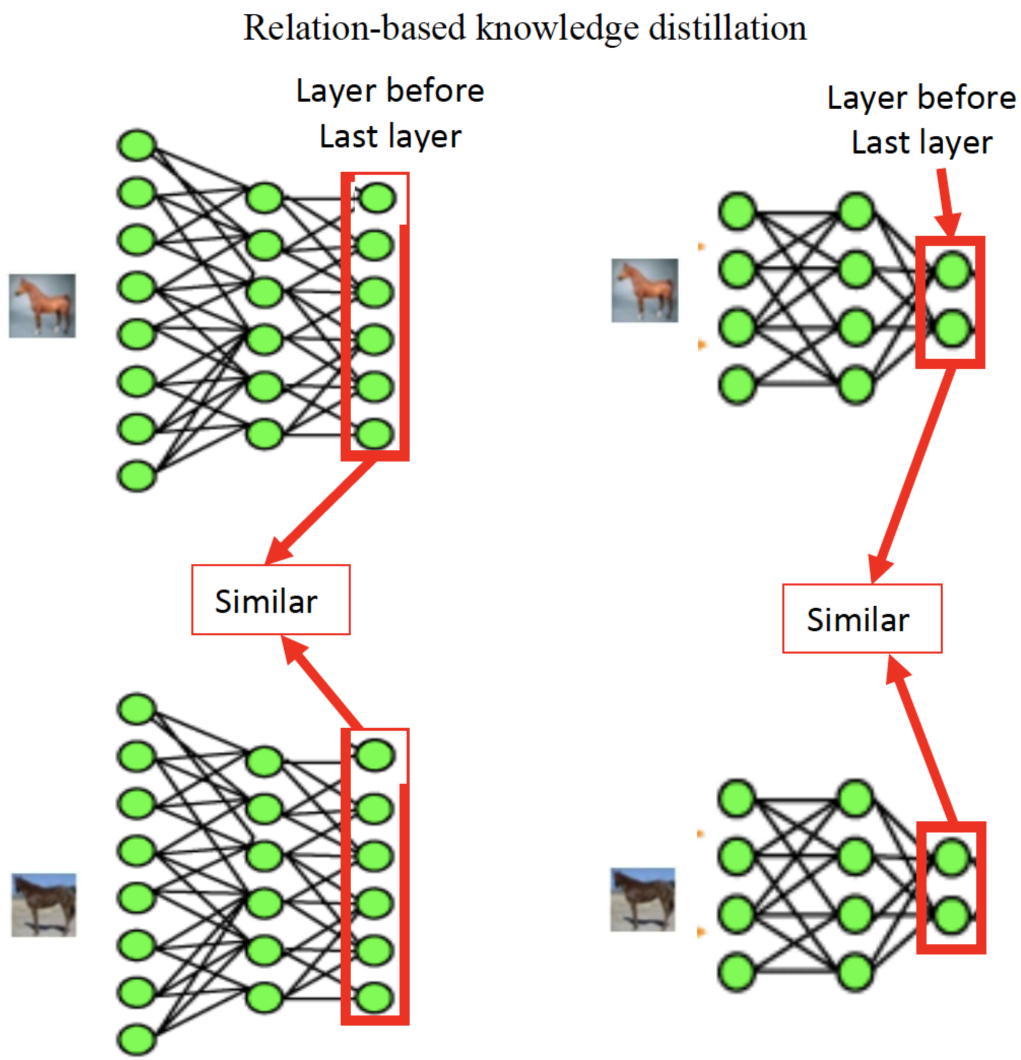
- **Feature-Based Knowledge**: Matching intermediate representations/features
- **Response-Based Knowledge**: Matching output logits or probabilities
- **Relation-Based Knowledge**: Matching the relational structure between data points or features.
  - i.e., if two inputs are deemed similar by the teacher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



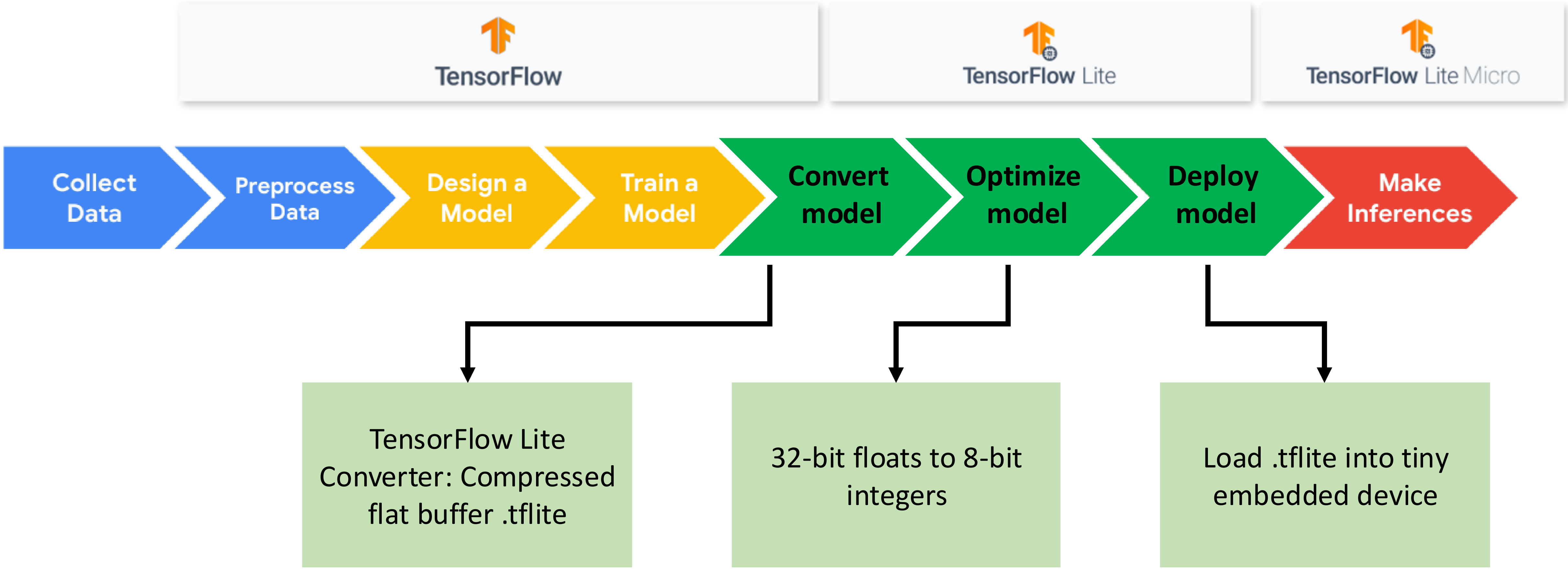
Source: https://neptune.ai/blog/knowledge-distillation

Knowledge Distillation: Feature-based Vs. Relation-based

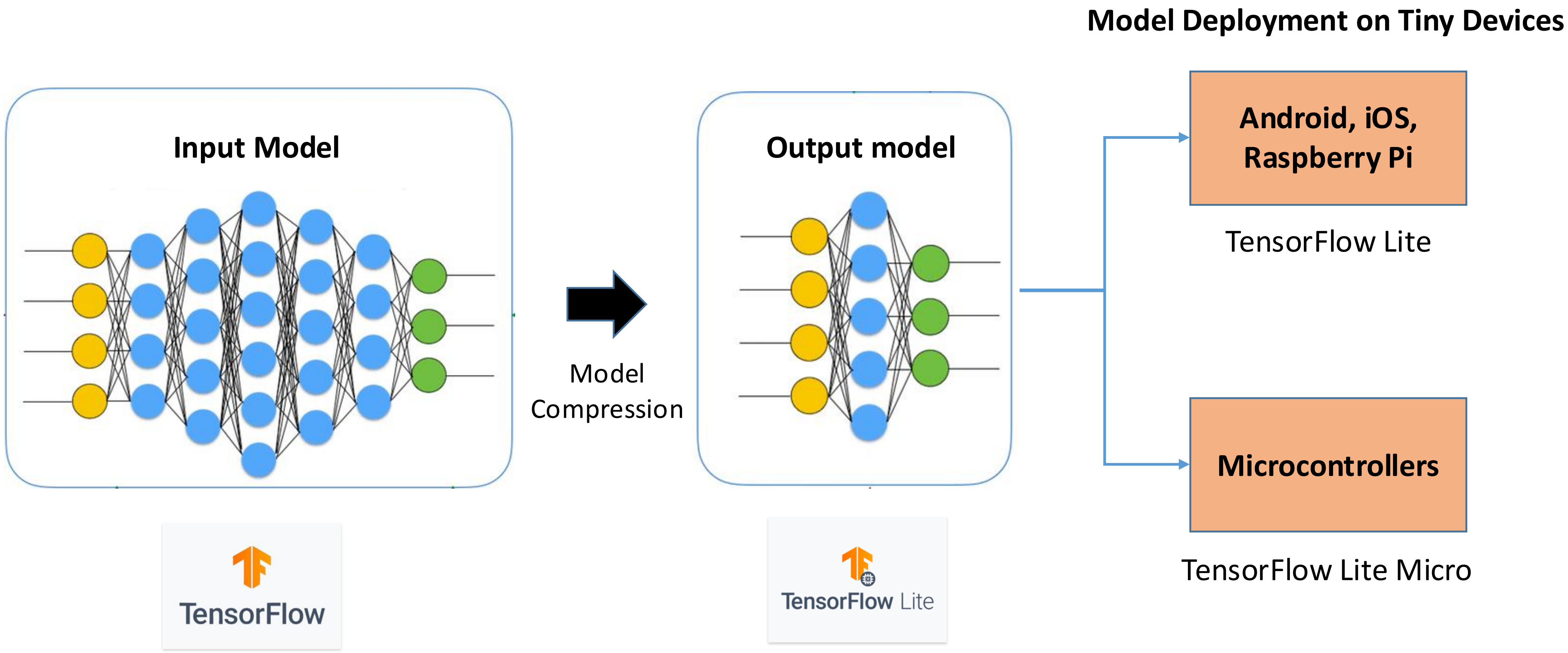




Putting it all together – TF -> TF Lite for TinyML



TFLite: Model Compression and Deployment

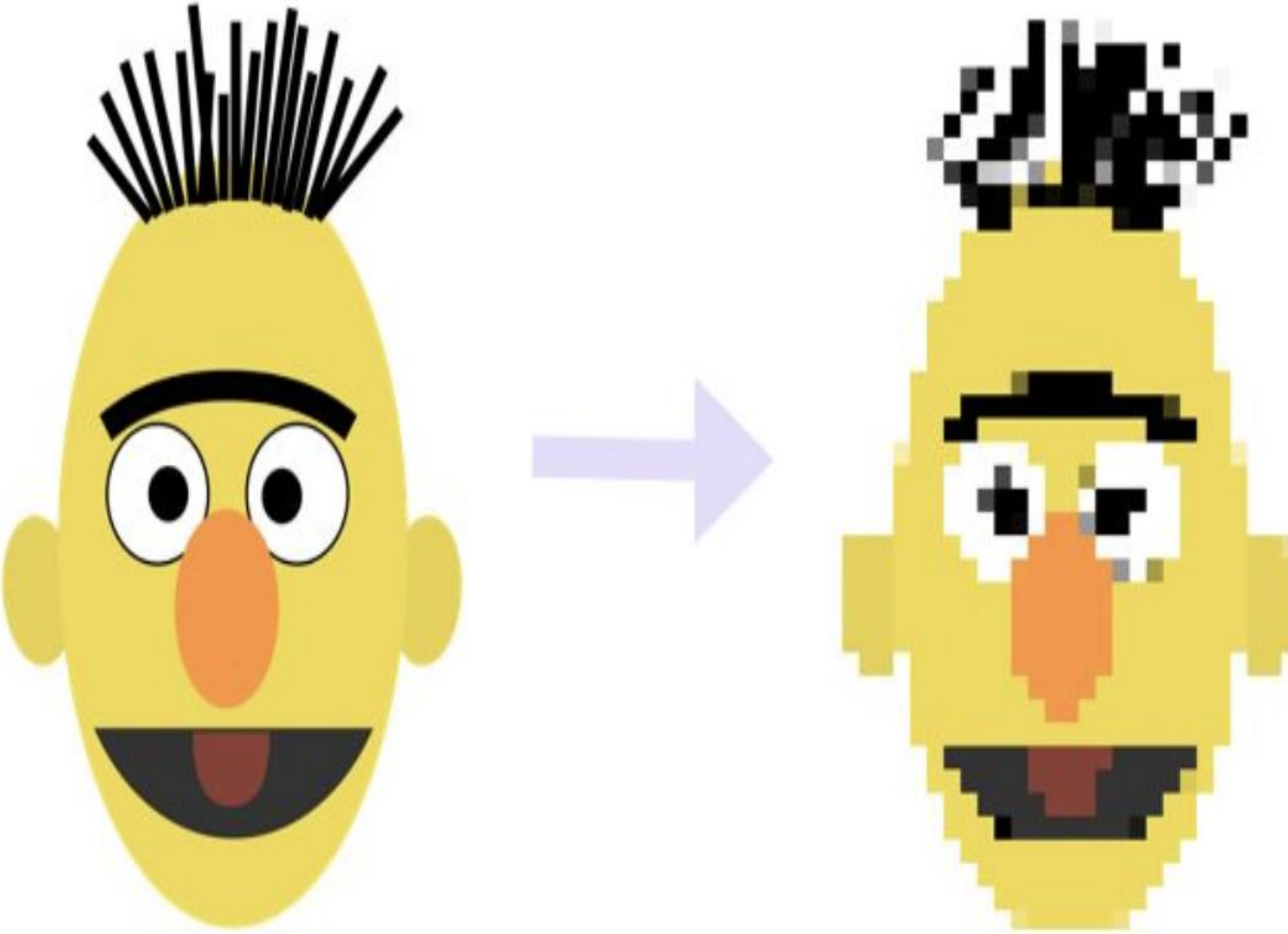


TF vs. TFLite – Summary of Key Differences

	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

Coding Session

- Training a simple CNN model using MNIST Dataset
- Quantizing CNN model
- Pruning CNN model
- Applying KD on CNN
- Evaluating the performance of compressed CNN Models



Road Map

1. Installing and Configuring Needed Python Libraries

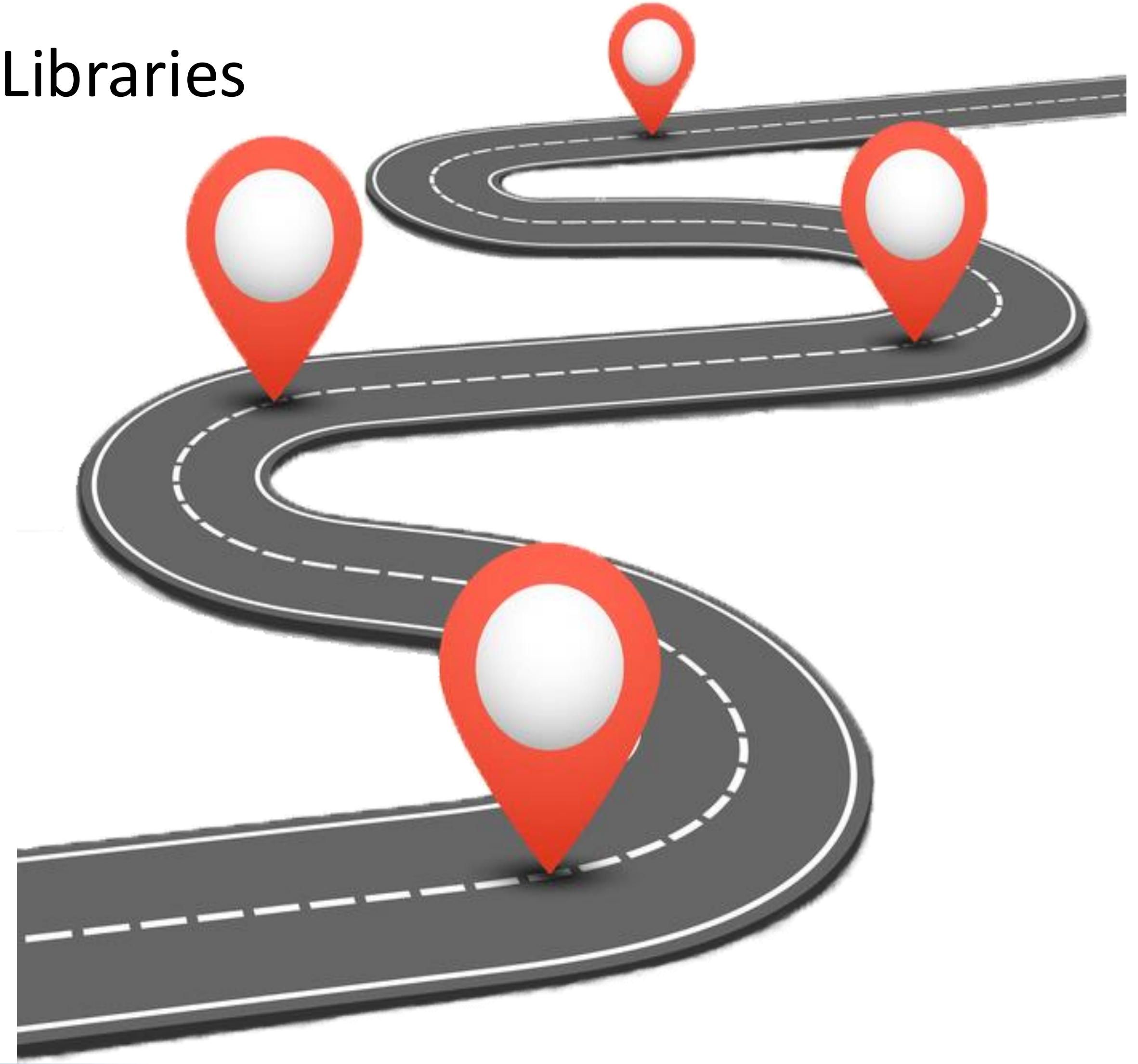
2. Preparing MNIST Dataset

3. Creating a simple CNN model

4. Compiling the CNN Model

5. Training the CNN Model

6. Evaluating the Trained CNN Model



Road Map

7. Integer Quantization on CNN Model

8. Dynamic Range Quantization on CNN Model

9. Float-16 Quantization on CNN Model

10.Weight Pruning on CNN Model

11. Knowledge Distillation on CNN Model