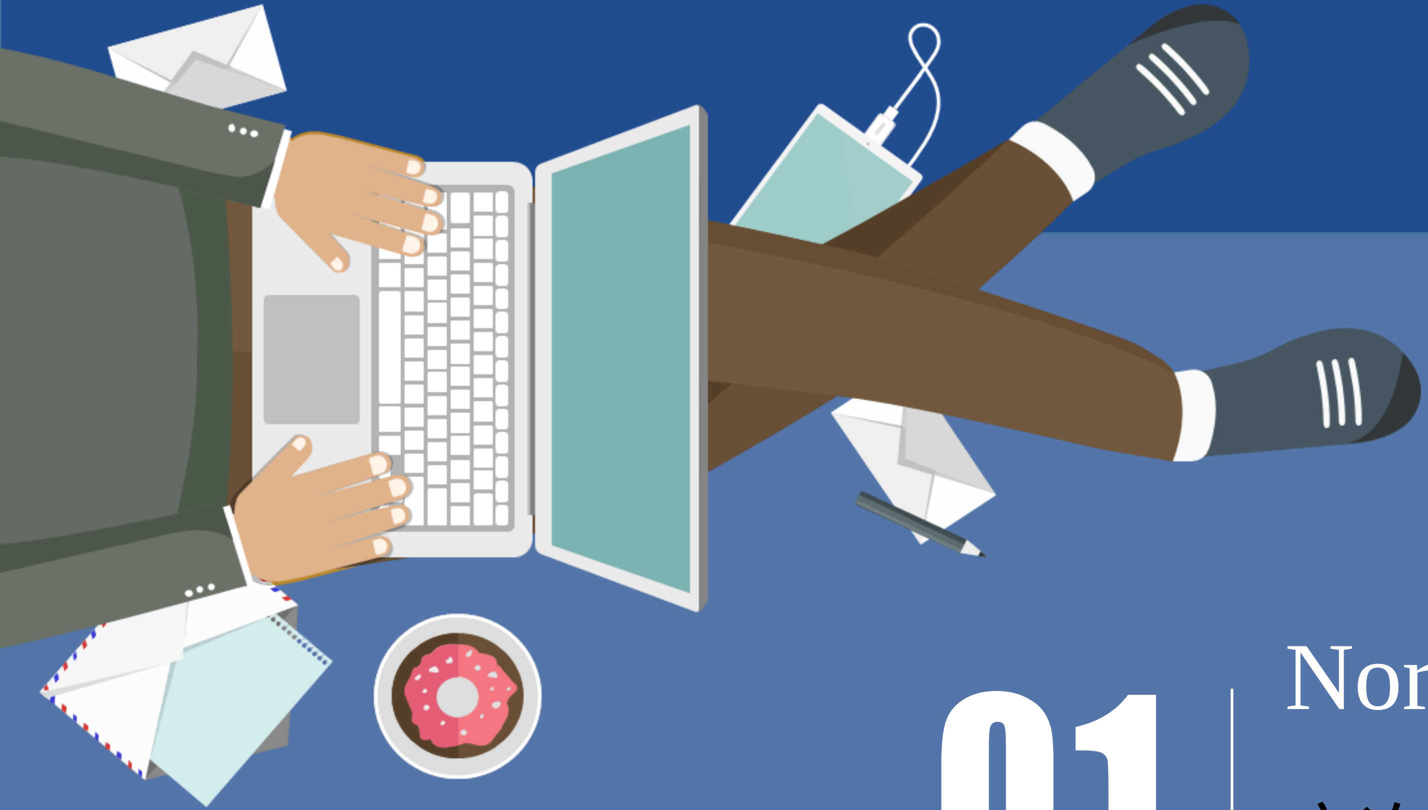
Natural Language Processing

第十三周 模型齐次与量化

庞彦
yanpang@gzhu.edu.cn

Spring 2023



01 Normalization

齐次化

Normalization

Why do we need Normalization ?

Normalization techniques can decrease your model’s training time by a huge factor. Let me state some of the benefits of using Normalization.

- ✓ It normalizes each feature so that they maintains the contribution of every feature, as some feature has higher numerical value than others. This way our network can be unbiased(to higher value features).
- ✓ It reduces **Internal Covariate Shift**. It is the change in the distribution of network activations due to the change in network parameters during training. To improve the training, we seek to reduce the internal covariate shift.
- ✓ It makes the Optimization faster because normalization doesn’t allow weights to explode all over the place and restricts them to a certain range.
- ✓ An unintended benefit of Normalization is that it helps network in Regularization(only slightly, not significantly).

Normalization

How Normalization layers behave in Distributed training ?

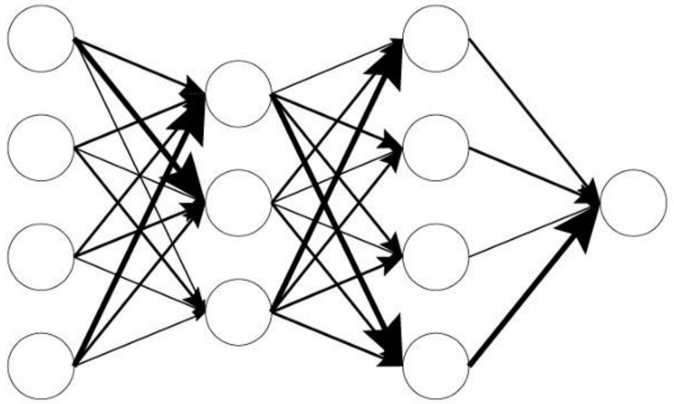
Which Normalization technique should you use for your task like CNN, RNN, style transfer etc ?

What happens when you change the batch size of dataset in your training ?

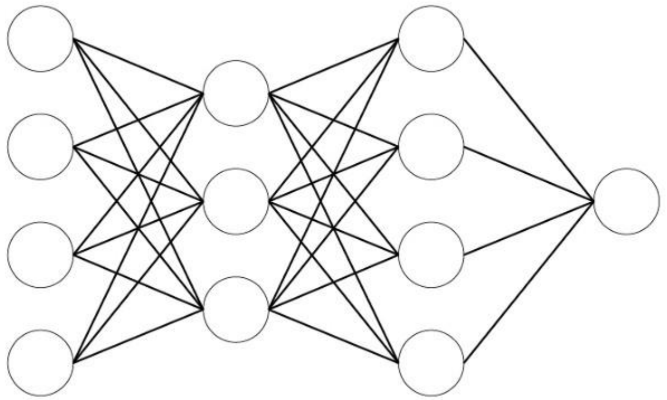
Which norm technique would be the best trade-off for computation and accuracy for your network ?

Batch Normalization

Batch normalization is a method that normalizes activations in a network across the mini-batch of definite size.



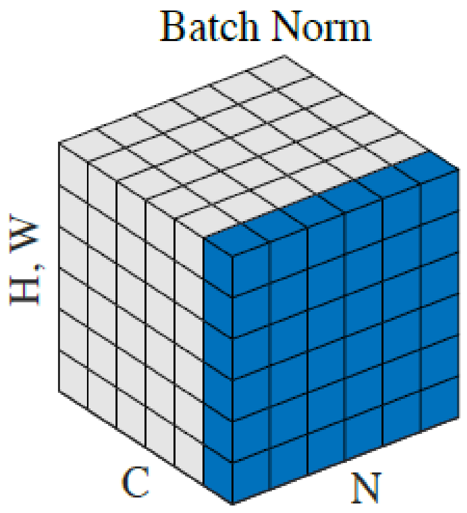
- **Raw** signal
- **High interdependancy** between distributions
- **Slow** and **unstable** training



- **Normalized** signal
- **Mitigated interdependancy** between distributions
- **Fast** and **stable** training

Batch Normalization

If we would like to represent real numbers, we have to give up perfect precision.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

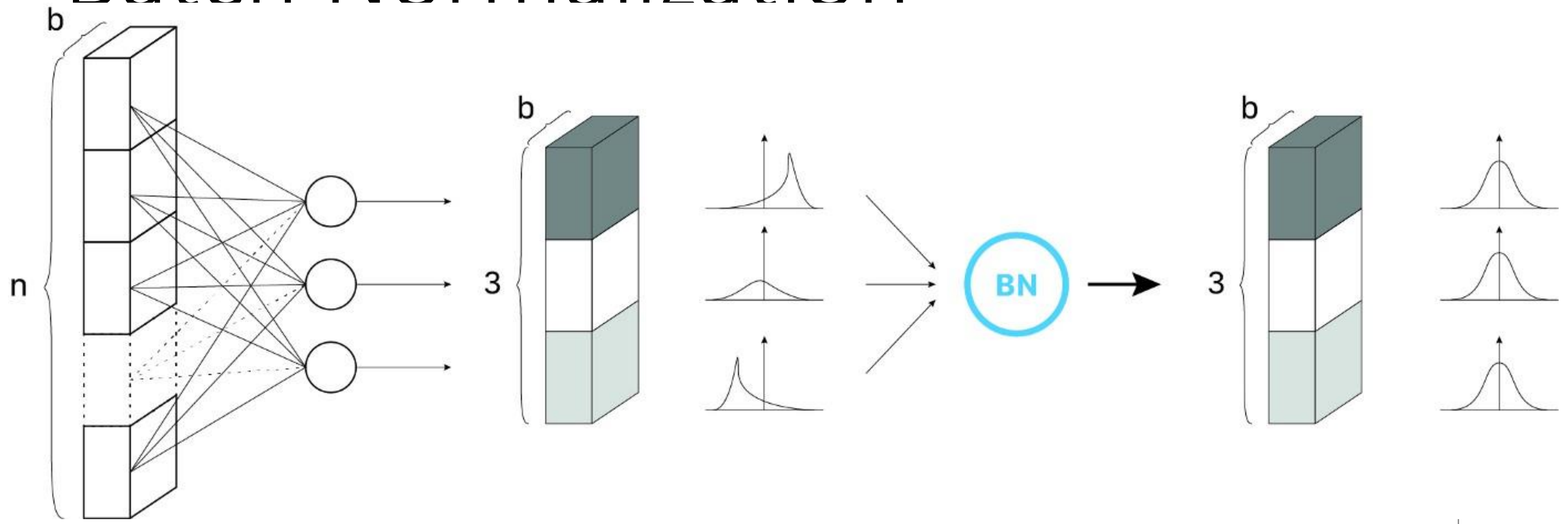
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

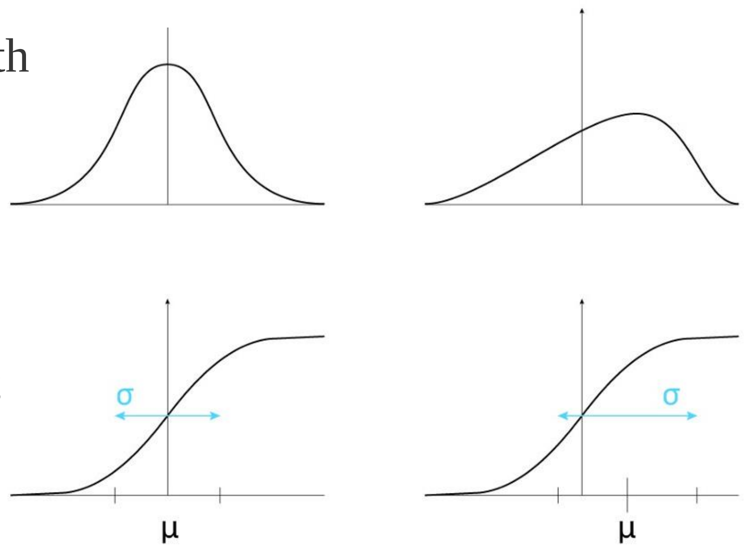
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization



Batch Normalization first step. Example of a 3-neurons hidden layer, with a batch of size b. Each neuron follows a standard normal distribution.



- γ allows to adjust the standard deviation ;
- β allows to adjust the bias, shifting the curve on the right or on the left side.

Batch Normalization

Problems associated with Batch Normalization:

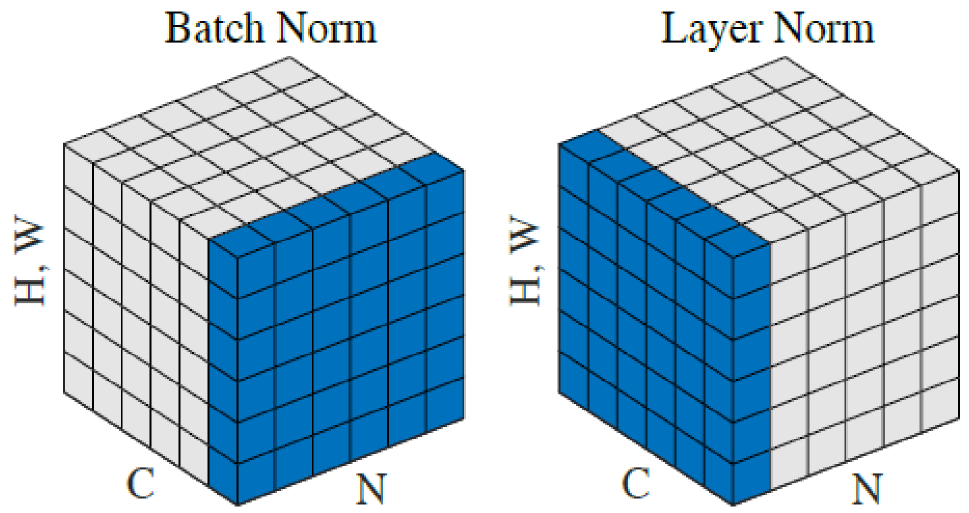
Variable Batch Size → If batch size is of 1, then variance would be 0 which doesn't allow batch norm to work. Furthermore, if we have small mini-batch size then it becomes too noisy and training might affect. There would also be a problem in distributed training. As, if you are computing in different machines then you have to take same batch size because otherwise γ and β will be different for different systems.

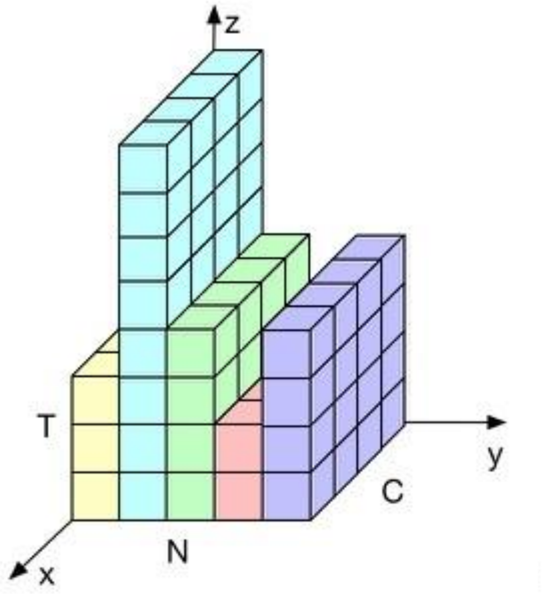
Recurrent Neural Network → In an RNN, the recurrent activations of each time-step will have a different story to tell(i.e. statistics). This means that we have to fit a separate batch norm layer for each time-step. This makes the model more complicated and space consuming because it forces us to store the statistics for each time-step during training.

Layer Normalization

Layer normalization improves the training speed for various neural network models.

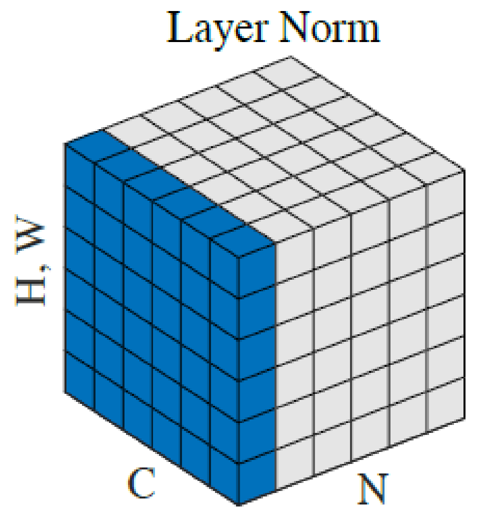
Layer normalization normalizes input across the features instead of normalizing input features across the batch dimension in batch normalization.





Layer Normalization

If we would like to represent real numbers, we have to give up perfect precision.

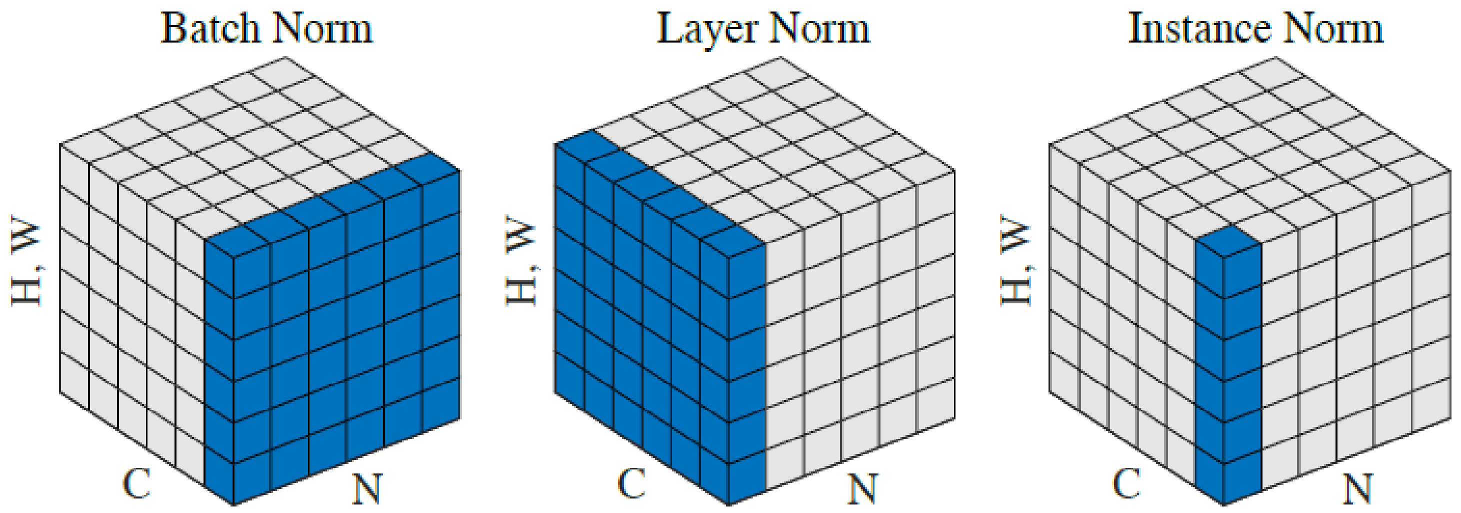


$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

Instance Normalization

Layer normalization and **instance normalization** is very similar to each other but the difference between them is that instance normalization normalizes across each channel in each training example instead of normalizing across input features in a training example.

Unlike batch normalization, the **instance normalization layer** is applied at test time as well(due to non-dependency of mini-batch).



Instance Normalization

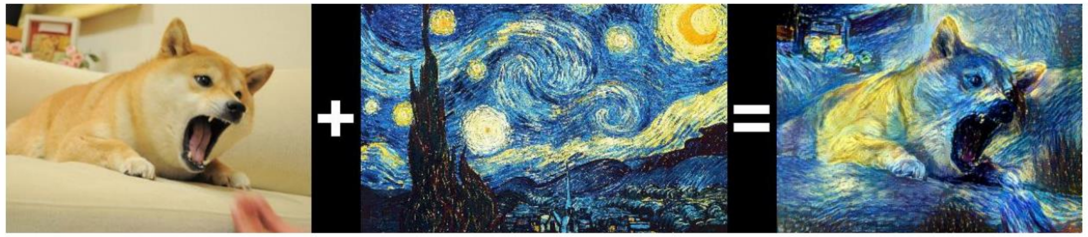
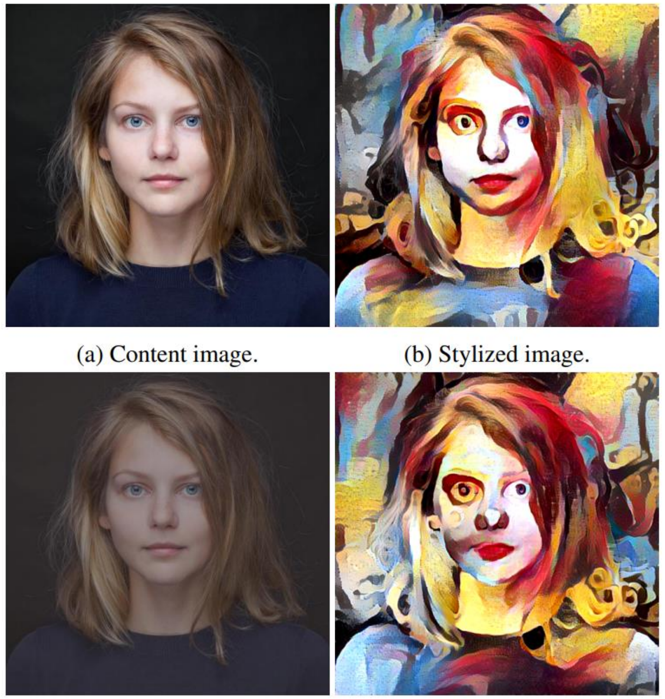


Figure 1: Artistic style transfer example of Gatys et al. (2016) method.

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{tilm},$$

$$\sigma_{ti}^2 = \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} (x_{tilm} - mu_{ti})^2.$$



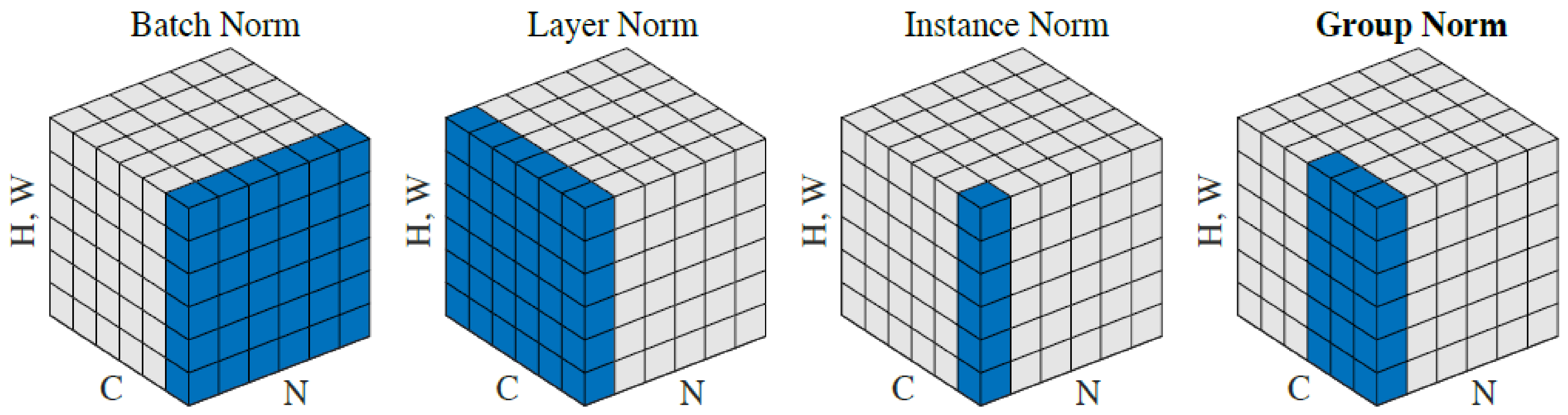
(a) Content image. (b) Stylized image.

(c) Low contrast content image. (d) Stylized low contrast image.

Group Normalization

Group Normalization normalizes over group of channels for each training examples.

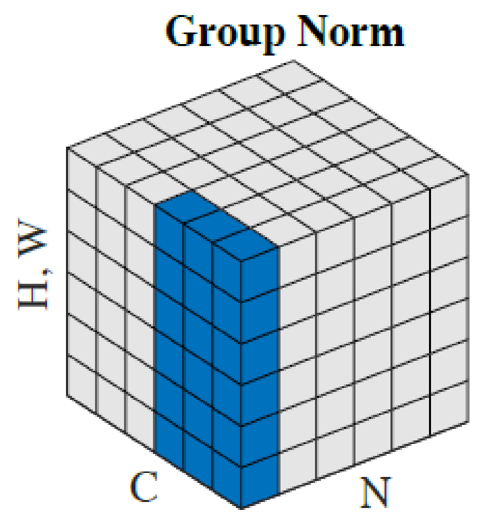
- When we put all the channels into a single group, group normalization becomes Layer normalization.
- When we put each channel into different groups it becomes Instance normalization.

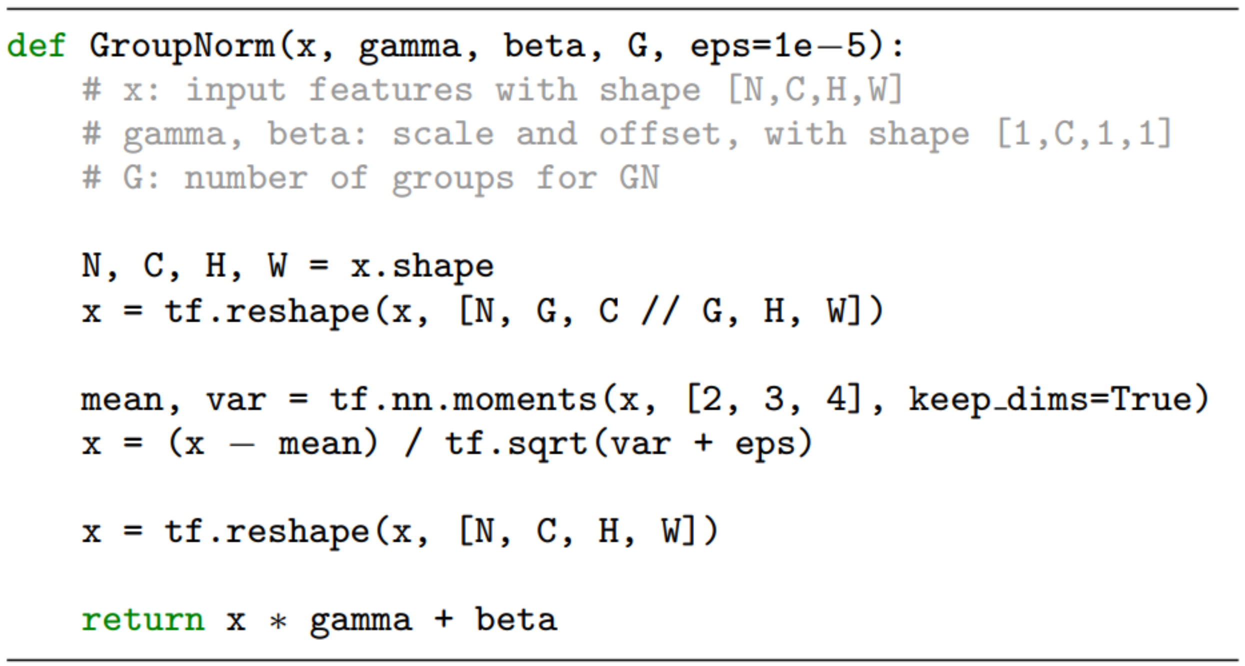


Group Normalization

Group Normalization divides the channels into groups and computes within each group the mean and variance for normalization.

The computation of group normalization is independent of batch sizes, and its accuracy is stable in a wide range of batch sizes.

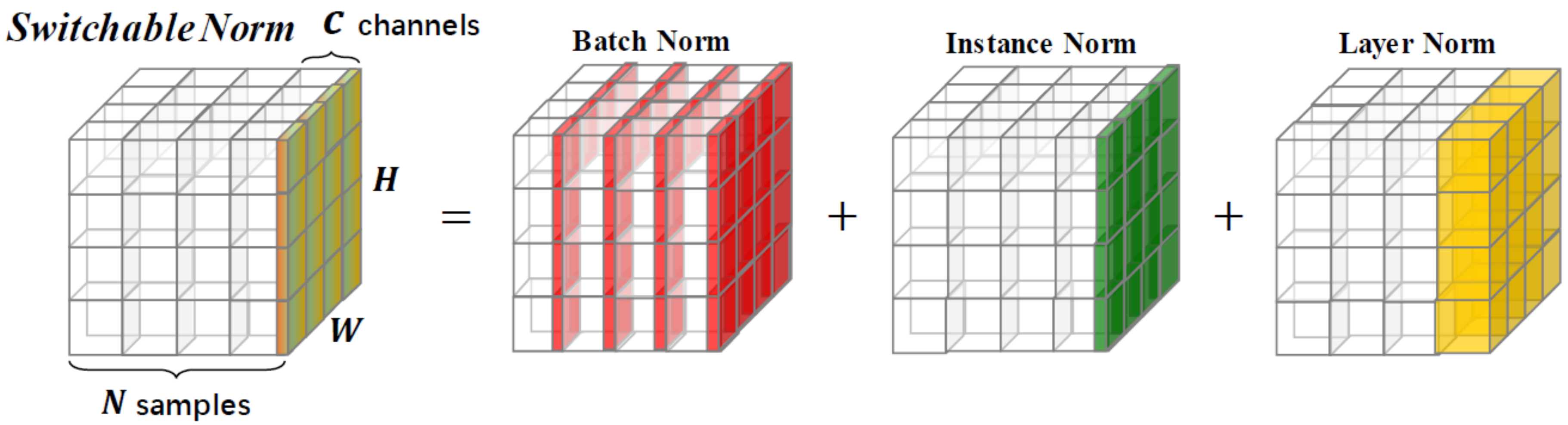




```
def GroupNorm(x, gamma, beta, G, eps=1e−5):
    # x: input features with shape [N,C,H,W]
    # gamma, beta: scale and offset, with shape [1,C,1,1]
    # G: number of groups for GN

    N, C, H, W = x.shape
    x = tf.reshape(x, [N, G, C // G, H, W])

    mean, var = tf.nn.moments(x, [2, 3, 4], keep_dims=True)
    x = (x − mean) / tf.sqrt(var + eps)

    x = tf.reshape(x, [N, C, H, W])

    return x ∗ gamma + beta
```

Switchable Normalization

Switchable Normalization is a normalization technique that is able to learn different normalization operations for different normalization layers in a deep neural network in an end-to-end manner.



https://github.com/switchablenorms/Switchable-Normalization

Switchable Normalization

SN has an intuitive expression

$$\hat{h}_{ncij} = \gamma \frac{h_{ncij} - \Sigma_{k\in\Omega} w_k \mu_k}{\sqrt{\Sigma_{k\in\Omega} w'_k \sigma_k^2 + \epsilon}} + \beta, \tag{3}$$

where Ω is a set of statistics estimated in different ways. In this work, we define $\Omega = \{\text{in}, \text{ln}, \text{bn}\}$ the same as above where μ_k and σ_k^2 can be calculated by following Eqn.(2). However, this strategy leads to large redundant computations. In fact, the three kinds of statistics of SN depend on each other. Therefore we could reduce redundancy by reusing computations,

$$\begin{aligned}
\mu_{\text{in}} &= \frac{1}{HW}\sum_{i,j}^{H,W} h_{ncij}, \quad \sigma_{\text{in}}^2 = \frac{1}{HW}\sum_{i,j}^{H,W}(h_{ncij} - \mu_{\text{in}})^2, \\
\mu_{\text{ln}} &= \frac{1}{C}\sum_{c=1}^{C}\mu_{\text{in}}, \quad \sigma_{\text{ln}}^2 = \frac{1}{C}\sum_{c=1}^{C}(\sigma_{\text{in}}^2 + \mu_{\text{in}}^2) - \mu_{\text{ln}}^2, \\
\mu_{\text{bn}} &= \frac{1}{N}\sum_{n=1}^{N}\mu_{\text{in}}, \quad \sigma_{\text{bn}}^2 = \frac{1}{N}\sum_{n=1}^{N}(\sigma_{\text{in}}^2 + \mu_{\text{in}}^2) - \mu_{\text{bn}}^2,
\end{aligned} \tag{4}$$

showing that the means and variances of LN and BN can be computed based on IN. Using Eqn.(4), the computational complexity of SN is $\mathcal{O}(NCHW)$, which is comparable to previous work.

Furthermore, w_k and w'_k in Eqn.(3) are importance ratios used to weighted average the means and variances respectively. Each w_k or w'_k is a scalar variable, which is shared across all channels. There are $3 \times 2 = 6$ importance weights in SN. We have $\Sigma_{k\in\Omega} w_k = 1$, $\Sigma_{k\in\Omega} w'_k = 1$, and $\forall w_k, w'_k \in [0, 1]$, and define

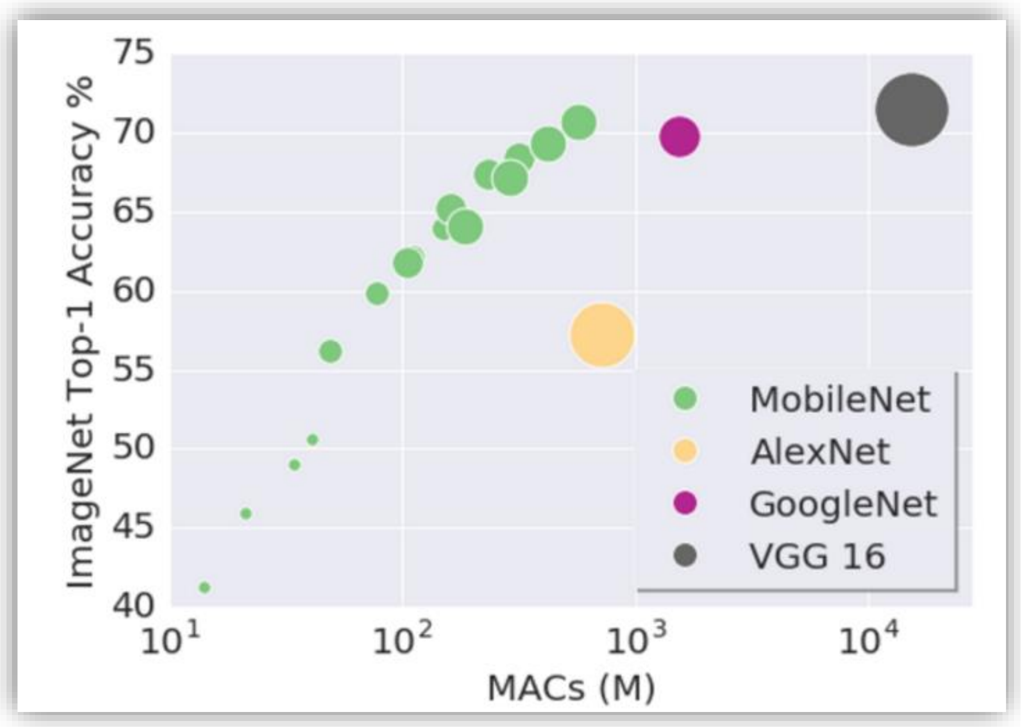
$$w_k = \frac{e^{\lambda_k}}{\Sigma_{z\in\{\text{in},\text{ln},\text{bn}\}} e^{\lambda_z}} \quad \text{and} \quad k \in \{\text{in}, \text{ln}, \text{bn}\}. \tag{5}$$

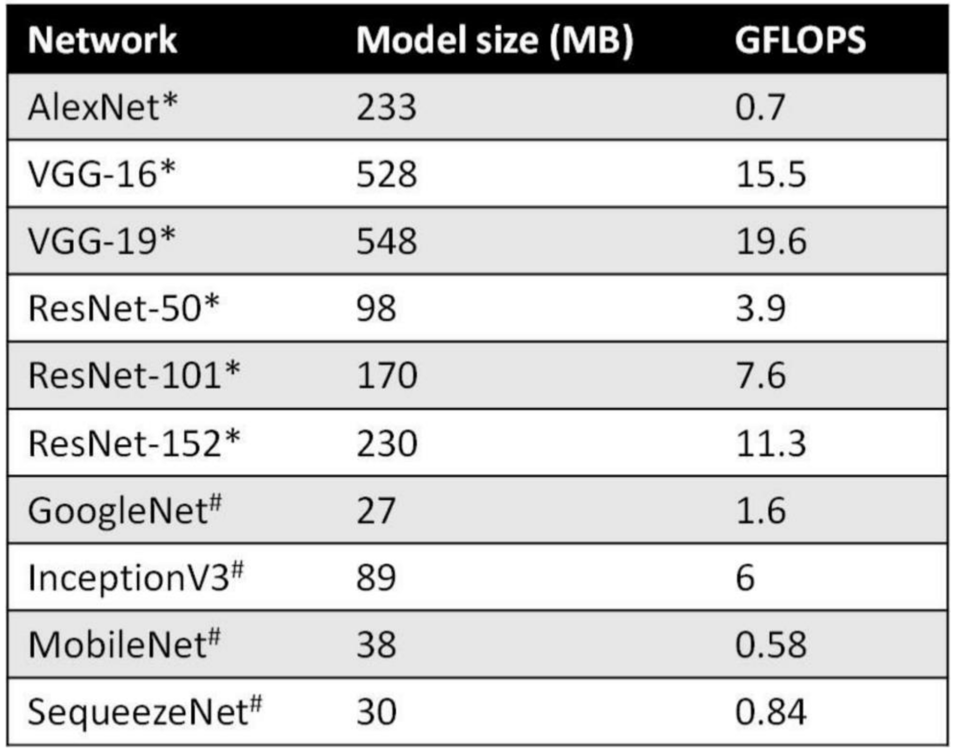
02 Quantization

模型量化

Deep Learning

In general, the larger deep neural network can receive better performance.

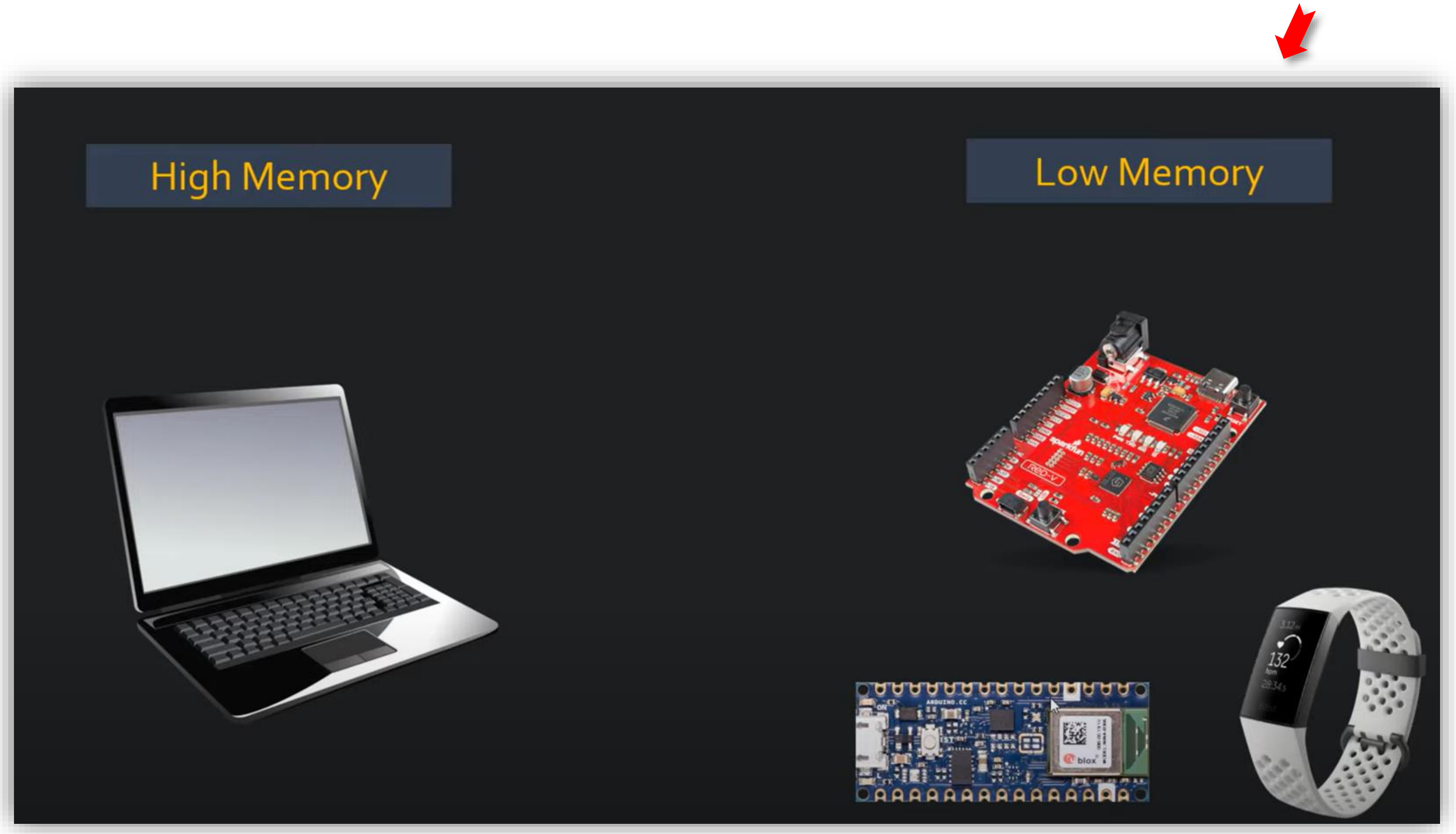




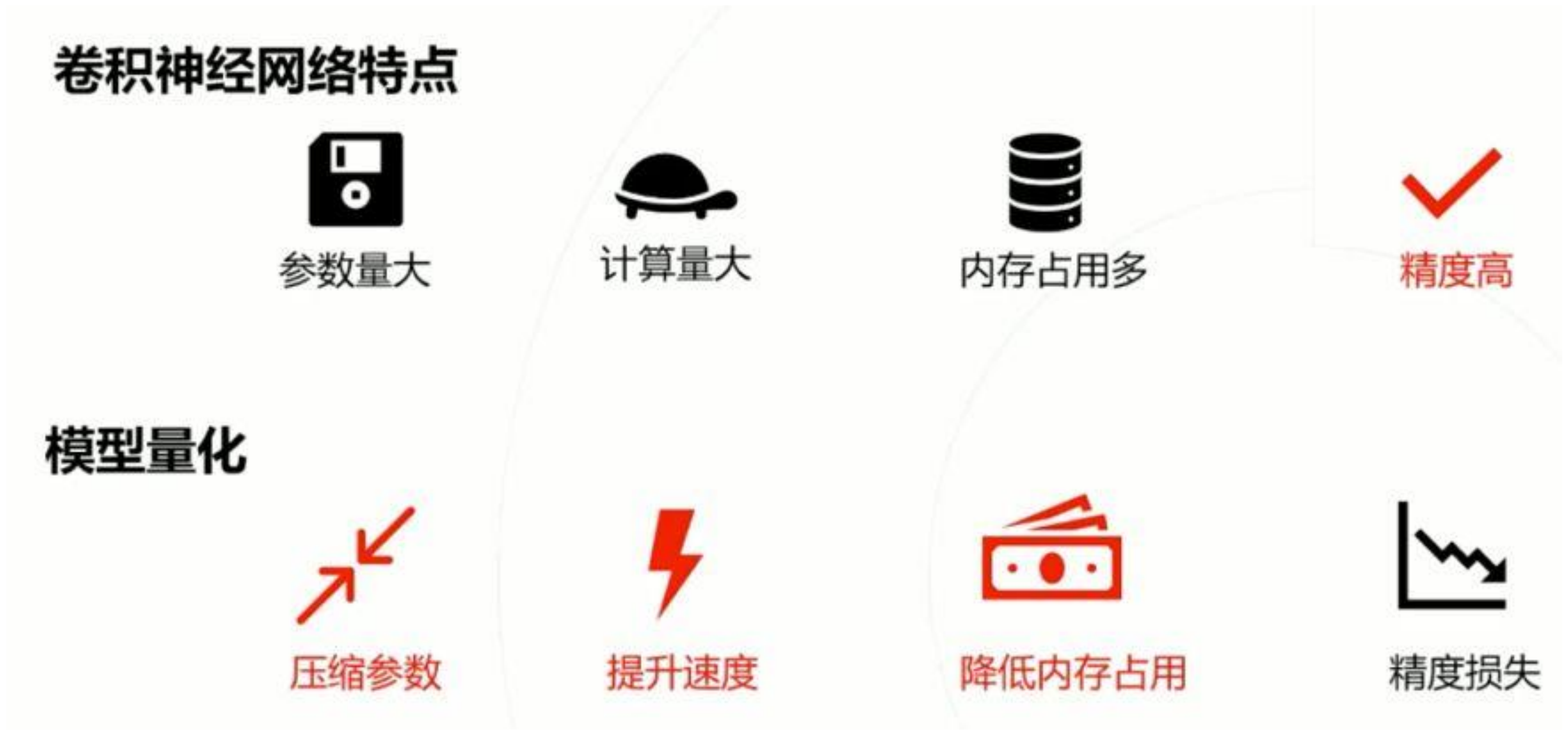
Network	Model size (MB)	GFLOPS
AlexNet*	233	0.7
VGG-16*	528	15.5
VGG-19*	548	19.6
ResNet-50*	98	3.9
ResNet-101*	170	7.6
ResNet-152*	230	11.3
GoogleNet#	27	1.6
InceptionV3#	89	6
MobileNet#	38	0.58
SequeezeNet#	30	0.84

Large Scale Models

It is difficult to transfer large models pretrained on high performance computers to edge devices.



Deep Learning



卷积神经网络特点

参数量大

计算量大

内存占用多

精度高

模型量化

压缩参数

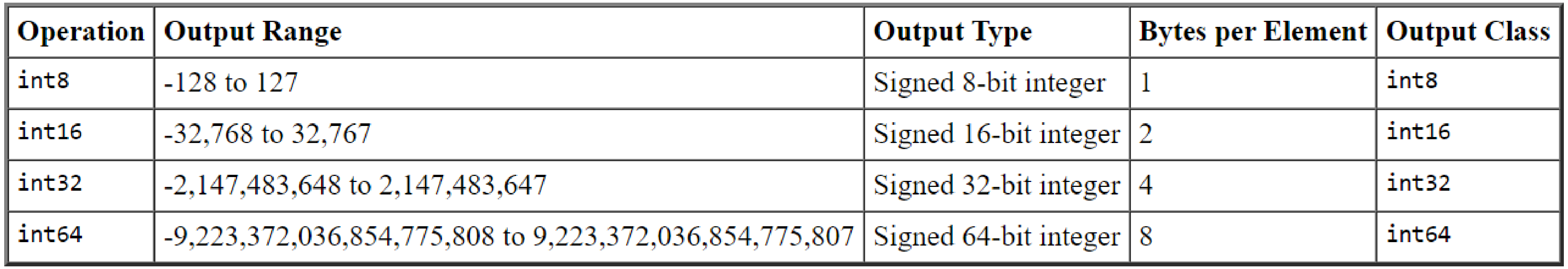
提升速度

降低内存占用

精度损失

Integers

Integers are represented with their form in base-2 numeral system. Depending on the number of digits used, an integer can take up several different sizes.



Operation	**Output Range**	**Output Type**	**Bytes per Element**	**Output Class**
`int8`	-128 to 127	Signed 8-bit integer	1	`int8`
`int16`	-32,768 to 32,767	Signed 16-bit integer	2	`int16`
`int32`	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer	4	`int32`
`int64`	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer	8	`int64`

Floating-point

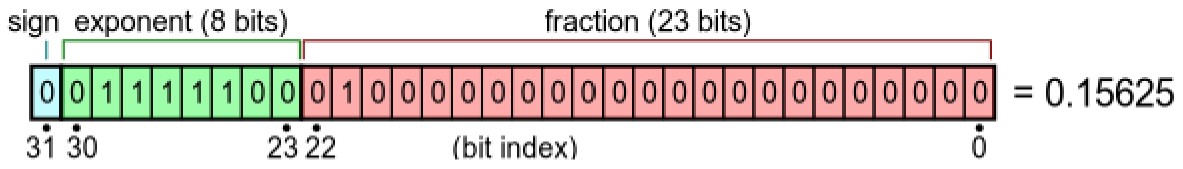
If we would like to represent real numbers, we have to give up perfect precision.

To give an example, the number 1/3 can be written in decimal form as 0.33333…, with infinitely many digits, which cannot be represented in the memory.

Essentially, a float is the scientific notation of the number in the form

$$significand \times base^{exponent}$$

where the base is most frequently 2.



Floating-point

Similarly to integers, there are different types of floats.

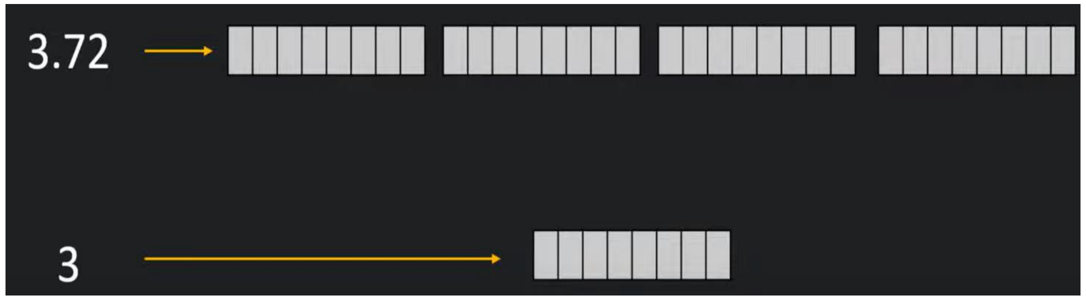
The most commonly used are:

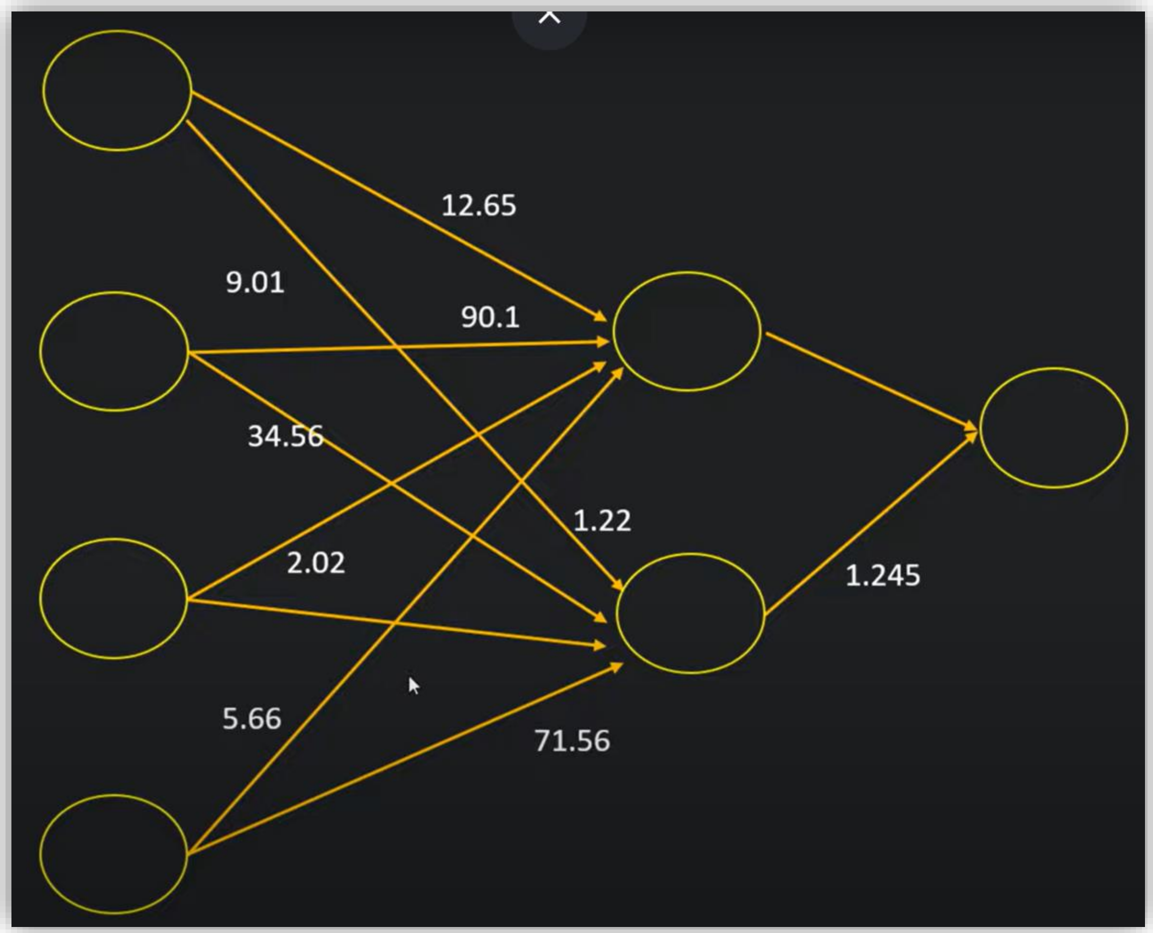
- half or float16 (1 bit sign, 5 bit exponent, 10 bit significand, so **16 bits in total**)
- single or float32 (1 bit sign, 8 bit exponent, 23 bit significand, so **32 bits in total**)
- double or float64 (1 bit sign, 11 bit exponent, 52 bit significand, so **64 bits in total**).

Neural Network

Weights --> FLOAT: FT64, FT32

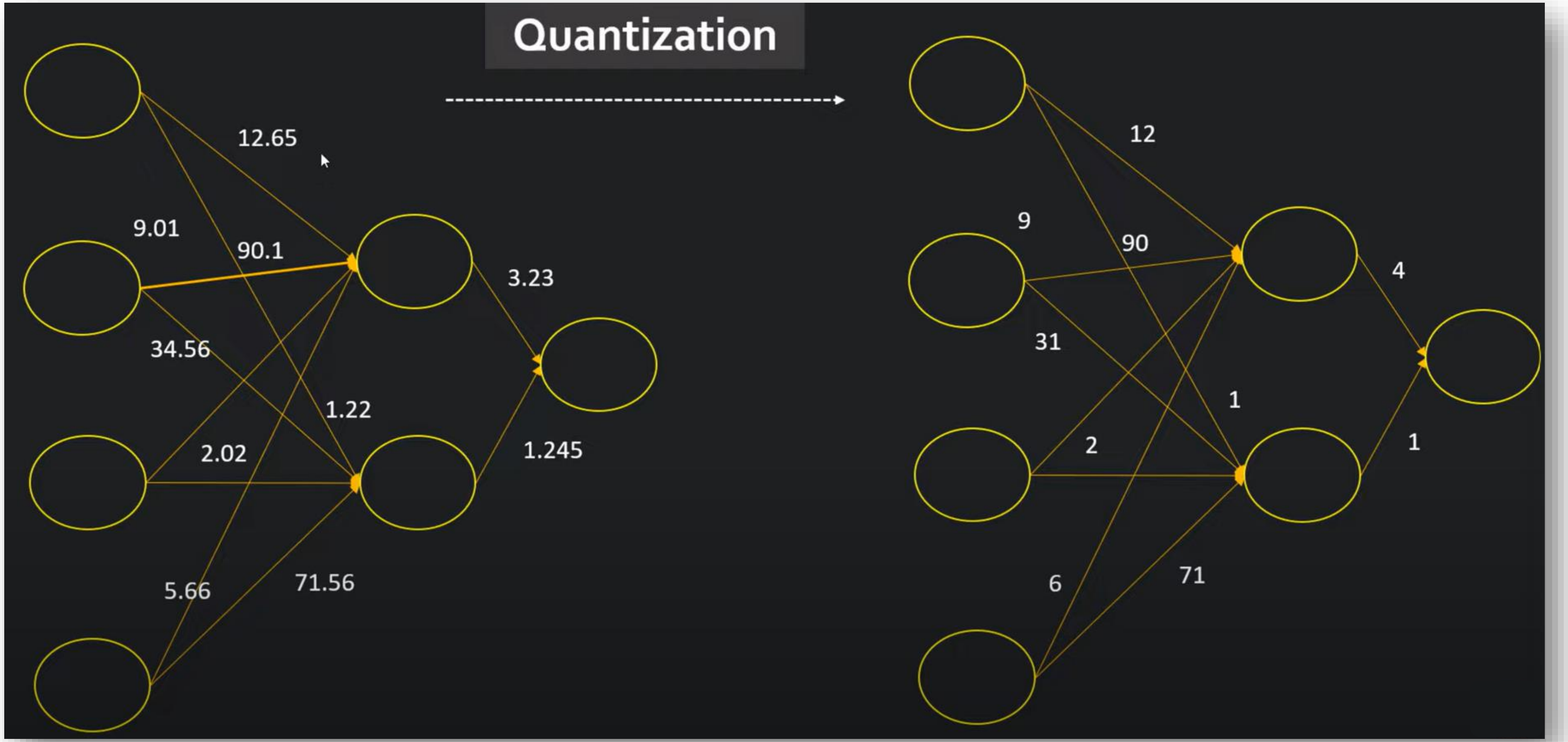
FT64： 8 Bytes
FT32： 4 Bytes





Quantization

Quantization is a process of reducing model size so that it can run on edge devices.



Benefits of Quantization

Quantizing Networks

Suppose that you have a layer with outputs in the range of [-a, a), where a is any real number.

First, we scale the output to [-128, 128), then we simply round down. That is, we use the transformation

$$x \mapsto \left\lfloor 128 \frac{x}{a} \right\rfloor.$$

Quantizing Networks

To give a concrete example, let's consider the calculation below.

$$x \mapsto \left\lfloor 128 \frac{x}{a} \right\rfloor .$$

$$\begin{pmatrix} -0.18120981 & -0.29043840 \\ 0.49722983 & 0.22141714 \end{pmatrix} \begin{pmatrix} 0.77412377 \\ 0.49299395 \end{pmatrix} = \begin{pmatrix} -0.28346319 \\ 0.49407474 \end{pmatrix}$$

The range of the values here is in (-1, 1), so if we quantize the matrix and the input, we get

$$\begin{pmatrix} -24 & -38 \\ 63 & 28 \end{pmatrix} \begin{pmatrix} 99 \\ 63 \end{pmatrix} = \begin{pmatrix} 4770 \\ 8001 \end{pmatrix} .$$

This is where we see that the result is not an int8. Since multiplying two 8-bit integers is a 16-bit integer, we can de-quantize the result with the transformation to obtain the result.

$$x \mapsto \frac{ax}{16384}$$

$$\begin{pmatrix} -0.2911377 \\ 0.48834229 \end{pmatrix} .$$

Quantization

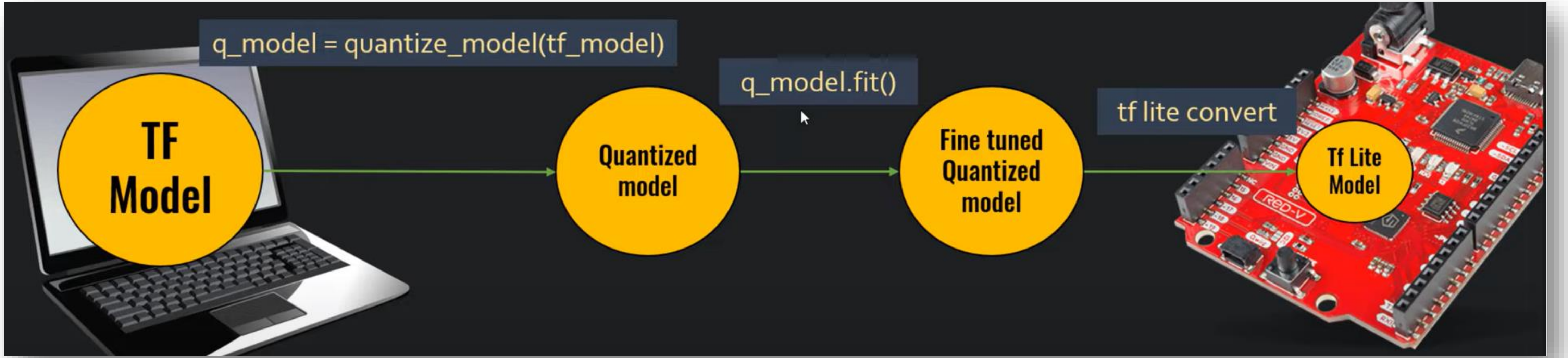
Two ways to perform quantization:

- **Post-training**: train the model using float32 weights and inputs, then quantize the weights. Its main advantage that it is simple to apply. Downside is, it can result in accuracy loss.

- **Quantization-aware training**: quantize the weights during training. Here, even the gradients are calculated for the quantized weights. When applying int8 quantization, this has the best result, but it is more involved than the other option.

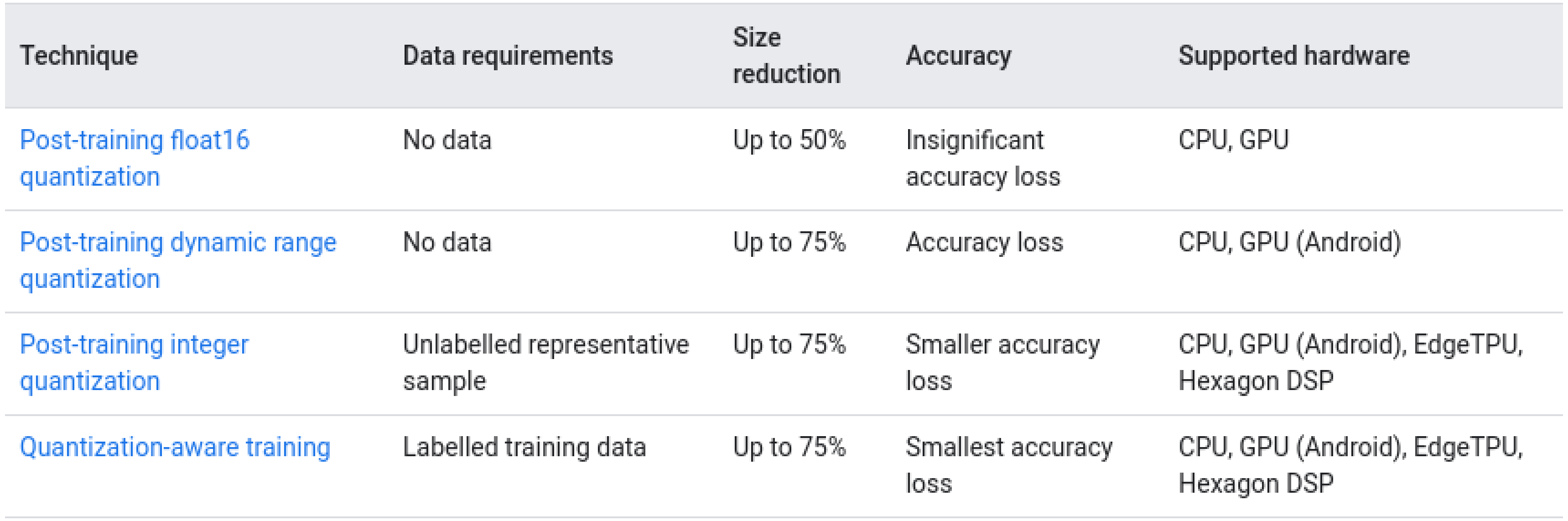
Quantization-aware training

Quantization is a process of reducing model size so that it can run on edge devices.



Quantization

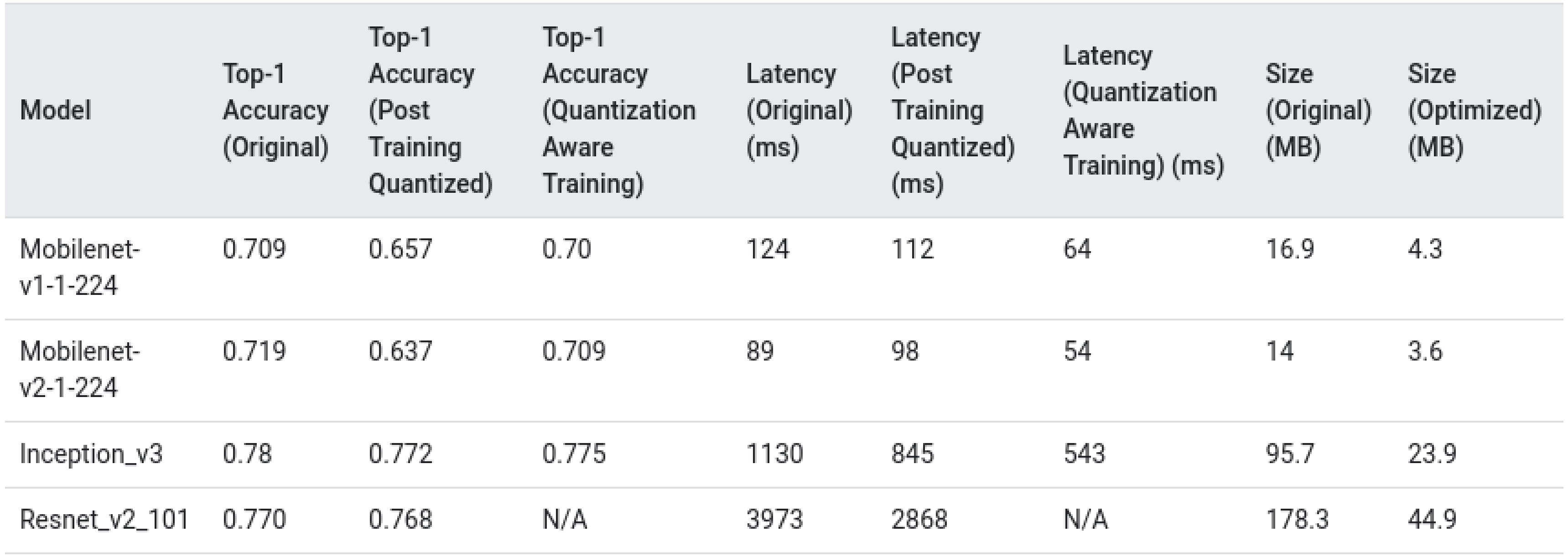
Quantization methods and their performance in TensorFlow Lite.



Technique	Data requirements	Size reduction	Accuracy	Supported hardware
Post-training float16 quantization	No data	Up to 50%	Insignificant accuracy loss	CPU, GPU
Post-training dynamic range quantization	No data	Up to 75%	Accuracy loss	CPU, GPU (Android)
Post-training integer quantization	Unlabelled representative sample	Up to 75%	Smaller accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP
Quantization-aware training	Labelled training data	Up to 75%	Smallest accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP

Quantization

Comparison of quantization methods in TensorFlow Lite for several convolutional network architectures.



Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

Q&A