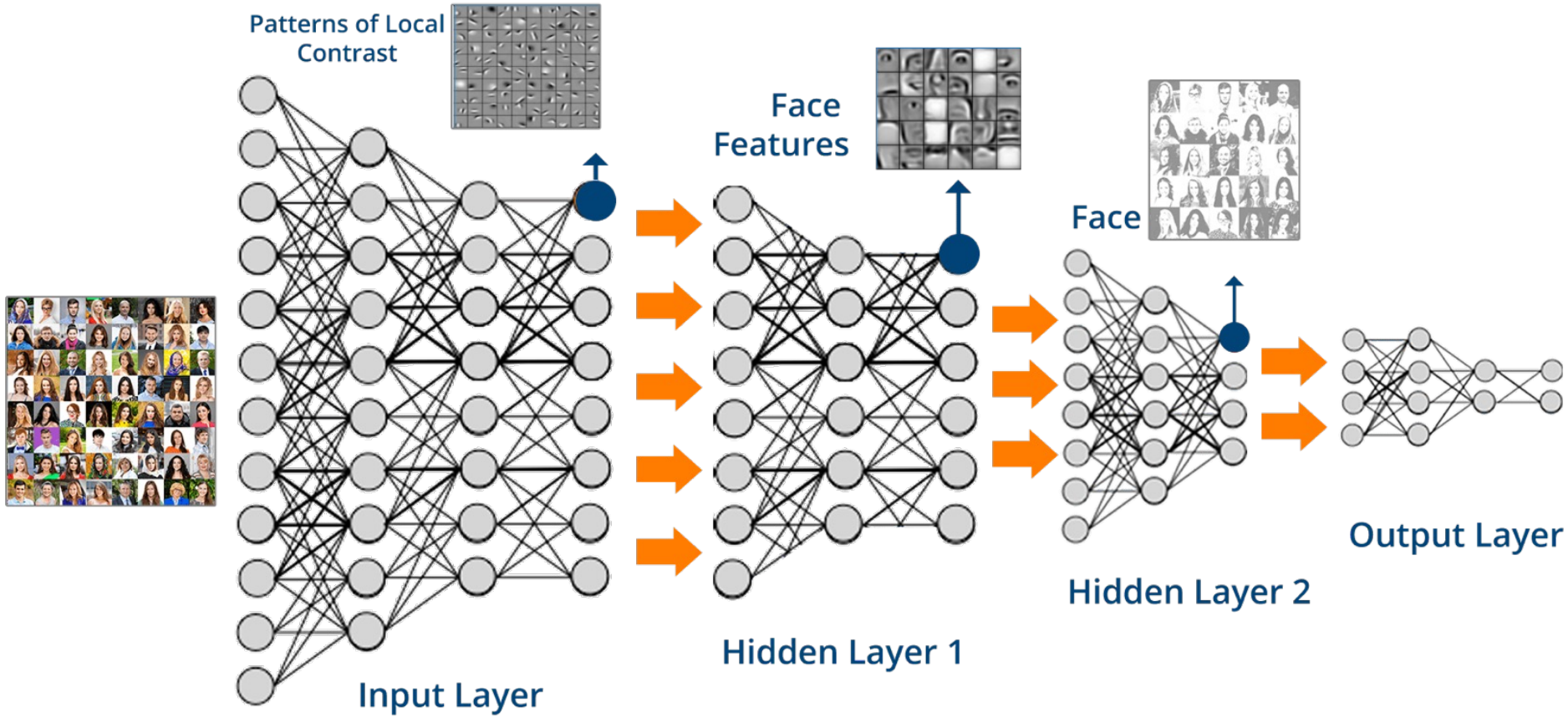
POLITECNICO
MILANO 1863

# Hardware Architectures for Embedded and Edge AI

*Prof Manuel Roveri – manuel.roveri@polimi.it*

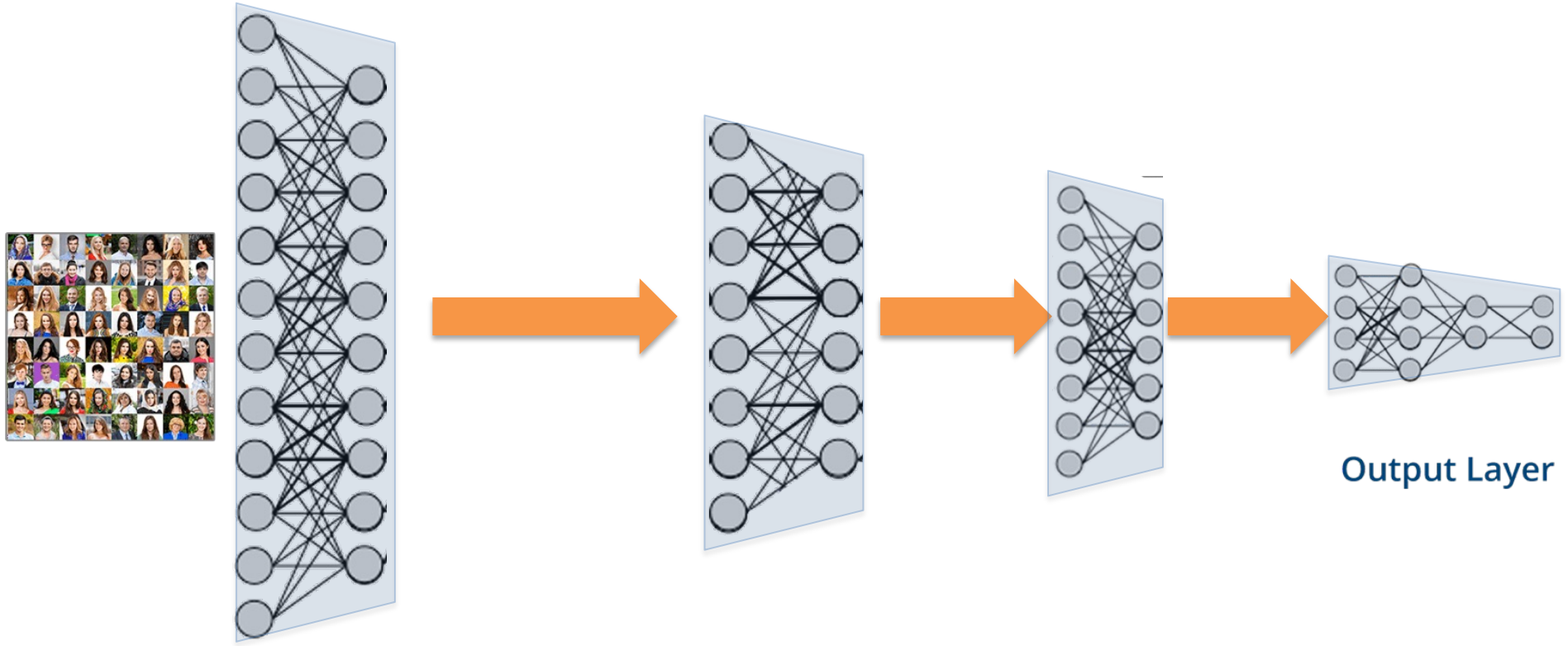
*Lecture 7 – Approximate Computing for Embedded and Edge AI*

# Approximate Deep Learning for IoT: Tiny Deep Learning

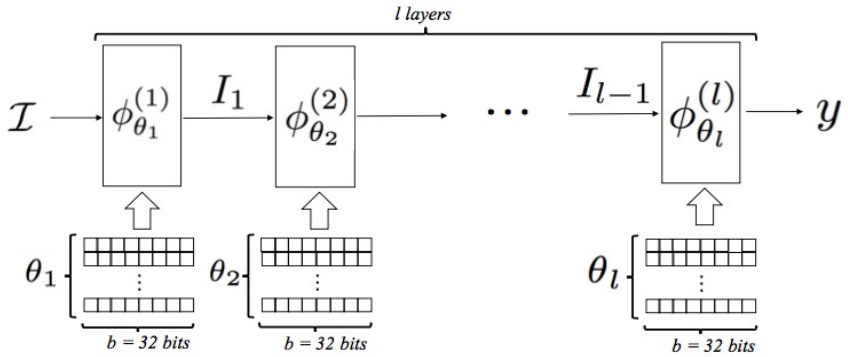


# Approximate Deep Learning for IoT: Tiny Deep Learning

Re-design of the CNN architecture → Approximate Computing → Embedded System Code Optimization



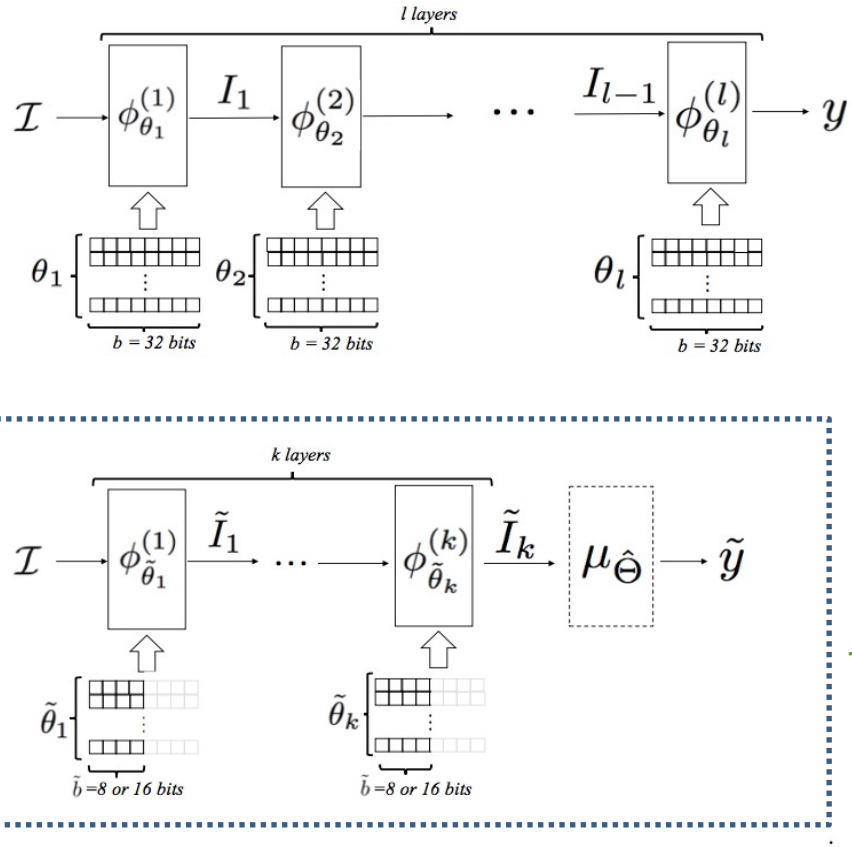
# Approximate computing in Tiny Deep Learning



# Approximate computing in Tiny Deep Learning

Precision scaling

Reducing the memory occupation of the CNN by changing the precision of the weights (#bits for the representation)



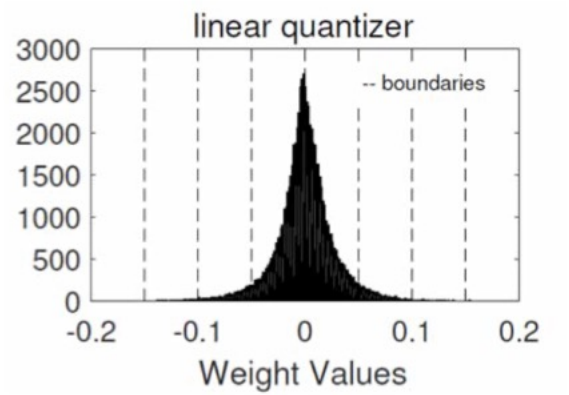

Reduce the computational load and the memory occupation by skipping the execution of some tasks associated with the processing chain

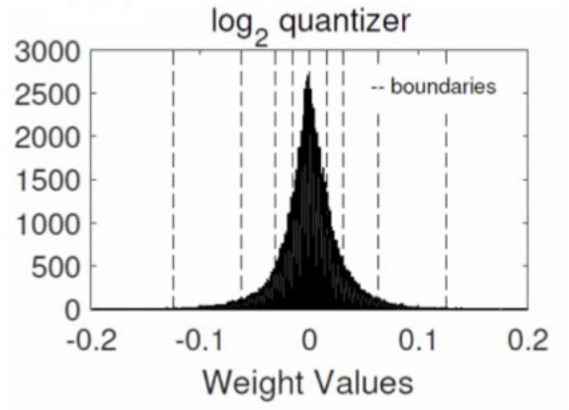
Task dropping

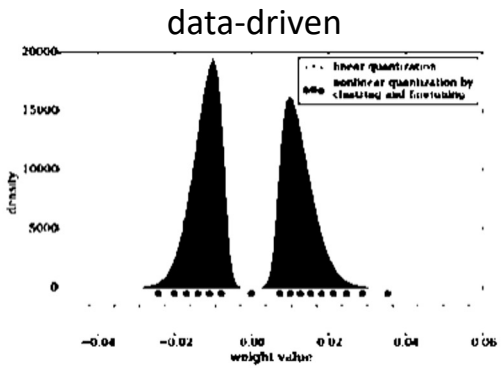
C. Alippi, S. Disabato, M. Roveri, "Moving Convolutional Neural Networks to Embedded Systems: the AlexNet and VGG-16 case", 2018 ACM/IEEE International Conference on Information Processing in Sensor Networks Porto (IPSN 2018), Portugal, April 11-13, 2018.

# 1) Precision Scaling

# Precision Scaling (Quantization)

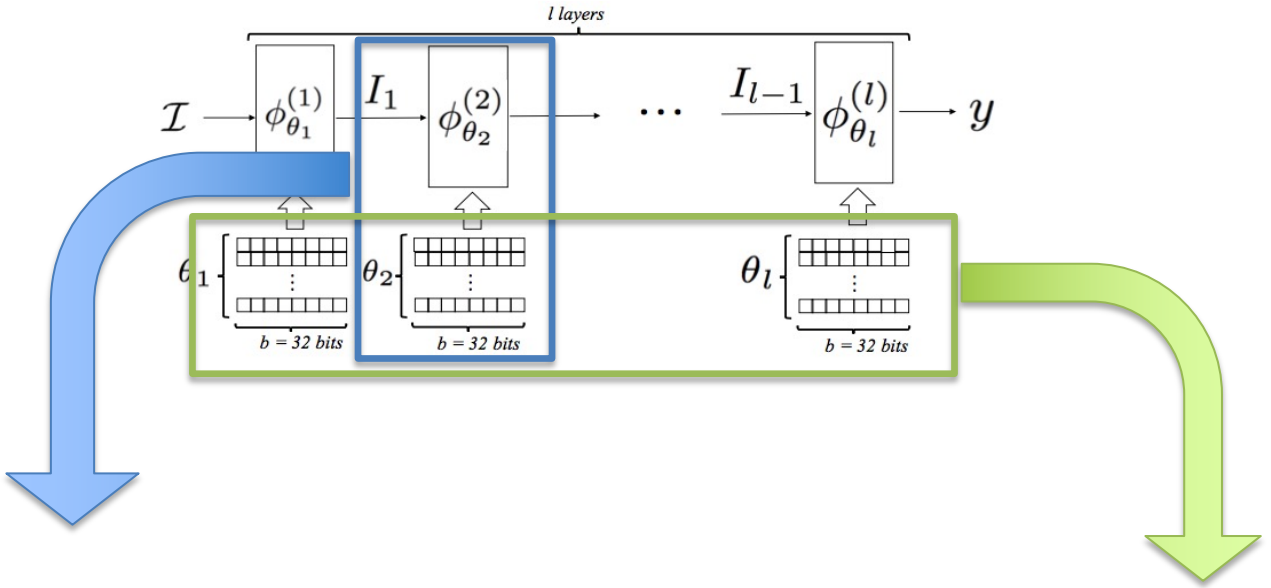



- **Linear quantizer**: uniform distance between each quantization level (number of bits: 32, 16, 8 or custom)
- **Log function**: the distance between the levels varies (e.g., log base2 quantization with logaritmic distribution of intervals). When weights are power of 2, multiplication -> bit shifts
- **Data-driven**: quantization levels are determined or learned from the data, e.g., using *k*-means clustering.

# Where and what to quantize in CNNs?




We can quantize:

- **weights** (reducing the storage capacity requirement)
- **activations** (but the effects depends on the network architecture and dataflow). Typically done during the inference

We can use:

- Fixed quantization: the **same quantization** for all the layers
- **Variable quantization: different quantization mechanisms** for different layers, filters or channels

# The effect of quantization on accuracy

Table 3 Methods to Reduce Numerical Precision for AlexNet. Accuracy Measured for Top-5 Error on ImageNet

| Reduce Precision Method | | bitwidth | | Accuracy loss vs. 32-bit float (%) |
|---|---|---|---|---|
| | | Weights | Activations | |
| Dynamic Fixed Point | w/o fine-tuning [123] | 8 | 10 | 0.4 |
| | w/ finetuning [122] | 8 | 8 | 0.6 |
| Reduce Weight | BinaryConnect [129] | 1 | 32 (float) | 19.2 |
| | Binary Weight Network (BWN) [131] | 1* | 32 (float) | 0.8 |
| | Ternary Weight Network (TWN) [133] | 2* | 32 (float) | 3.7 |
| | Trained Ternary Quantization (TTQ) [134] | 2* | 32 (float) | 0.6 |
| Reduce Weight and Activation | XNOR-Net [131] | 1* | 1* | 11 |
| | Binarized Neural Network (BNN) [130] | 1 | 1 | 29.8 |
| | DoReFa-Net [122] | 1* | 2* | 7.63 |
| | Quantized Neural Network (QNN) [121] | 1 | 2* | 6.5 |
| | HWGQ-Net [132] | 1* | 2* | 5.2 |
| Nonuniform Quantization | LogNet [119] | 5 (conv), 4 (fc) | 4 | 3.2 |
| | Incremental Network Quantization (INQ) [138] | 5 | 32 (float) | -0.2 |
| | Deep Compression [120] | 8 (conv), 4 (fc) | 16 | 0 |
| | | 4 (conv), 2 (fc) | 16 | 2.6 |

*Not Applied to First and/or Last Layers.

Picture taken from [1]

# 8-bit quantization: impact on performance and energy

*Advantages on performance*:

- Four 8-bit operations ≈ one 32-bit operation for a given clock cycle

*Advantages on the energy consumption*:

- An **8bit fixed-point add** consumes
  - 3.3 × less energy than a 32bit fixed-point add,
  - 30 × less energy than a 32bit floating-point add.
- An **8bit fixed-point multiply** consumes
  - 15.5 × less energy than a 32bit fixed-point multiply,
  - 18.5 × less energy than a 32bit floating-point multiply

# "Extreme" quantization: Binary and Ternary NNs

**Binary Nets (-1,+1):**

- Binary Connect (with binary weights)
- Binary NNs (with binary weights and activations)

➢ Accuracy loss ≈ 20%-30%

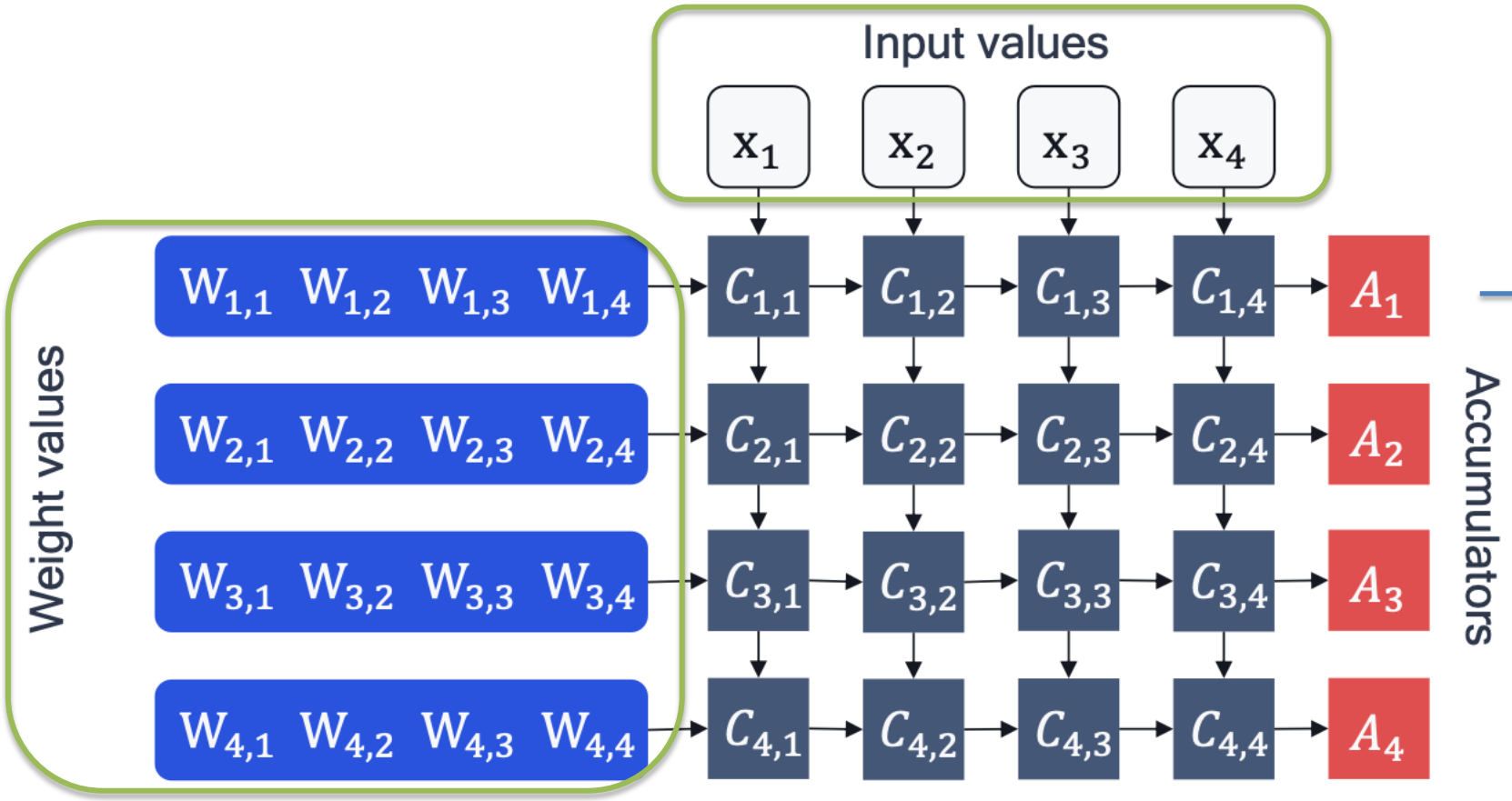
**Ternary Nets (-w,0,+w):**

- Ternary Weights Nets (all the weights share the same values)
- Trained Ternary Quantization (a different scale is trained for each weight)

➢ Accuracy loss ≈ 0.5%

# How is quantization implemented?

# A schematic overview of matrix-multiply logic in NN accelerator hardware




$$A_n = \mathbf{b}_n + \sum_m C_{n,m}$$

This is exactly the Multiply-and-Accumulate (MAC) in NN processing

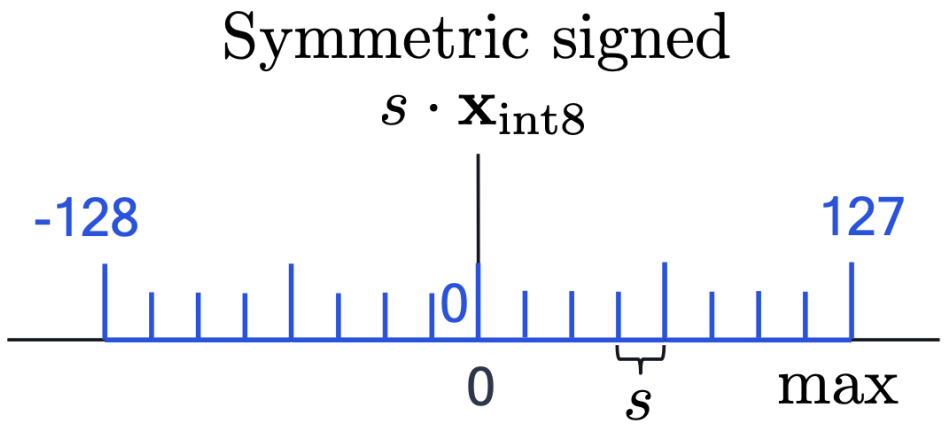
How to quantize inputs and weights?

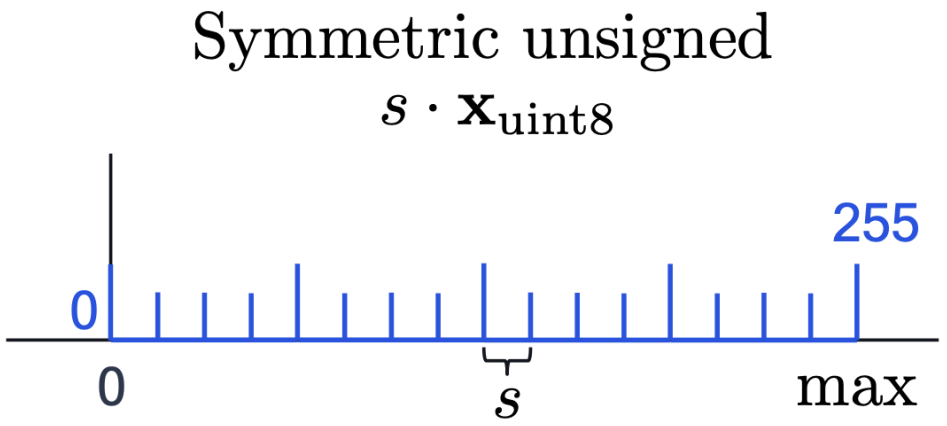
Picture taken from [2]

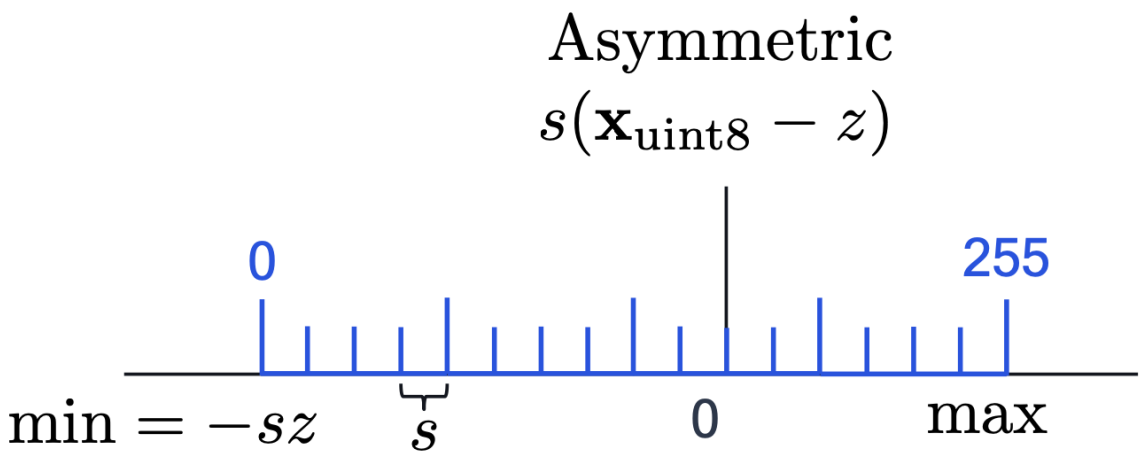
# Symmetric and asymmetric quantization

- The parameters:
  - Bit-width b
  - Scale factor s
  - Zero-point z (only for asymmetric quantization)
- The scale factor and the zero-point are used to to map a floating point value to the integer grid, whose size depends on the bit-width.
- The scale factor is commonly represented as a floating-point number and specifies the step-size of the quantizer.
- The zero-point is an integer that ensures that real zero is quantized without error.

# Symmetric and asymmetric quantization

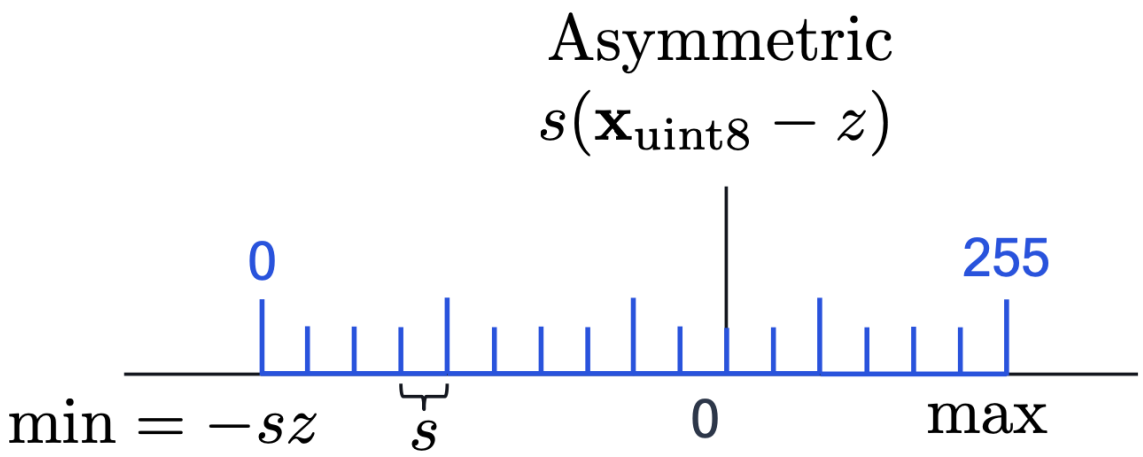



Picture taken from [2]

# Symmetric and asymmetric quantization



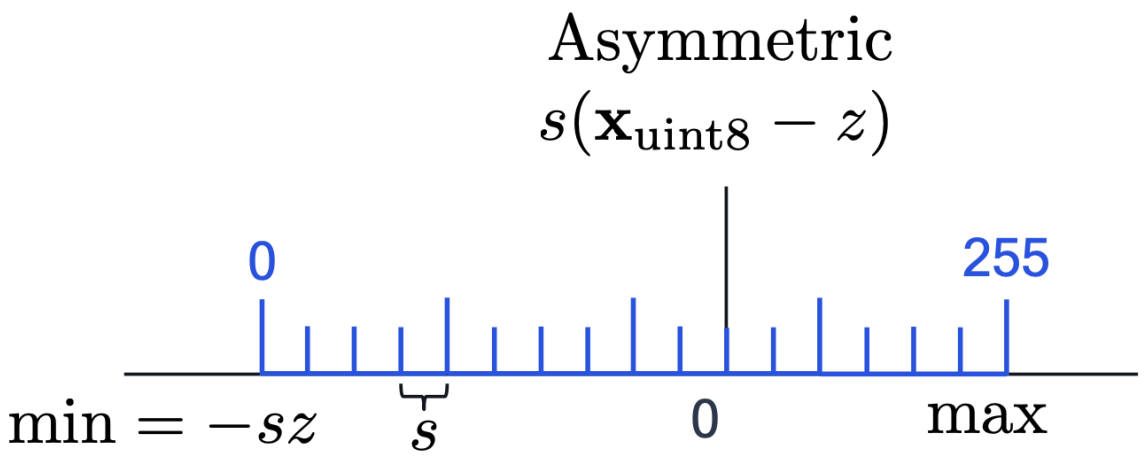

Picture taken from [2]

# Symmetric and asymmetric quantization

$$\mathbf{x}_{\text{int}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil + z; 0, 2^b - 1\right)$$

Quantization

$$\text{clamp}(x; a, c) = \begin{cases} a, & x < a, \\ x, & a \leq x \leq c, \\ c, & x > c. \end{cases}$$




Picture taken from [2]

# Symmetric and asymmetric quantization

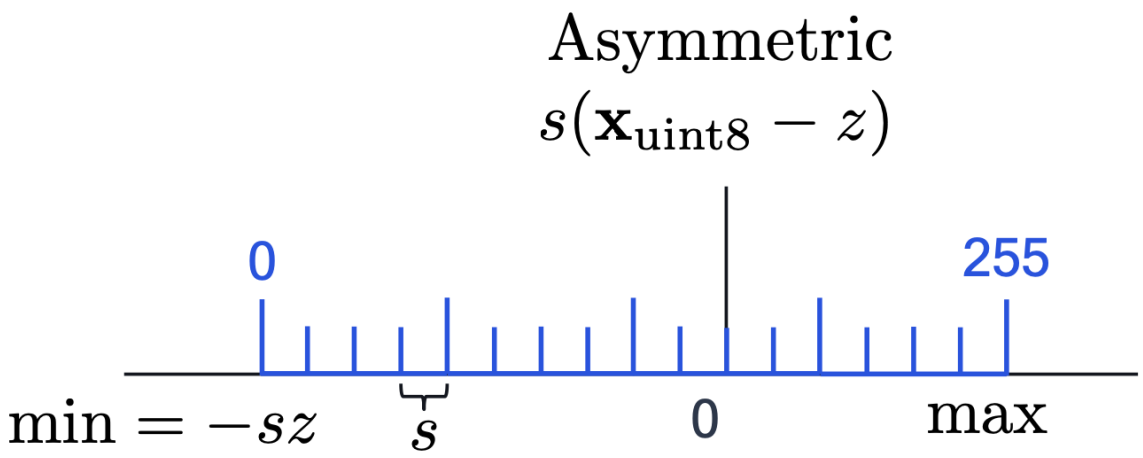
$$\mathbf{x}_{\text{int}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil + z; 0, 2^b - 1\right) \quad \Rightarrow \quad \text{clamp}\,(x; a, c) = \begin{cases} a, & x < a, \\ x, & a \leq x \leq c, \\ c, & x > c. \end{cases}$$

Quantization

$$\mathbf{x} \approx \widehat{\mathbf{x}} = s\,(\mathbf{x}_{\text{int}} - z) \quad \Rightarrow \quad \begin{cases} q_{\text{min}} = -sz \\ q_{\text{max}} = s(2^b - 1 - z) \end{cases}$$

De-quantization

Quantization grid limits




Picture taken from [2]

# Clipping and rounding errors

- Any values of x that lie outside the range

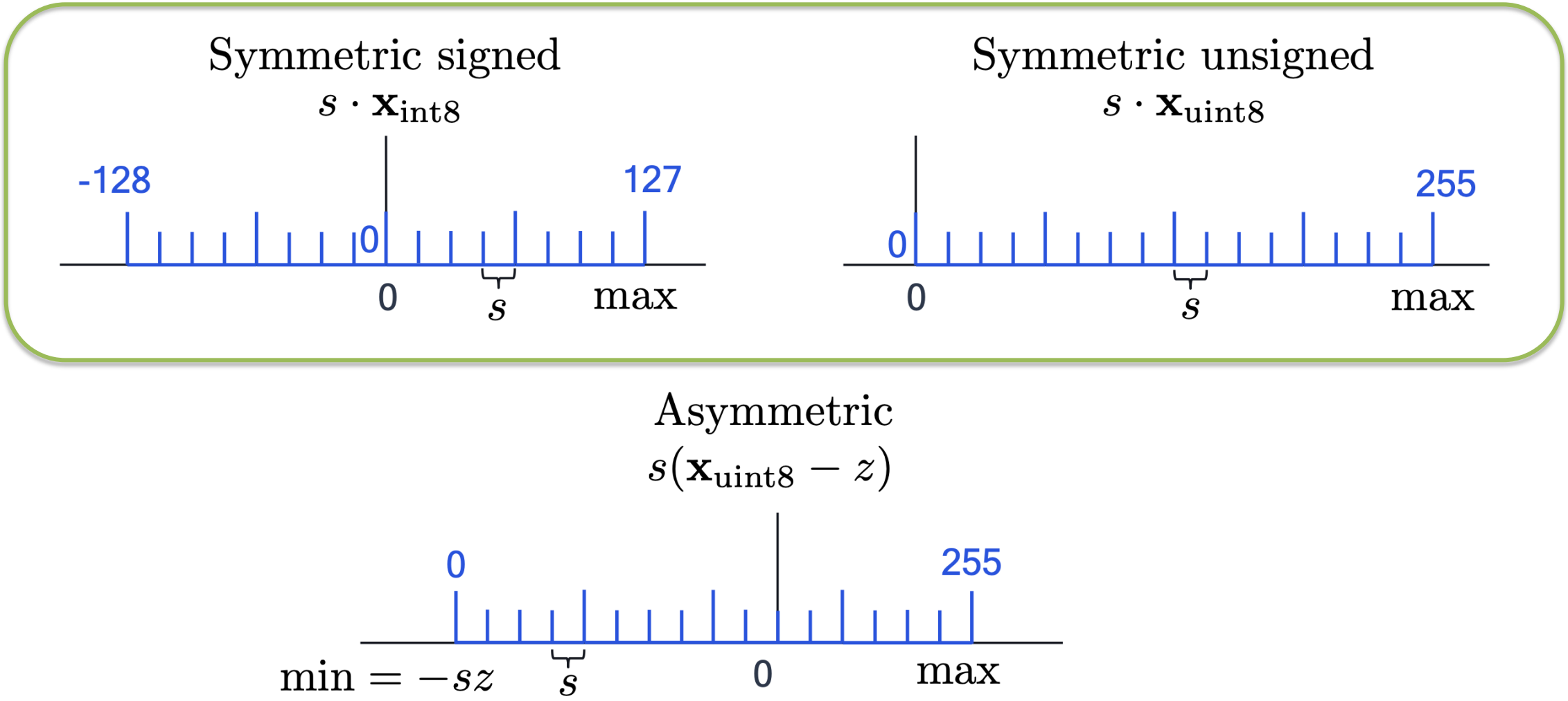
$$\begin{cases} q_{\min} = -sz \\ q_{\max} = s(2^b - 1 - z) \end{cases}$$

  will be clipped to its limits, incurring a clipping error.
- Increasing the scale factor $s$ allows to expand the quantization range, hence reducing the clipping error
- However, increasing the scale factor leads to increased rounding error:

$$\left[-\frac{1}{2}s, \frac{1}{2}s\right]$$

Picture taken from [2]

# Symmetric and asymmetric quantization

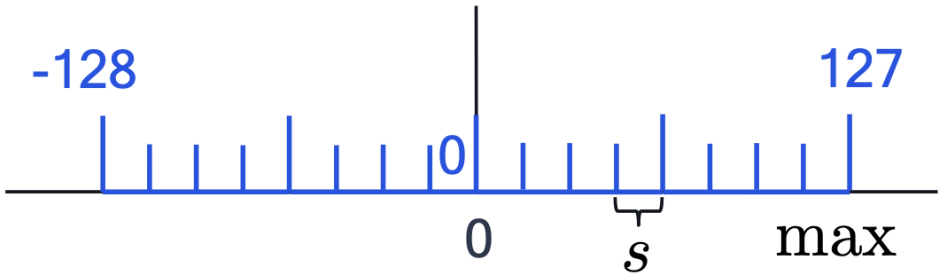



Picture taken from [2]

# Symmetric and asymmetric quantization

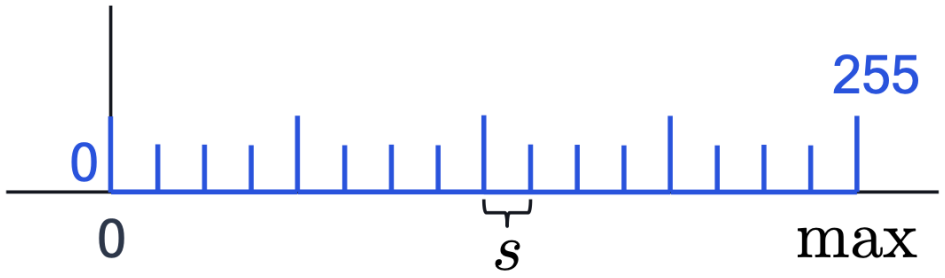
Symmetric signed

$s \cdot \mathbf{x}_{\text{int8}}$




Symmetric unsigned

$s \cdot \mathbf{x}_{\text{uint8}}$




$$\mathbf{x}_{\text{int}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil; 0, 2^b - 1\right) \quad \text{for unsigned integers}$$

$$\mathbf{x}_{\text{int}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil; -2^{b-1}, 2^{b-1} - 1\right) \quad \text{for signed integers}$$

$$\widehat{\mathbf{x}} = s\,\mathbf{x}_{\text{int}}$$

De-quantization step

Picture taken from [2]

# Expressing FP inputs and weights with INT8

$$\widehat{\mathbf{x}} = s_{\mathbf{x}} \cdot \mathbf{x}_{\text{int}} \approx \mathbf{x}$$

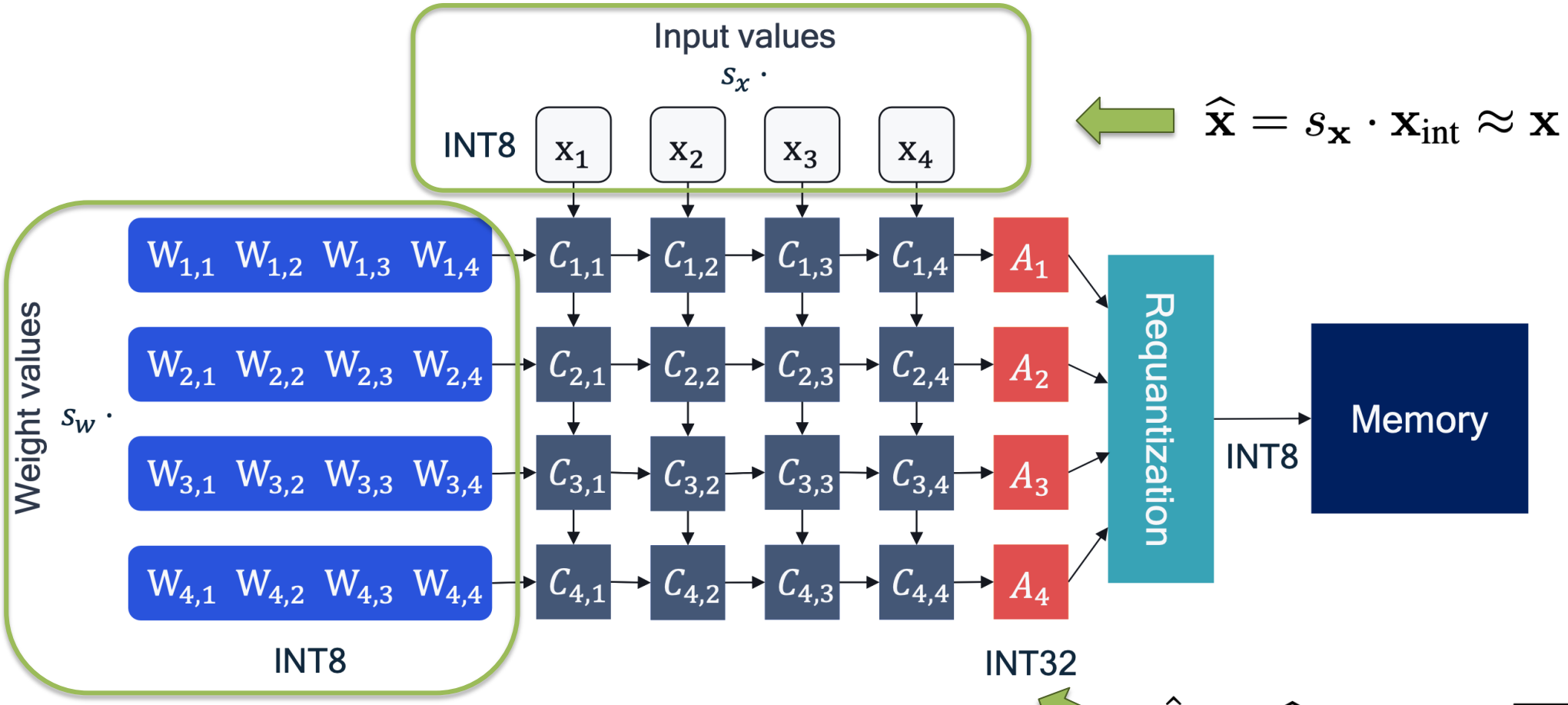
A floating-point vector x can be expressed approximately as a scalar multiplied by a vector of integer values

$$\begin{aligned}\hat{A}_n &= \widehat{\mathbf{b}}_n + \sum_m \widehat{\mathbf{W}}_{n,m}\, \widehat{\mathbf{x}}_m \\ &= \widehat{\mathbf{b}}_n + \sum_m \left(s_{\mathbf{w}} \mathbf{W}^{\text{int}}_{n,m}\right)\left(s_{\mathbf{x}} \mathbf{x}^{\text{int}}_m\right) \\ &= \widehat{\mathbf{b}}_n + s_{\mathbf{w}} s_{\mathbf{x}} \sum_m \mathbf{W}^{\text{int}}_{n,m}\, \mathbf{x}^{\text{int}}_m\end{aligned}$$

- By quantizing the weights and activations we can write the quantized version of the MAC
- Separate scale factors:
  - Weights $s_w$
  - Activations $s_x$

Picture taken from [2]

# A schematic overview of matrix-multiply logic in NN accelerator hardware: the quantized inference




$$\hat{\mathbf{x}} = s_{\mathbf{x}} \cdot \mathbf{x}_{\text{int}} \approx \mathbf{x}$$

$$\hat{A}_n = \hat{\mathbf{b}}_n + s_{\mathbf{w}} s_{\mathbf{x}} \sum_m \mathbf{W}^{\text{int}}_{n,m} \mathbf{x}^{\text{int}}_m$$

Picture taken from [2]

# When is quantization implemented?

# Two different ways to implement quantization

- Post-training: quantization (PTQ): a pre-trained FP32 network is directly converted into a fixed-point network without the need for the original training pipeline
- Quantization-aware training (QAT): quantization and training are jointly addressed (back-propagation implemented with simulated quantization within the training pipeline)

# Post-training quantization (PTQ)

Post-training quantization (PTQ) algorithms:

- Receives in **input** a pre-trained FP32 network
- Produces in **output** a fixed-point network without retraining

The main challenge is the definition of the quantization range settings.

Two main approaches:

$$q_{\min} = \min \mathbf{V},$$
$$q_{\max} = \max \mathbf{V},$$

Min-Max

$$\arg\min_{q_{\min}, q_{\max}} \left\| \mathbf{V} - \hat{\mathbf{V}}(q_{\min}, q_{\max}) \right\|_F^2,$$

Mean Squared Error (MSE)

Picture taken from [2]

# PTQ: advantages and disadvantages

- Effective and fast to implement since they do not require retraining of the network with labeled data

- Limited performance when low-bit quantization of activations (e.g., 4-bit and below) is considered
- Post-training techniques may not be enough to mitigate the large quantization error incurred by low-bit quantization

# Quantization-aware training (QAT)

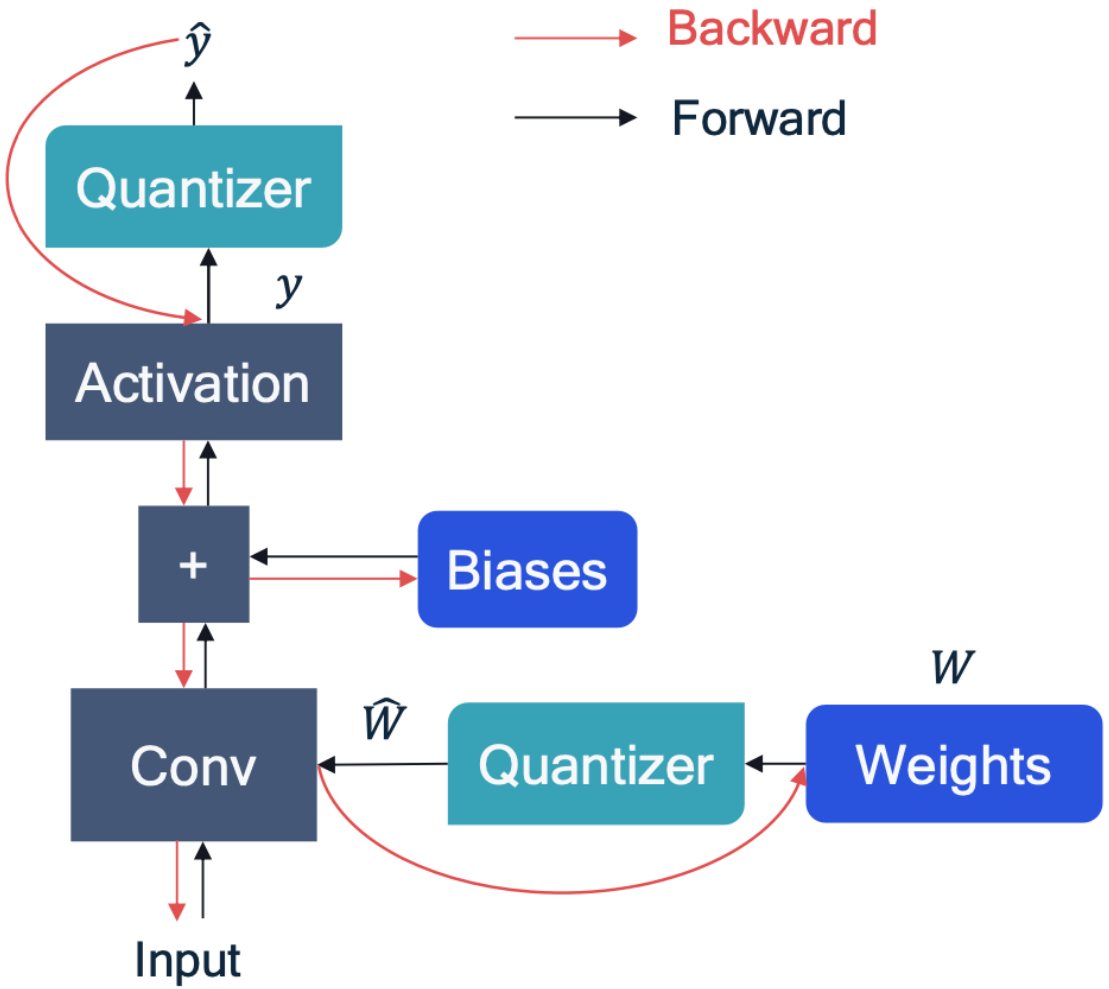
Higher accuracy comes with the usual costs of neural network training, i.e., longer training times, need for labeled data and hyper-parameter search

**The challenge:** back-propagate through the simulated quantizer block

**The solution:** approximate the gradient using the *straight-through estimator* (Bengio et al. 2013), which approximates the gradient of the rounding operator as 1

$$\frac{\partial \lfloor y \rceil}{\partial y} = 1$$

# QAT: forward and backward propagation




Picture taken from [2]

# 2) Task dropping

# Task dropping

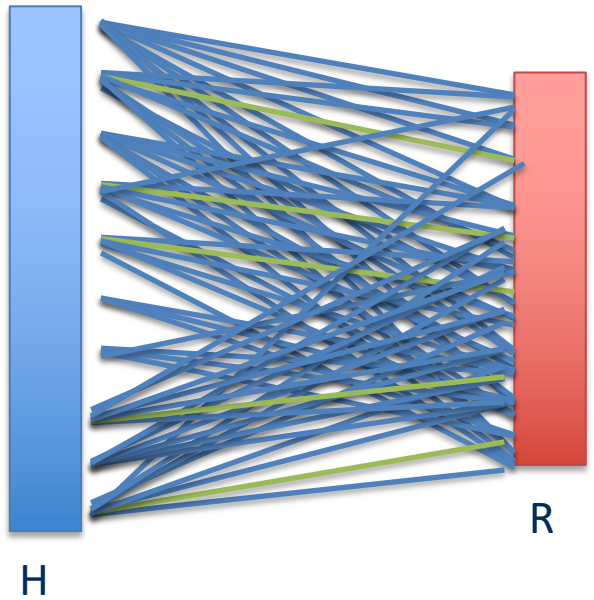
- Reduce the number of operations and model size. This might have a positive effect on computational demand and memory footprint!
- These techniques can be classified as:
  - ✓ **network pruning**
  - ✓ **network architecture design**
  - ✓ **transfer learning**
  - ✓ **knowledge distillation**

# 1) Network Pruning

- To make training easier, NNs are usually overparameterized
- Therefore, a large amount of the weights in a NN is redundant and can be removed (i.e., set to zero)
- Aggressive network pruning often requires some fine tuning of the weights to maintain the original accuracy

Simple pruning

Structured pruning



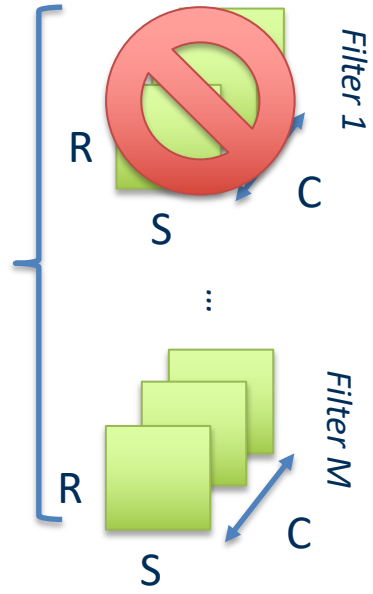
# 1) Network Pruning

- To make training easier, NNs are usually overparameterized.
- Therefore, a large amount of the weights in a NN is redundant and can be removed (i.e., set to zero)
- Aggressive network pruning often requires some fine tuning of the weights to maintain the original accuracy

Simple pruning

Structured pruning

Groups of weights are pruned
(entire row, entire column, filters, etc…)



| Metrics | AlexNet | | GoogLeNet v1 | |
|---|---|---|---|---|
| | dense | sparse | dense | sparse |
| Top-5 error | 19.6 | 20.4 | 11.7 | 12.7 |
| Number of CONV Layers | 5 | 5 | 57 | 57 |
| Depth in (Number of CONV Layers) | 5 | 5 | 21 | 21 |
| Filter Sizes | 3,5,11 | | 1,3,5,7 | |
| Number of Channels | 3-256 | | 3-832 | |
| Number of Filters | 96-384 | | 16-384 | |
| Stride | 1,4 | | 1,2 | |
| NZ Weights | 2.3M | 351k | 6.0M | 1.5M |
| NZ MACs | 395M | 56.4M | 806M | 220M |
| FC Layers | 3 | 3 | 1 | 1 |
| Filter Sizes | 1,6 | | 1 | |
| Number of Channels | 256-4096 | | 1024 | |
| Number of Filters | 1000-4096 | | 1000 | |
| NZ Weights | 58.6M | 5.4M | 1M | 870k |
| NZ MACs | 14.5M | 1.9M | 635k | 663k |
| Total NZ Weights | 61M | 5.7M | 7M | 2.4M |
| Total NZ MACs | 410M | 58.3M | 806M | 221M |

**3-10x reduction in complexity, 1% loss in accuracy**

Picture taken from [1]

# 2) Network architecture design

Replace a large filter with a series of small filters (fewer weights in total)

**Before Training**
(concatenating smaller filters to emulate
larger ones)

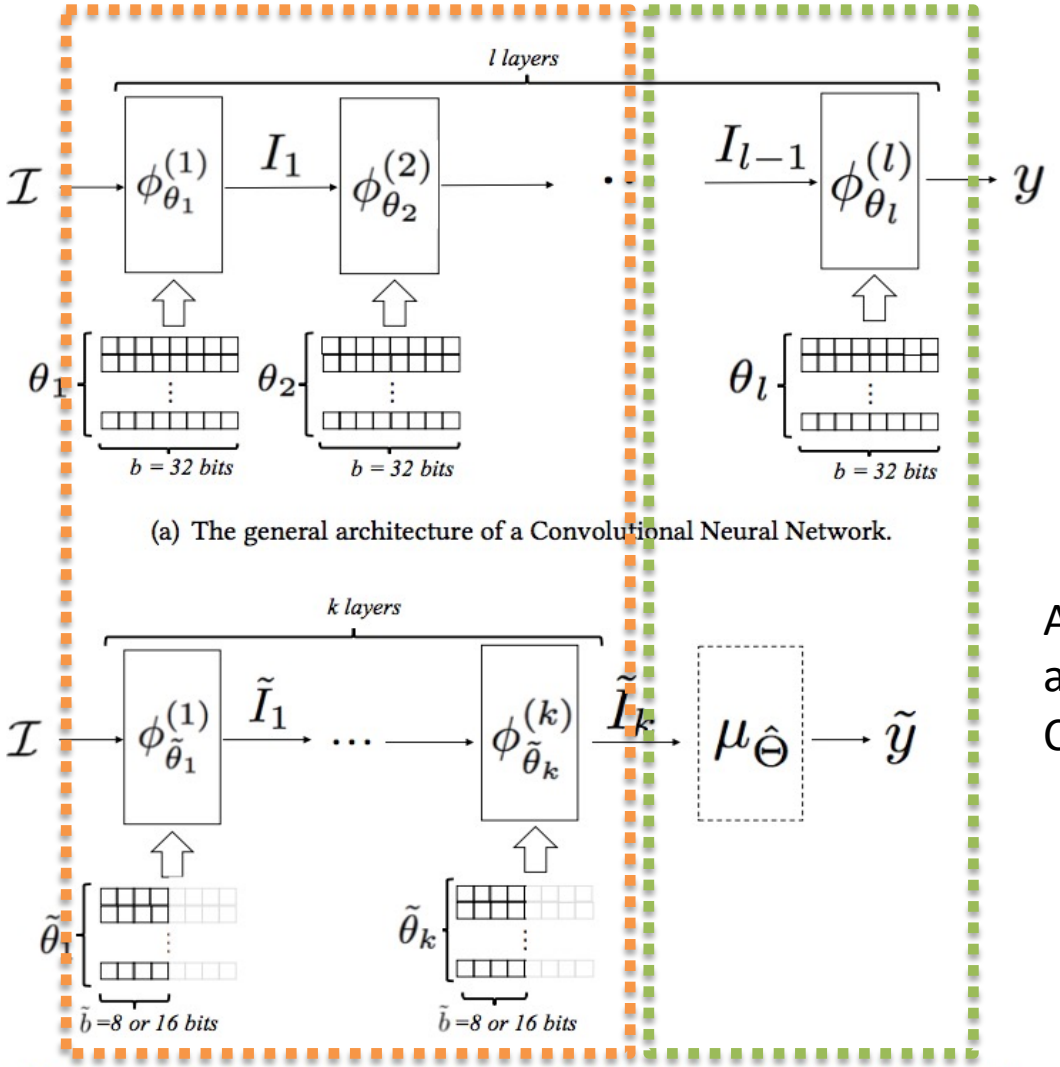
- From 5x5 to two 3x3 filters
- From NxN to the combination of 1xN and Nx1 filter

**After Training**
(tensor decomposition to decompose filters w/o impacting accuracy)

- Low rank approximation
- Canonical Polyadic decomposition

# 3) Transfer learning

**Transfer learning of the first *k* processing layers**



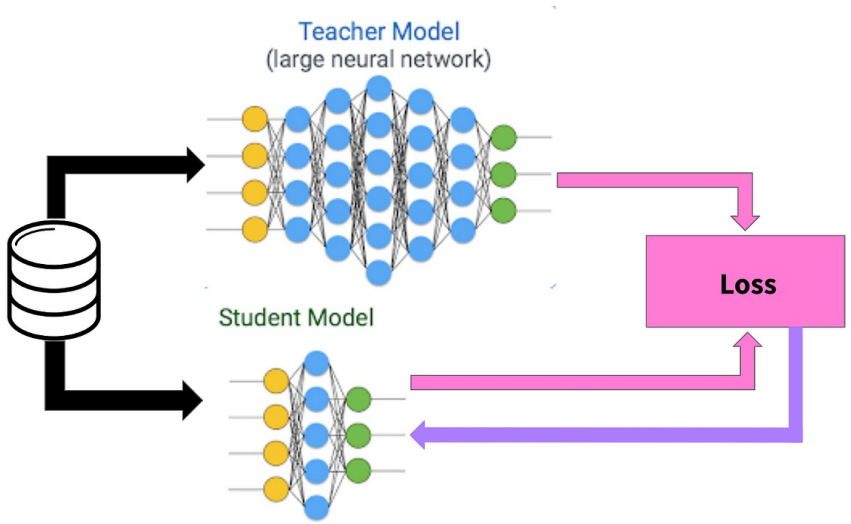

(a) The general architecture of a Convolutional Neural Network.

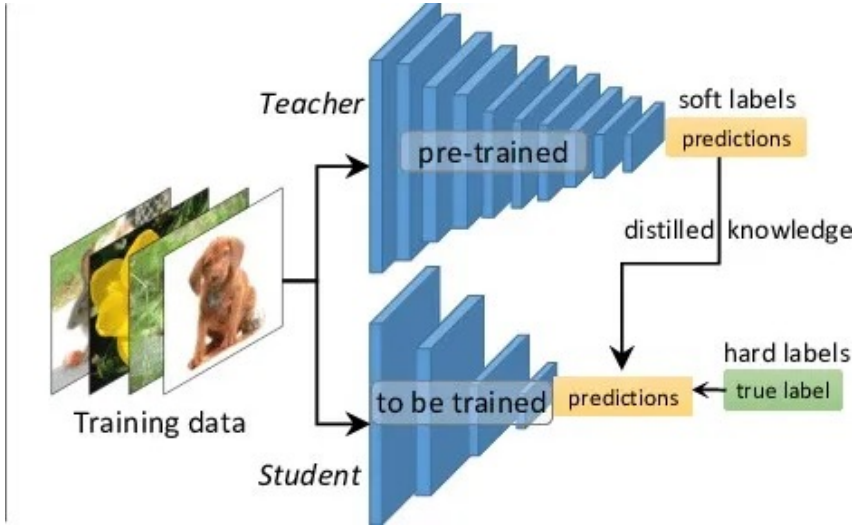
(b) The general architecture of a Convolutional Neural Network for Embedded Systems.

**Replacing the last *l-k* layers with a unique processing layer**

Application-specific and approximated CNN

# 4) Knowledge distillation




Cross-entropy gradient w.r.t. each logit $z_i$

$$\frac{\partial C}{\partial z_i} = \frac{1}{T}(q_i - p_i) = \frac{1}{T}\left(\frac{e^{z_i/T}}{\sum_j e^{z_j/T}} - \frac{e^{v_i/T}}{\sum_j e^{v_j/T}}\right)$$

Temperature T

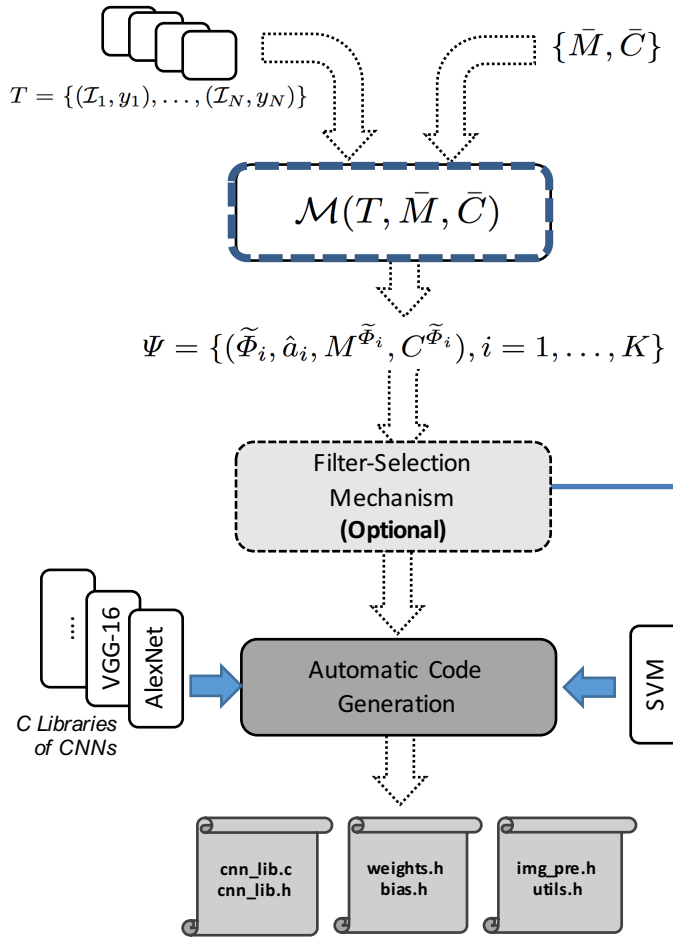
Student Model

Teacher Model

Picture taken from [3]

# Summing up …

# The methodology for designing Tiny Deep Learning solutions




M receives in input:

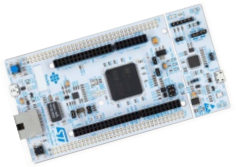
- a dataset T= $\{(\tilde{I}_k^1, y_1^o), \ldots, (\tilde{I}_k^N, y_N^o)\}$
- the constraints on memory M and computation C

The output of M is the Pareto set of $\widetilde{\Phi}$ satisfying the constrains on M and C

Identify those filters providing output features that are truly beneficial to support the classification by μ and discard the others:

$$f = 1, \ldots, F^{max}$$

# Porting the approximated AlexNet to two off-the-shelf IoT units



| Platform | STM32F7 | Raspberry Pi 3B | | |
|---|---|---|---|---|
| CPU | ARM M7 @ 167 MHz | ARM11 @ 1.2 GHz | | |
| RAM | 512 KB | 1024 MB | | |
| $M$ | 52 KB | 102 KB | | |
| $\overline{C}$ | $100.0 \times 10^6$ | $100.0 \times 10^6$ | | |
| CNN $\Phi$ | AlexNet | AlexNet | AlexNet | AlexNet |
| k | 5 | 5 | 5 | 5 |
| q | 0 | 0 | 0 | 4 |
| Filter-selection mechanism | yes | yes | no | no |
| f | 1 | 1 | - | - |
| $\mu_C$ | Decision Tree | Decision Tree | SVM | SVM |
| $\hat{a}$ | 87.9 | 87.9 | 99.3 | 99.4 |
| $M^{\tilde{\Phi}}$ | 1.4 KB | 1.4 KB | 68 KB | 34 KB |
| $C^{\tilde{\Phi}}$ | $1.09 \times 10^6$ | $1.09 \times 10^6$ | $52.7 \times 10^6$ | $52.7 \times 10^6$ |
| Exec. Time (ms) | 2700 | 178 | 8687 | 8687 |