PhD Course: Optimized Execution of Neural Networks at the Edge

Pruning, Quantization, and Adaptive Inference

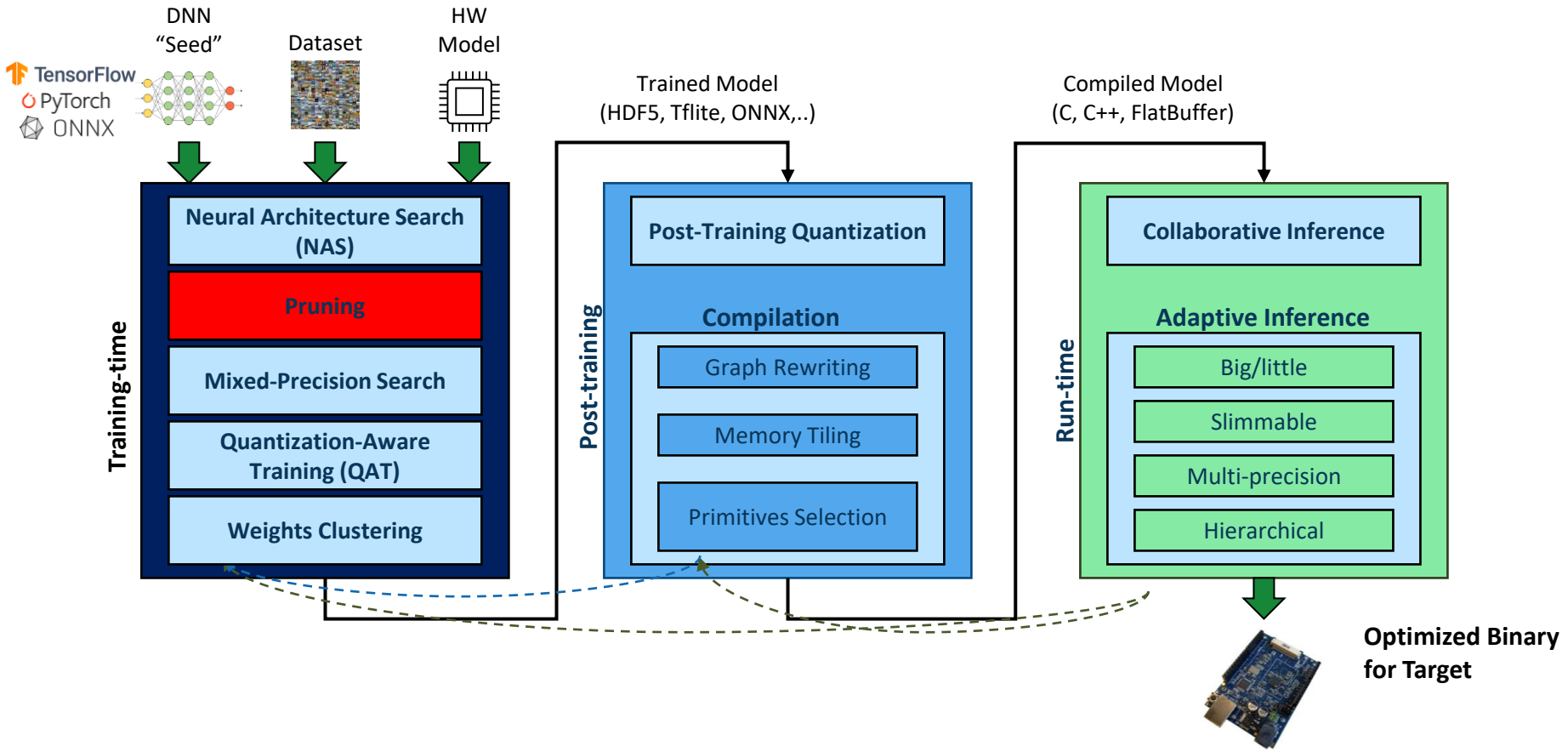
Daniele Jahier Pagliari, Alessio Burrello

daniele.jahier@polito.it, alessio.burrello@polito.it

Department of Control and Computer Engineering, Politecnico di Torino

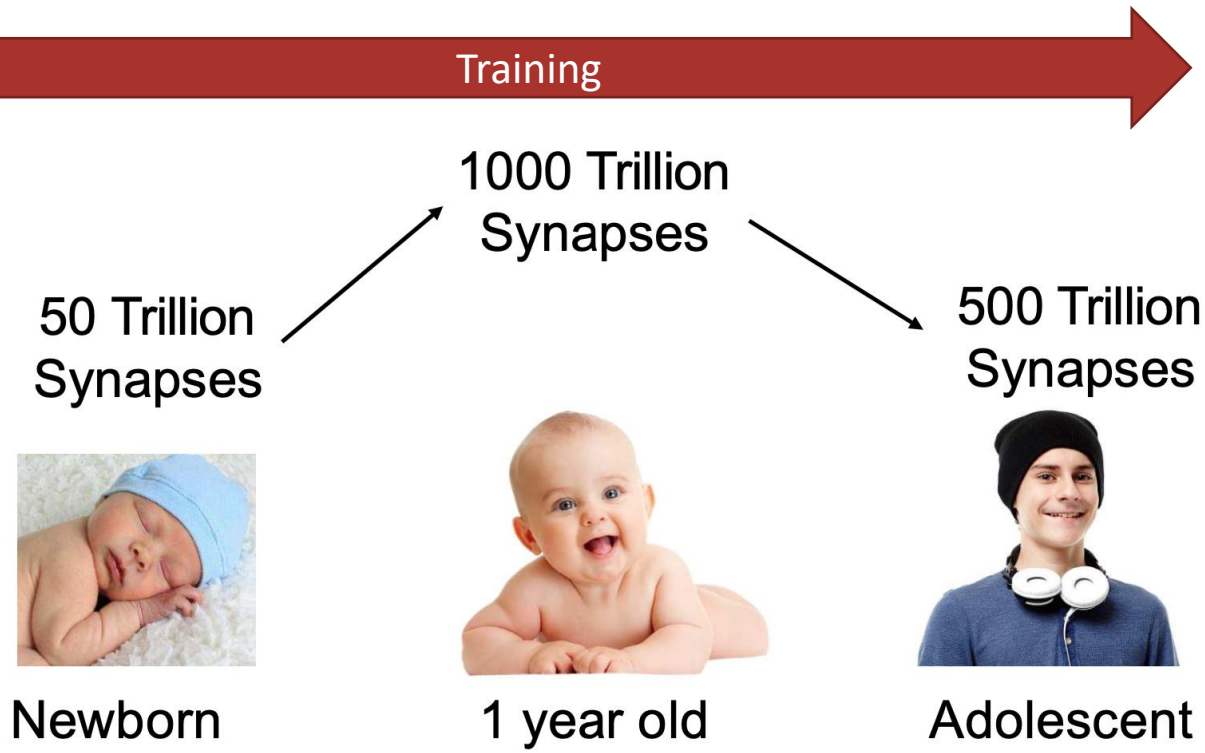
Politecnico di Torino

Recap: Our Flow



Pruning

- Pruning: eliminate unimportant portions of a DNN (and relative computations)
- Rough biological analysis:

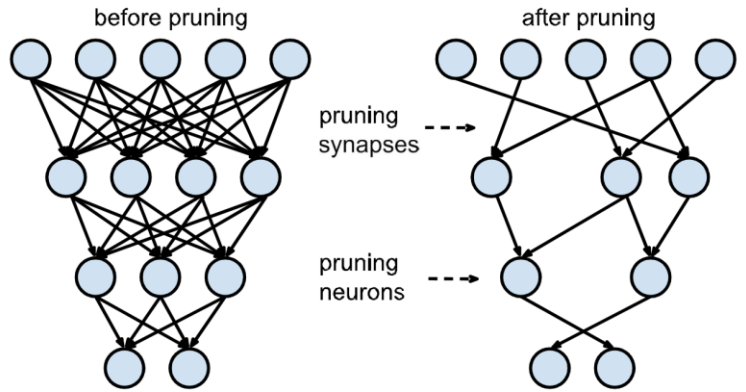


Christopher A Walsh. Peter Huttenlocher (1931-2013). Nature, 502(7470):172–172, 2013.

Pruning

- **Weights Pruning**
  - Eliminate **some weights** and the corresponding input/output connections, individually or in sets
  - Usually done at training-time.
  - Most common (and usually most beneficial) pruning approaches.

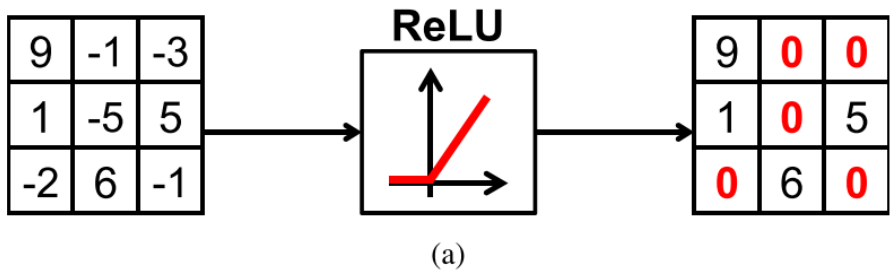
- **Activations Pruning**
  - Exploit sparsity in activations (e.g. after ReLU)
  - Skip computations relative to zero (or small) acti
  - **Runtime optimization**!



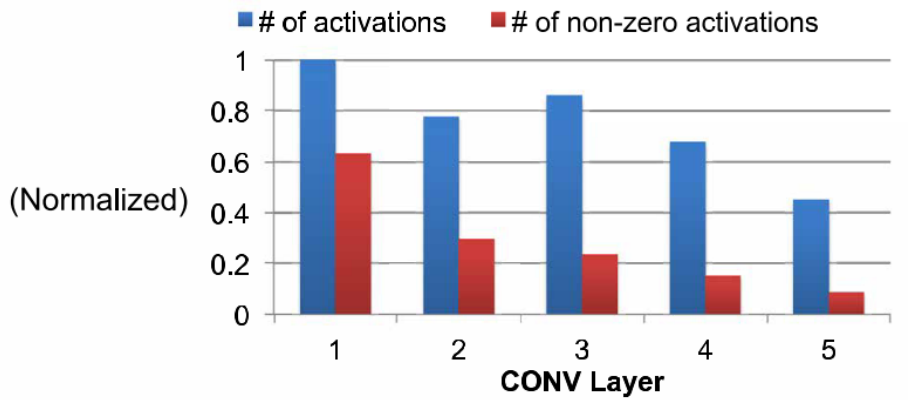
Han et al. Learning both Weights and Connections for Efficient Neural Networks, NIPS'15

Activations Pruning (Briefly)

- Deep NNs have <u>natural</u> sparsity properties, especially when using certain kinds of activation functions (e.g. ReLU):



(a)



Sparsity in AlexNet layers

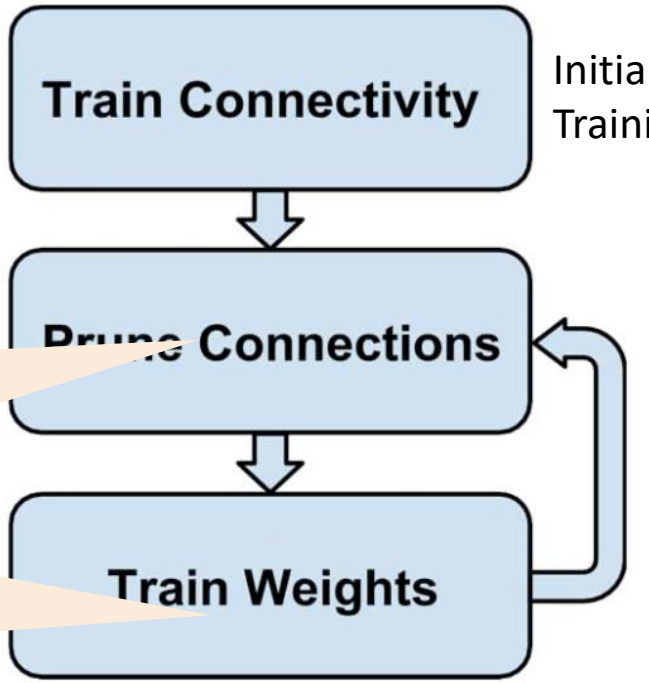
[1] V. Sze, Y. -H. Chen, T. -J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, 2017

Benefits of Weights Pruning

- Reduce model size.
  - Sparse weights can be stored in compressed formats (e.g. CSR).
- Main memory occupation and latency/energy are reduced only if it is possible to directly perform computations on the compressed format.
- ..and if computations involving zero values may be skipped. Not trivial!
  - Sparse computations are difficult to accelerate
- Excellent reference:
  - T. Hoefler et al, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks", JMLR 2021

Unstructured (Magnitude-based) Pruning

- The most famous approach to weights pruning is based on iteratively eliminating the weights whose magnitude is small
- Unstructured: prunes individual weights.
- Often > 80% of the weights can be pruned with no impact on accuracy.



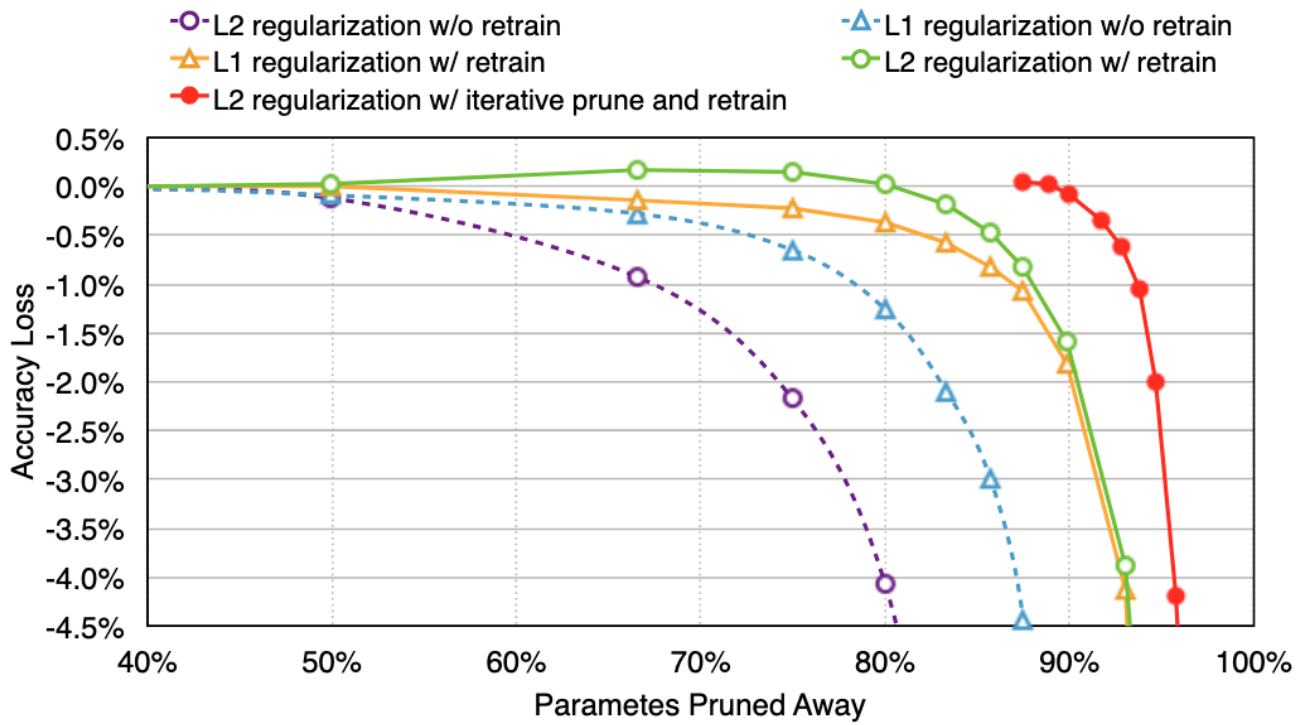
Sparsity Target = 50%

- Retrain model while **masking** pruned weights' gradients
- Ensure those weights are not modified by gradient descent.

.Han et al, "Learning both Weights and Connections for Efficient ral Networks, NIPS 2015.

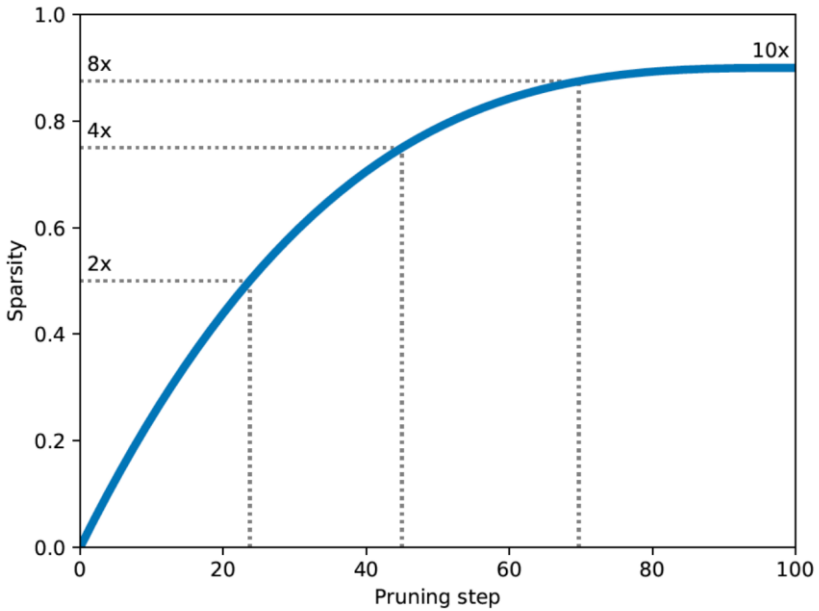
Magnitude-Based Pruning

- Retraining and iterating are fundamental (Example on AlexNet)



Pruning Schedules

- Gradually increasing the amount of pruned weights may yield superior performance compared to “constant” pruning.
- Many so-called “pruning schedules” have been proposed
- Example: Polynomial decay
  - Increase the sparsity faster at the beginning, when the model contains a lot of redundant connections, then gradually reduce the pruning rate.



Pruning Objectives

- Pure magnitude-based pruning aims at eliminating as many weights as possible
  - However, depending on the hardware and objective, **not all layers are equally critical**.

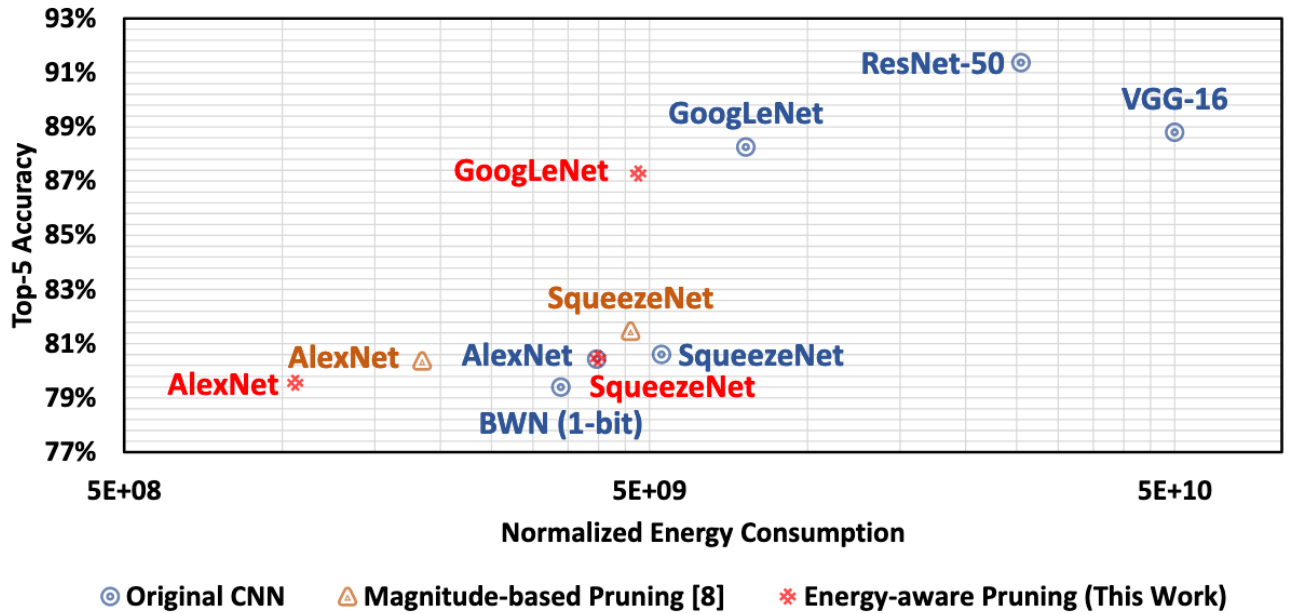
- For instance, a layer's criticality in terms of **energy consumption** depends on the amount of memory accesses at different levels (hence the data reuse patterns), the number of computations, etc.

[Source] Yang et al, Designing Energy-Efficient Convolutional Neural Networksusing Energy-Aware Pruning

Pruning Objectives

- Pruning methods can be adapted to other objectives, for instance assigning a priority to different layers.

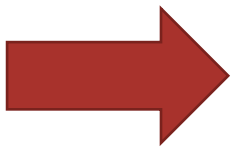
- Energy-aware pruning requires an energy model of the target HW.

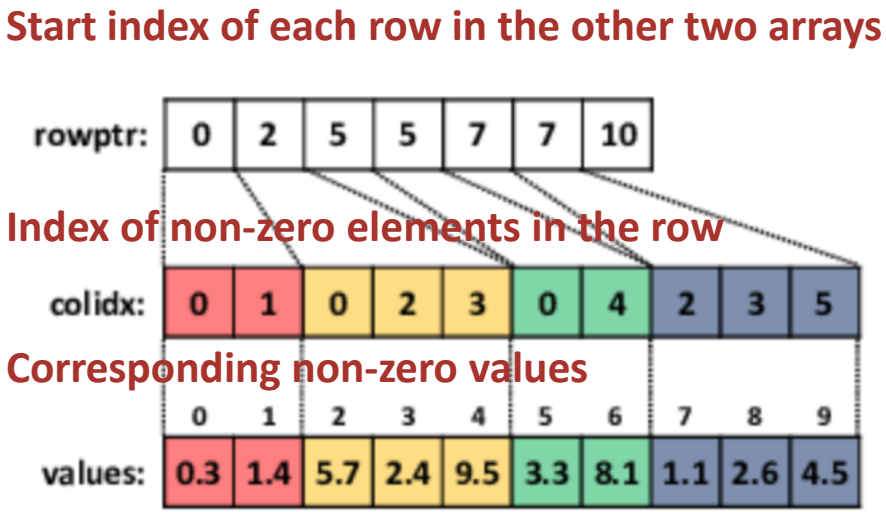


[Source] Yang et al, Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning

Sparse Weight Matrices Representation

- The Compressed Sparse Row (CSR) format is one simple format to represent sparse matrices in main memory, designed to allow a simple decoding scheme to access elements for computations.





36 float elements (144 Bytes), 10 non-zero

10 float elements + 17 int indices (108 Bytes)

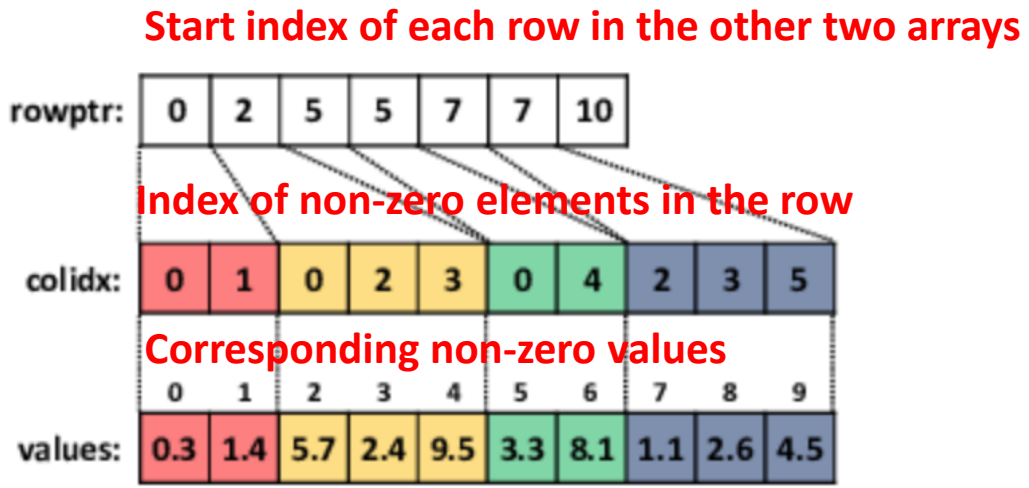
- Real compression ratios are higher than this.

[1] V. Sze, Y. -H. Chen, T. -J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, 2017

Sparse Weight Matrices Representation

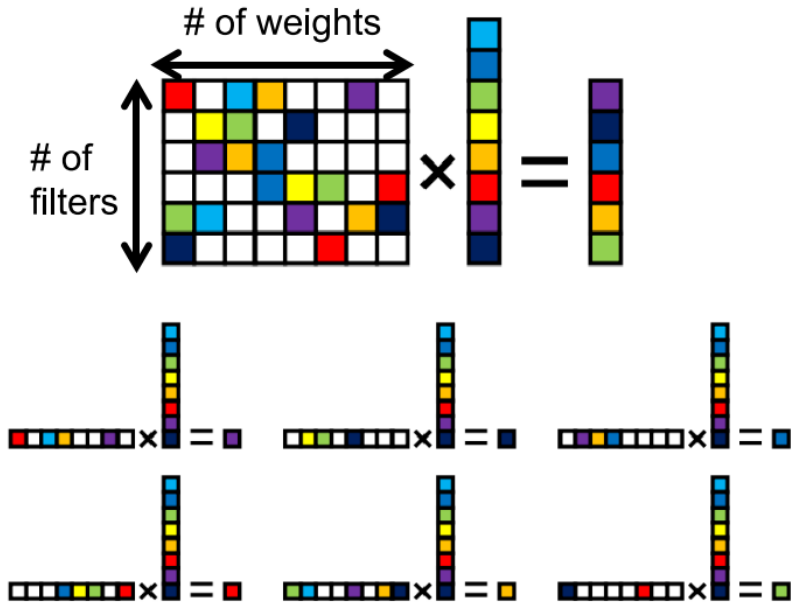
- Iterate over a CSR matrix (by row):

```
for i in len(rowptr) - 1:
  for k in range(rowptr[i],
rowptr[i+1]):
    j = colidx[k]
    v = values[k]
    // v contains m[i][j]
```



Sparse Weight Matrices Representation

- CSR is only efficient if you read the matrix by row. Reading by column requires “searching” for the index.
- For each row, you need a different sub-set of the input vector elements to perform the MAC.

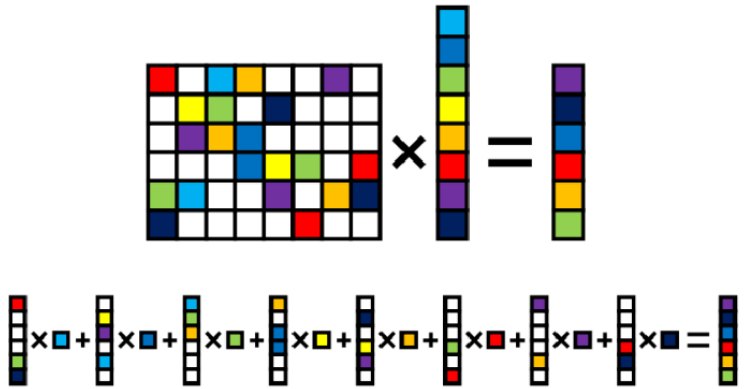


Sparse Weight Matrices Representation

- So, you must either store the entire input vector in registers, or read it multiple times with an irregular access pattern
- Difficult to optimize across memory hierarchies!!

Sparse Weight Matrices Representation

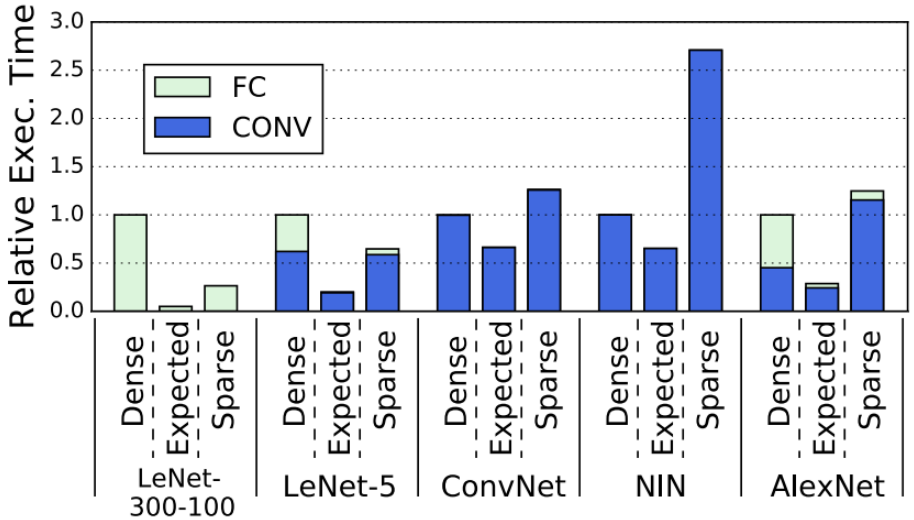
- Compressed Sparse Column (CSC):
  - Opposite of CSR (store column pointers and row indices)
  - Advantages, read matrix by column. Allows to read one input element at a time, but the (partial) output vector is accessed irregularly



- So, CSC is more convenient when the output size is smaller than the input size

Unstructured Pruning Problem

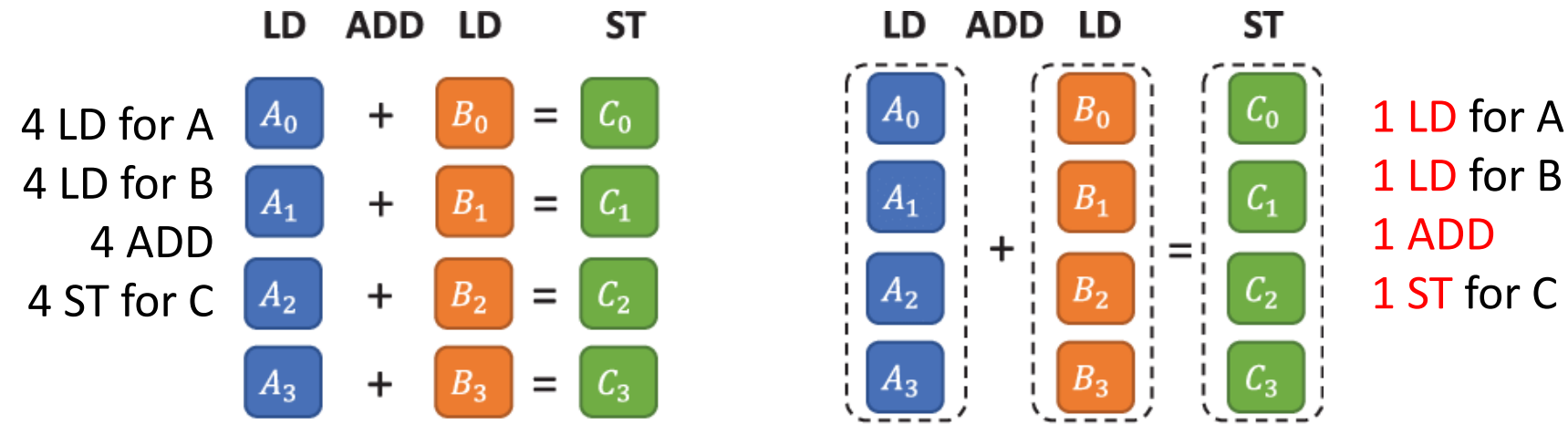
- **Common problem: accelerating sparse computations is tricky.**
  - Irregular access patterns for activations (no vector load/stores, more cache misses, etc)
  - Harder load balancing and SIMD vectorization
  - **Reduced benefits or even slow-downs**



[3] Yu, J., Lukefahr, A., Palframan, D., Dasika, G., Das, R., & Mahlke, S. (2017). Scalpel: Customizing dnn pruning to the underlying hardware parallelism. ACM SIGARCH Computer Architecture News, 45(2), 548-560.

SISD vs SIMD (Reminder)

- SIMD exploits parallelism on different data: operation concurrently applied to different pieces of data



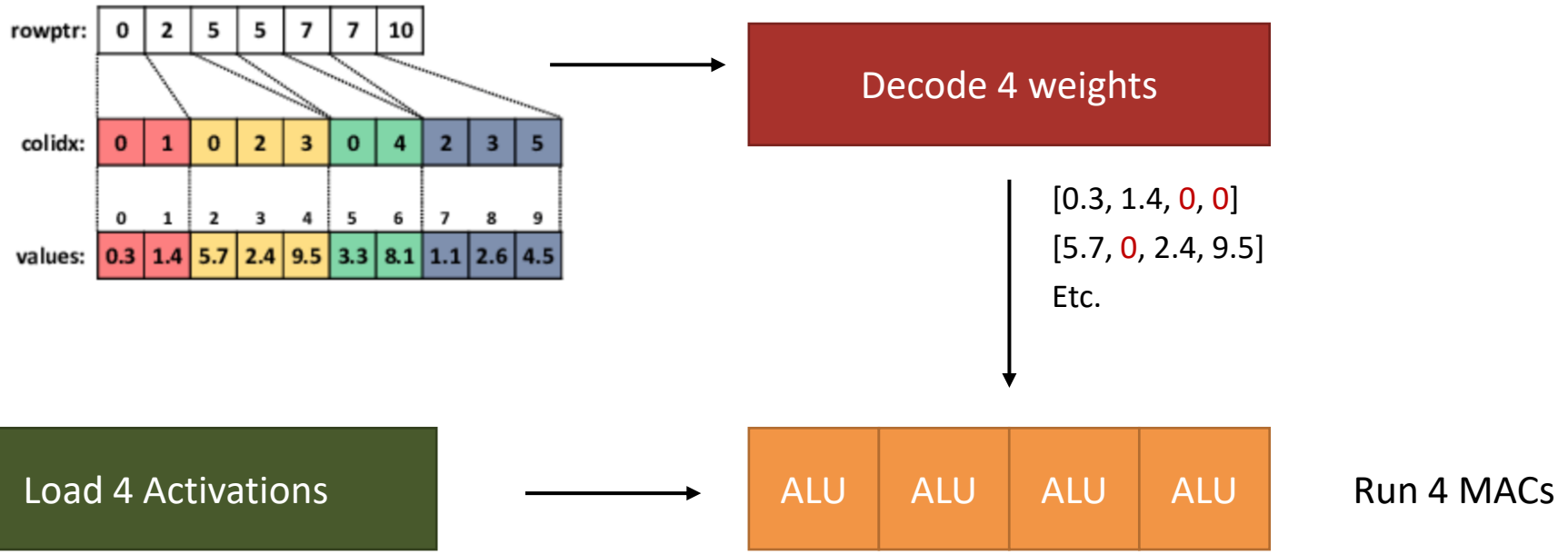

```
for (i=0; i < N; i++)
  C[i] = A[i] + B[i];
```

⇒

```
for (i=0; i < N/4; i++)
  C[i:4*i] = A[i:4*i] + B[i:4*i];
```

With a parallelism of 4 (memory access = 1/4)

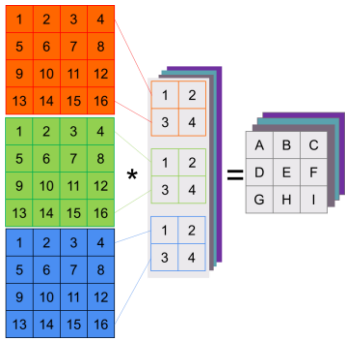
Unstructured vs Structured Pruning (SIMD)



- **Approach 1:** Load only activations corresponding to non-zero weights
  - Cannot use SIMD Load/Store if not consecutive

- **Approach 2:** Load 4 consecutive activations regardless:
  - Cannot skip a LOAD+MAC unless we have #ALU consecutive zeros
  - Cannot reduce energy unless we can shut-off individual ALUs

Convolution as GEMM (Reminder)



1	2	5	6	1	2	5	6	1	2	5	6
2	3	6	7	2	3	6	7	2	3	6	7
3	4	7	8	3	4	7	8	3	4	7	8
5	6	9	10	5	6	9	10	5	6	9	10
6	7	10	11	6	7	10	11	6	7	10	11
7	8	11	12	7	8	11	12	7	8	11	12
9	10	13	14	9	10	13	14	9	10	13	14
10	11	14	15	10	11	14	15	10	11	14	15
11	12	15	16	11	12	15	16	11	12	15	16

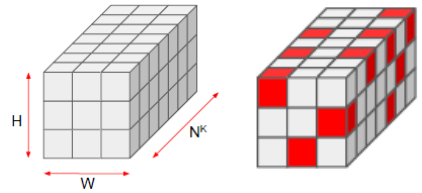
·

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

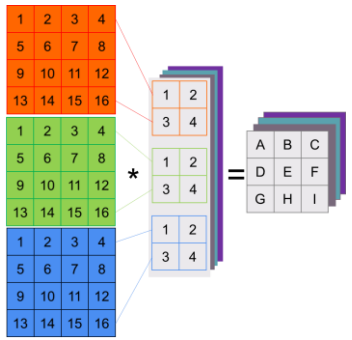
=

A	A	A	A
B	B	B	B
C	C	C	C
D	D	D	D
E	E	E	E
F	F	F	F
G	G	G	G
H	H	H	H
I	I	I	I

Sparsity ≠ Optimization



- Weight zeroing is not simplification (unless sparse compute libraries available)
  - Same memory footprint
  - Same num. accesses
  - Same power consumption
    - Same #MAC



1	2	5	6	1	2	5	6	1	2	5	6
2	3	6	7	2	3	6	7	2	3	6	7
3	4	7	8	3	4	7	8	3	4	7	8
5	6	9	10	5	6	9	10	5	6	9	10
6	7	10	11	6	7	10	11	6	7	10	11
7	8	11	12	7	8	11	12	7	8	11	12
9	10	13	14	9	10	13	14	9	10	13	14
10	11	14	15	10	11	14	15	10	11	14	15
11	12	15	16	11	12	15	16	11	12	15	16

*

1	0	1	0
0	2	0	2
0	3	0	3
4	0	4	0
0	1	1	1
2	2	0	0
0	0	0	3
4	0	4	0
0	1	0	0
0	0	2	0
0	3	0	0
4	0	0	4

sparsity 50%

=

A	A	A	A
B	B	B	B
C	C	C	C
D	D	D	D
E	E	E	E
F	F	F	F
G	G	G	G
H	H	H	H
I	I	I	I

Unstructured vs Structured Pruning

- Cache misses increase with sparsity, increasing energy consumption and latency.

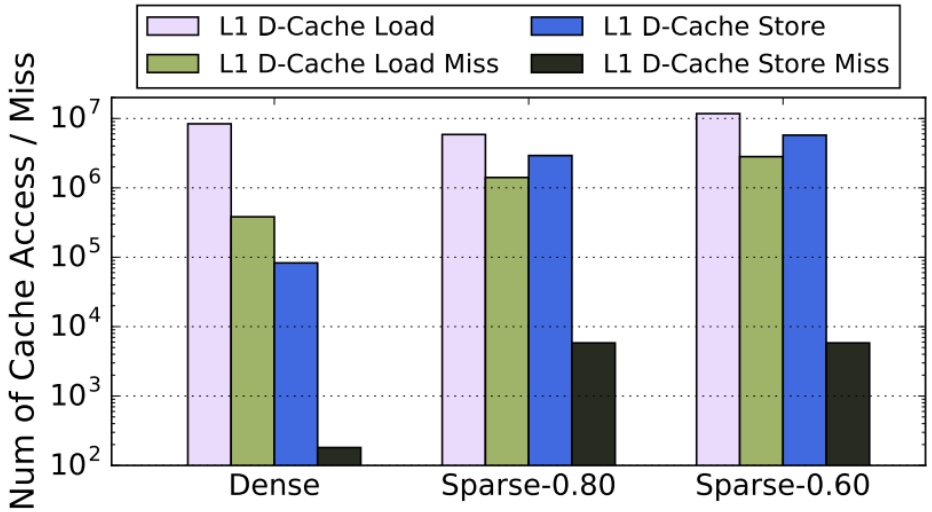


Figure 5: Numbers of cache access and cache miss in the computation of the second CONV layer (*conv2*) in AlexNet on Intel Core i7-6700 CPU. The original dense layer (Dense), the sparse layer with 80% (Sparse-0.80) and 60% (Sparse-0.60) of weights removed are tested. The matrices are randomly generated.

[3] Yu, J., Lukefahr, A., Palframan, D., Dasika, G., Das, R., & Mahlke, S. (2017). Scalpel: Customizing dnn pruning to the underlying hardware parallelism. ACM SIGARCH Computer Architecture News, 45(2), 548-560.

Structured Pruning

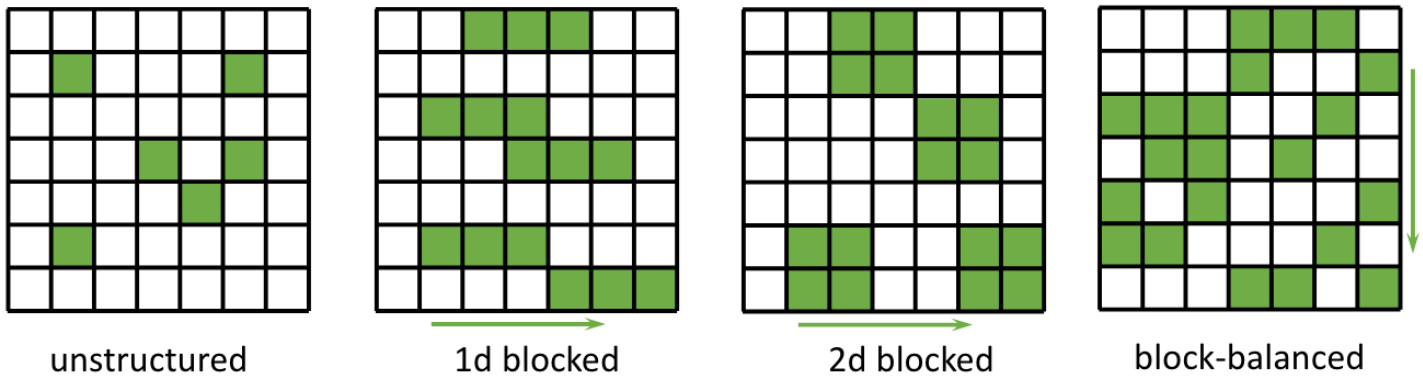
- Solutions to accelerate pruning:
  1. Design custom HW to support sparsity (but then, what if you have a dense NN?)
  2. **Constrain the pruning process** to make it more HW-compatible.

- Solution #2 is called structured pruning

- 1000s of options here too…we will just see two macro-categories:
  1. Blocked and block-balanced pruning
  2. Node (or channel)-based pruning.

Blocked / Block-Balanced Pruning

- Group weights together
  - E.g. based on the underlying HW parallelism (SIMD vector length, etc).
- Prune entire groups of weights (blocked)
  - Based on an aggregated metric, e.g. Root Mean Square value.
- …or, prune M every N weights (block-balanced)

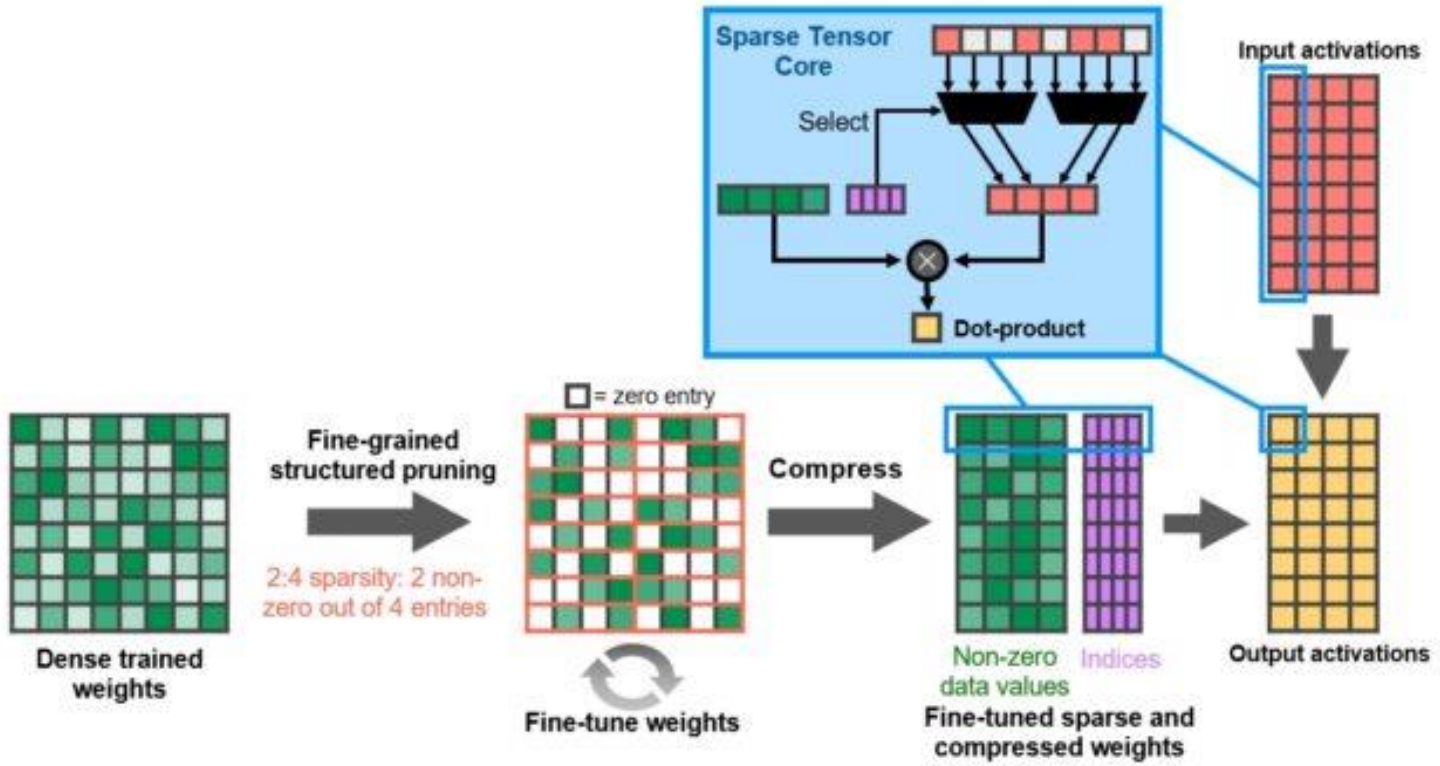


- Works well for SIMD-based HW, to maximize utilization

[4] Hoefler, T., et al.. (2021). Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. Journal of Machine Learning Research, 22(241), 1-124.

Block-Balanced Pruning

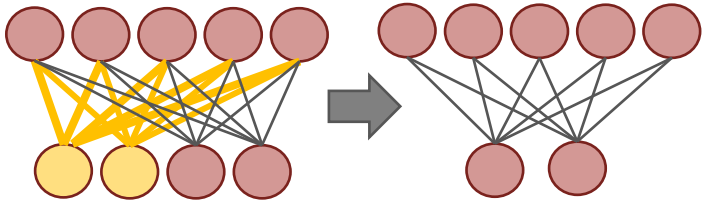
- HW Support still necessary to obtain max. speedups
- E.g. NVIDIA Tensor Cores (since Ampere):

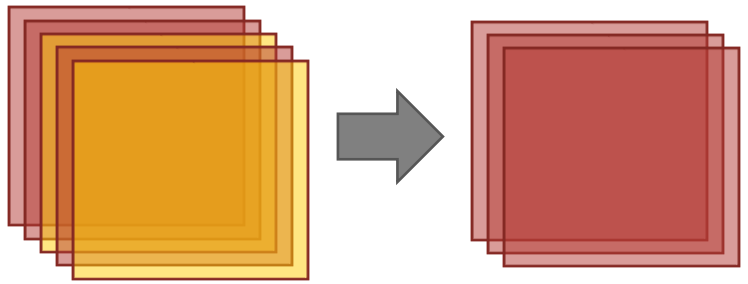


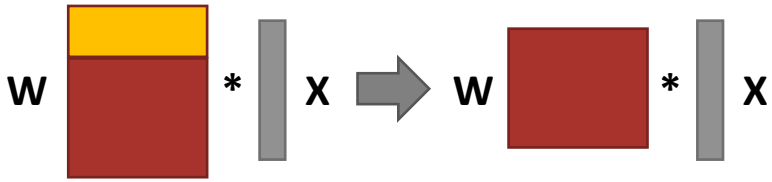
[Source] https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

Node-Based Pruning

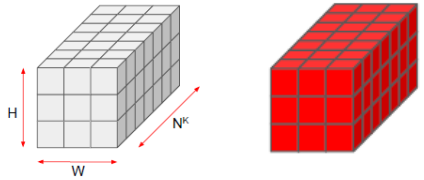
- Eliminate entire nodes instead of individual weights (connections). Each channel of a Linear layer, and each feature map of a Conv layer, is considered a “node”.
- It does not actually make the layer sparse, it just shrinks it….
  - The final model is equivalent to a DNN with a smaller number of neurons/channels
  - Entire “rows” removed from the weights matrix, which remains dense (**special case of blocked pruning**)
  - Memory and compute benefits become “obvious”.



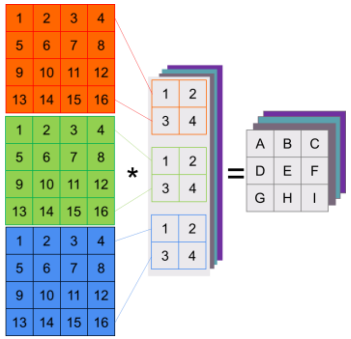


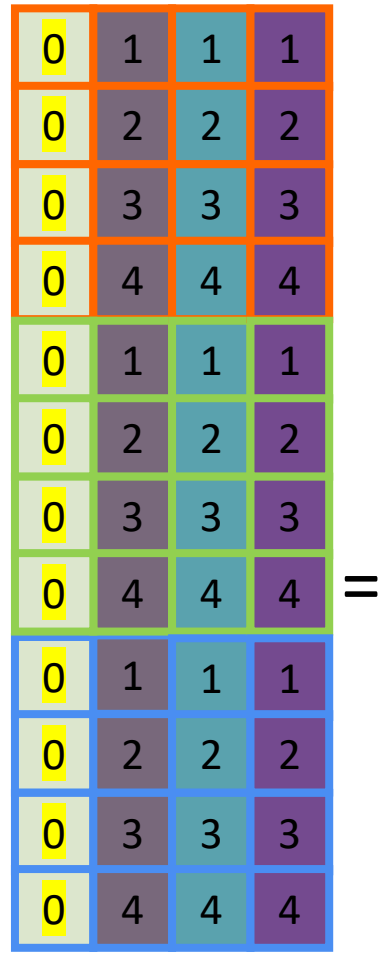


Node-Based Pruning

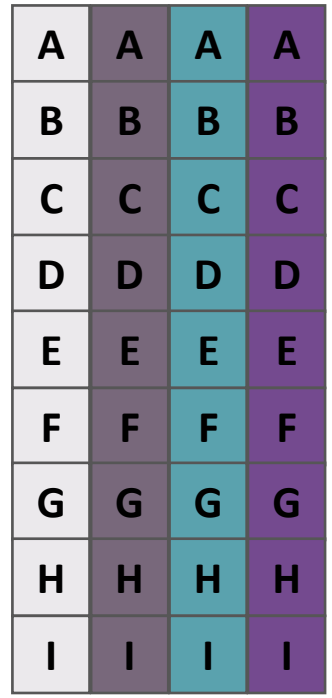


- Both footprint and latency improve
  - Smaller weight matrix
  - Smaller output feature matrix
  - Less MACs



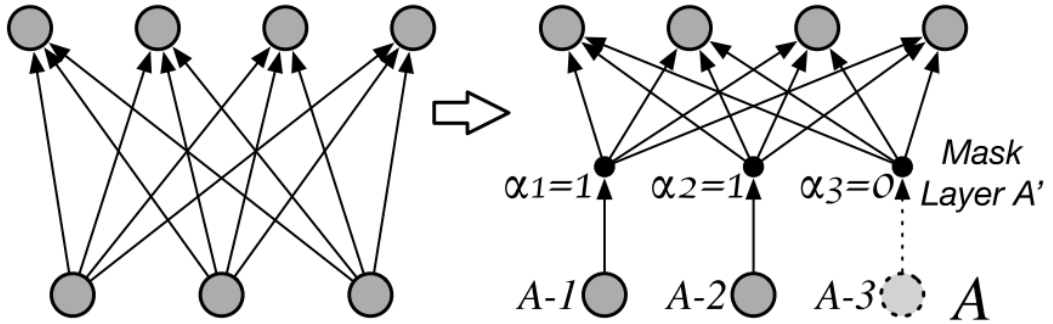


sparsity 25%



Node-Based Pruning

- One of the possible ways to select nodes to prune uses so-called "mask" layers:



[Source] Yu et al, *Scalpel: Customizing DNN pruning to the underlying hardware parallelism, ISCA 2017*

  - Multiply each channel/feature times a binary mask α_i, with values {0, 1}.

- Train the α_i together with the model weights, **forcing them to 0** for the least important nodes

Node-Based Pruning

- One of the possible ways to select nodes to prune uses so-called "mask" layers:

[Source] Yu et al, *Scalpel: Cu* *are parallelism, ISCA 2017*

- Add a **L1 regularization t** force α_i to zero for unimportant nodes

- At the end of training, remove mask layers and **eliminate neurons/channel corresponding to $\alpha_i = 0$**, making the network smaller (but still dense).

Node-Based Pruning

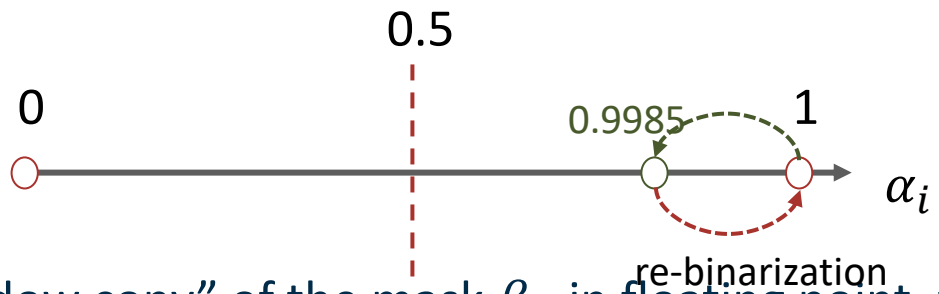
- This is very similar to a DNAS!!!
  - In fact, there's not a net distinction between node-based pruning and NAS…
  - …with node-based pruning, we are effectively reshaping the DNN architecture.

- Examples of DNAS/Structured Pruning hybrids include:
  - A. Gordon et al, "MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks", CVPR 2018
  - A. Wan et al, "FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions", CVPR 2020
  - M. Risso et al, "Lightweight Neural Architecture Search for Temporal Convolutional Networks at the Edge", IEEE TCOMP 2022

This is the NAS/Pruning tool that you'll use in the hands-on sessions.

Training a Binary Mask

- Important "caveat" with the "mask layers" techniques: training a binary value with gradient descent is not easy.
  - Small value updates do not have an effect.
  - Example: $\alpha = 1$, $\alpha' = 1.5$, LR = 0.001 → $\alpha = \alpha$ – LR * α' = 1 – 0.0015 = 0.9985 **= 1**



- Solution: keep a "shadow copy" of the mask β_i, in floating point, where $\alpha_i = H(\beta_i)$ and H is a binarization function
  - From simple Heaviside: $\alpha_i = \begin{cases} 1 & if \ \beta_i > 0.5 \\ 0 & otherwise \end{cases}$ to trainable threshold, histeresis, etc
  - Update β_i in every training iteration, and re-generate the binarized α_i on-the-fly

Training a Binary Mask

- New problem: the gradient of H(...) is 0 almost everywhere.
  - Gradients won't back-propagate from α_i to β_i

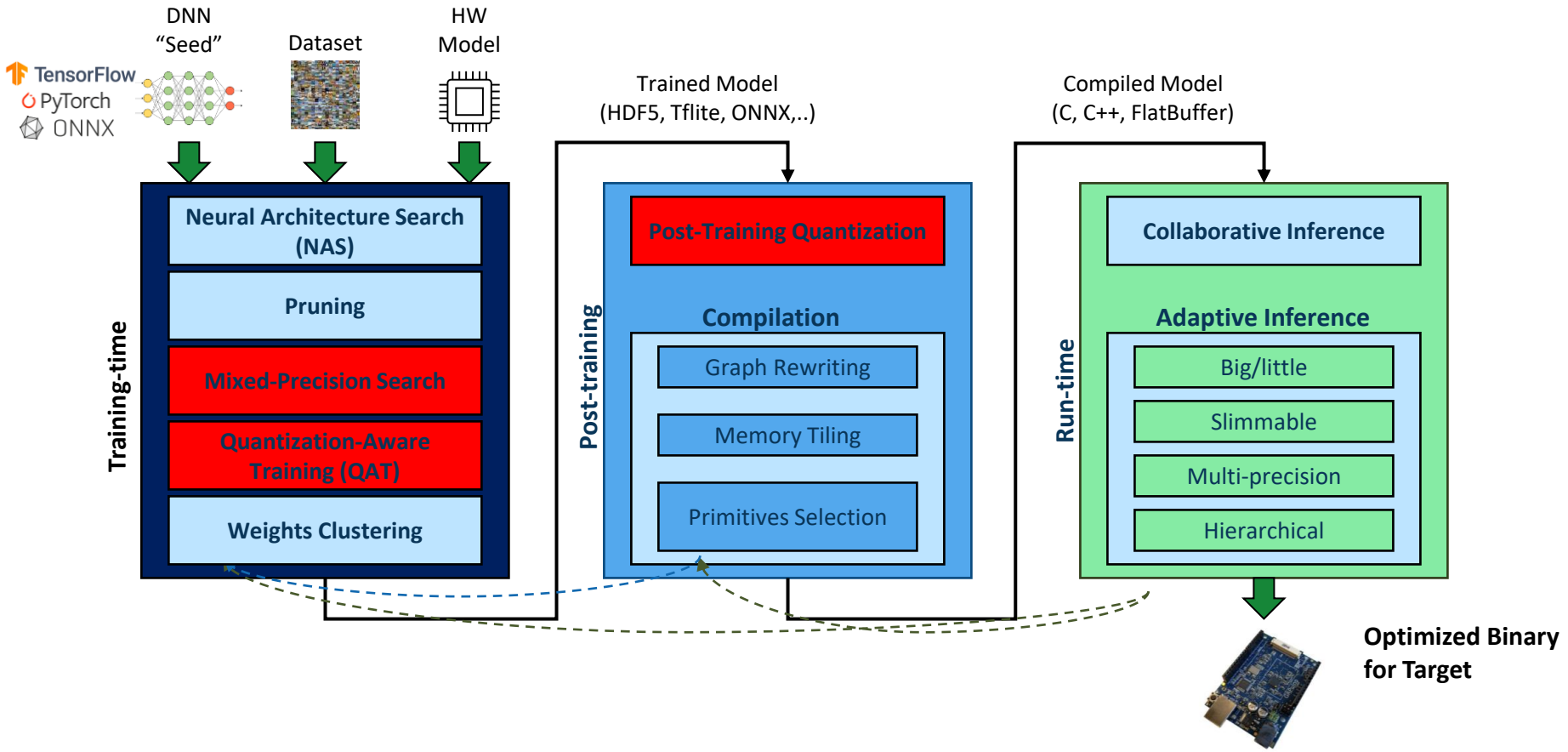
- This is addressed by **discrete function gradients estimators**. The simplest being the so-called Straight-Through Estimator (STE)
  - Approximate $y = H(x)$ with $y = x$ when back-propagating gradients

- Also used in Quantization-aware Training (QAT), see later…

Politecnico
di Torino

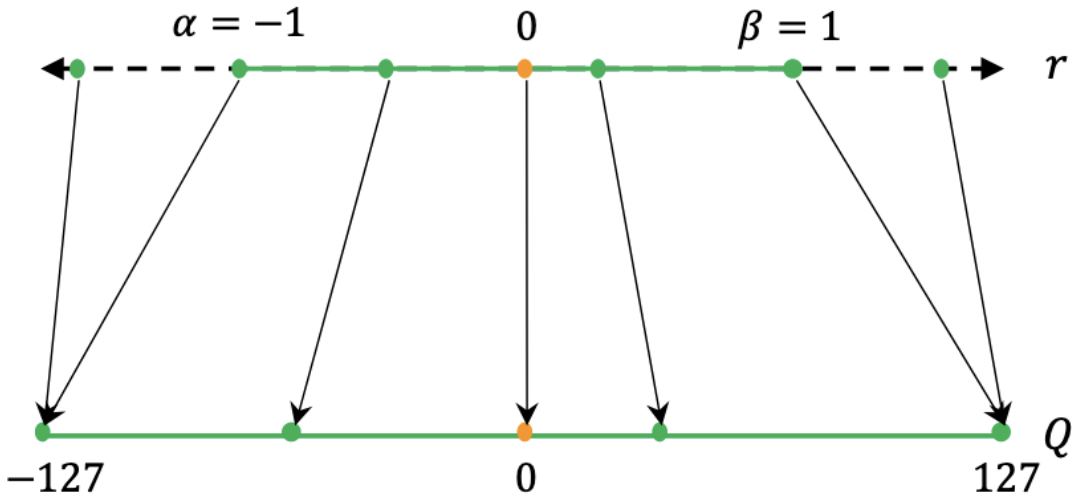
Quantization & Mixed Search

Recap: Our Flow



Quantization

- Standard DL training and inference are performed with 32-bit float weights and activations.
- Both tasks, but especially inference, can be performed at reduced precision often with little accuracy loss
  - Using smaller float formats (e.g. 16-bit mini-float, bfloat, etc)
  - Using low-precision integers (with appropriate scaling)
  - Using custom data format



- Excellent Reference:
  - M. Nagel et al, “A White Paper on Neural Network Quantization, arXiv 2021

Benefits of Quantization

1. Some edge platforms do not have Floating Point Units (FPUs)

2. Store and transfer less bits
   - Remember that data transfers are fundamental contributors to the energy/latency of DNNs

3. Using low precision data can also improve the efficiency of computations (HW-dependent):
   - GEMM on 8bit integer can be faster and more energy-efficient than on floats (ALU vs FPUs)

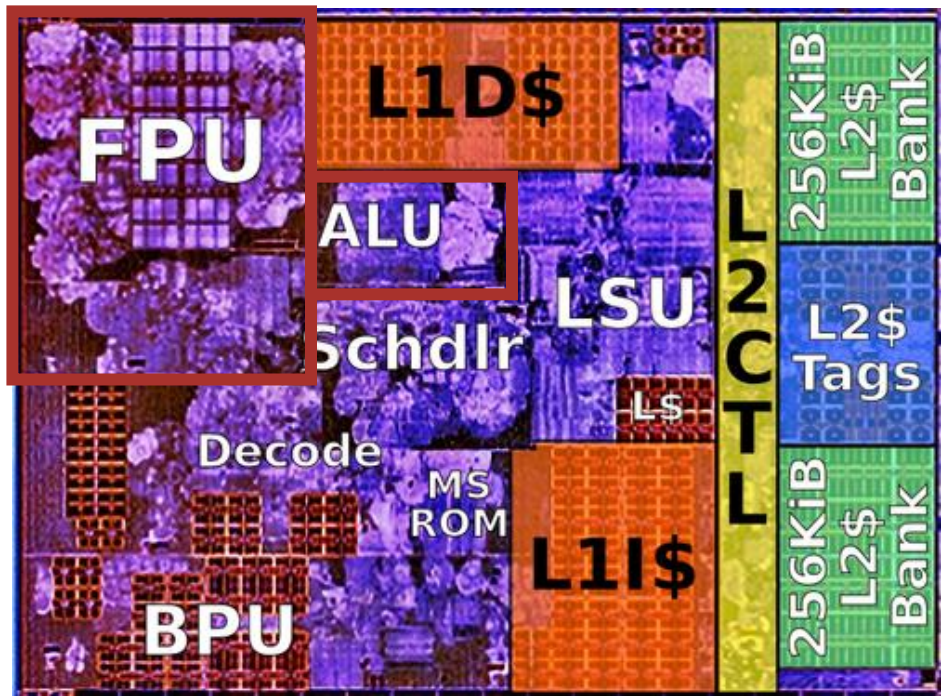
Quantization @ Edge

- To deploy DL models on IoT edge devices, a popular quantized data format is **8-bit integer** (fixed point).

- **Why integer?**
  - ALUs are smaller and more efficient than FPUs
  - Some extreme-edge devices (microcontrollers) don't even have a FPU

  --> **Quantization can be mandatory!**

Example of a non-edge CPU: AMD Zen



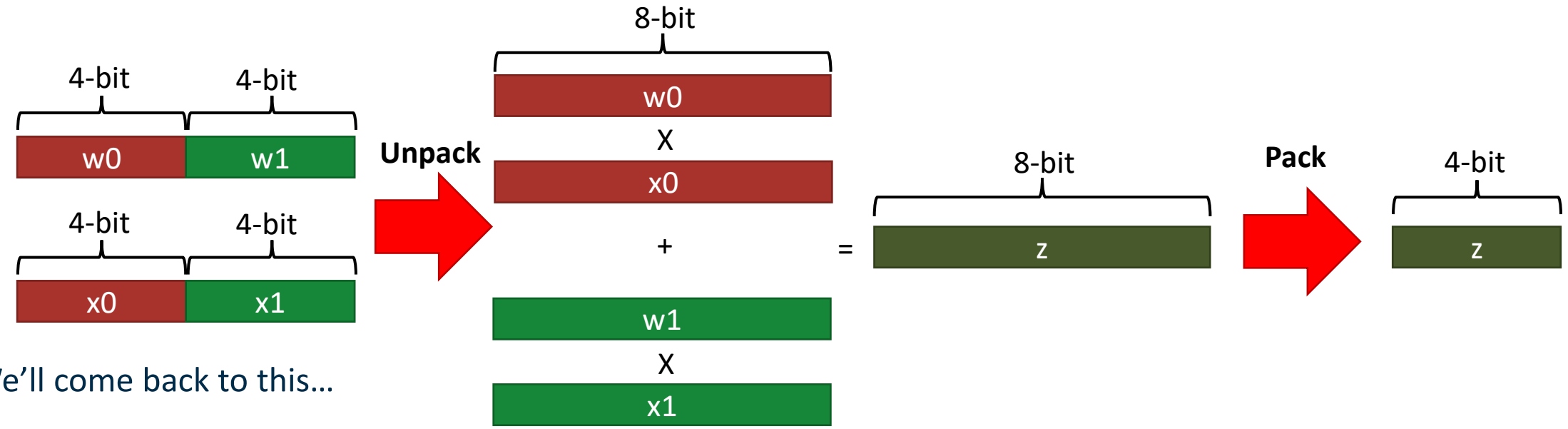
Quantization @ Edge

- **Why 8-bit?**
  - Most CPUs have 1 byte as the minimum unit of processing. Instruction only work on single or multiple bytes, not sub-bytes.
  - RAM memories are byte-addressed

- Also, 8-bit quantization is (empirically) a *"sweet spot"* in terms of accuracy for many tasks.

Quantization @ Edge

- Using less than 8-bits (and sometimes even less than 16) requires:
  - Custom hardware support or…
  - …complex packing/unpacking of data



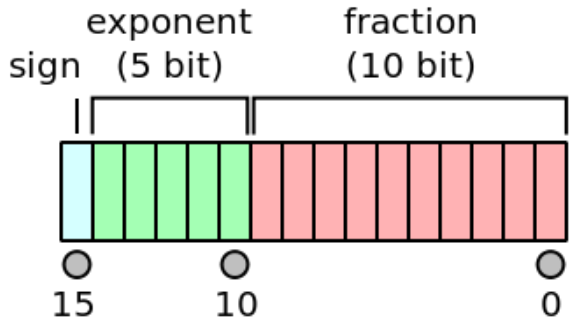
- We'll come back to this…

Quantization @ Edge

- Int8 quantization benefits:
  - Reduce memory by 4x
  - Reduce data transfer energy by 4x
  - Improve computation efficiency (with SIMD MACs)

Quantization - Cloud

- Quantization in the cloud is often done using a **16-bit minifloat** format (half precision, bfloat16, others):



- The goal in this case is mainly to increase execution speed while maintaining high accuracy. Energy is relevant, but somewhat secondary.

- Minifloats are more complex to process than integers, but can represent a higher dynamic range for the same bit-width, and therefore typically result in higher accuracy.

Quantization Formats - Summary

- **16-bit floats** basically come for free:
  - Can be used for both training and inference with no drop in accuracy in most cases
  - But they're still floats
  - Increasingly adopted for **training/inference** in the cloud. Less common at the edge

- **Int-8** is the current standard at the edge:
  - Drop in accuracy often minimal (especially with Quantization-Aware Training).
  - Most ISAs have byte-level instructions

- **Sub-byte** integer quantization is attracting attention:
  - 1-bit quantization (BNN) is already popular for simple tasks
  - 2/4-bit is increasingly used, despite its packing/unpacking overheads.

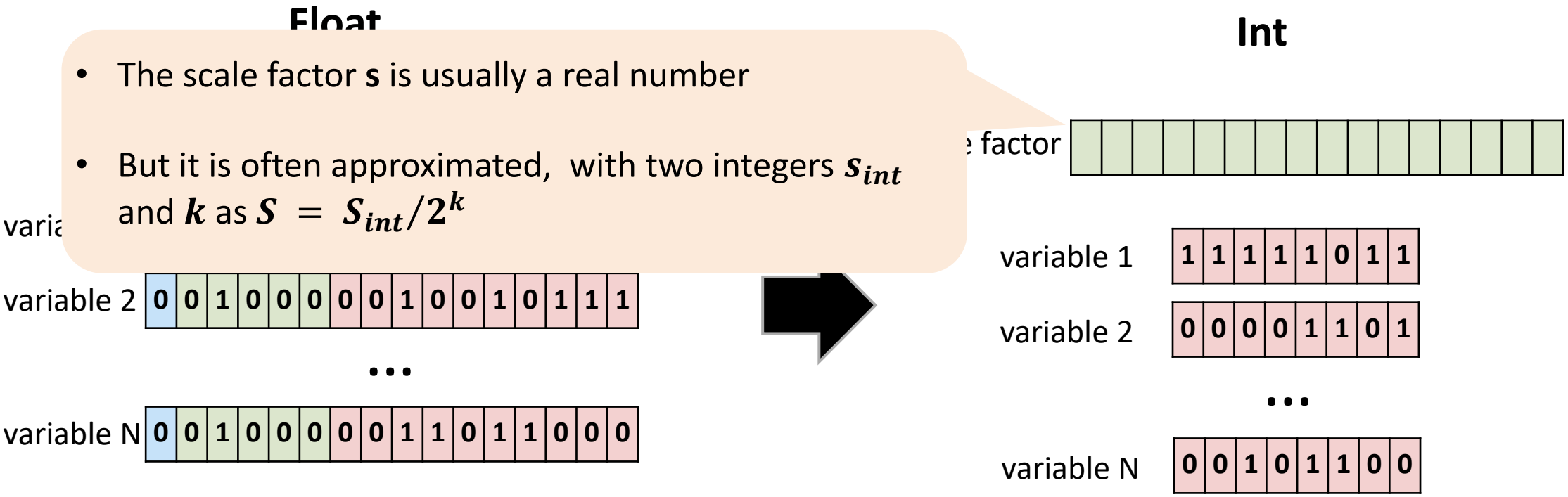
Benefits of Quantization - Summary

- Storage and memory:
  - Reduce ROM/disk occupation
  - Reduce RAM occupation
  - Increase data-transfers bandwidth (e.g., with SIMD load/store)

- Compute:
  - Speed-up thanks to ALU vs FPU
  - Further speed-up thanks to SIMD operations
  - Enable deployment on FPU-less Microcontrollers

Integer Quantization

- Many ways to represent real numbers using an integer.
  - No universal standard as for floats
- The most common approach is to use integer multiples of a common scale factor



- The scale factor **s** is usually a real number
- But it is often approximated, with two integers s_{int} and k as $S = S_{int}/2^k$

Integer Quantization

- Example (symmetric quantization):

$$s = 0.05$$

$$x_{1q} = -5 \quad \boxed{1\,1\,1\,1\,1\,0\,1\,1}$$

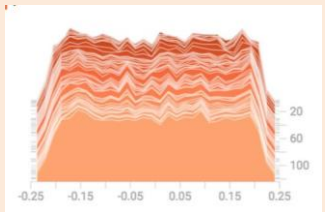
$$x_{2q} = +13 \quad \boxed{0\,0\,0\,0\,1\,1\,0\,1}$$

$$x_{1real} \approx -5 * 0.05 = -0.25$$

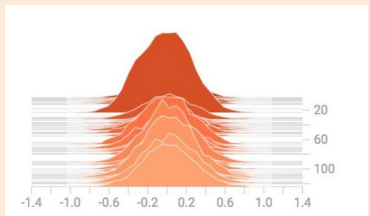
$$x_{2real} \approx +13 * 0.05 = 0.65$$

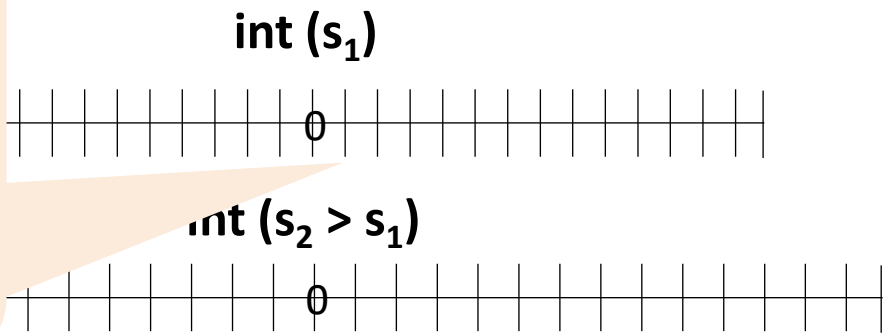
- Differently from floats, data density is uniform in the entire range
- Using a different s for different tensors/slices allows covering different ranges

- **Different layers (or channels)** can have widely different ranges of values for weights and activations:



VS



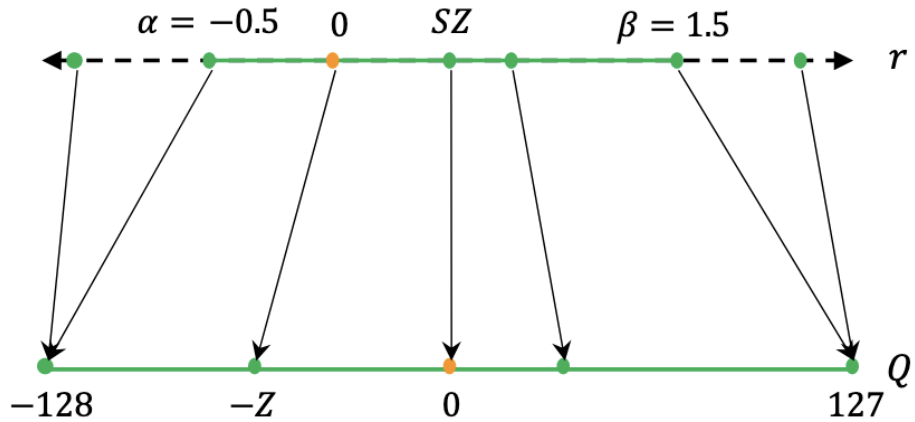


Integer Quantization

- Generalization of the previous scheme:

$$\mathbf{x}_q = \mathbf{clamp}\left(\left\lfloor \frac{x_{real}}{s} \right\rfloor + z, a, b\right) \quad \Longleftrightarrow \quad \mathbf{x}_{real} \approx s(x_q - z)$$

  - where: $\mathrm{clamp}(x, a, b) = \min(\max(x, a), b)$
  - E.g. for uint8 a = 0, b = 255
  - Also called **asymmetric** integer quantization.



- Typically, symmetric (z=0) is used for weights, asymmetric for activations
  - **Activations**: often all-positive outputs (e.g. ReLU)
  - **Weights and biases**: usually centered around 0. Moreover, z=0 avoids an additional multiplication loop

Integer Quantization

- Why is z always 0 for weights?
  - Weights distributions tend to be more symmetric compared to activations.
  - But most importantly, z can be heavily optimized for activations, not for weights…

- Case 1: both asymmetric

Requires an additional MUL for each activation
Since xi is variable!

$$w_{i,real} = (w_{i,int} - z_w) * s_w, \qquad x_{i,real} = (x_{i,int} - z_x) * s_x$$

$$out_{real} = f\left(\sum^{N}(w_{i,real} * x_{i,real})\right) = f\left(\sum^{N}((w_{i,int} - z_w) * s_w * (x_{i,int} - z_x) * s_x)\right) =$$

$$= f\left(s_w s_x(\sum^{N}(w_{i,int} * x_{i,int} - w_{i,int} * z_x - x_{i,int} * z_w) + N * z_x * z_w)\right)$$

Integer Quantization

- Case 2: symmetric weights

- Weights are constant after training
  - This entire summation can be pre-computed.
- No additional MUL required, just a **single** subtraction
  - Already there if you have biases or BatchNorm

$$w_{i,real} = w_{i,int} * s_w, \qquad x_{i,real} = (x_{i,int} - z_x) * s_x$$

$$out_{real} = f\left(\sum^{N}(w_{i,real} * x_{i,real})\right) = f\left(\sum^{N}(w_{i,int} * s_w * (x_{i,int} - z_x) * s_x)\right) =$$

$$= f\left(s_w s_x \sum^{N}(w_{i,int} * x_{i,int} - w_{i,int} * z_x)\right)$$

$$= f\left(s_w s_x \sum^{N}(w_{i,int} * x_{i,int}) - s_w s_x \sum^{n}(w_{i,int} * z_x)\right)$$

Integer Quantization

- How to determine s and z for a given tensor (**a**)?

- The basic way is to make sure the entire range (a_{min}, a_{max}) is correctly mapped (min-max quantization)

Activations	$s = \frac{a_{max} - a_{min}}{2^{Nbit} - 1}$	$z = \frac{a_{max} - a_{min}}{2}$
Weights	$s = \frac{2 * \max(\lvert a_{max} \rvert, \lvert a_{min} \rvert)}{2^{Nbit} - 1}$	$z = 0$

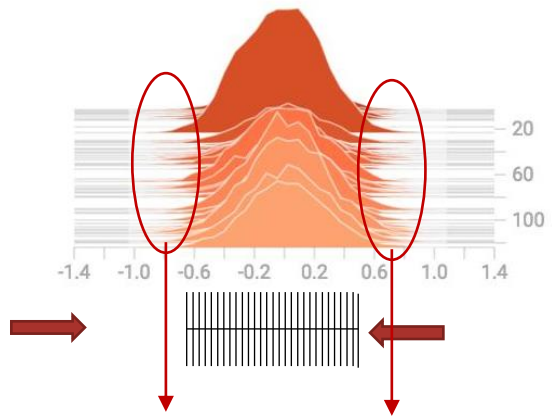
Integer Quantization

- Some approaches use a smaller s, worsening the approximation of some outlier weights, in exchange for more precision around z:
  - Can sometimes yield higher accuracy
  - Many algorithms proposed, e.g.:
    - *Cai et al. Deep learning with low precision by half-wave gaussian quantization, CVPR 2017.*
    - *… and many others*

- In this case, the quantization procedure has two steps:

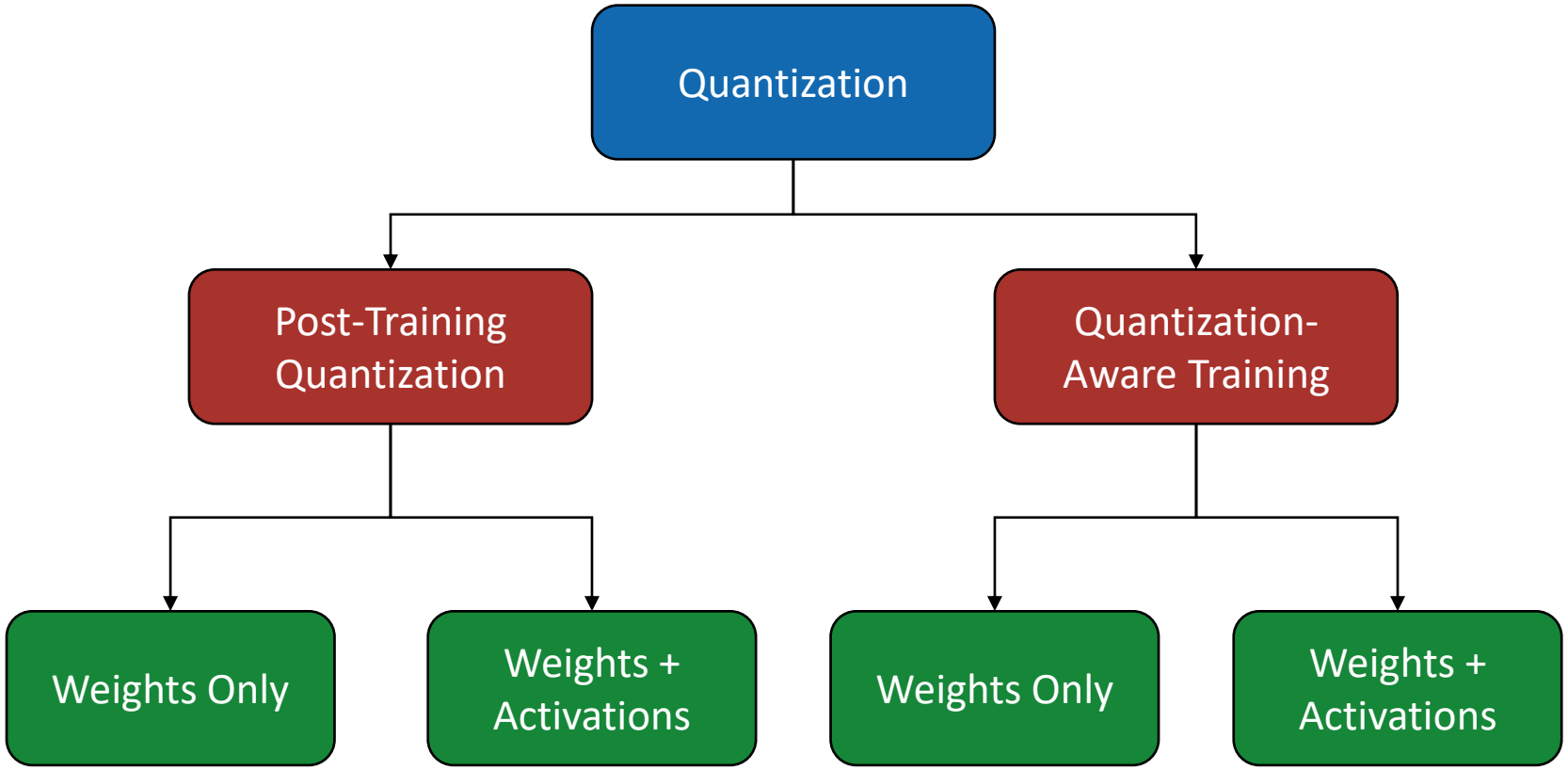
① $tmp = round(\frac{a}{s} + z)$

② $a_q = clamp(tmp, -127, 127) = \min(\max(-127, tmp), 127)$



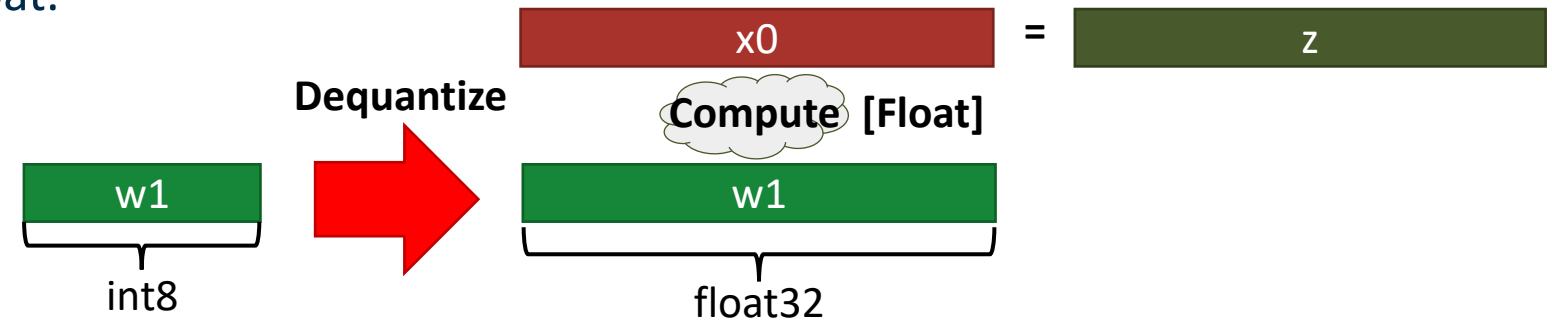
Not in the range

Quantization: When and What?

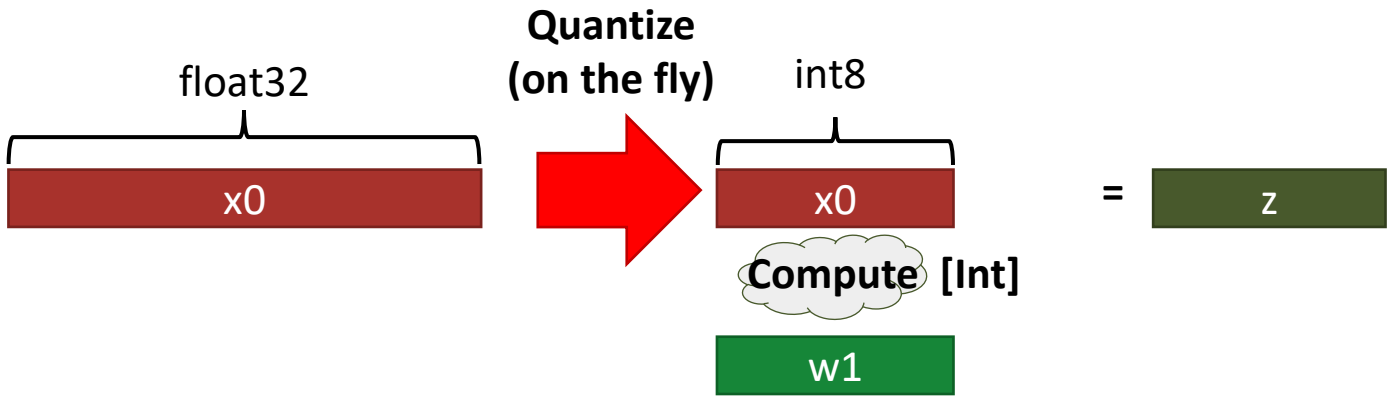


Weights Only Quantization

- Basic version: during inference, weights are dequantized and operations are performed in high-precision float.

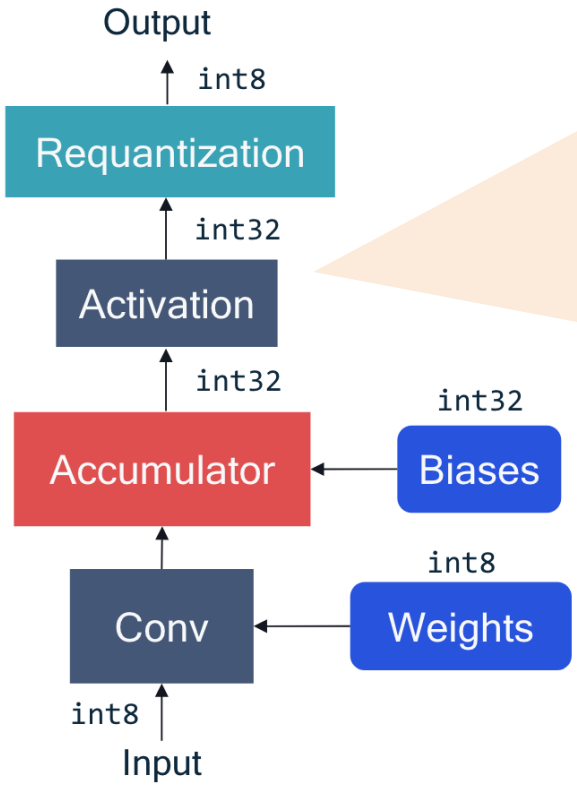


- Advanced version ("dynamic range quantization"): Weights are kept in low-precision format, and activations are quantized on-the-fly so that operations can be performed at low-precision



Weights + Activations Quantization

- Use low-precision for both weights and activations, and for both storage and computation.
- Accumulation and bias addition usually at higher precision, then output requantized



- Desired operation: $\boldsymbol{Y_{real} = W_{real} \cdot X_{real} + B_{real}}$
- Approximated as: $\boldsymbol{Y_{real} \approx s_w W_q \cdot s_x (X_q - z_x) + s_b B_q}$
- Impose $\boldsymbol{s_b = s_w s_x}$ we obtain: $\boldsymbol{Y_{real} \approx s_x s_w (W_q \cdot X_q + B_q - zxWq)}$

  $= ACC$ (all integers, accumulated on int32)
- We want: $\boldsymbol{Y_q = \mathbf{clamp}\left(\left\lfloor \frac{Y_{real}}{s_y} \right\rceil + z_y\right)}$
- Requantization: $\boldsymbol{Y_q = \mathbf{clamp}\left(\left\lfloor \frac{s_x s_w}{s_y} ACC \right\rceil + z_y\right)}$

[Source] M. Nagel et al. (Qualcomm), A White Paper on Neural Network Quantization, arXiv 2021

Weights + Activations Quantization

- **WARNING:** w+x quantization does not necessarily mean “full integer” model. What is actually turned to integer is tool-dependent
- Some tools (e.g. TFLite by default) leave DNN input/outputs in float for direct compatibility with float scripts.
- Others use floating point for non-MatMul layers
  - Some activation functions
  - BatchNorm.

Batch Normalization Folding

- Batch Normalization can be “fused” with the “nearest” FC or Conv layer
- Example for **FC/Conv+BN+ReLU:**
  - If, instead, ReLU is before BN, the latter can be fused with the “next” FC/Conv, rather than the “previous”

	Original	**Fused/Folded**
BN	$y_i = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} x_i - \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} \mu + \beta$	$y_i = W'.z_{ci} + b'$ where $W' = W \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$
FC	$x_i = W.z_{ci} + b$	
FC+BN	$y_i = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} (W.z_{ci} + b) - \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} \mu + \beta$	$b' = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} (b - \mu) + \beta$

Batch Normalization Folding

- Useful in general to **speed—up inference**:
  - Reduce tot. number of parameters (a little)
  - Reduce tot. number of operations (a little)
  - Avoid 1 additional layer (often processed sequentially → **DATA REUSE!!!**)
  - Layer fusions are quite common in DNN compilers

- Also helps in making a quantized model **full-integer**:
  - If hardware does not support quantized BN.
  - Better combined with Quantization-aware Training (QAT)

Post-Training Quantization (PTQ)

- The model is trained normally, in float
- After training, W and X (or just W) are converted to lower precision (e.g. int8)

- Pros:
  - Simpler.
  - Can be applied to pre-trained models, with **no re-training** and (in some cases) even without input data available.

- Cons:
  - It can cause **a significant accuracy drop** on complex tasks.

PTQ: How to determine s and z?

- The basic way is to make sure the entire range of values (a_{min}, a_{max}) of the generic tensor A (W or X) is correctly mapped: min-max quantization

Activations	$s = \frac{a_{max} - a_{min}}{2^{Nbit}}$	$z = \frac{a_{max} - a_{min}}{2}$
Weights	$s = \frac{2 * \max(\lvert a_{max} \rvert, \lvert a_{min} \rvert)}{2^{Nbit}}$	$z = 0$

- Problem:
  - For weights tensors, w_{min} and w_{max} can be extracted from a trained float model
  - For activation tensors, x_{min} and x_{max} are unknown, as they depend on the inputs!
- Solution: use a small dataset to compute x_{min} and x_{max}
  - A few 1000s of samples are normally sufficient
  - Outliers can be squashed in the range with *clamp()* during re-quantization

PTQ: How to determine s and z?

- More advanced approaches prefer using a smaller “s” , worsening the approximation of some outliers, in exchange for more precision around z:
  - Can sometimes yield higher accuracy

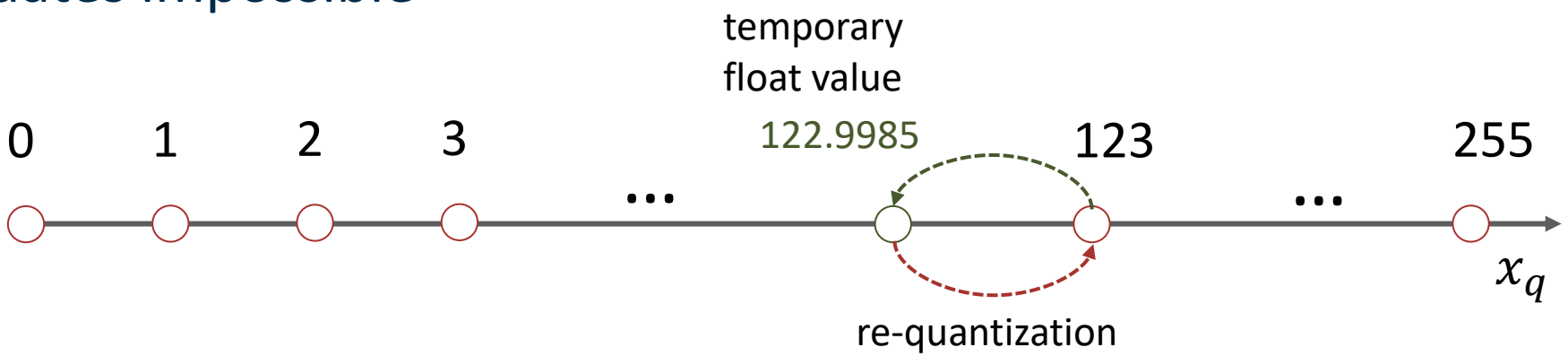
  - Example: minimize MSE between float and quantized tensor:
    - *Banner, R. et al Post training 4-bit quantization of convolutional networks for rapid-deployment, NeurIPS 2019.*

$$\text{argmin}_{a_{min},a_{max}} \left|\left|A_{real} - A_q\right|\right|_2$$

Quantization-Aware Training (QAT)

- Reduce accuracy drops by simulating quantization during training.
  - Optimizer can take into account the limited set of values that weights/activations can assume.
  - **Activation ranges** are available at all times (and can be trained as well!!)
  - Almost always a good idea (when re-training is possible).

- But…using true low-precision integers during back-prop makes small weight updates impossible



Quantization-Aware Training (QAT)

- Fake quantization:
  - Keep data in float. Simulate the limited set of values that can be assumed using round & clamp

Real Quant

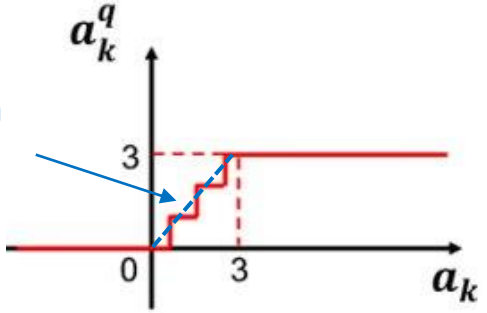
$$A_q = \mathbf{clamp}\left(\left\lfloor \frac{A_{real}}{s} \right\rceil + z\right)$$

Fake Quant

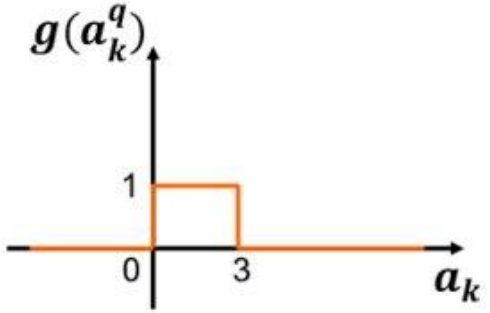
$$A_{fq} = \mathbf{s} * \mathbf{clamp}\left(\left\lfloor \frac{A_{real}}{s} \right\rceil + z\right) - z$$

- Back-propagate gradients of A_{fq} to A_{real}
  - Simulate quantization only in the forward pass, and replace it with a differentiable function (e.g. identity, with the STE trick) in the backward pass.

Approximated by the blue line in backwards passes



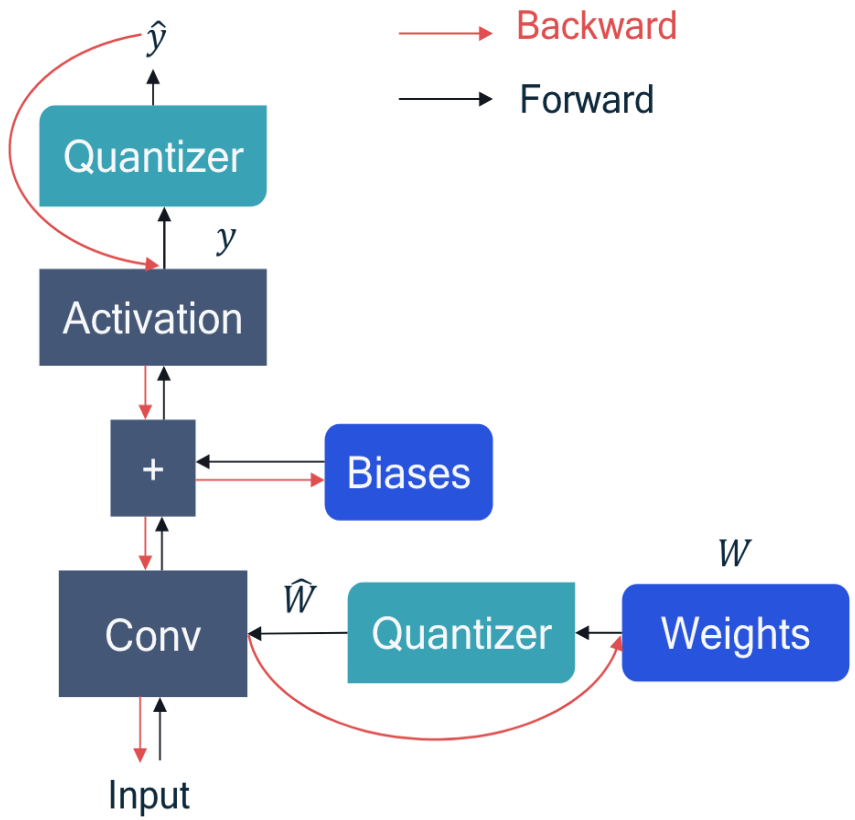
Quantization function (2bit)



Quantization gradient

Quantization-Aware Training (QAT): Summary

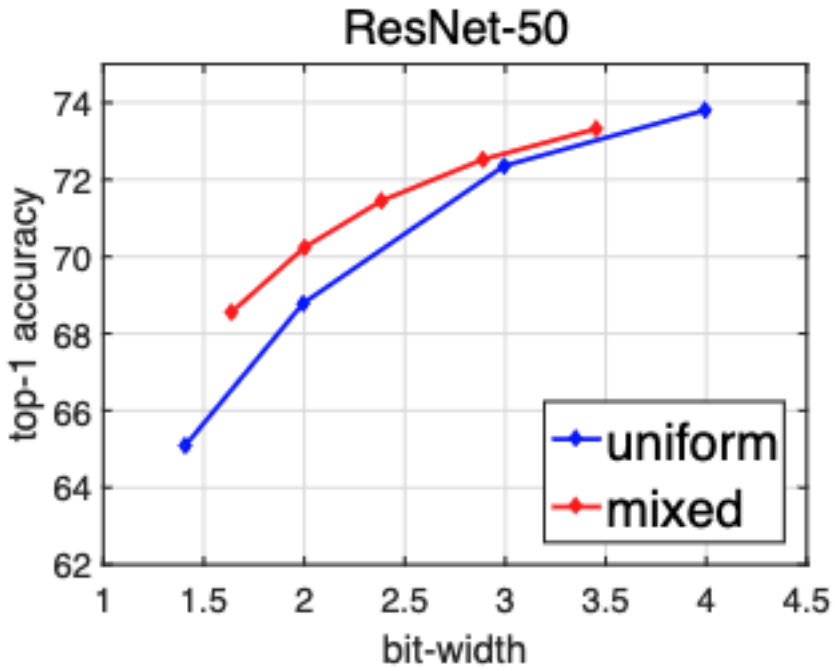
- Add “fake quantization” ops to simulate quantization in the forward pass
  - Actual computation still in float
- In the backward pass, use a STE to propagate gradients
- For **min-max quantization**, fake quant nodes also record w_{min} w_{max} x_{min} x_{max} to compute s and z
- Alternative: **learn the quantization parameters** during QAT:
  - Choi et al, PACT: Parameterized Clipping Activation for Quantized Neural Networks, CVPR 2018.
  - Esser et al, Learned Step Size Quantization, ICLR 2020
  - etc.



Fixed- vs Mixed-Precision

- Fixed-precision: s and z change per-tensor (or per channel), but the bit-width N is fixed for the entire network

- Mixed-precision: uses a different N layer-wise or channel-wise.
  - Example: **2, 4, 8-bit**
  - N_w (weights) can differ from N_x (activations)
  - Possibly higher compression for the same accuracy



[9] Z. Cai et al, Rethinking Differentiable Search for Mixed-Precision Neural Networks, CVPR 2020

Mixed-Precision Quantization

- **Bit-width assignment** problem:
  - How to assign N to *x* and *w* of different layers?
  - Huge search space: $((N_{prec})^2)^{N_{layers}}$

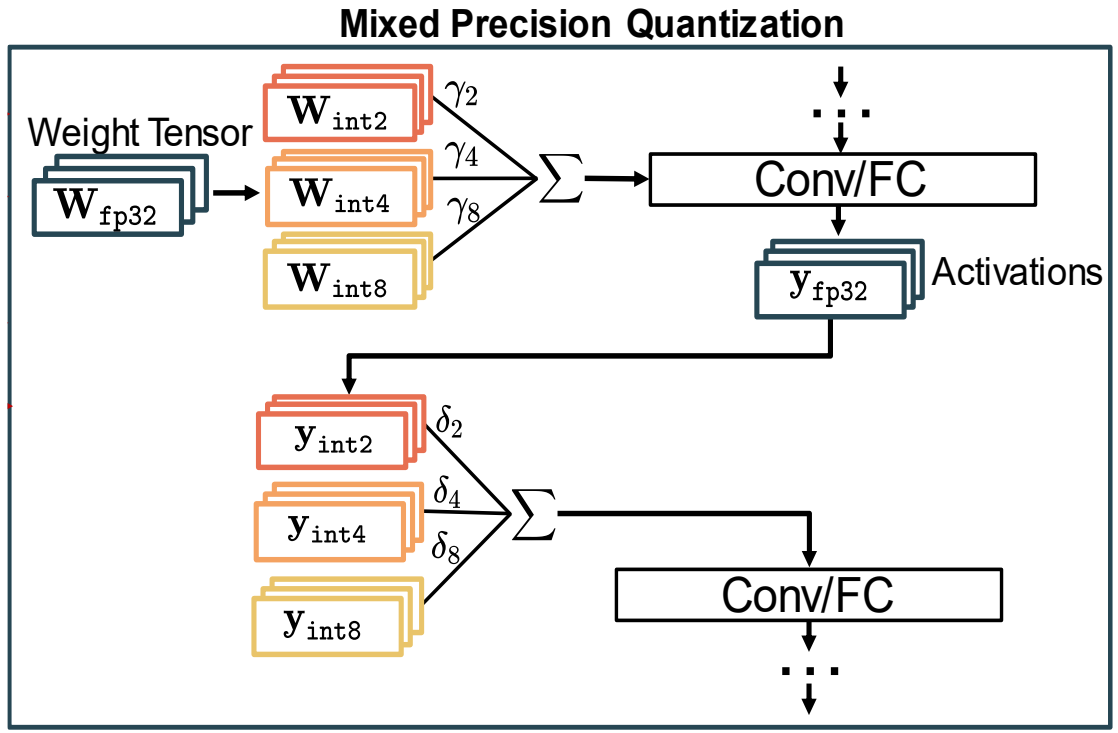
- Classical solutions:
  - Black-box meta-heuristics (RL, Genetic, etc.), Greedy, Simulated Annealing
  - A. T. Elthakeb et al, *"ReLeQ: A Reinforcement Learning Approach for Deep Quantization of Neural Networks"*, NeurIPS 2018
  - K. Wang et al, *"HAQ: Hardware-Aware Automated Quantization with Mixed Precision"*, CVPR 2019

- Can be approached with a method **similar to DNAS!**

Differentiable Mixed Precision Search

- **EdMIPS:**
  - Quantize each W/X tensor at multiple precisions.
  - Combine different quantizations with trainable weights, e.g.:

$$\widehat{\mathbf{W}} = \sum_{p=0}^{P-1} \mathbf{W}\mathbf{q}_p \cdot \gamma_p$$



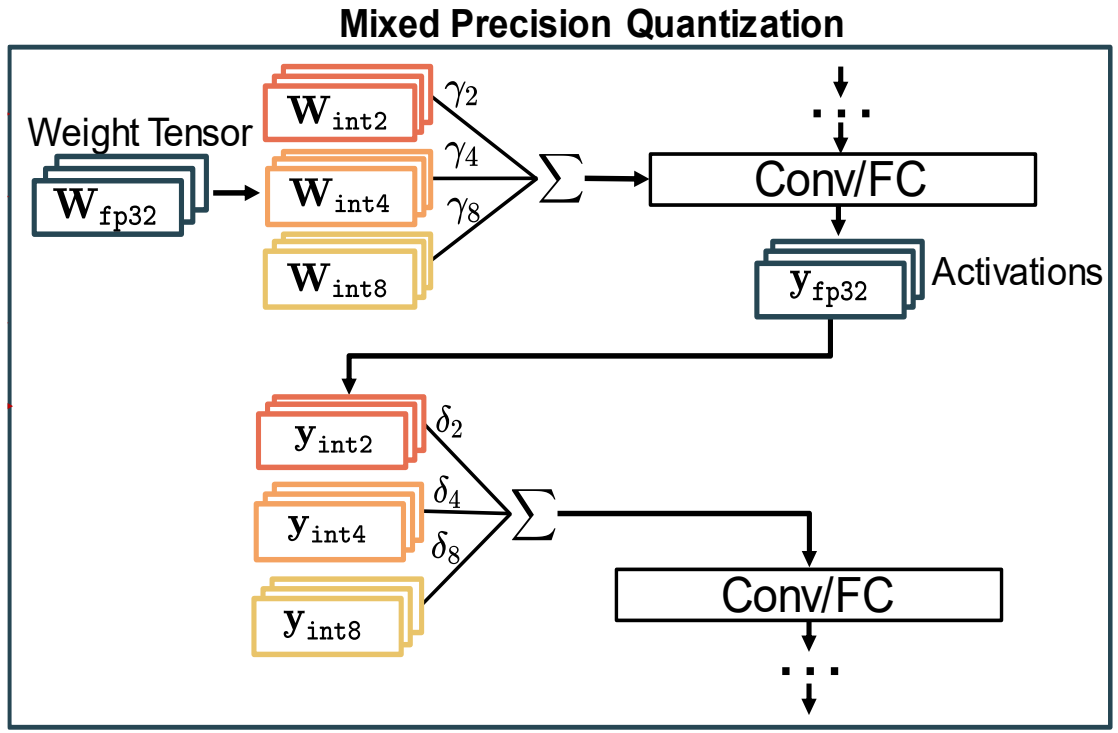
[9] Z. Cai et al. "Rethinking differentiable search for mixed-precision neural networks." Proc. IEEE/CVF CVPR, 2020.

Differentiable Mixed Precision Search

- **EdMIPS:**
  - Train with cost regularizer that depends on bit-width
  - Similar to constrained DNAS
  - E.g. size regularizer:

Cost = $C_{in} \bullet C_{out} \bullet K^2 \bullet$ **bit**$_{weights}$

$$bit_{weights} = \gamma_2 * 2 + \gamma_4 * 4 + \gamma_8 * 8$$



[9] Z. Cai et al. "Rethinking differentiable search for mixed-precision neural networks." Proc. IEEE/CVF CVPR, 2020.

Politecnico
di Torino

Advanced Quantization

Binary Neural Networks

Binary Neural Networks

- Extreme form of quantization, where data are constrained to have a binary value.
  - Initially used only for weights, (BinaryConnect)
  - Then extended also to activations (XNOR-Net)

- **Obvious advantage**: 32x reduction in model size

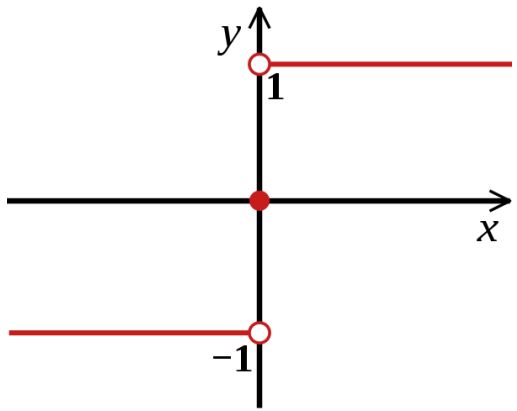
- **Less obvious advantage**: MAC operations can be replaced by bit manipulations.
  - Significant speed-up and energy efficiency improvement, also on general purpose HW.

- **Drawback:** accuracy can drop significantly on complex tasks

Binary Neural Networks

- W and X are constrained to have one of two values: {-1, 1}
  - In practice, -1 is represented by logic-0 for HW simplicity.
  - More advanced variants use {-alpha, alpha}, where alpha is set per-tensor or per-channel.

- Re-quantization is done using the sign() function



Binary Neural Networks

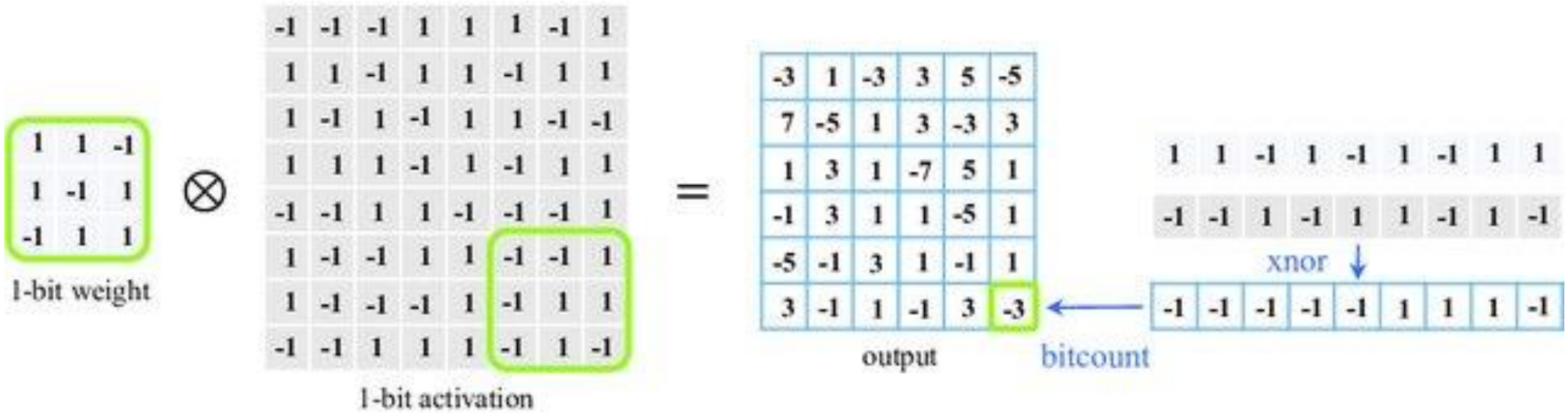
- Multiplication of binary weights represented in this format becomes a Binary XNOR.

Value of A	Value of B	Value of A * B	Logic A	Logic B	Logic A XNOR Logic B
-1	-1	1	0	0	1
-1	1	-1	0	1	0
1	-1	-1	1	0	0
1	1	1	1	1	1

Binary Neural Networks

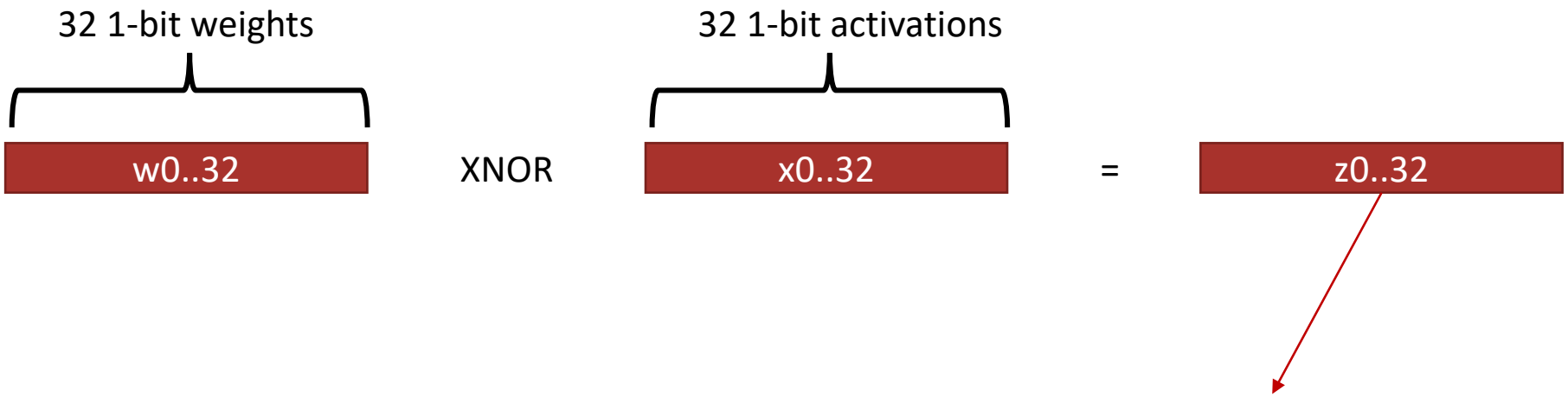
- Accumulation reduces to counting the number of 1s (***bitcount*** or ***popcount***):
- Assume we want to accumulate the results of N bit-wise XNORs (multiplications)
- Let us call:
  - p = count of 1s
  - n = count of 0s (-1s),
- $Accum = p - n = p - (N - p) = 2 * p - N$
  - Since 2 and N are constants, equivalent to *p*

Binary Neural Networks



Binary Neural Networks

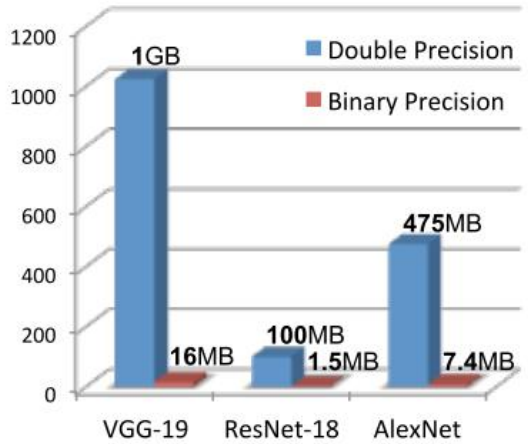
- BNNs are also easy to implement on general purpose HW (differently from other sub-byte quantization formats).
- All general purpose CPUs have instructions for bit-wise logic.



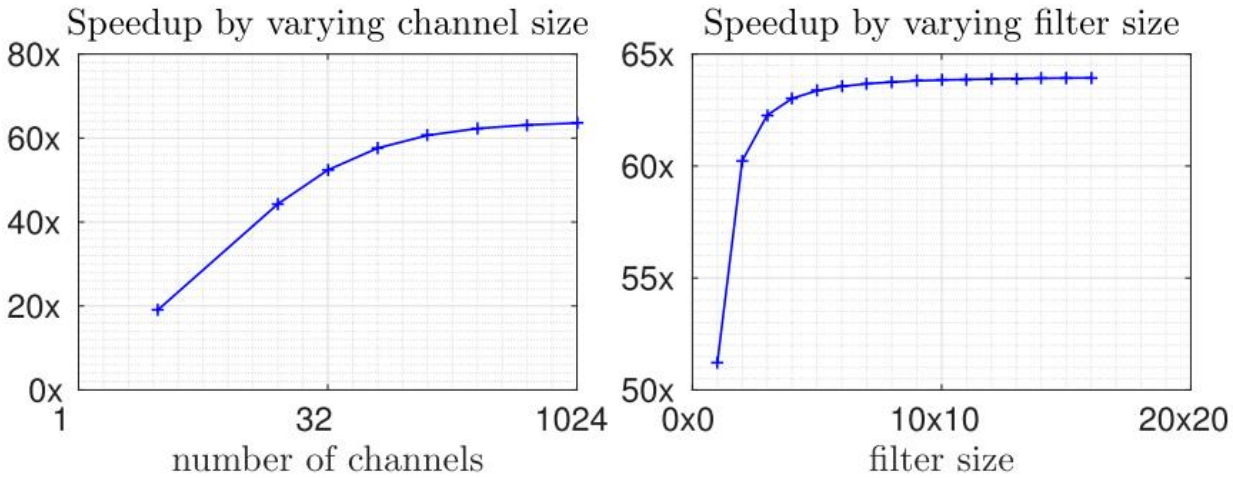
32 multiplications can be performed in a single instruction (without considering multi-core)

Binary Neural Networks

- BNN results:



Memory Occupation

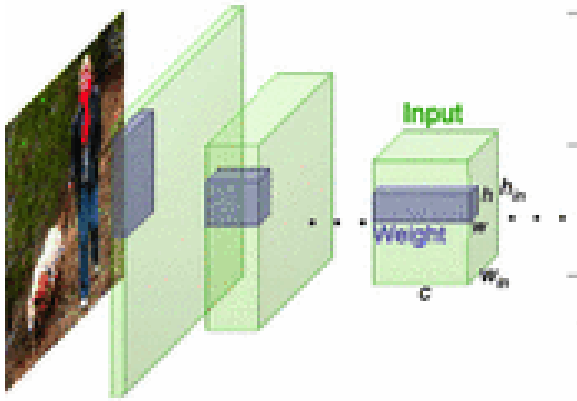


Speed-up

[10] Rastegari, M., et al.. Xnor-net: Imagenet classification using binary convolutional neural networks. In European conference on computer vision. Cham: Springer International Publishing, 2016.

Binary Neural Networks

- BNN results (cont'd):



Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution (Real-Value Inputs, Real-Value Weights)	+ , − , ×	1x	1x	%56.7
Binary Weight (Real-Value Inputs, Binary Weights)	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (XNOR-Net) (Binary Inputs, Binary Weights)	XNOR , bitcount	~32x	~58x	%44.2

[10] Rastegari, M., et al.. Xnor-net: Imagenet classification using binary convolutional neural networks. In European conference on computer vision. Cham: Springer International Publishing, 2016.

Politecnico
di Torino

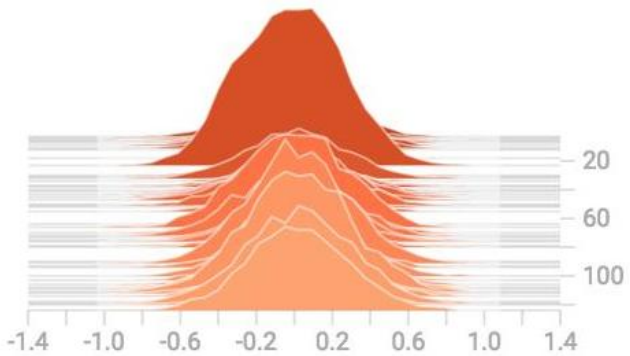
Advanced Quantization

Non-Uniform Quantization

Non-Uniform Quantization

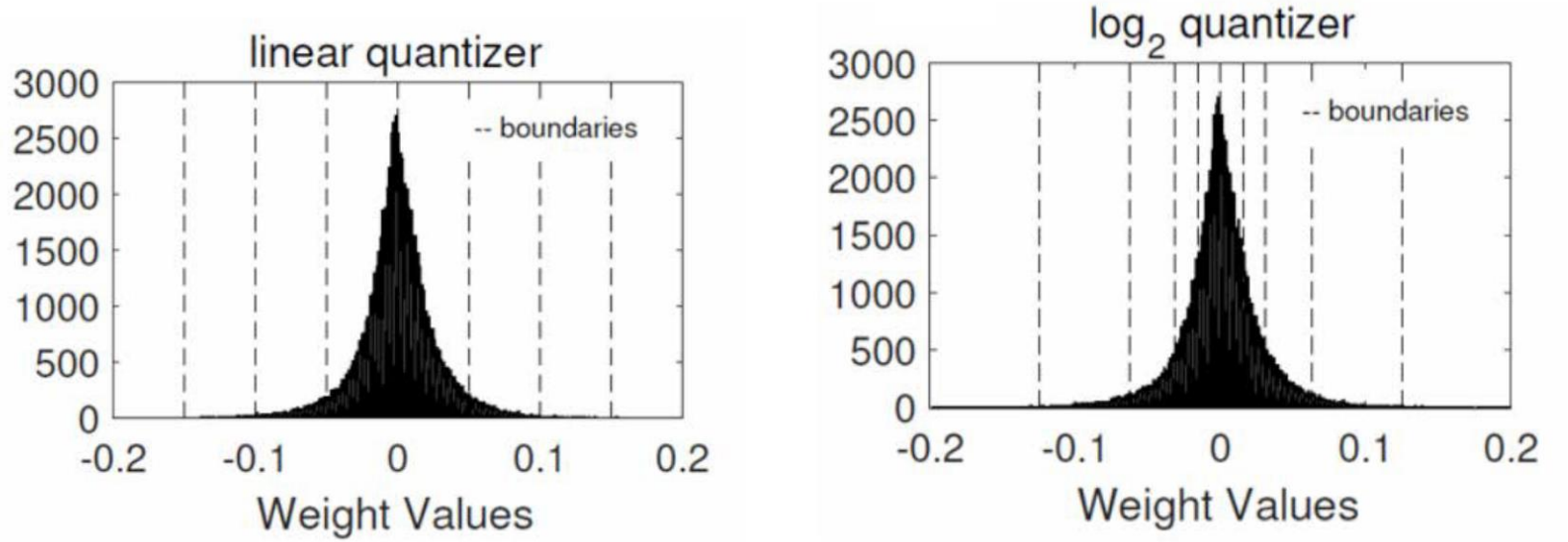
- Distributions of weights and activations are almost never uniform.

- Non-uniformly spaced quantization points may improve accuracy
  - Also called “non-linear” quantization



Log-domain quantization

- **Idea:** more quantization levels close to 0, since that is where most values fall.
- This is achieved using a logarithmic mapping, rather than a linear one.

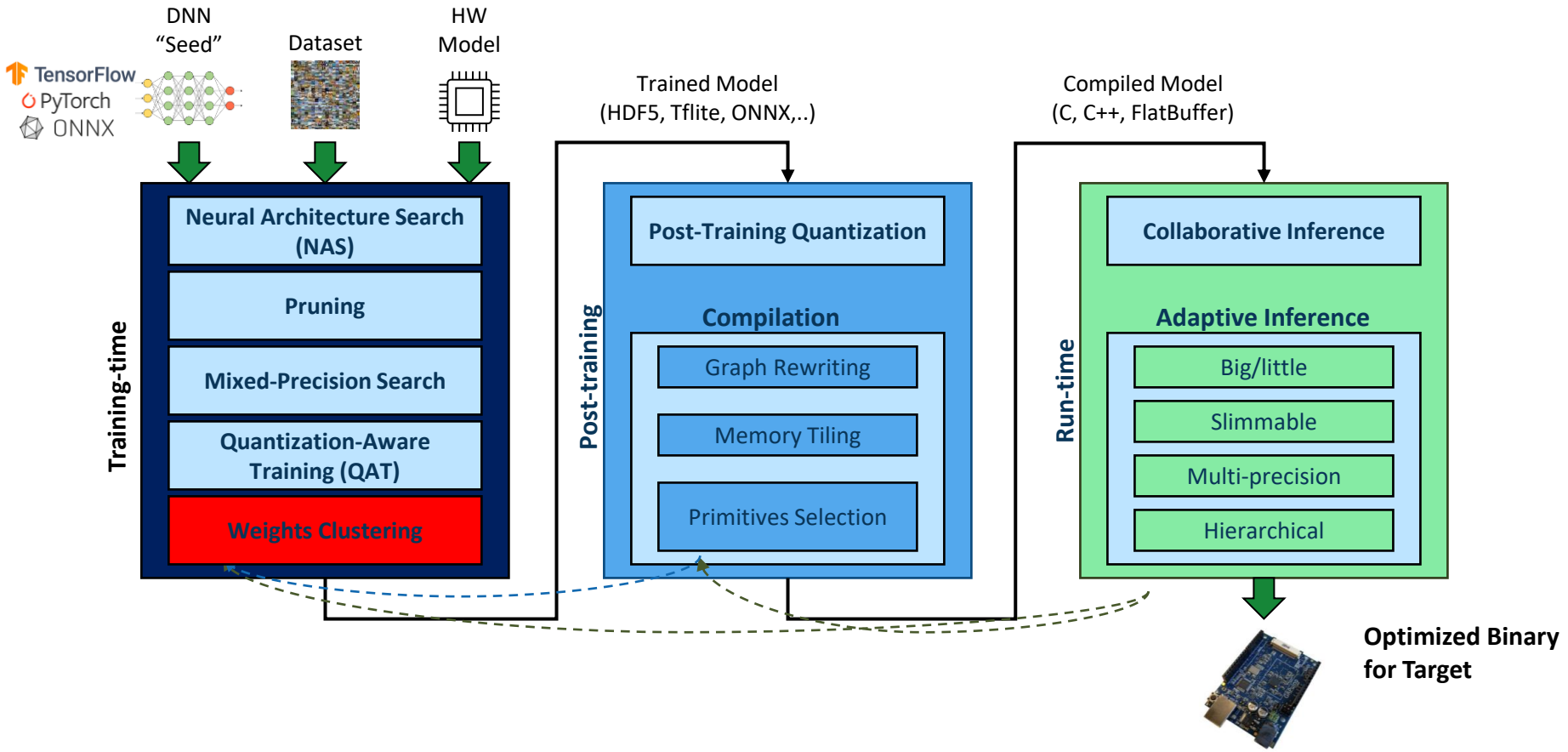


[11] Lee, E. H., et al., Lognet: Energy-efficient neural networks using logarithmic computation. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

Politecnico
di Torino

Weights Clustering

Recap: Our Flow



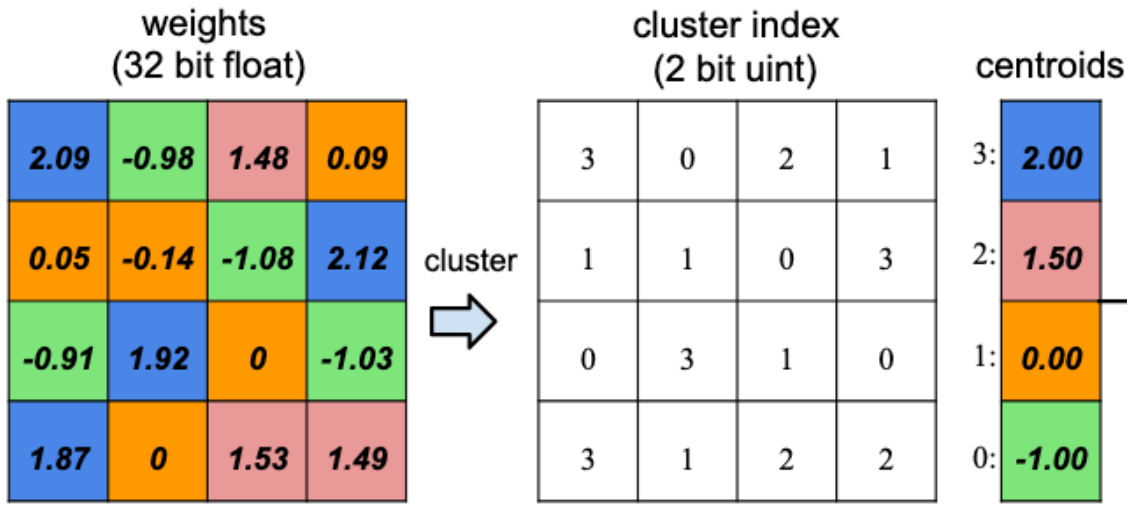
Weights Clustering

- Can be considered a form of quantization in which rather than reducing the precision of each weight/activation value, we reduce the number of distinct values allowed.

- Store the allowed values in a codebook (i.e., *look-up table*)
  - Store weight as codebook index → reduce storage size

Weights Clustering

- Idea: replace each float weight with its closest allowed value (cluster centroid). Then, store only the values of centroids plus one index per weight.

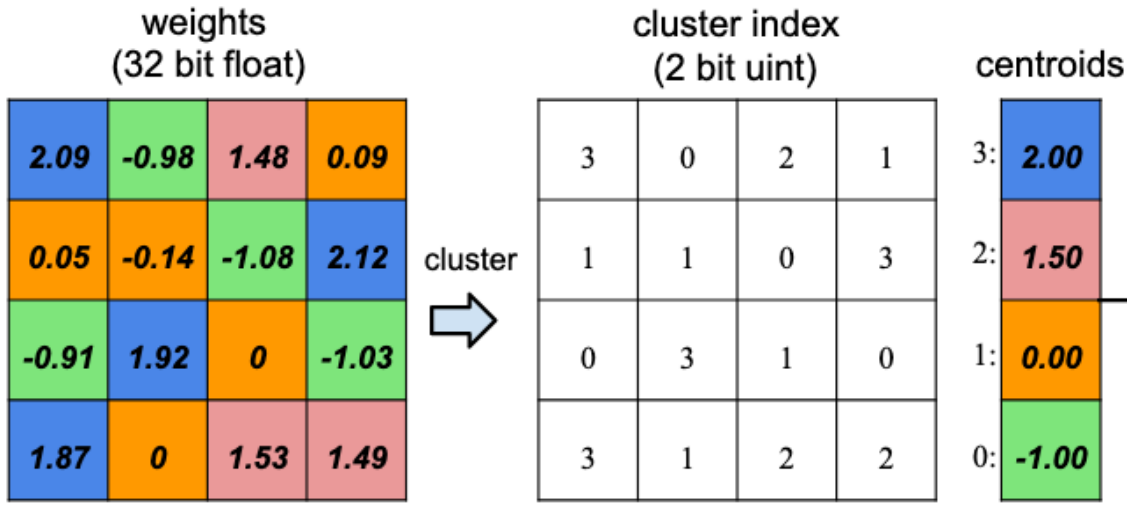


[12] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

Weights Clustering

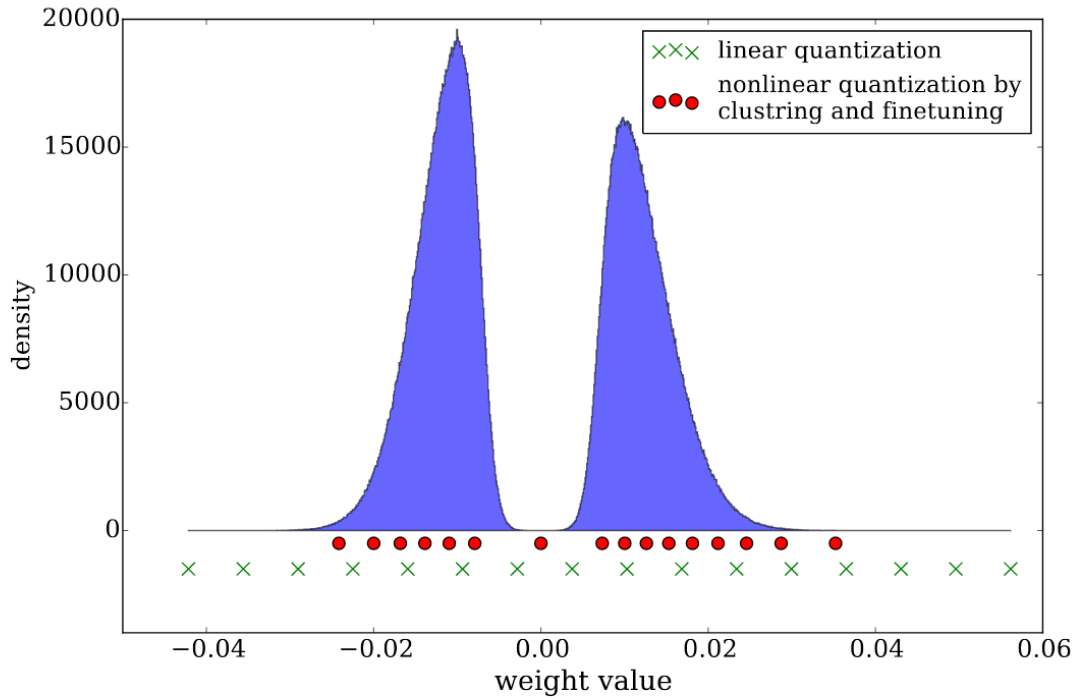
- Example: 4x4 matrix with 4 allowed values:
  - Original storage: 4x4x32bit = 512bit
  - Clustered storage: 4x32bit (centroids) + 4x4x**log2(4)** (indexes) = 160bit

- Model size reduction, but possibly slower inference (2-step weight access).



Weights Clustering

- Match the quantization levels and the data distribution:



[12] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

Weights Clustering

- How to determine the centroids during training? The most famous algorithm works as follows:

1. First, the centroids are initialized using a pre-trained float model.
   - Different initialization methods
   - E.g., uniformly spaced between the min and max value of each weight tensor

2. Then, a k-means clustering is applied to obtain the initial optimized centroids.

[12] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

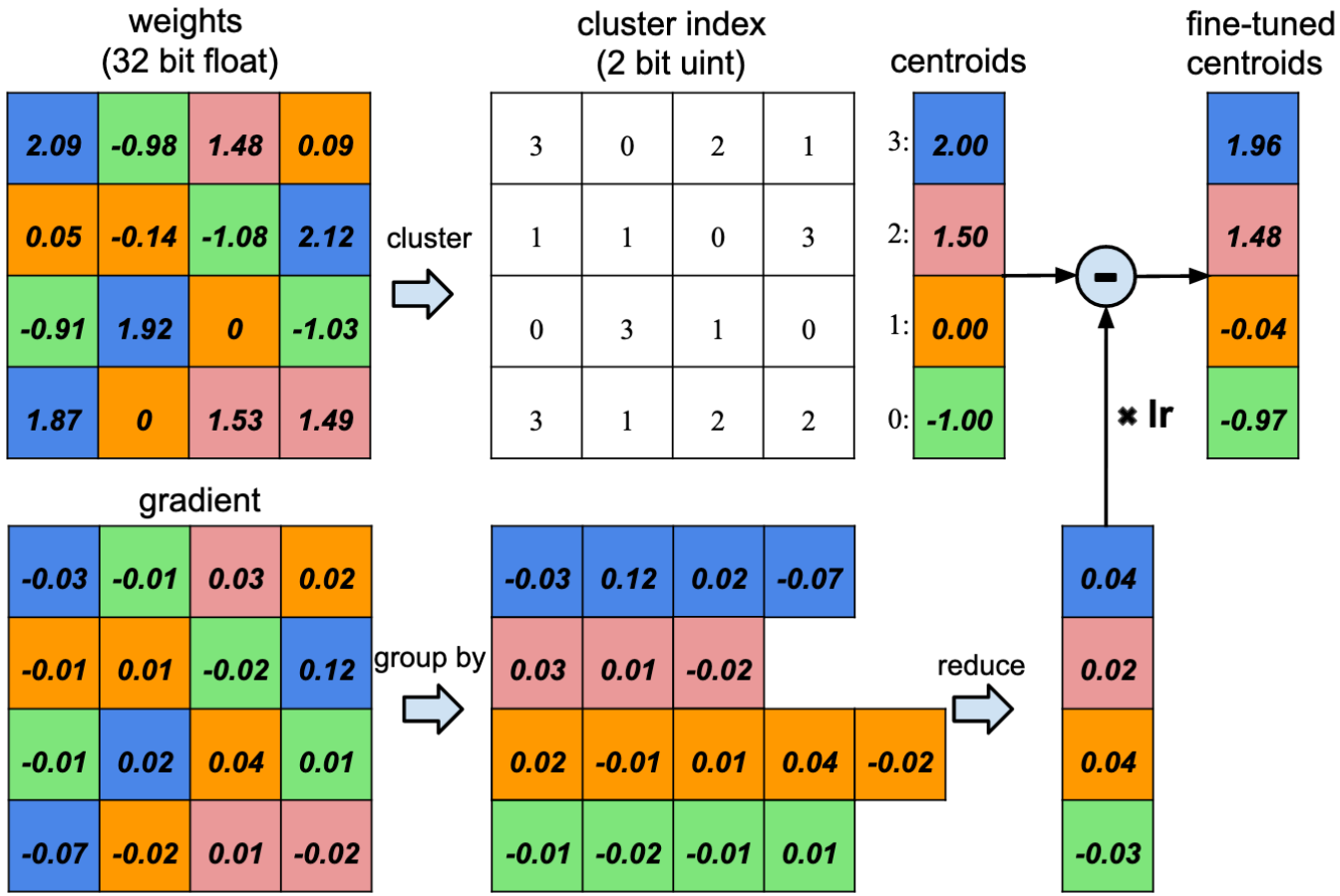
Weights Clustering

3. Lastly, the model is then fine-tuned as follows:

   a) In the forward pass, each weight is replaced with the closest centroid

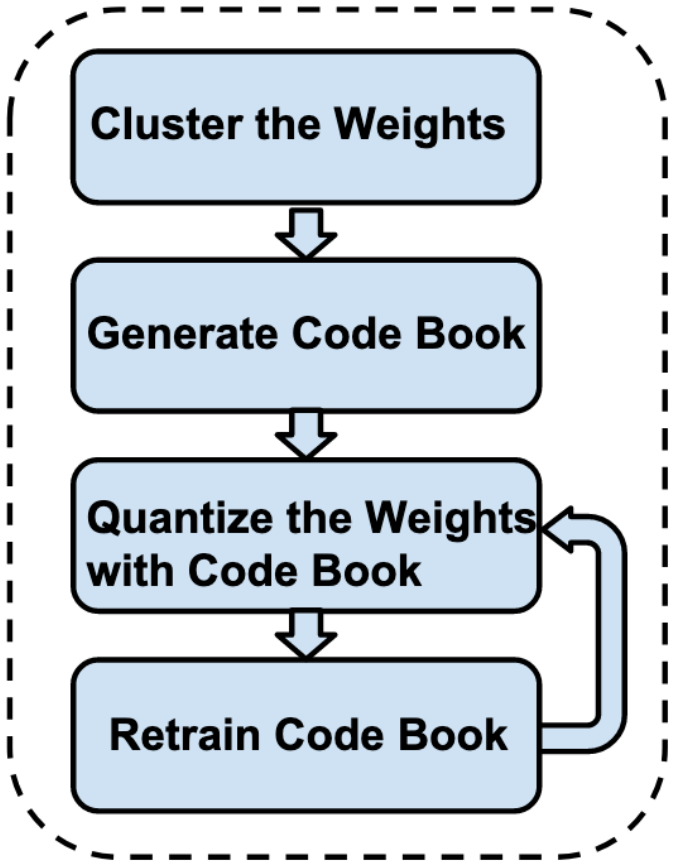
   b) In the backward pass, the gradients of all weights that were assigned to the same cluster are summed together, and used to update the centroid for the next iteration.

Weights Clustering



Weights Clustering

- Recap:



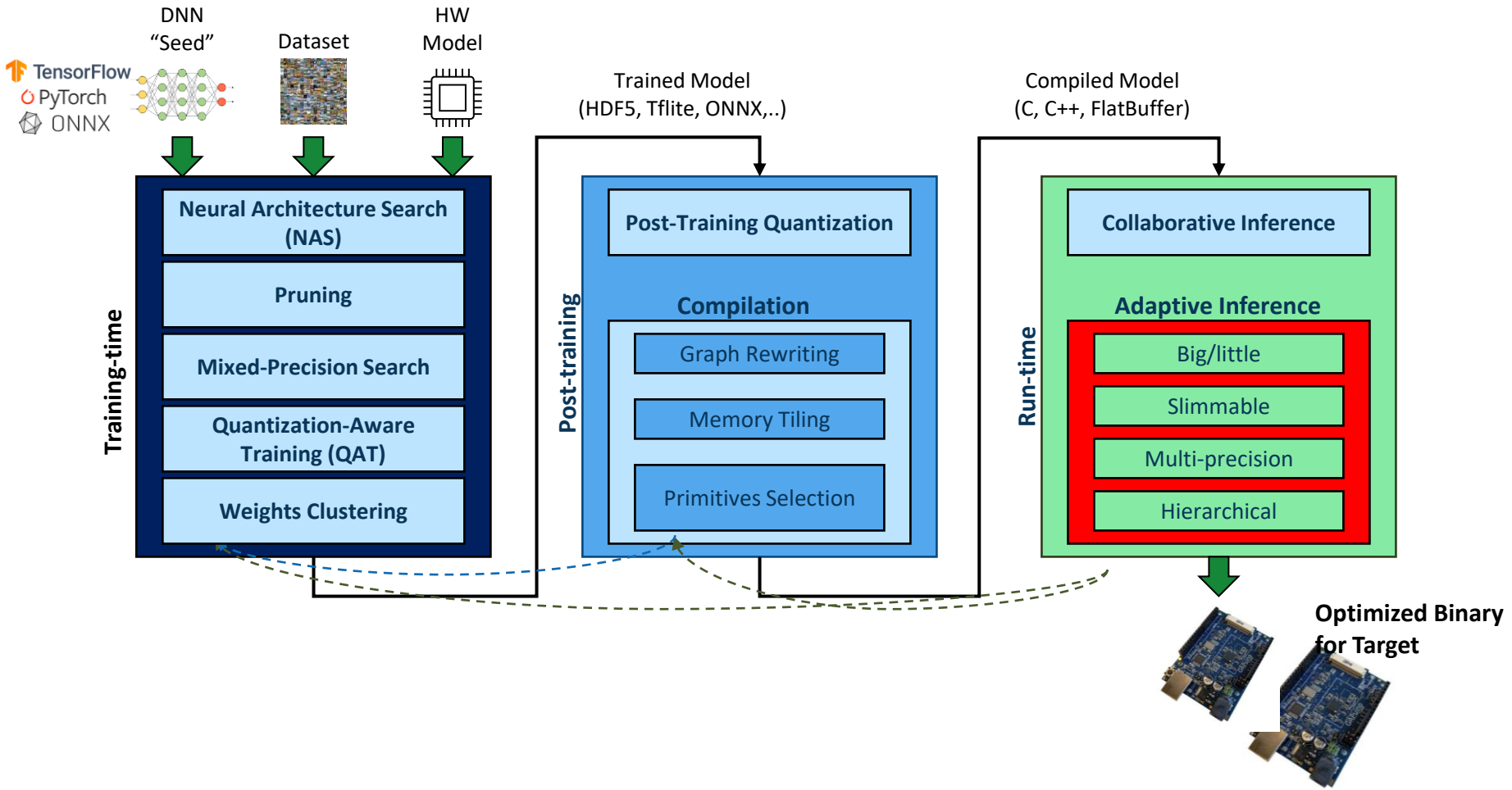
Weights Clustering

- Weight clustering can also be combined with quantization, so that centroids are stored as int8 rather than float32.

Politecnico
di Torino

Adaptive Inference

DNN Deployment Flow

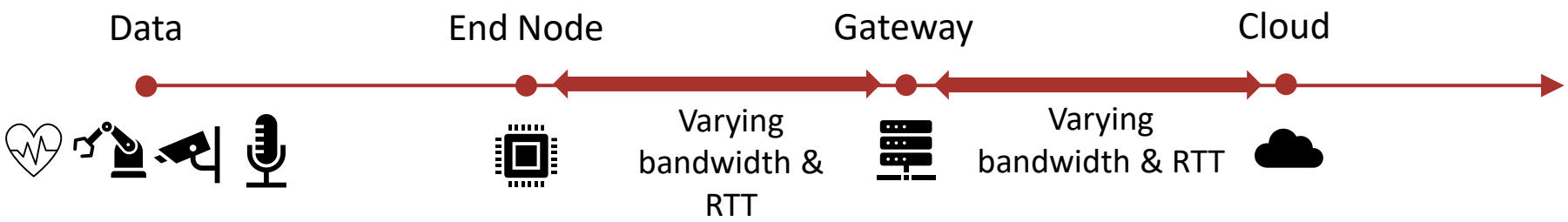


Motivation

- Static design-time optimizations (quant, pruning, etc) are not sufficient

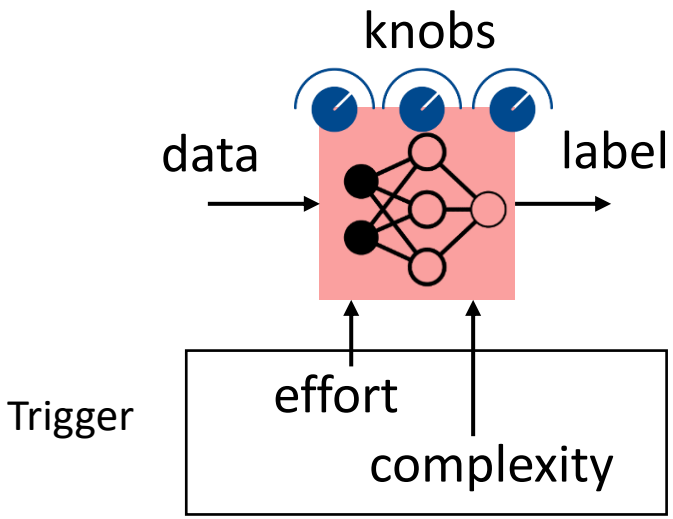
- Constant compute effort:
  - Can’t adapt to task complexity
  - Can’t adapt to external conditions (e.g. battery life)

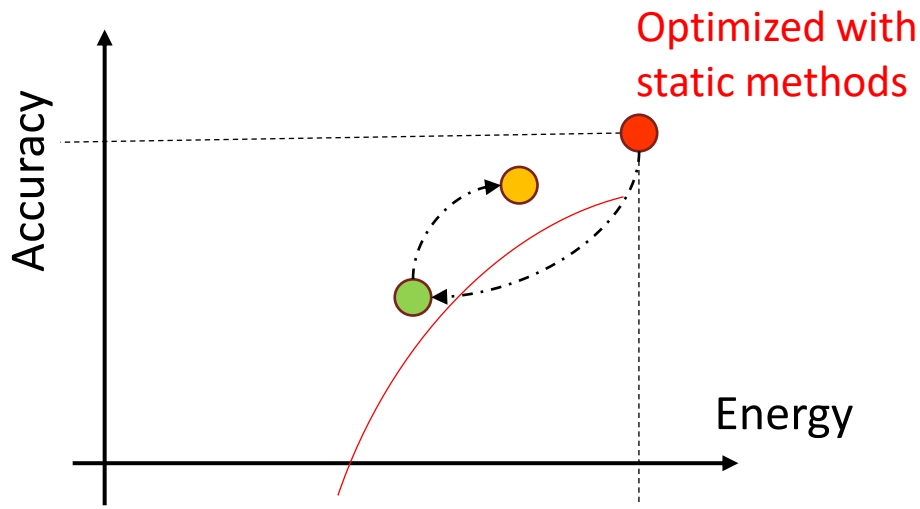
- The ”best” inference device is time-dependent:
  - The varying speed/energy of compute (esp. on cloud)
  - The varying speed/energy of communication



Adaptive/Dynamic Optimizations

- Shift among different operating points at runtime based on trigger:
  - **Effort**, i.e., resource budget → **Energy/Quality Scaling or Dynamic Inference**
  - **Complexity**, i.e., task hardness → **Adaptive Inference**

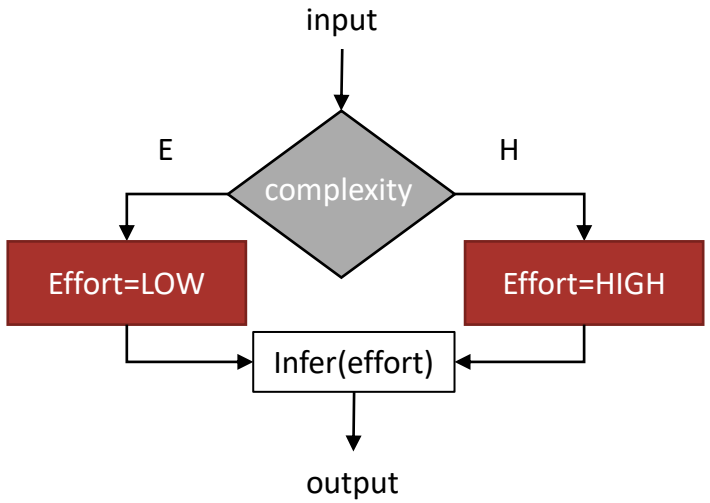




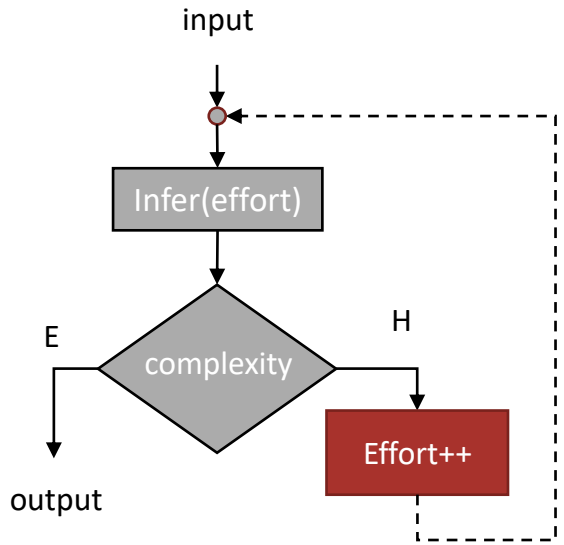
Adaptive Inference

- Naïve approach: train and alternate between **multiple models**
  - Big/little systems (see later)
  - Main drawback: huge **memory overhead**

- How to decide which model to run?



One-shot (difficult, task-specific)



Iterative (easy, general, less effective)

Politecnico
di Torino

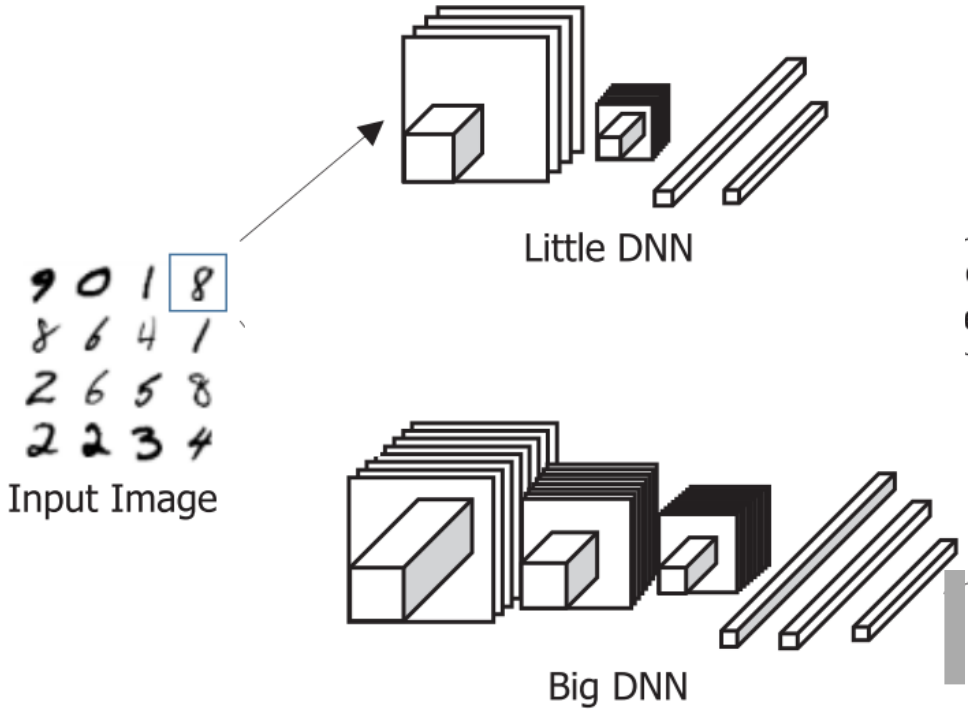
Iterative Adaptive Methods

Big/Little Models

- Naïve approach to adaptive inference: multiple models

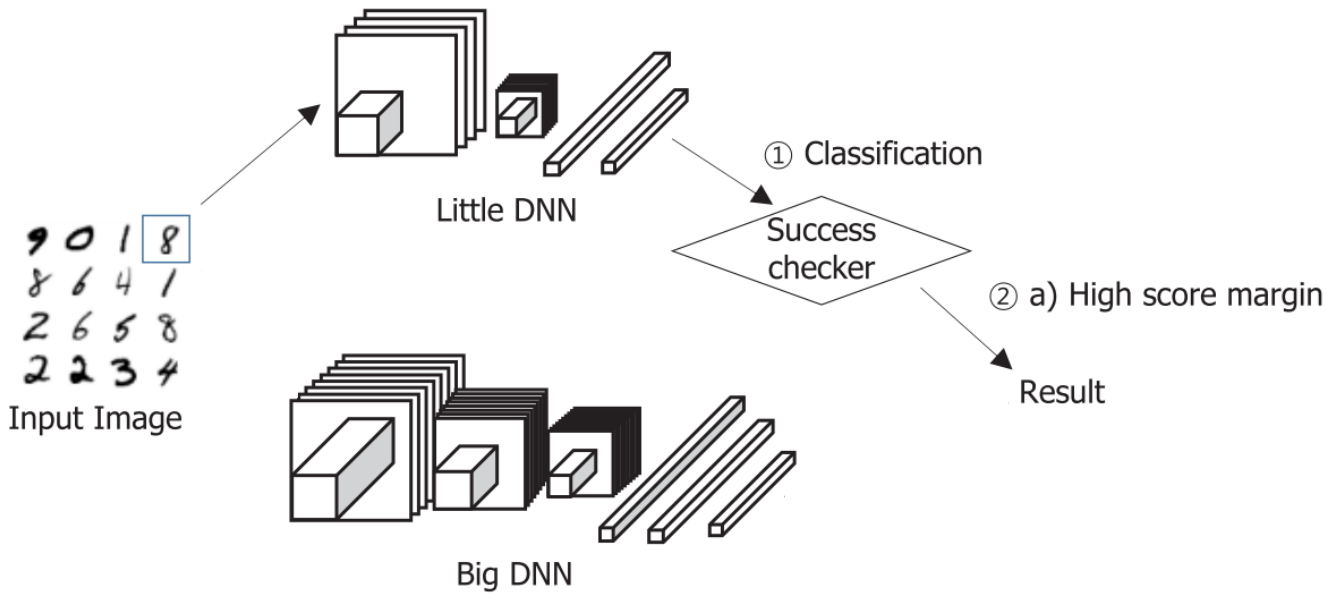
1. Train two models of different size on the same task
   - Differences can include the number of layers, channels/nodes, filter sizes, etc.

2. Process each new input with the ”**little**” model first.



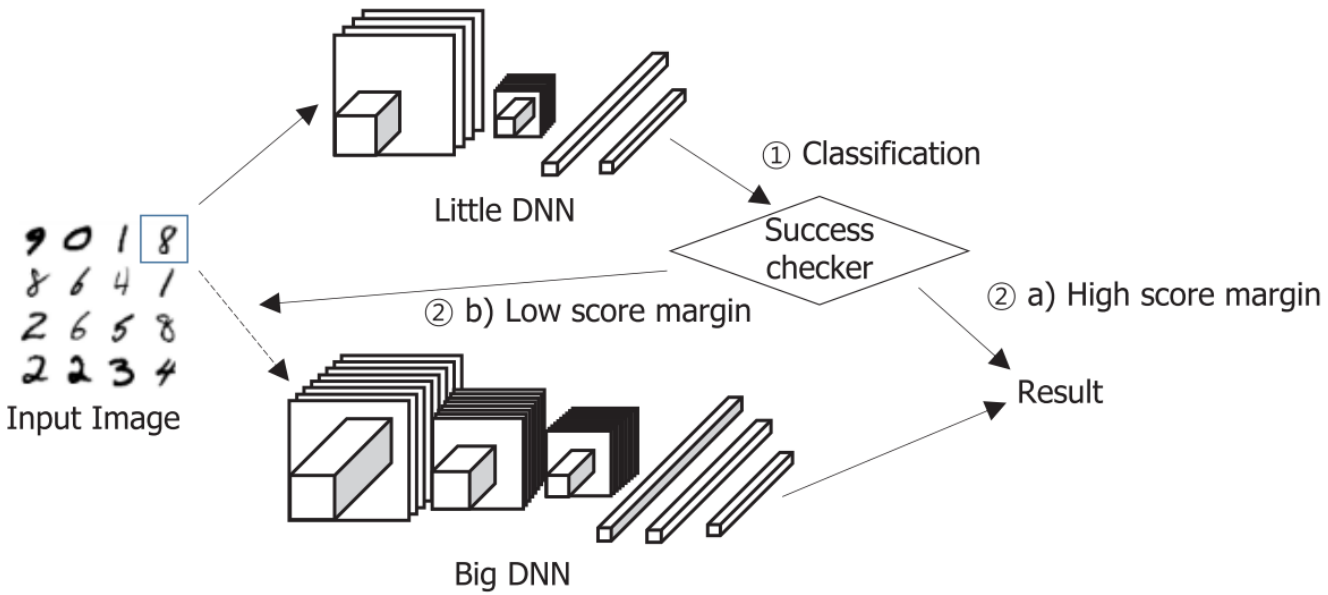
Big/Little Models

3. Process the result of the little model with a “success checker”.
   - Determines if the little model was “confident” about its prediction.
   - If the prediction is confident, it is simply used as the final output and inference is stopped.



Big/Little Models

3. Otherwise, send the input to the “big” model and perform a second inference. Use the big model output as the final result



Big/Little Models

- Assumptions:
  1. Inputs are not all equally difficult to classify (adaptive)
  2. **Easy inputs are the majority**

- If those assumptions holds, the B/L system will:
  - Reach an **accuracy comparable to that of the big model** alone.
  - But **using only the little model for the majority of inputs**, saving time and energy (on average)

- If assumption n.2 is false, then there will be a time/energy overhead
  - Two inferences per input…

Big/Little Models

- The "**success checker**" is key!
  - Must correctly identify which predictions of the little model are correct.

- The most common method (for classifiers) is the so-called **Score Margin (SM)**:

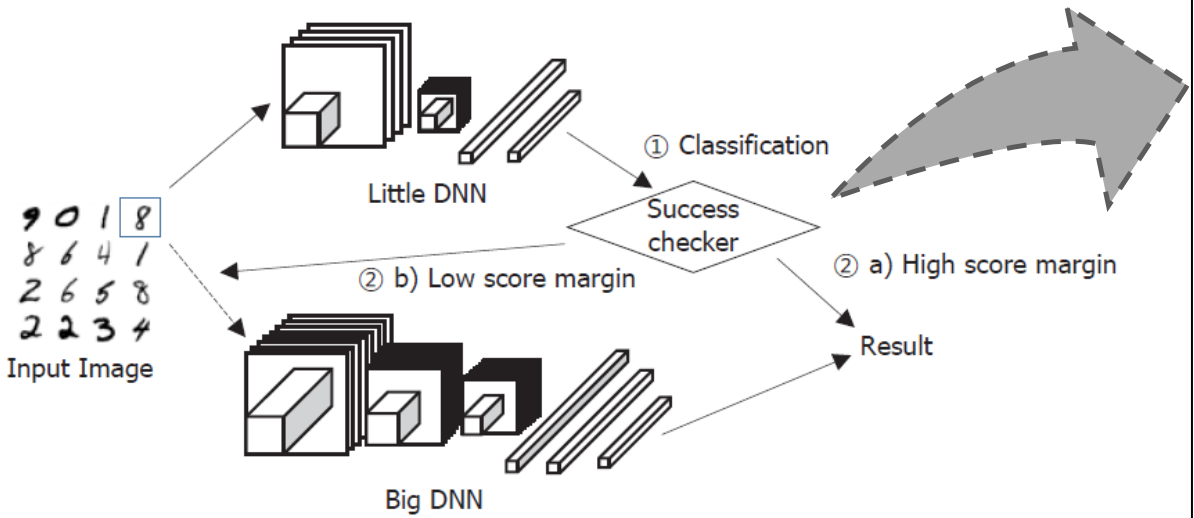
$$SM = \ largest\ prob. - 2nd\ largest\ prob.$$

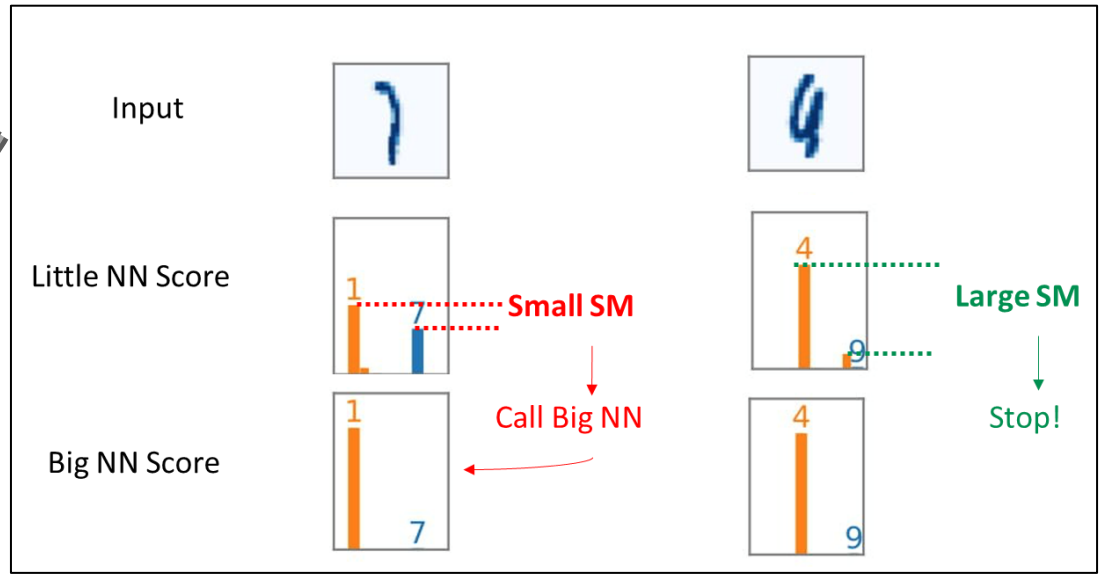
  - The larger the SM, the "more confident" the prediction

Big/Little Models

- Success Checker rule:
  - *"invoke the "big" model iff SM <= T_h , stop otherwise"*

  - T_h is a user-defined threshold. Can be changed at runtime to tune the energy/latency vs accuracy trade-off
    - **Smaller T_h saves more energy/time** and vice versa



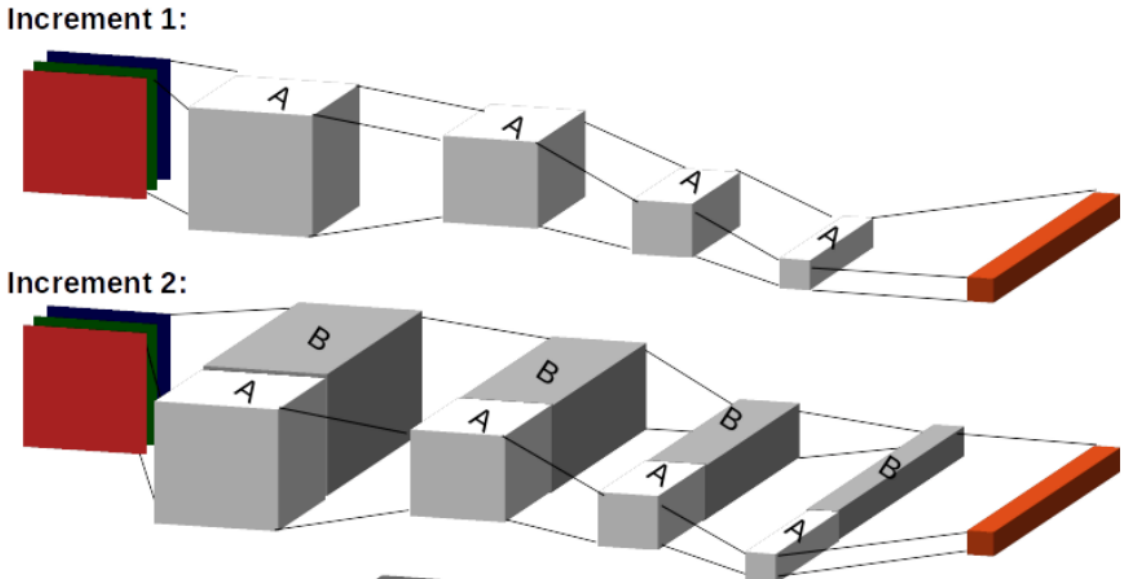


Variable-width NNs

- **Variable-Width DNNs:**
  - Reduce memory overhead of big/little systems
  - ”Little” model generated from a **subset of the big model’s features:** width multiplier or channel pruning
  - Can be easily extended to more than two “increments”

”easy input"

”hard input"



[13] H. Tann et al, “Runtime configurable deep neural networks for energy-accuracy trade-off”. Proc. IEEE/ACM/IFIP CODES, 2016

Incremental training

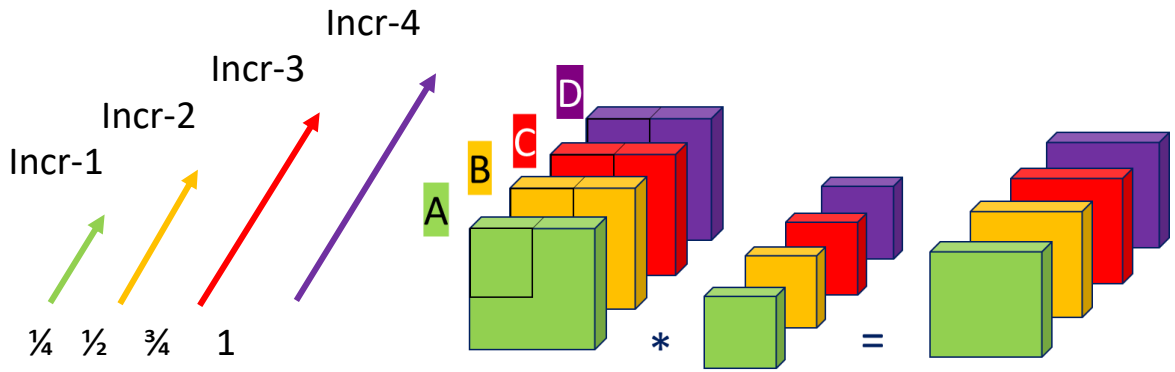
Num_Incr = #sub-networks

Incr_Arch = sub-network architectures

Algorithm 1: Incremental Training

Input : Num_Incr, Incr_Arch
Output: Trained Network

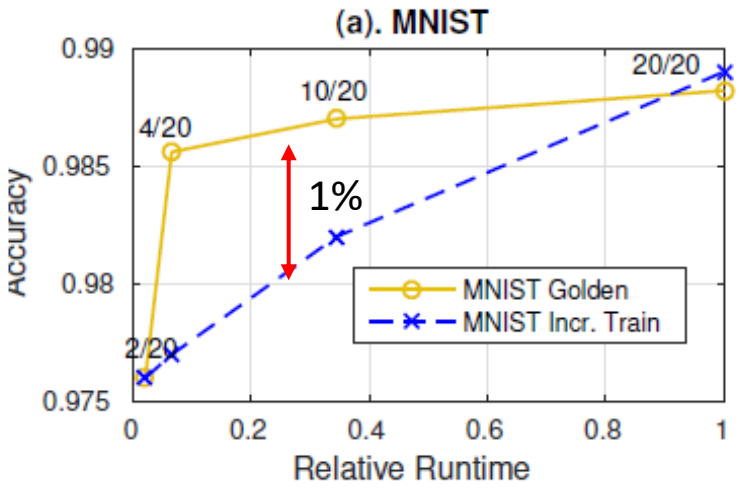
```
1 net = initialize(Incr_Arch[1])
  // Train all weights in net
2 net = train(net, KEEP_FIXED(NULL))
3 for i = 2 to Num_Incr do
      // Add more channels and initialize their
      // weights
4     tmp_net = {net ∪ initialize(Incr_Arch[i])}
      // Keep all weights corresponding to net
      // fixed
5     tmp_net = train(tmp_net, KEEP_FIXED(net))
6     net = tmp_net
7 end
8 return net
```

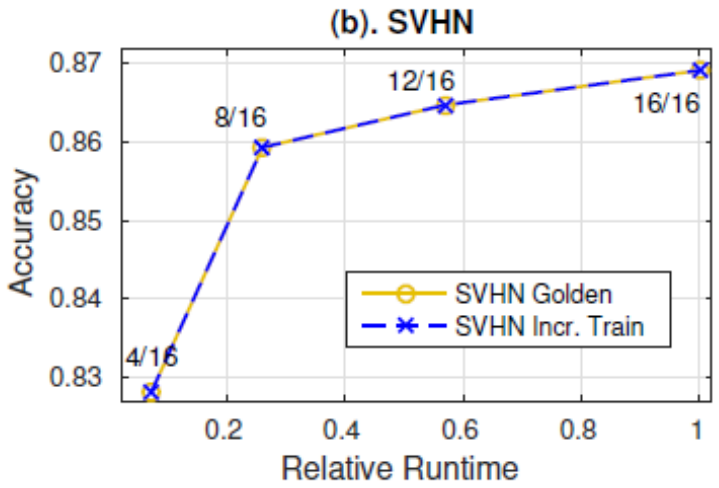


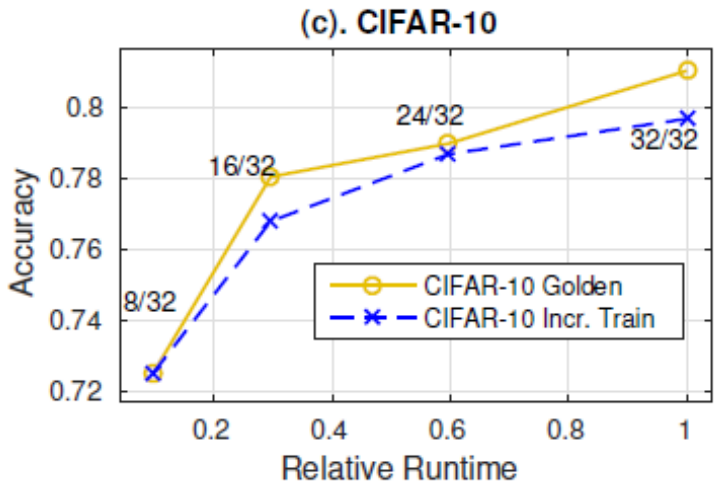
[13] H. Tann et al, "Runtime configurable deep neural networks for energy-accuracy trade-off". Proc. IEEE/ACM/IFIP CODES, 2016

Results on incremental training

Golden = multiple separate models

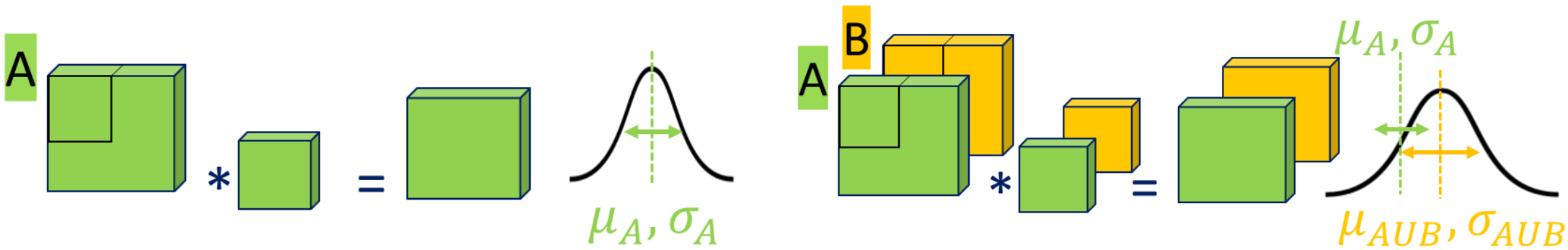




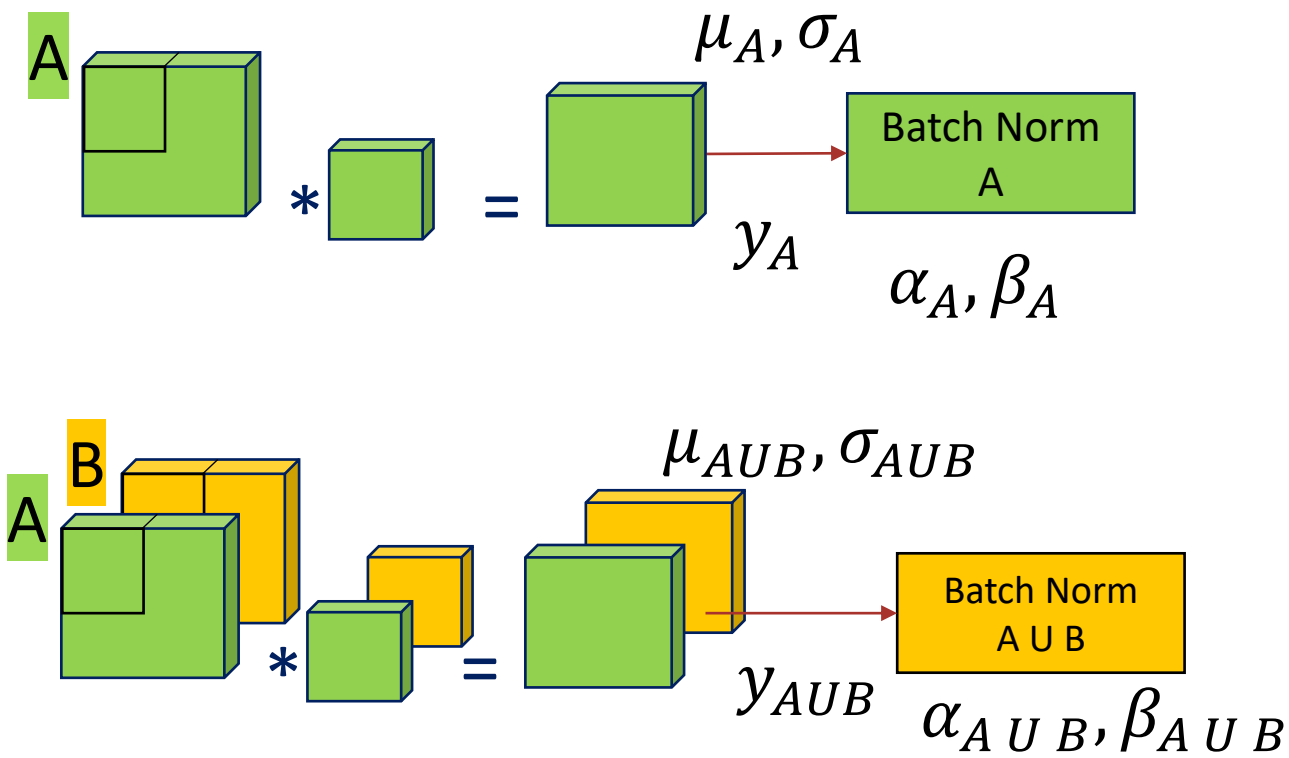


Limitations of incremental training

- Feature aggregation inconsistency:
  - Expanding sub-net A to A U B, new connections (A → B, B→A) are added
  - Accumulating different numbers of channels results in different μ, σ
  - This affects negatively **BatchNorm** layers:
    - Training is stable, but quite far from optimal
    - High loss w.r.t. the golden solution



Switchable Batch Norm



- Privatized Batch Norm
  - Each sub-net has its own BN setting

Sb1:: A $\leftarrow BN_A$

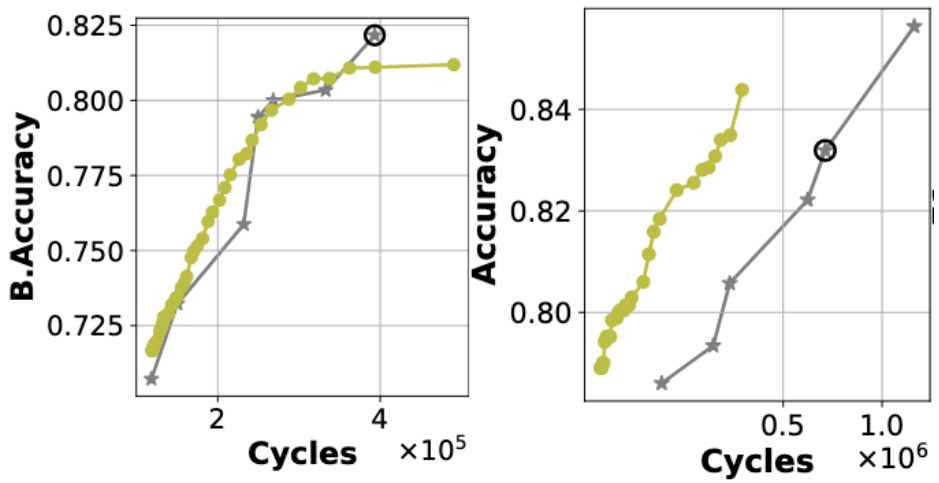
Sb2:: A U B $\leftarrow BN_{A\ U\ B}$

Sb3:: A U B U C $\leftarrow BN_{A\ U\ B\ U\ C}$

[14] Yu, J., Yang, L., Xu, N., Yang, J., & Huang, T. (2018). Slimmable neural networks. arXiv preprint arXiv:1812.08928.

Variable-Width NNs Results

- **Variable-Width NNs for Human Activity Recognition (HAR):**



(a) UniMiB-SHAR (b) UCI HAPT

Method	Operating Points	Tot. Memory [kB]
Multiple Static Models	5/4	50.6/39.4
Variable Width	**47/34**	**26.3/14.6**

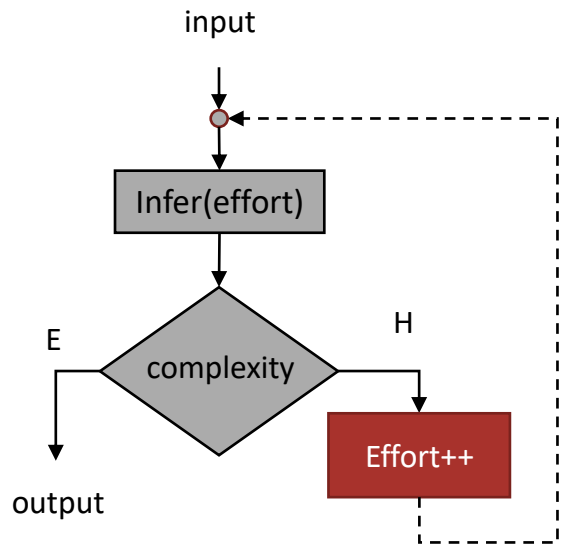
- Multiple operating points obtained changing the **score margin threshold** that identifies an "easy" input
- **2x/3x less memory** for same runtime flexibility
- Comparable latency/enegy per inference.

[15] F. Daghero et al, "Human Activity Recognition on Microcontrollers with Quantized and Adaptive Deep Neural Networks", ACM Trans. on Embedded Systems, 2022.

Other Iterative Systems

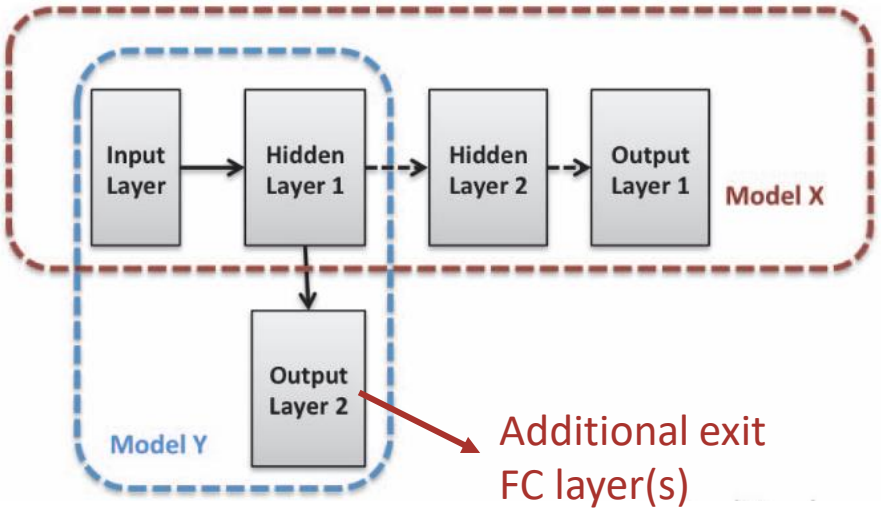
- Besides naïve big/little and variable-width, adaptive “iterative” systems can be implemented using:
  - Different number of layers (**early-exit**)
  - Different quantization bit-width (**dynamic precision scaling**)
  - Gates/mixtures of experts
  - Sparsity
  - Etc.

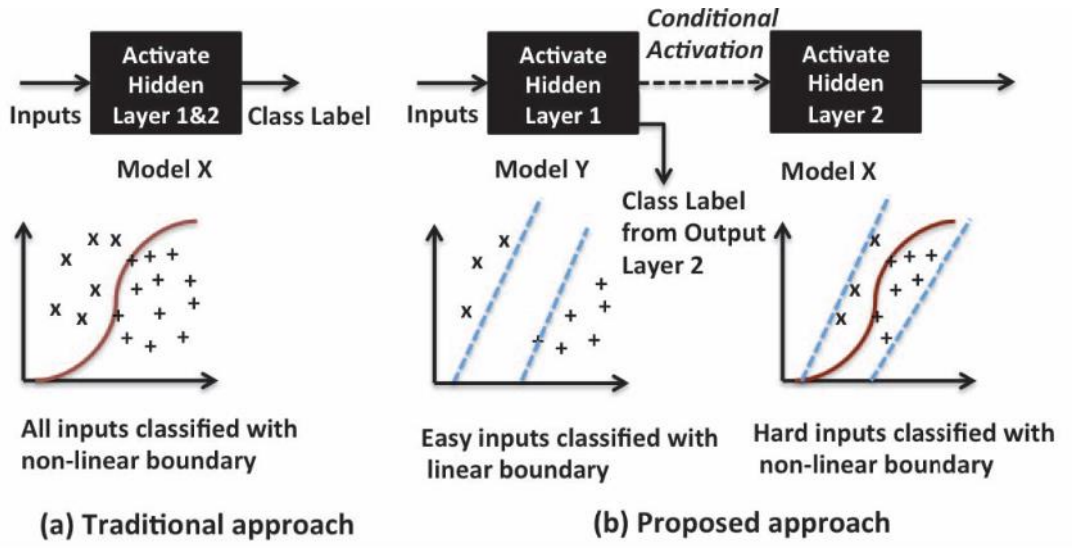
- Many use the Score Margin as complexity estimator.
  - Some issues here…
  - But there are other approaches.



Early-Exit NNs

- Principle:
  - Shallower models can correctly classify data that can be separated by a simple decision boundary (e.g. linear)
  - Deeper models are required to obtain more complex boundaries.

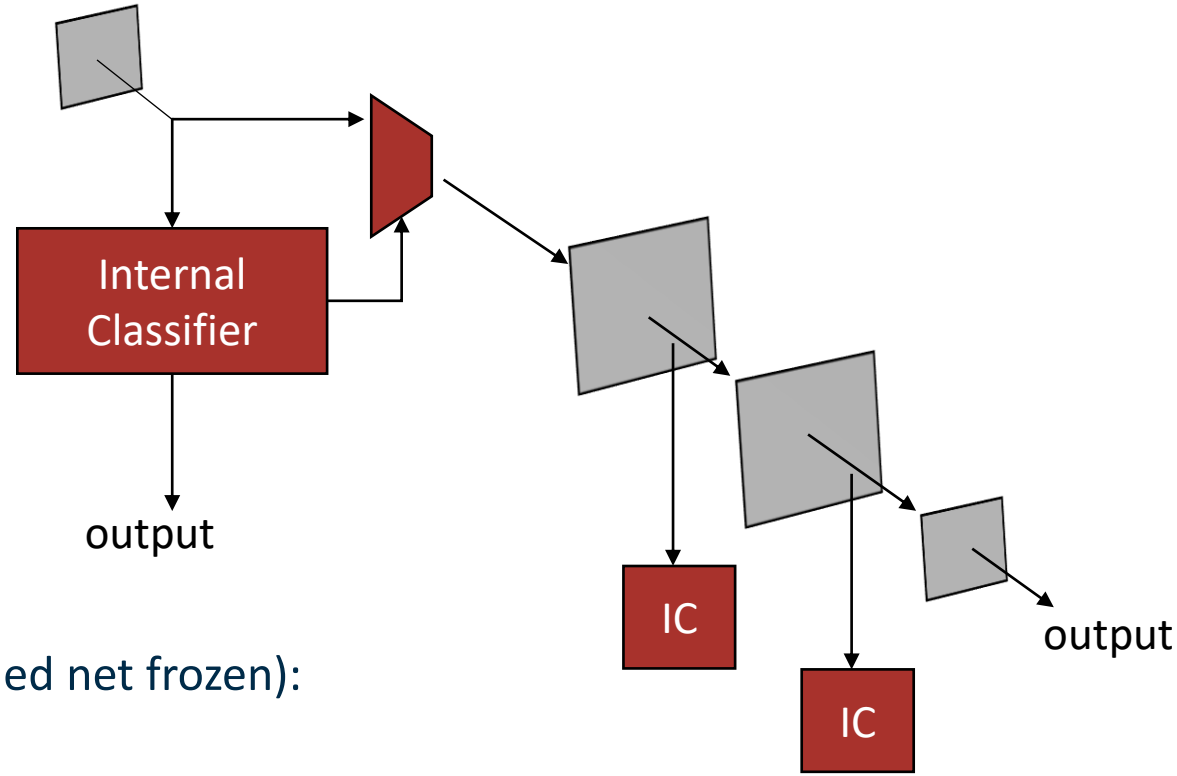




[16] P. Panda et al. *Conditional Deep Learning for Energy-Efficient and Enhanced Pattern Recognition*.
[17] Shallow-Deep Networks: Understanding and Mitigating Network Overthinking, 2018.

Early-Exit NNs

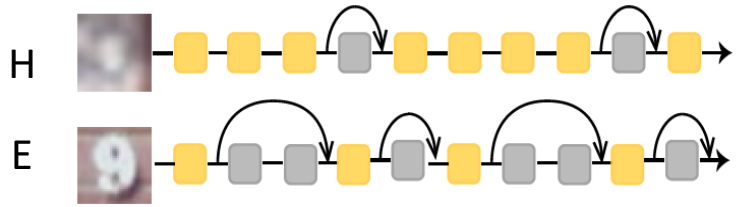
- Issues:
  - Early-exit branches overhead
  - Training the system:
    - Train early—exit branches only (pre-trained net frozen): Fast, but weak
    - Train whole network with a weighted loss function

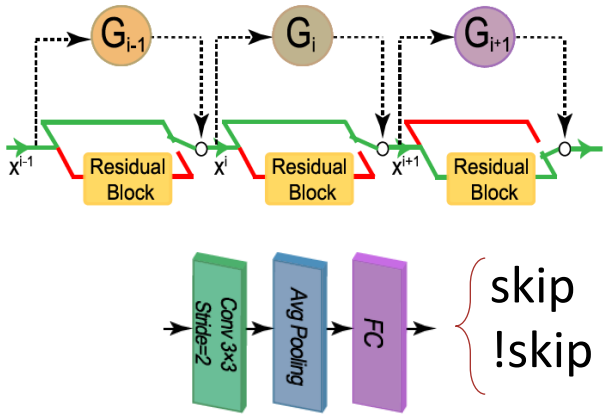


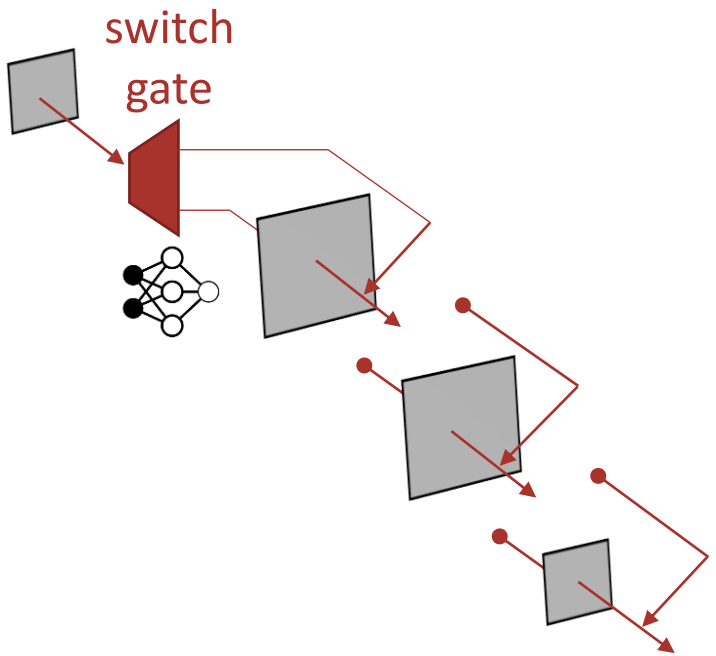
[source] P. Panda et al. *Conditional Deep Learning for Energy-Efficient and Enhanced Pattern Recognition*.
[source] Shallow-Deep Networks: Understanding and Mitigating Network Overthinking, 2018.

Switch Gates

- Run-time «depth modulation»
- Switch gates
  - Layers that infer when the next layer is needed
  - Can be less complex than “early-exit”
  - Hard to train (binary output)
  - Works well mostly for **very deep networks**!





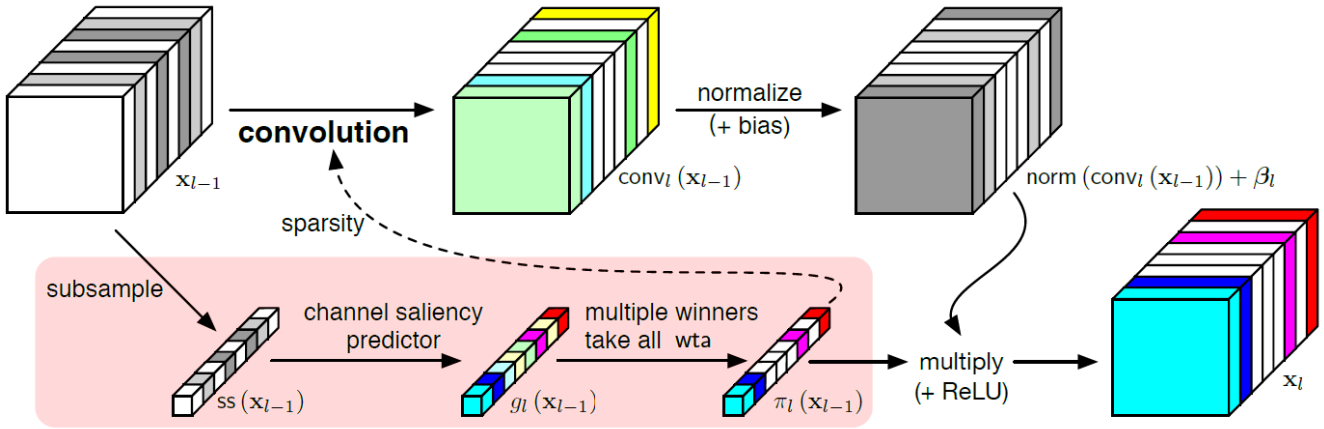


[18] Wang, Xin, et al. "Skipnet: Learning dynamic routing in convolutional networks." Proceedings of the European conference on computer vision (ECCV). 2018.

Feature Boosting and Suppression

- Adaptive Gate at the level of features, instead of layers
  - Channel-wise subsampling to a scalar (pooling)
  - Channel saliency prediction: fully connected layer
  - Voting: K-winners-take-all
  - Convolution w/o suppressed input channels
  - Boost features during Batch Norm to compensate

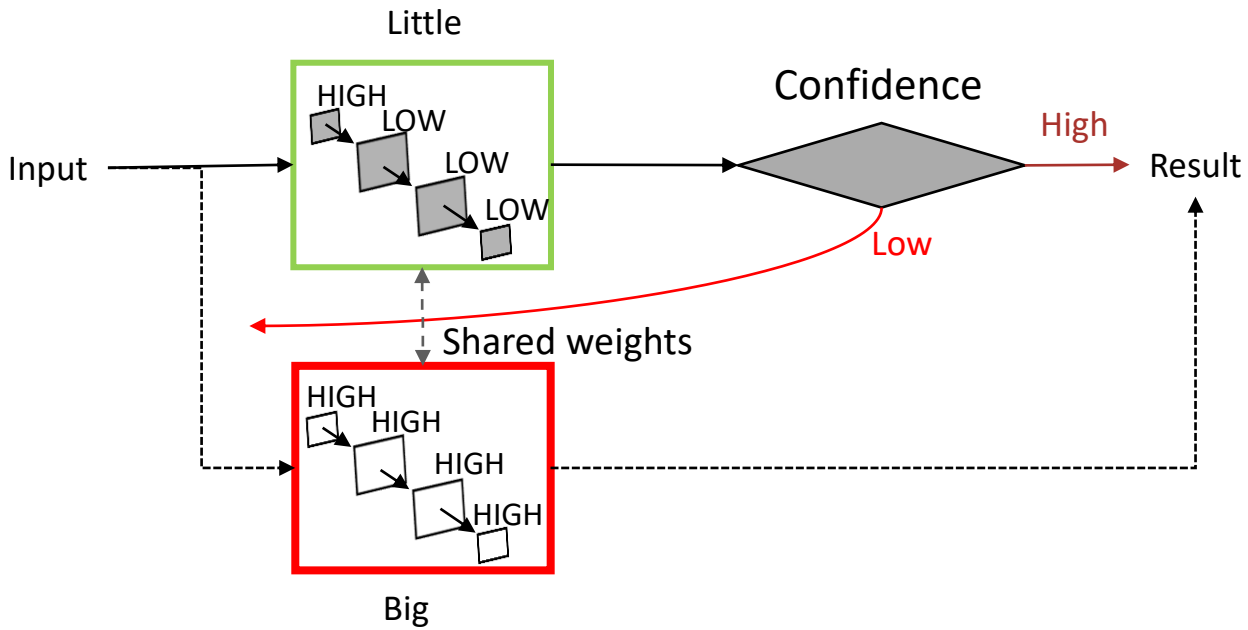
→ Not trivial to optimize..

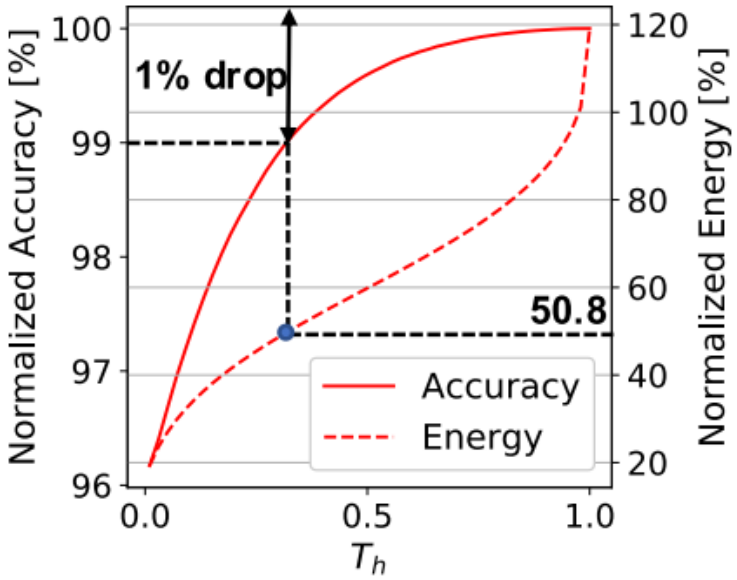


[19] Gao, Xitong, et al. "Dynamic channel pruning: Feature boosting and suppression." *arXiv preprint arXiv:1810.05331* (2018).

Dynamic Precision Scaling

- "Big-little" using the same model with different quantizations





[20] D. Jahier Pagliari et al, *"Dynamic bit-width reconfiguration for energy-efficient deep learning hardware"*, Proc. ISLPED 2018.
[21] V. Peluso et al, *"Energy-Accuracy Scalable Deep Convolutional Neural Networks: A Pareto Analysis"*

Politecnico
di Torino

Hierarchical / StagedAdaptive Methods

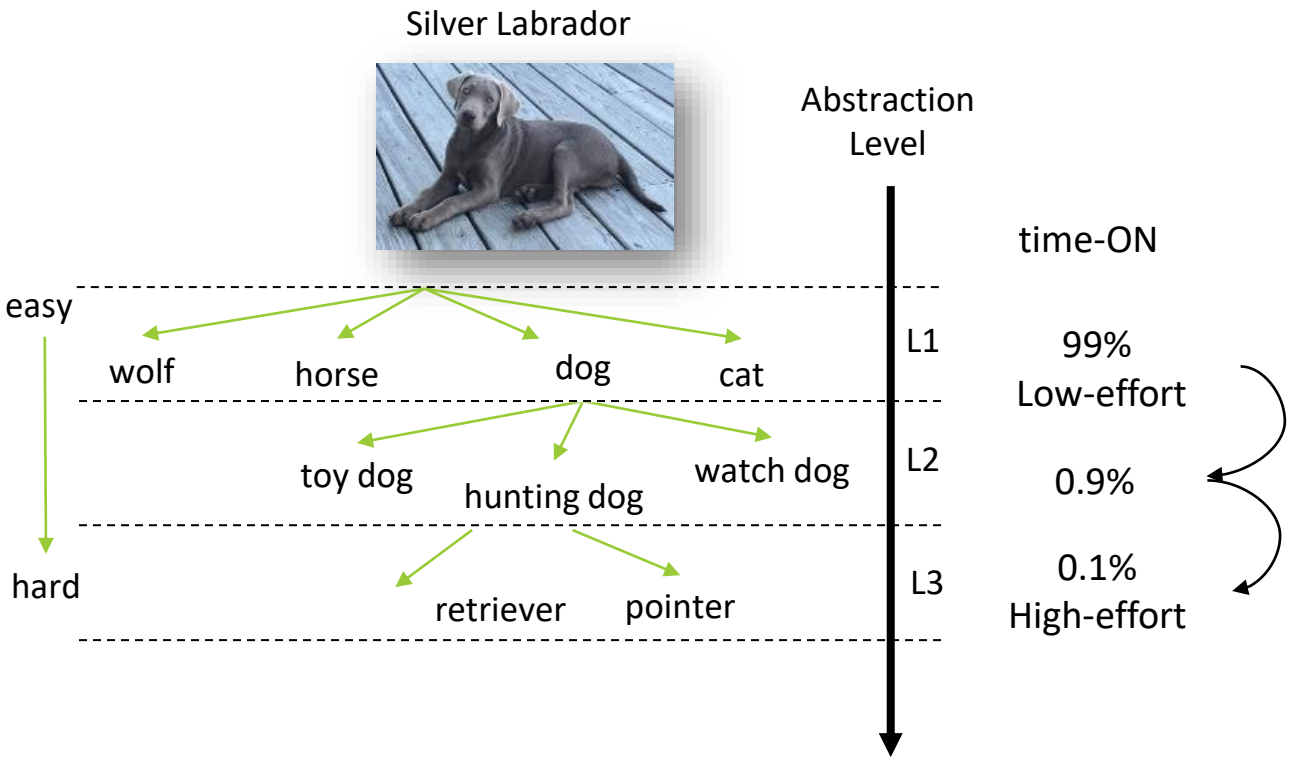
Staged/Hierarchical Inference

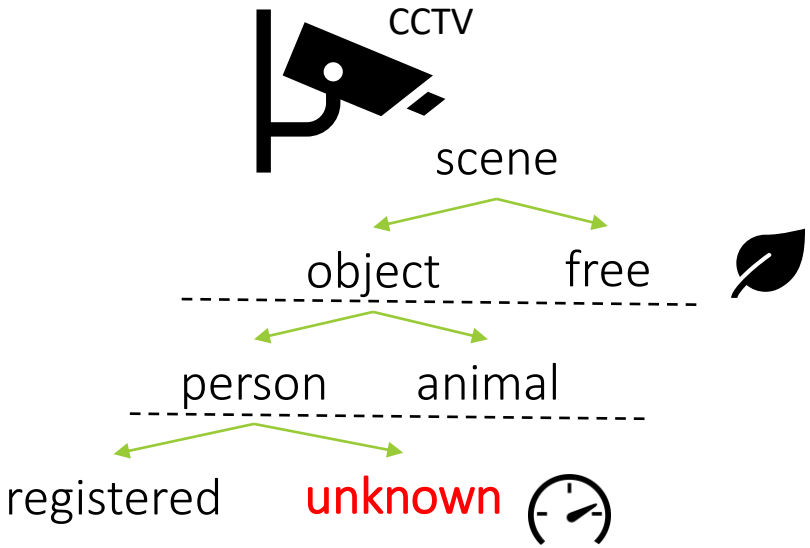
- Score Margin based systems offer a systematic way to perform input-dependent inference:
  - The system decides automatically if an input is ”easy” or “difficult”

- Using prior knowledge on the target application to distinguish easy and hard inputs may lead to superior results.
  - Potentially more effective, but less general…

- So-called staged/hierarchical inference approach.

Staged/Hierarchical inference





Staged/Hierarchical Inference

- Example: use classifiers of different complexity within a face recognition task

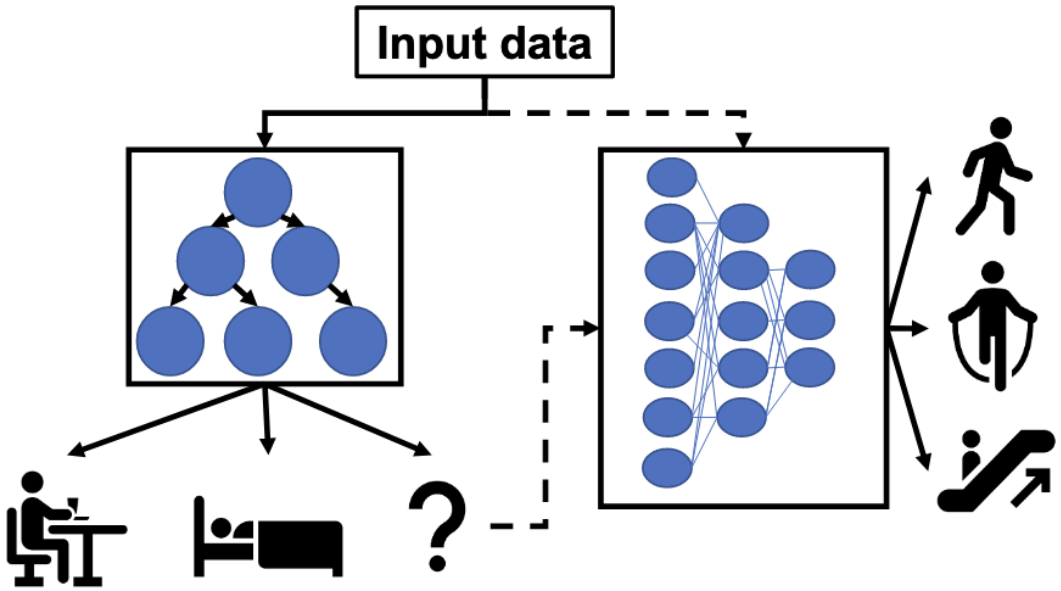
Simpler tasks
use smaller NNs

Staged/Hierarchical Inference

- **Example:**
  - Cascade of classifiers, implementing an increasingly difficult variant of the task
  - Example: Human Activity Recognition.

  - **Decision Tree** to detect “easy” classes (sitting, sleeping)

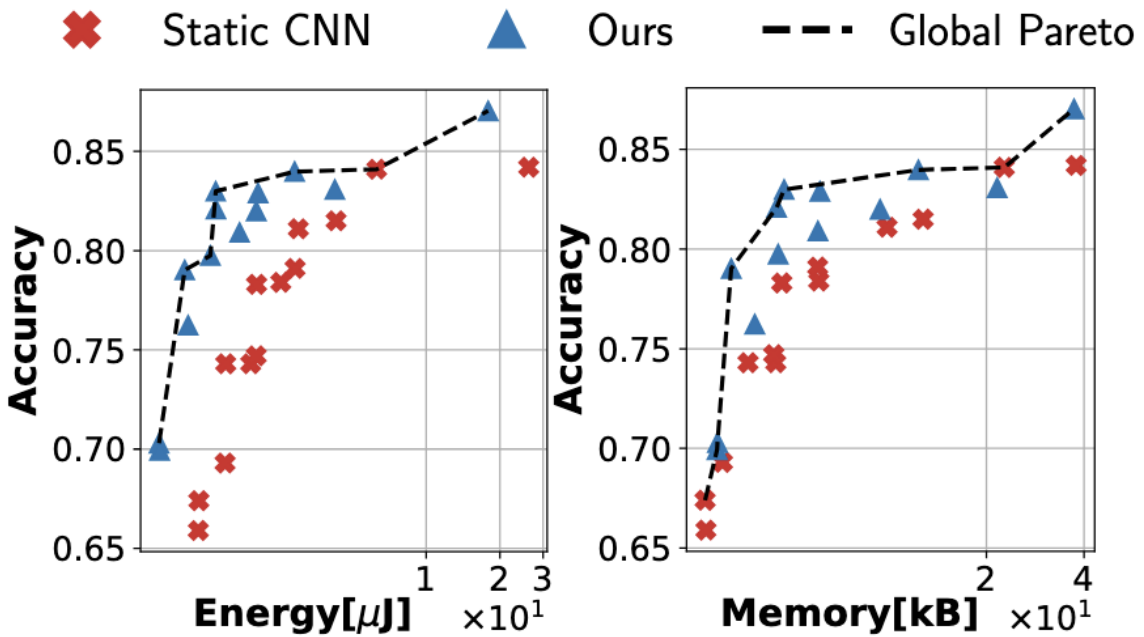
  - Fallback class for “everything else”, processed by a **DNN** (run, cycle, etc)



[22] F. Daghero et al, *“Two-stage Human Activity Recognition on Microcontrollers with Decision Trees and CNNs”,* to appear in the Proc. of IEEE PRIME 2022.

Adaptive Inference Results (4)

- Up to **67% energy reduction** at iso-accuracy w.r.t. a static CNN
- Or, up to **+12% accuracy** at iso-energy
- Even **memory is reduced**: DT has negligible memory overhead, and last CNN FC layer reduced thanks to smaller number of classes

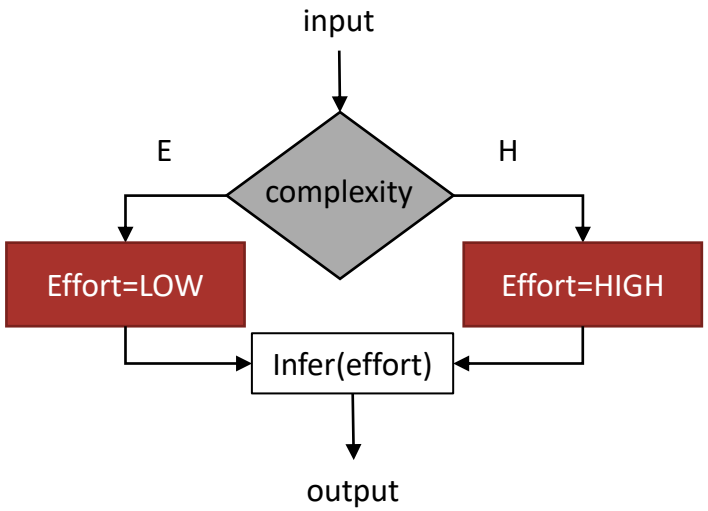


[22] F. Daghero et al, *"Two-stage Human Activity Recognition on Microcontrollers with Decision Trees and CNNs"*, to appear in the Proc. of IEEE PRIME 2022.

One-shot Adaptive Inference

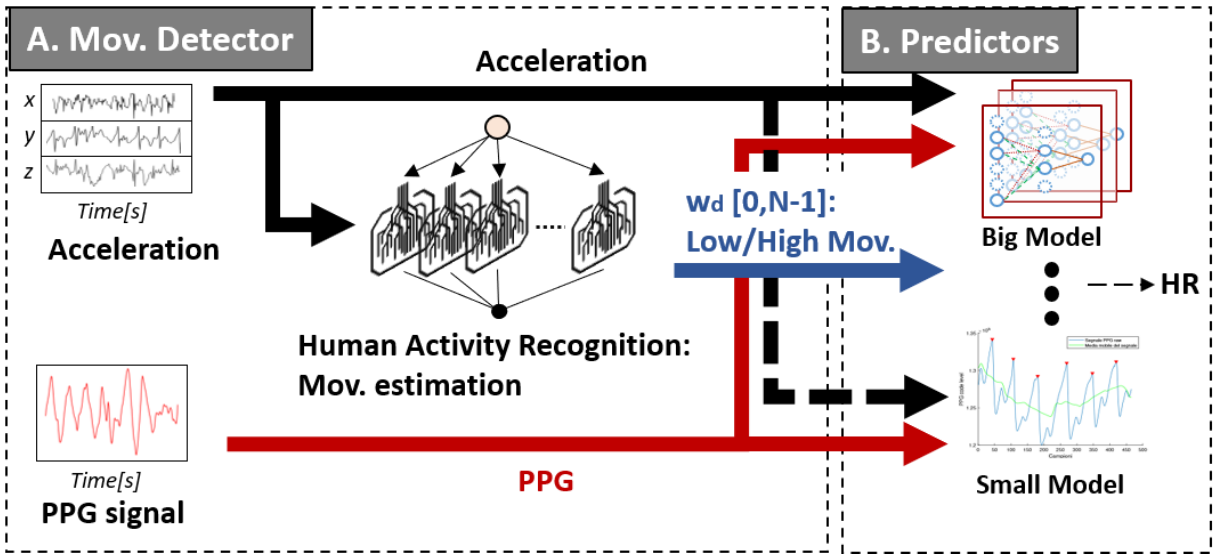
- In some cases, the one-shot adaptive approach is feasible.

- Usually thanks to task-related a priori knowledge



One-Shot Adaptive Inference

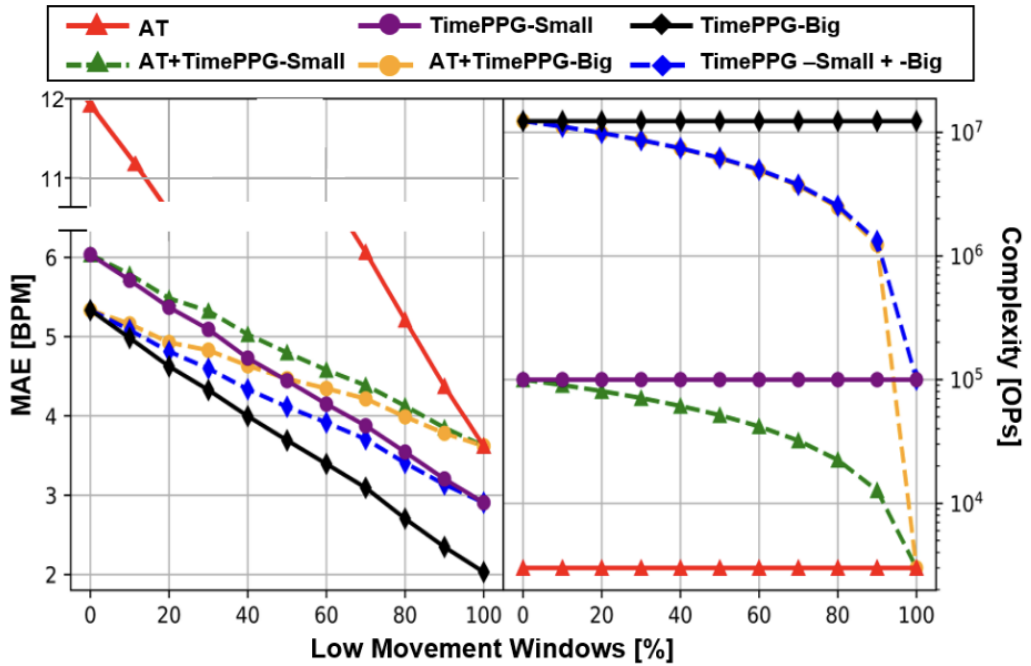
- Example: PPG-based HR monitoring is harder when user is moving (**motion artifacts**)
- Switch between a simple filter-based HR estimator and a DNN based on **accelerometer values**



[23] A. Burrello et al, *"Embedding temporal convolutional networks for energy-efficient PPG-based heart rate monitoring"*, ACM Trans on Computing for Healthcare, 2022.

One-Short Adaptive Inference Results

- E.g., AT + TimePPG-Small: similar perf to TimePPG small, much lower complexity



[23] A. Burrello et al, *"Embedding temporal convolutional networks for energy-efficient PPG-based heart rate monitoring"*, ACM Trans on Computing for Healthcare, 2022.

Politecnico
di Torino

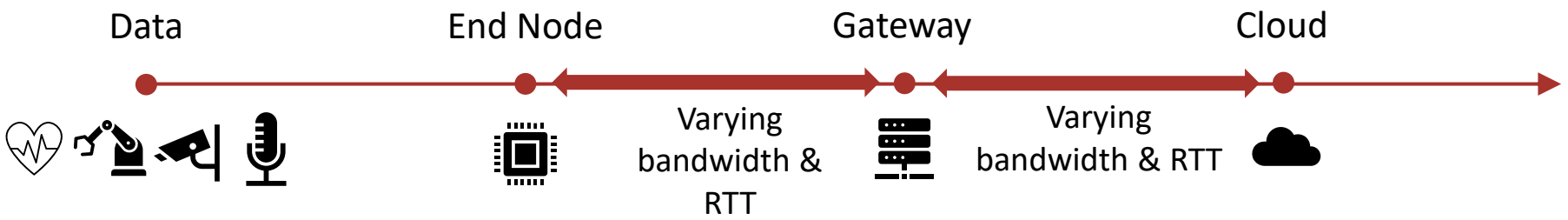
Collaborative Inference

Introduction and Context

- Despite the “theoretical” benefits of pure edge computing…

- In reality, the optimal inference device is **time-dependent,** due to:
  - The varying speed/energy of compute devices
  - The varying speed/energy of TX/RX.

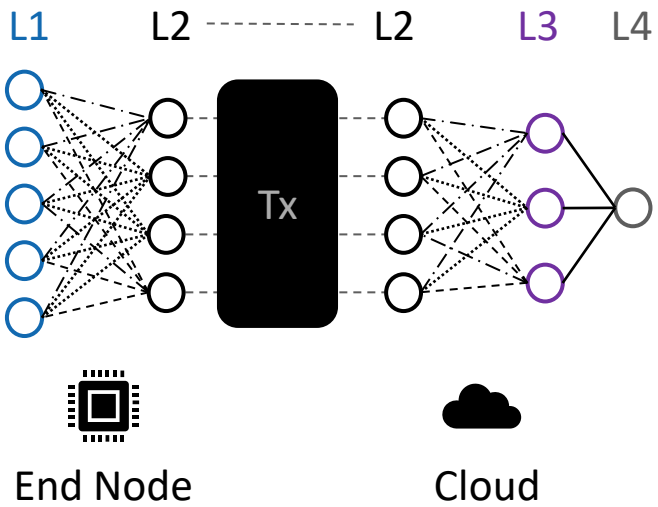
- **Collaborative Inference (CI)** dynamically maps inference tasks on a network of collaborating devices



Collaborative Inference

- Seminal works on CI focus on feed-forward NNs:
  - Y. Kang et al, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge”, Proc. ASPLOS 2017

- Neurosurgeon:
  - Partition the network (e.g., layer-wise)
  - Initial layers on the edge, final layers in the cloud
  - Compressed features → Reduced TX
  - Change partition point dynamically



Collaborative Inference

- Neurosurgeon: motivaiton

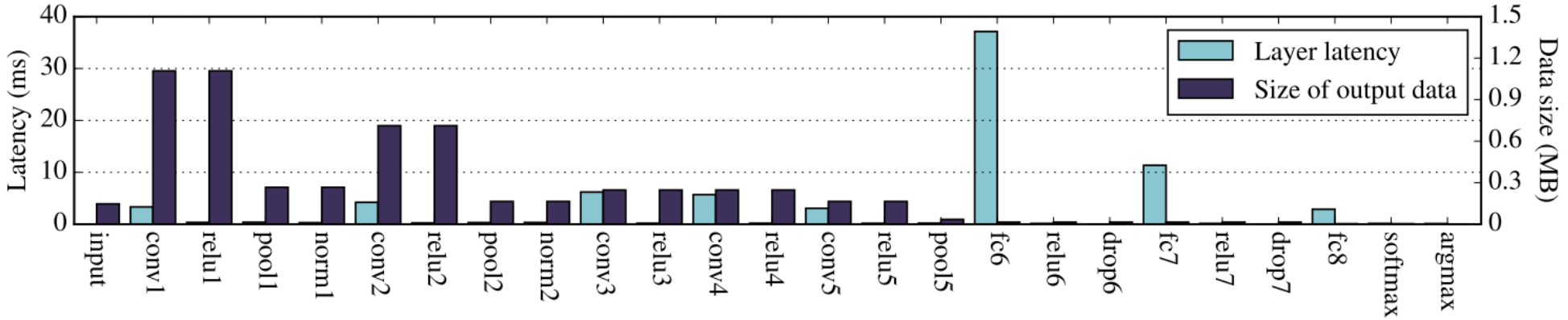
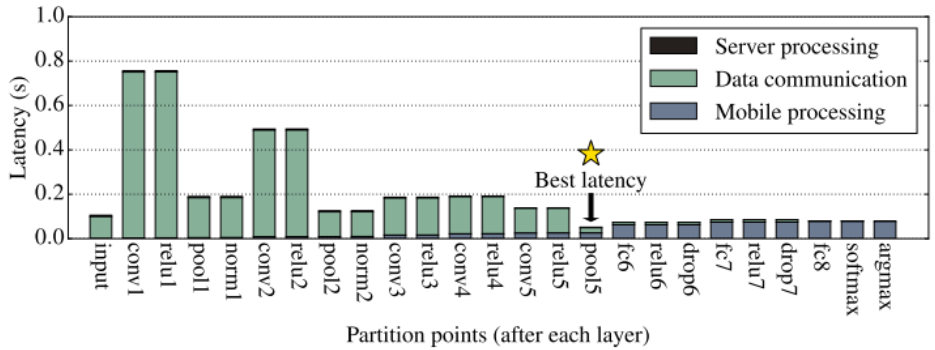
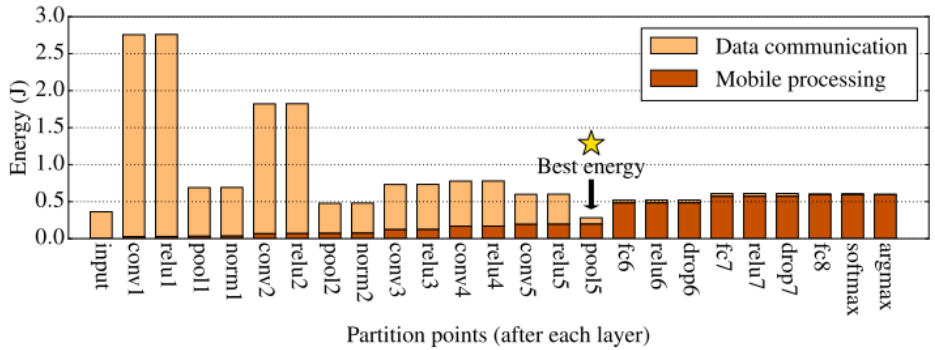


Figure 5: The per layer execution time (the light-colored left bar) and size of data (the dark-colored right bar) after each layer's execution (input for next layer) in AlexNet. Data size sharply increases then decreases while computation generally increases through the network's execution.



(a) AlexNet latency



(b) AlexNet energy consumption

Collaborative Inference

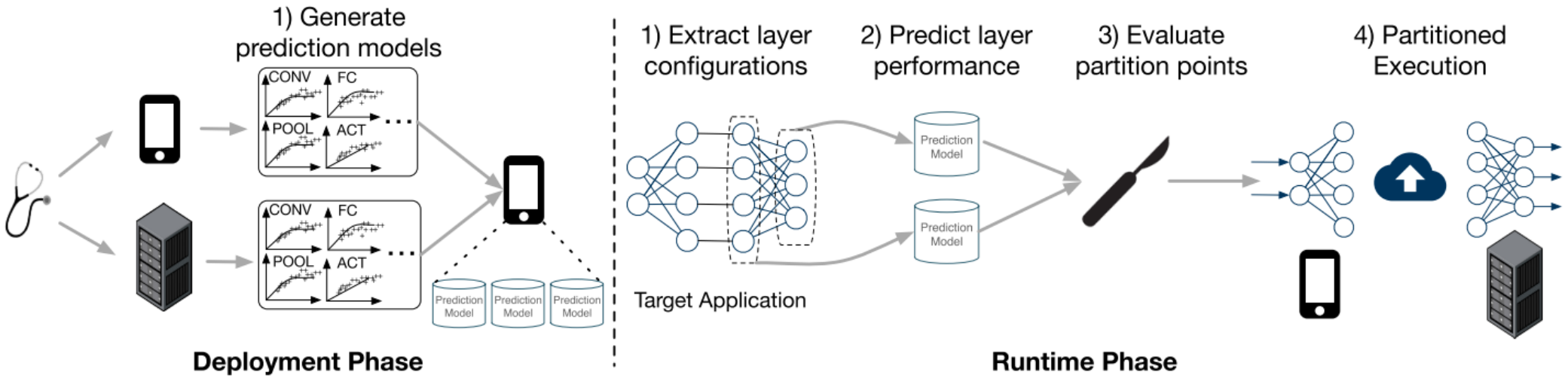
- Neurosurgeon: **Runtime**

  1/2. Evaluate latency/energy for each layer present in the target NN

  3. Consider all possible partition points and select the best.

  4. Partition the execution:

  - Remote Procedure Call (RPC) for mobile/cloud offloading
  - Both devices **host the complete DNN model** (no memory saving)



Collaborative Inference

- **Partition point selection:**
  - Done by the edge node
  - Find **argmin** of sum between:
    - **Local** Latency/Energy for **layers 1:j**
    - **Transmission** Latency/En. for **layer j**
    - **Remote** latency for **layers j:N**

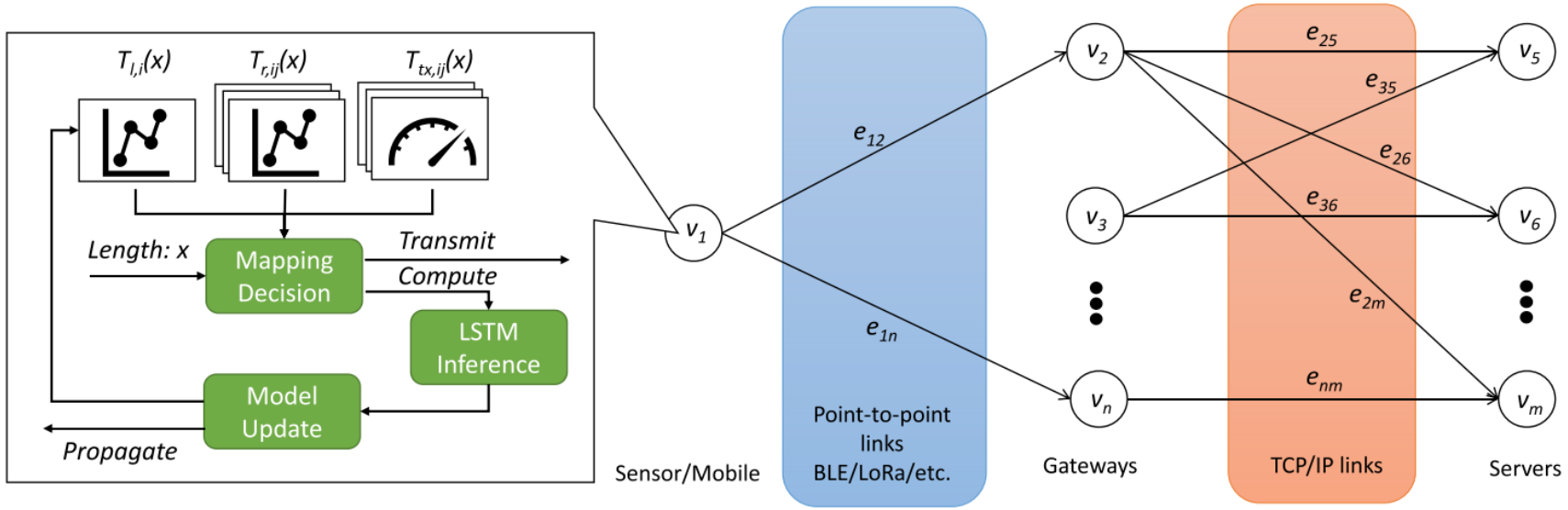
- Remote time depends on server load (K)

- Issues:
  - Estimate server load (K)
  - Estimate transmission time (variable connection speed).

9: **procedure** PARTITIONDECISION
10: **for each** i *in* $1 \cdots N$ **do**
11: $TM_i \leftarrow f_{mobile}(L_i)$
12: $TC_i \leftarrow f_{cloud}(L_i, K)$
13: $PM_i \leftarrow g_{mobile}(L_i)$
14: $TU_i \leftarrow D_i/B$
15: **if** $OptTarget == latency$ **then**
16: **return** $\arg\min_{j=1\cdots N}(\sum_{i=1}^{j} TM_i + \sum_{k=j+1}^{N} TC_k + TU_j)$
17: **else if** $OptTarget == energy$ **then**
18: **return** $\arg\min_{j=1\cdots N}(\sum_{i=1}^{j} TM_i \times PM_i + TU_j \times PU)$

Collaborative Inference for RNNs

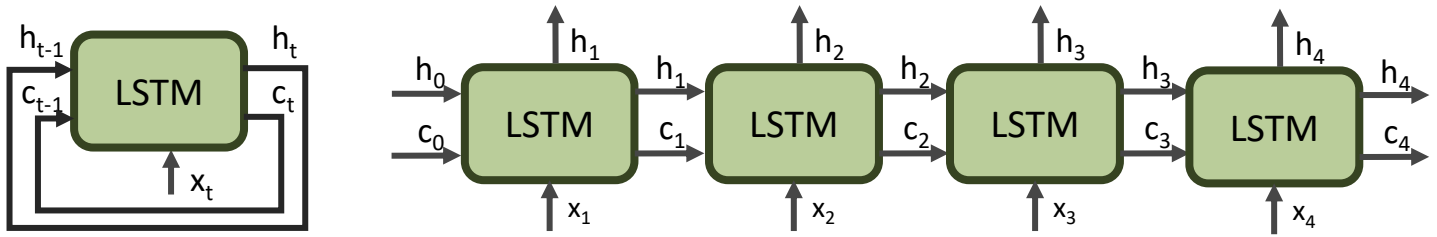
- **CRIME:** Collaborative RNN Inference Mapping Engine
- Map RNN-based inference on a network of edge and cloud devices dynamically



[24] Y. Chen et al, “C-NMT: A Collaborative Inference Framework for Neural Machine Translation”, ISCAS 2022
[25] D. Jahier Pagliari et al, "CRIME: Input-Dependent Collaborative Inference for Recurrent Neural Networks," in IEEE TCOMP 2021
[26] D. Jahier Pagliari et al, “Input-dependent edge-cloud mapping of recurrent neural networks inference”, DAC 2020

CRIME

- RNNs **process data sequences of variable length**.
  - Text, speech, time-series, etc.
  - CI for RNNs requires a different approach
- Example: Long-Short Term Memory (LSTM) RNNs
  - Input at each step t: new datum (x_t) and prev. output (h_t, c_t)
  - At inference time, the NN is **unrolled n times** (n = input length)

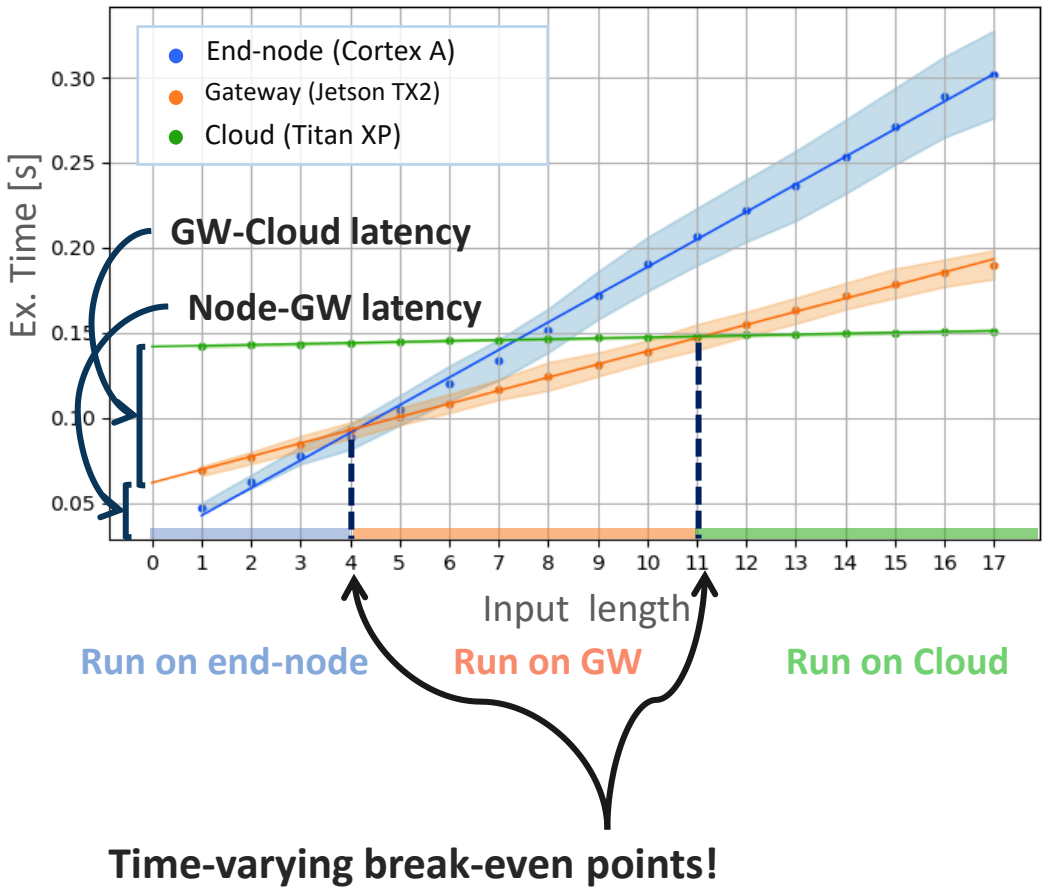


a) LSTM feedback loop

b) Unrolled LSTM for N=4

CRIME

- RNN inputs are smaller than CNNs
  - E.g., NLP, few 100s of bytes per sentence
- Less “compression” at layer outputs
  - No benefit from partitioned execution
- Transmission dominated by Round-trip time (T_{rtt})
  - ≈ fixed latency cost, independent of n
- **Optimal mapping:** short inputs locally, long inputs remotely



The CRIME Framework

- Local execution time models are **linear:** $T_{l,i}(n) = \alpha_{l,i} n + \beta_{l,i}$
  - Simple to store and transmit, low evaluation overhead
- Mapping decision:

1. Find best remote device

$$\hat{\jmath} = \arg \min_{j:e_{ij} \in E} (T_{tx,ij}(n) + T_{r,ij}(n)).$$

2. Select between local and remote

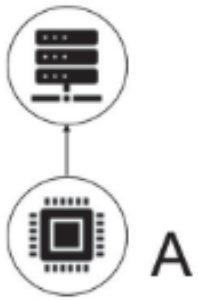
$$v_{target} = \begin{cases} v_i \text{ if } T_{l,i}(n) \leq T_{tx,i\hat{\jmath}}(n) + T_{r,i\hat{\jmath}}(n) \\ v_{\hat{\jmath}} \text{ otherwise} \end{cases}$$

CRIME Results

- Up to 50 % avg. inference time reduction w.r.t. static mapping (edge/cloud)

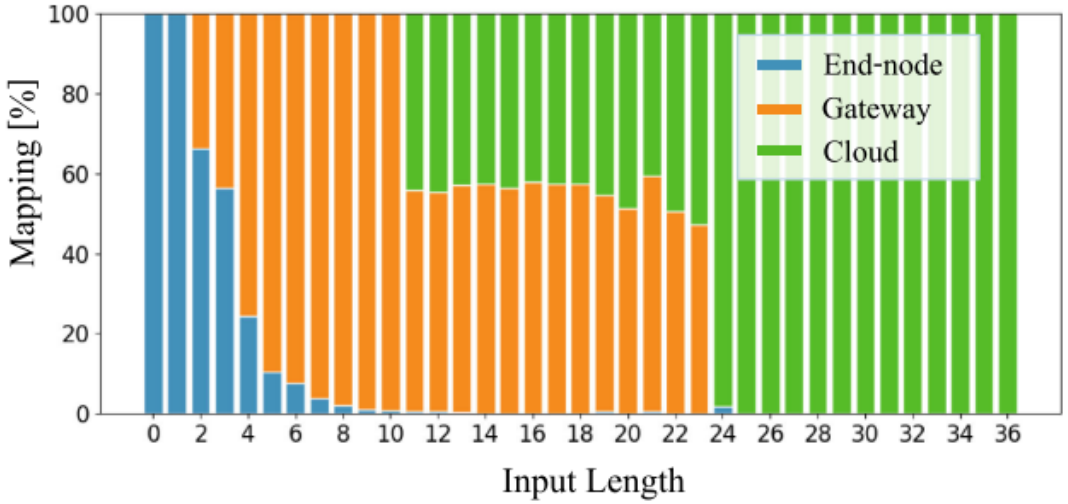
Profile	Test	Ex. timereduction [%] vs end-node	Ex. timereduction [%] vs cloud	Ex. timeincrement [%] vs oracle
Artificial	SNLI	25.66	17.29	0.28
	SQuAD	15.67	25.99	0.40
	$SNLI_{200}$	20.93	7.84	0.18
	IMDB	1.51	23.47	0.01
RIPE Atlas	SNLI	21.35	8.64	0.24
	SQuAD	8.06	15.72	0.28
	$SNLI_{200}$	3.48	36.51	0.032
	IMDB	0.12	56.23	0.002

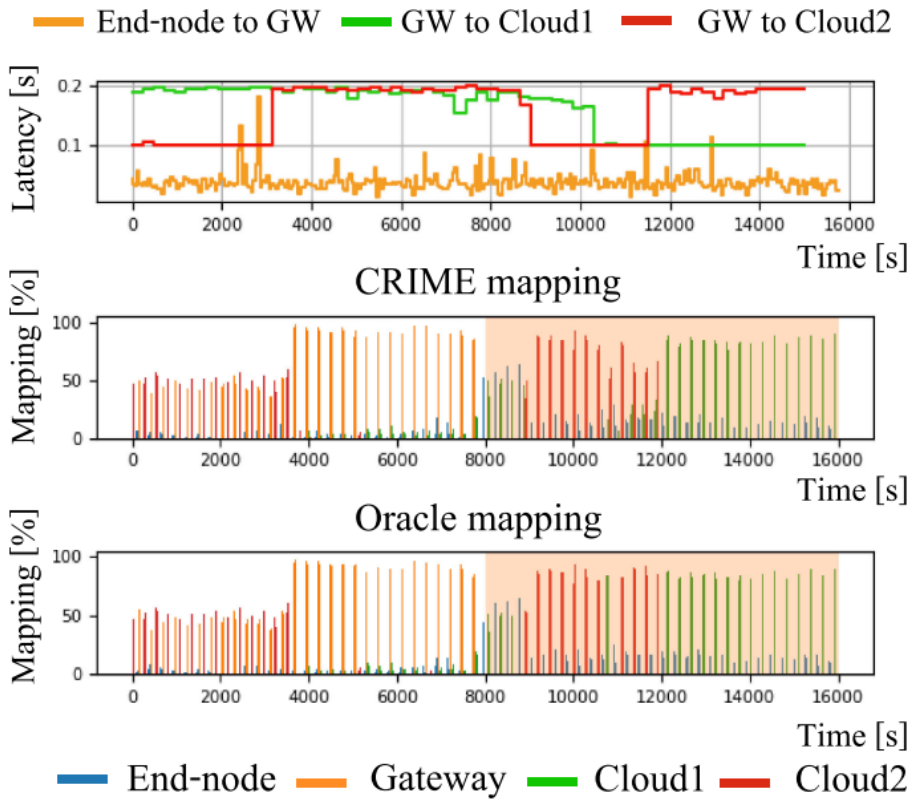
RIPE ATLAS Meas. ID: 1437285, Probe ID: 6222, Date and time: May 3rd 2018, 3-6 p.m.



CRIME Results

- CRIME's dynamic mapping:





References

[1] V. Sze, Y. -H. Chen, T. -J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, 2017

[2] S.Han et al, ”Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015.

[3] Yu, J., Lukefahr, A., Palframan, D., Dasika, G., Das, R., & Mahlke, S. (2017). Scalpel: Customizing dnn pruning to the underlying hardware parallelism. ACM SIGARCH Computer Architecture News, 45(2), 548-560.

[4] Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., & Peste, A. (2021). Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. Journal of Machine Learning Research, 22(241), 1-124.

[5] M. Nagel et al, “A White Paper on Neural Network Quantization, arXiv 2021

[6] Banner, R. et al Post training 4-bit quantization of convolutional networks for rapid-deployment, NeurIPS 2019

[7] Choi et al, PACT: Parameterized Clipping Activation for Quantized Neural Networks, CVPR 2018.

[8] Esser et al, Learned Step Size Quantization, ICLR 2020

[9] Z. Cai et al, Rethinking Differentiable Search for Mixed-Precision Neural Networks, CVPR 2020

[10] Rastegari, M., et al.. Xnor-net: Imagenet classification using binary convolutional neural networks. In European conference on computer vision. Cham: Springer International Publishing, 2016.

[11] Lee, E. H., et al., Lognet: Energy-efficient neural networks using logarithmic computation. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

[12] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

[13] H. Tann et al, “Runtime configurable deep neural networks for energy-accuracy trade-off”. Proc. IEEE/ACM/IFIP CODES, 2016

References

[14] Yu, J., Yang, L., Xu, N., Yang, J., & Huang, T. (2018). Slimmable neural networks. arXiv preprint arXiv:1812.08928.
[15] F. Daghero et al, “Human Activity Recognition on Microcontrollers with Quantized and Adaptive Deep Neural Networks”, ACM Trans. on Embedded Systems, 2022.
[16] P. Panda et al. Conditional Deep Learning for Energy-Efficient and Enhanced Pattern Recognition.
[17] Shallow-Deep Networks: Understanding and Mitigating Network Overthinking, 2018.
[18] Wang, Xin, et al. "Skipnet: Learning dynamic routing in convolutional networks." Proceedings of the European conference on computer vision (ECCV). 2018.
[19] Gao, Xitong, et al. "Dynamic channel pruning: Feature boosting and suppression." arXiv preprint arXiv:1810.05331 (2018).
[20] D. Jahier Pagliari et al, “Dynamic bit-width reconfiguration for energy-efficient deep learning hardware”, Proc. ISLPED 2018.
[21] V. Peluso et al, ”Energy-Accuracy Scalable Deep Convolutional Neural Networks: A Pareto Analysis”
[22] F. Daghero et al, “Two-stage Human Activity Recognition on Microcontrollers with Decision Trees and CNNs”, to appear in the Proc. of IEEE PRIME 2022
[23] A. Burrello et al, “Embedding temporal convolutional networks for energy-efficient PPG-based heart rate monitoring”, ACM Trans on Computing for Healthcare, 2022
[24] Y. Chen et al, “C-NMT: A Collaborative Inference Framework for Neural Machine Translation”, ISCAS 2022
[25] D. Jahier Pagliari et al, "CRIME: Input-Dependent Collaborative Inference for Recurrent Neural Networks," in IEEE TCOMP 2021
[26] D. Jahier Pagliari et al, “Input-dependent edge-cloud mapping of recurrent neural networks inference”, DAC 2020

PULP Platform

Open Source Hardware, the way it should be!

Alessio Burrello alessio.burrello@polito.it
Daniele Jahier Pagliari daniele.jahier@polito.it

DAUIN – Politecnico di Torino
Corso Duca degli Abruzzi 24
Torino, Italy

Institut für Integrierte Systeme – ETH Zürich
Gloriastrasse 35
Zürich, Switzerland

DEI – Università di Bologna
Viale del Risorgimento 2
Bologna, Italy

@pulp_platform

pulp-platform.org

youtube.com/pulp_platform

ETH zürich

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA