IWES 2018
Energy-Efficient Deep Learning

Valentino Peluso

POLITECNICO DI TORINO

Department of Computer Engineering

Deep Learning at the edge of the IoT

Data-Analytics in-the-Cloud

High data volume

High communication latency

Huge capacity storage

In-situ Data-Analytics

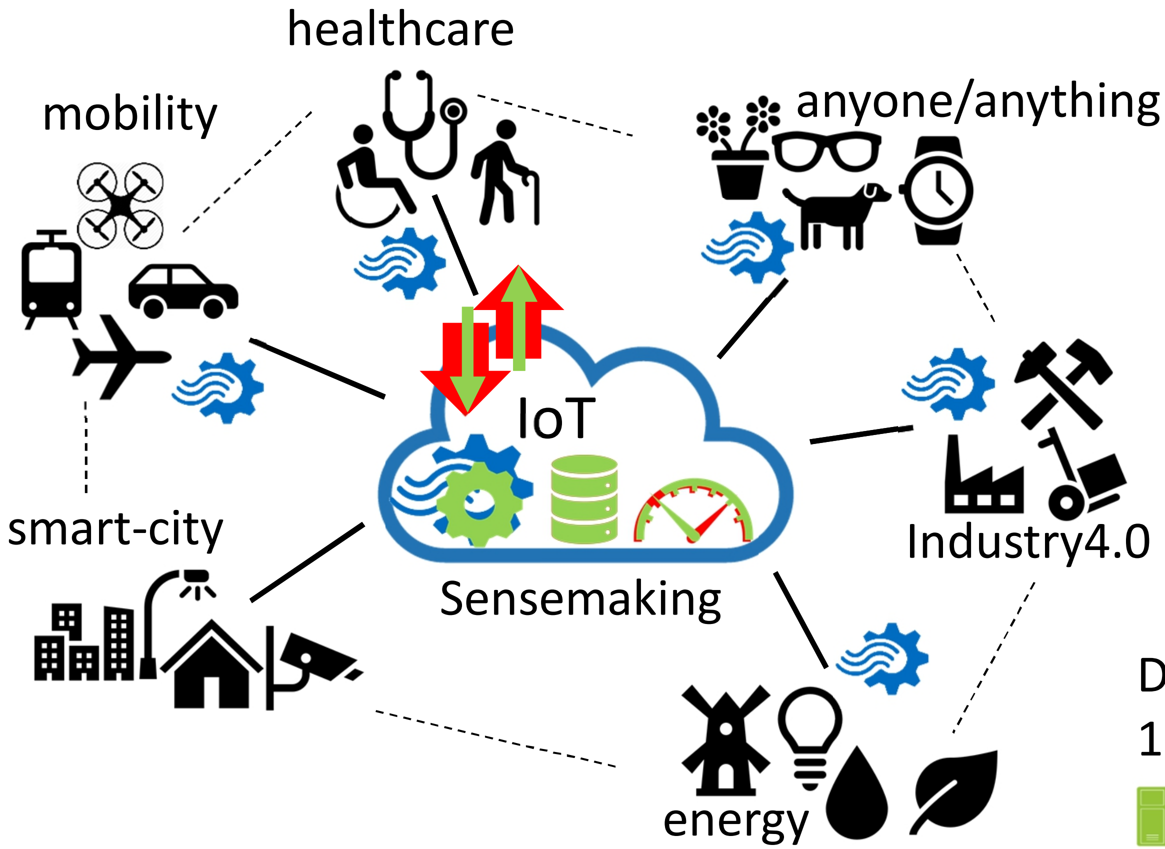
High information throughput

Low data volume

Low storage and network usage

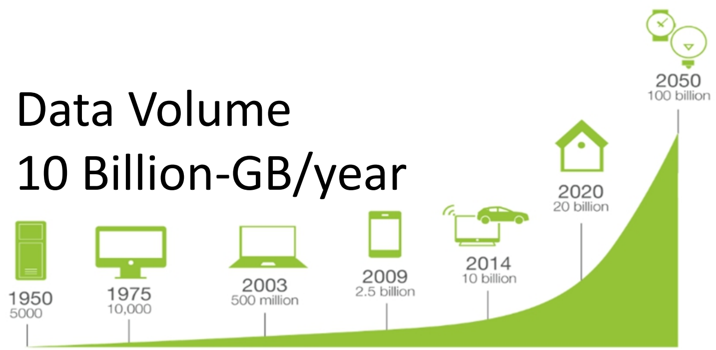
Real-Time M2M

Privacy



Scalability?

Data Volume
10 Billion-GB/year

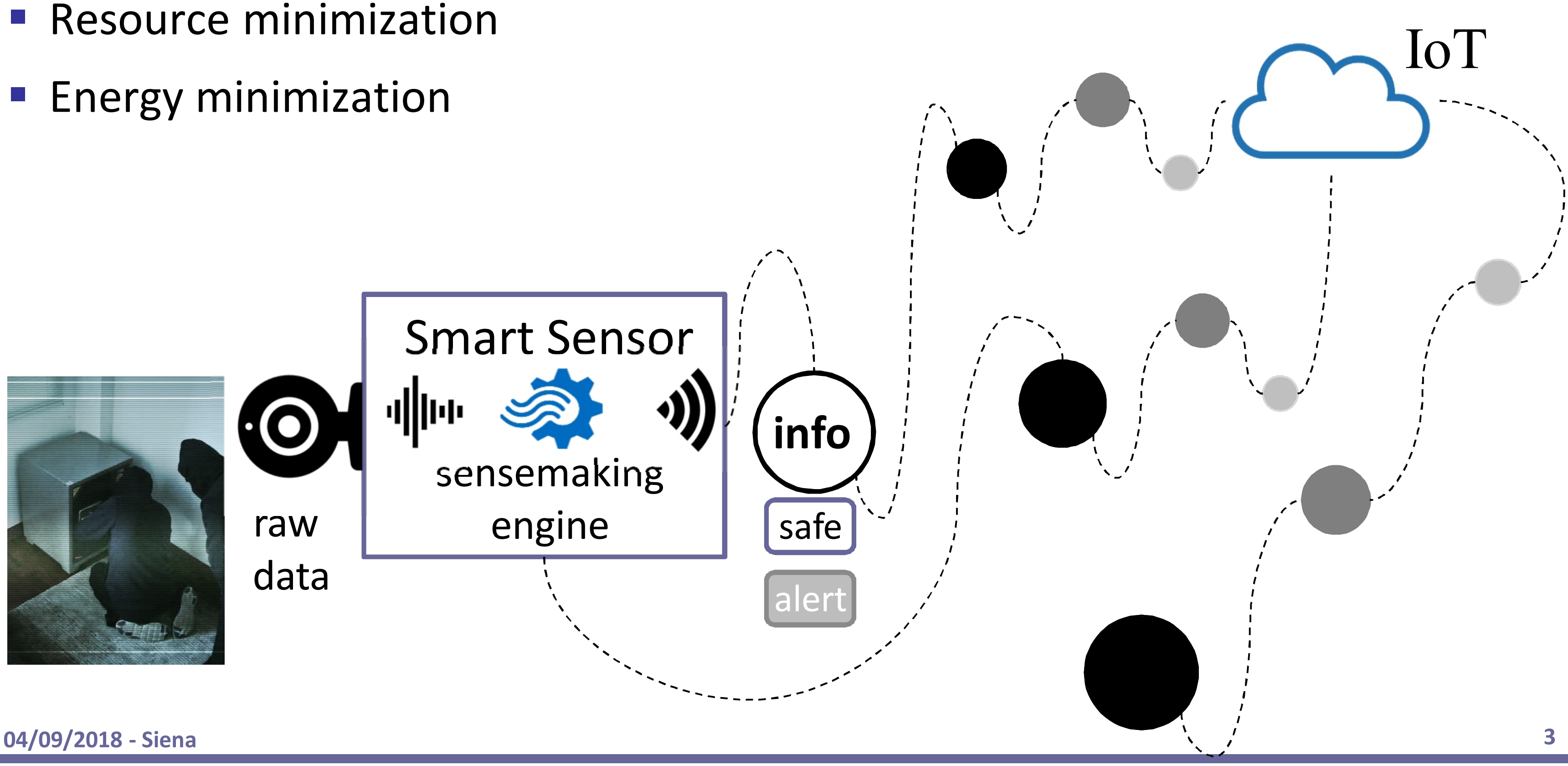


Bring CNNs at the edge of the IoT

- **Data-flow computing on embedded platforms**
  - Ultra-low power architecture
  - Reconfigurable architecture (host different CNNs)

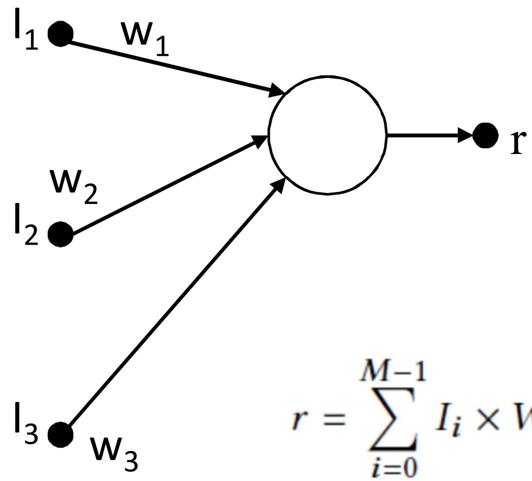
SENSEI

- **Optimal software-to-silicon mapping (design automation)**
  - Resource minimization
  - Energy minimization



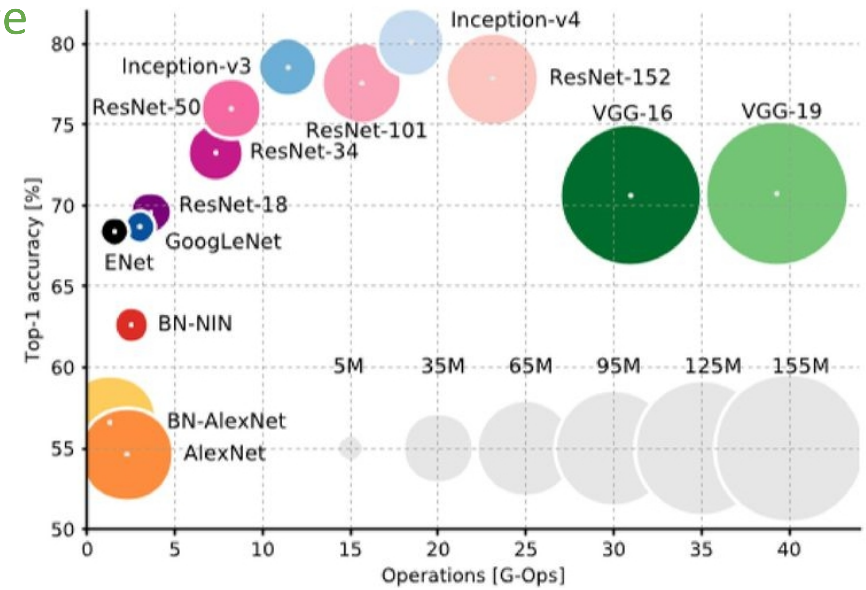
Fixed-Point CNNs

- **Fixed-point (INT) arithmetic instead of floating-point (FP32) is a well established strategy for inference**
  - FP32: ±S 1.M x 2^{EXP} [S=1b, M=23b, Exp=8b]
  - INT16: I.Q [8b.8b]



$$r = \sum_{i=0}^{M-1} I_i \times W_i$$

- **Fixed-point arithmetic**
  - require less memory
    - →fit in resource-constrained HW
  - minimize memory bandwidth usage
    - →leverage SIMD data-paths
  - achieve lower accuracy
    - reduced range
    - reduced precision



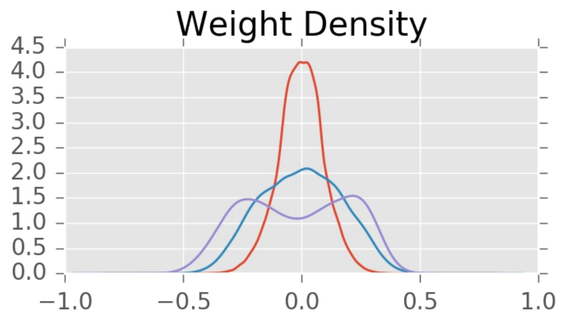
How to choose optimal precision

- **State-of-the-art: accuracy driven**

  →Need to consider also energy

1. **Input-dependant**
   - Dynamic bit-widith reconfiguration

2. **Network-dependant**
   - Different DNNs show different tolerance
     - Topology (Size, Depth, #Kernels)
     - Trainable parameters (sparsity)
   - Per-net quantization
     - Same precision for each and every layer
   - Per-layer quantization
     - Exploit the dynamic range of activations and weights across different layers



Energy-Efficient Precision Scaling for ConvNets

- **Broad Objective:**
  - Enable Dynamic Energy-Accuracy Scaling
    - Run-time adaptation according
      - application requirements and/or the context
      - available energy budget
- **Knobs:**
  - Arithmetic precision
- **Key feature:**
  - No Retraining
    - Costly
    - Training data may be not always available
  - Make use of a single weight-model rather than multiple models

Outline

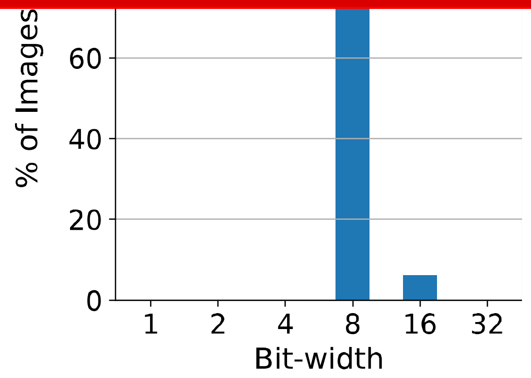
- Dynamic Bit-width Reconfiguration
- Energy-Aware Precision Scaling for ConvNets

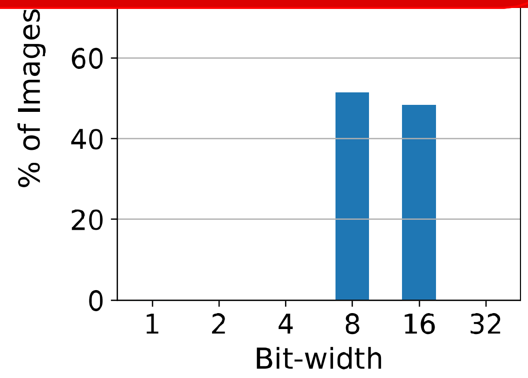
Motivation

- Idea: not all inputs are **equally hard to classify**.
- Using the **same fixed-point bit-width** for all inputs may be sub-optimal.
- **Example:**

assifying

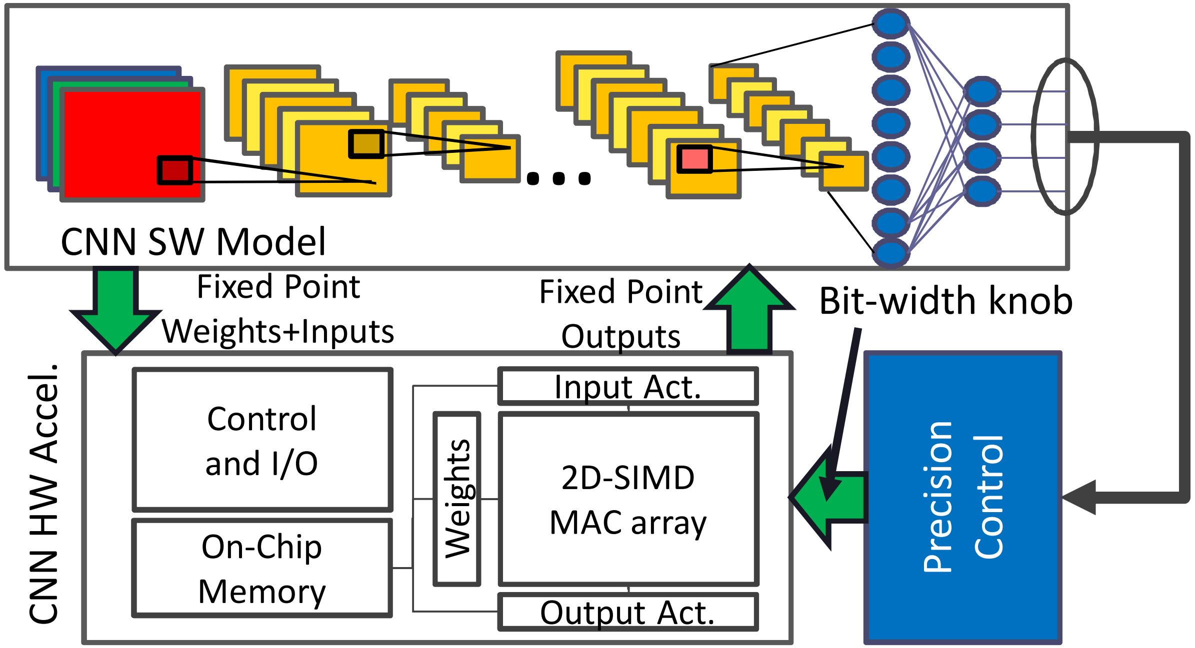
Consequence: static bit-width solutions require a costly retraining to (partially or totally) restore accuracy!





Proposed Approach

- Automatically **adapt the bit-width to the current input**.
- **Reduce energy consumption** compared to a *conservative* static bit-width (e.g. 16-bit in the previous two examples).
- **Increase accuracy** compared to lower precision (e.g. 8-bit)
- **General Framework:**

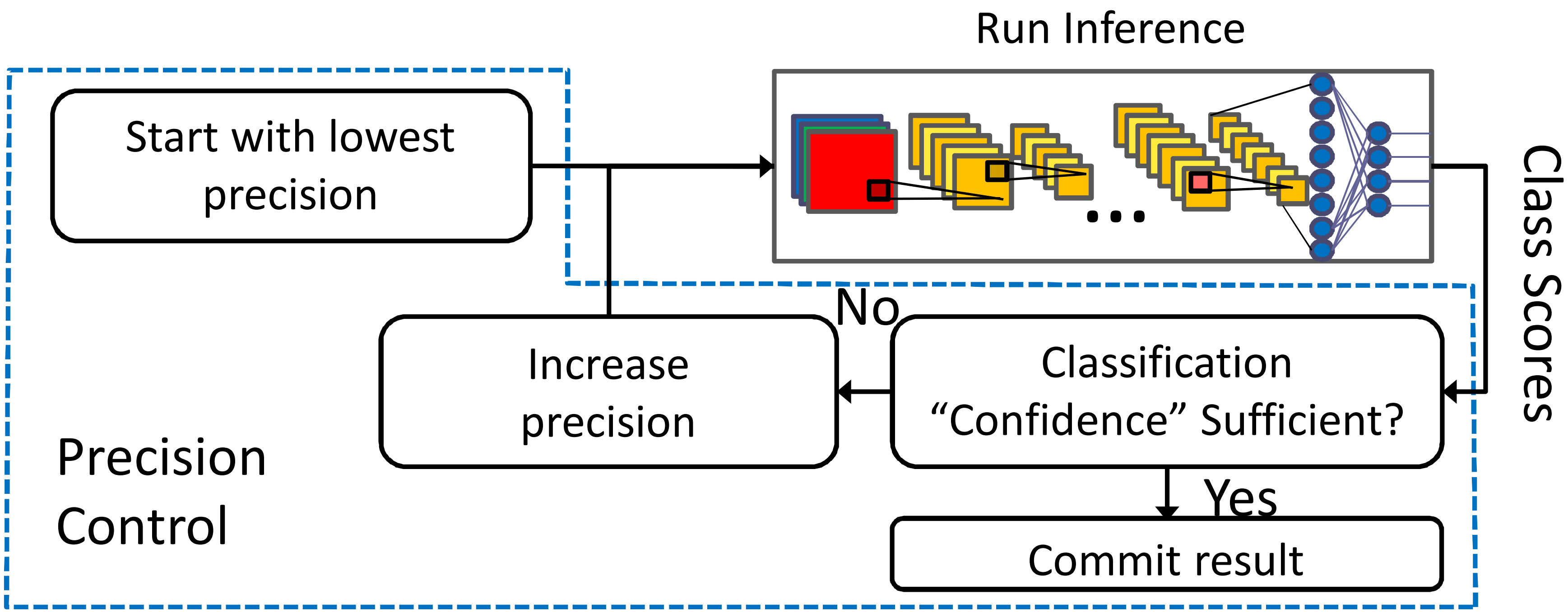


Implementation Details

- **Data format: Dynamic Fixed Point**
  - **Integer bits:** fixed, determined from variation ranges of activations and weights in each layer
  - **Fractional bits:** change depending on overall bit-width (may be negative)

- **Advantage: No storage duplication.**
  - Weights store at <u>maximum precision</u>, precision is reduced by LSB-truncation.

Runtime Bit-width Tuning



Classification "Confidence" Estimation

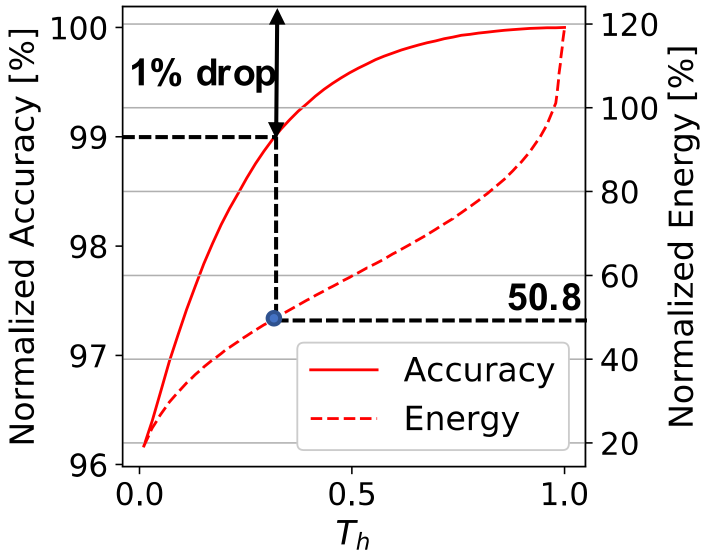
- Uses the **Score Margin (SM)**

$$SM = P(y_i|x) - P(y_j|x)$$

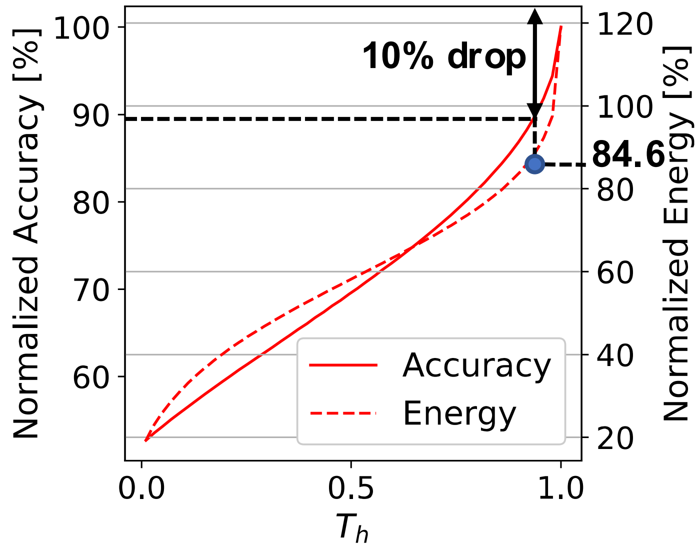
  - $P(y_i|x)$ = output probability of the selected class
  - $P(y_j|x)$ = second largest probability.
- **Idea:**
  - Large difference → only one class is probable
  - Small difference → "uncertain" between at least two classes.
- **Decision Strategy:**
  - Compare SM with a **Threshold (*Th*)**.
  - Changing the threshold allows exploring the **energy** (i.e. number of inferences per image) **versus accuracy trade-off**.

Results

- **Target CNNs:** <u>CaffeNet</u> *(*i.e. Caffe's AlexNet, *Krizhevsky et al., NIPS2012)* and <u>CNN-M</u> *(Chatfield et al, arXiv2014)*
  - Same task: <u>ImageNet</u> classification.
- **Target Accelerator:** Envisions *(Moons et al., ISSCC2017)*
- Different networks yield very different tradeoffs:



CaffeNet



CNN-M

Baseline: Static 16-bit Top-1 Accuracy

Outline

- Dynamic Bit-width Reconfiguration
- Energy-Aware Precision Scaling for ConvNets

Per-net assignment

- **Four options:**
  - 16b activations, 16b weights (16x16)
  - 8b activations, 16b weights (8x16)
  - 16b activations, 8b weights (16x8)
  - 8b activations, 8b weights (8x8)
- **Unconstrained assignment**

	CNN-1	CNN-2	CNN-3
ork Architecture	input 32×32×3 conv 3×3×32 conv 3×3×64 maxpool dense 128 dense 10 softmax	input 32×32×3 conv 3×3×32 conv 3×3×32 maxpool conv 3×3×64 conv 3×3×64 maxpool dense 512 dense 10 softmax	input 32×32×3 conv 3×3×32 conv 3×3×32 maxpool conv 3×3×64 conv 3×3×64 maxpool conv 3×3×128 conv 3×3×128 maxpool dense 1024 dense 512 dense 10 softmax
		CIFAR-10	CIFAR-100
		20 828 426	25 718 538

Need for a finer precision assignment
→ cross-layer optimization

	32-bit FP	16×16 Fix	8×16 Fix	16×8 Fix	8×8 Fix
CNN-1	90.02%	90.01%	89.66%	76.64%	74.54%
CNN-2	81.77%	81.75%	80.99%	77.80%	72.17%
CNN-3	55.80%	55.82%	52.79%	22.27%	14.69%

Optimization Flow

- FP-32 pretrained model
- Calibration Set (5000k samples)

- HW energy model
- Accuracy Constraint
- Calibration Set (5000k samples)

- Test Set

Fixed-Point Conversion (INT16) → **Per-Layer Precision Assignment** → **Model Evaluation**

- Per-layer radix point scaling based on range analysis
- GPU-accelerated emulation tool fully integrated in TensorFlow

- Energy Aware
- Based on Simulated Annealing

Results

- **Accuracy Results**

Max. Drop λ_{max}	1%	5%	10%
CNN-1	89.69% (0.35%)	86.82% (3.55%)	86.82% (3.55%)
CNN-2	80.76% (1.21%)	77.77% (4.87%)	75.57% (7.56%)
CNN-3	55.20% (1.11%)	54.01% (3.24%)	51.54% (7.67%)

- **Per-layer Precision Scaling**

CNN-1			CNN-2			CNN-3		
1%	5%	10%	1%	5%	10%	1%	5%	10%
8×16	8×8	8×8	8×16	8×8	8×8	16×16	8×16	8×16
8×16	8×8	8×8	8×16	8×16	8×8	16×16	8×16	8×16
8×16	8×16	8×16	8×8	8×8	8×8	16×16	8×16	8×16
8×8	8×8	8×8	8×8	8×8	8×8	8×16	8×16	8×16
8	8	8	8×16	8×8	8×8	8×16	8×16	8×16
			16×8	8×16	8×8	16×16	8×16	8×8
			16	16	16	8×16	8×16	8×8
						8×8	8×8	8×8
						16×8	8×8	8×8
						16	16	16

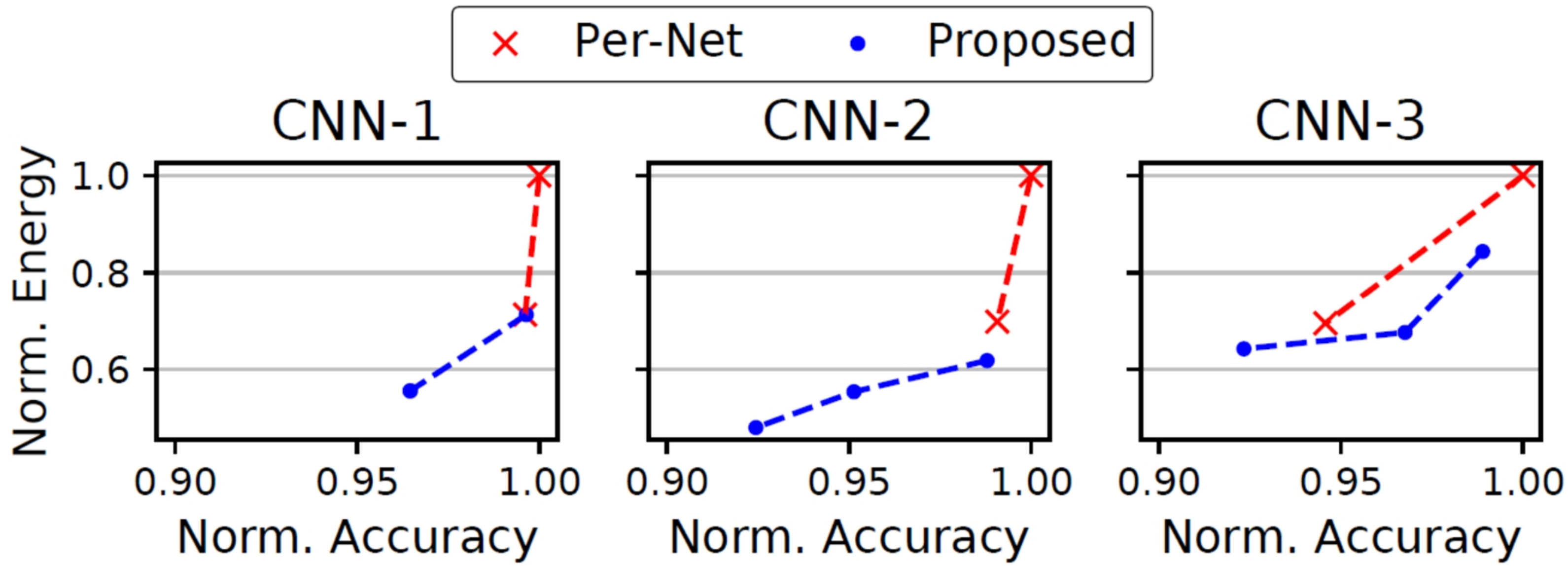
Comparison and Energy Savings

Per-Net

	32-bit FP	16×16 Fix	8×16 Fix	16×8 Fix	8×8 Fix
CNN-1	90.02%	90.01%	89.66%	76.64%	74.54%
CNN-2	81.77%	81.75%	80.99%	77.80%	72.17%
				7%	14.69%

10%
86.82% (3.55%)
75.57% (7.56%)
51.54% (7.67%)

Extended Pareto-points
→ dynamic adaptation to accuracy constraint or energy budget



Conclusion

- **Dynamic Energy Accuracy Scaling**
  - Allows to weigh the importance of accuracy vs energy depending on application/context
  - Especially interesting when retraining is not an option

- **Future works**
  - Combination of two techniques

- **Question?**

Dyn. Bit-width Reconfig. Results (2)

- **Comparison with other input-dependent strategies (on CaffeNet):**

Method	Energy Saving	Top-1 Drop	Multiple CNNs	Multiple Trainings
Park et al, CODES2015	53.7%	0.9%	Yes	Yes
Tann et al, CODES2016	32.61%	0.29%	No	Yes
This work	**49.2%**	**0.89%**	**No**	**No**

- **Warning:** different underlying hardware!

HW energy model

- **Computation Energy**

$$E^{comp} = \sum_{i=1}^{L} E^{\mathrm{MAC}} N_{cycles}(x_i) \cdot N_i^{\mathrm{MAC}} + E^{\mathrm{zero}} \cdot N_i^{\mathrm{zero}}$$

8x8 MAC energy

depends on precision

zero-skipping logic

- **Memory Energy**

$$E^{mem} = \sum_{i=1}^{2 \cdot L+1} E^{\mathrm{MAC}} \cdot [\alpha_i(x_i) + \beta_i(x_i) + \gamma_i(x_i)]$$

input activations

weights

output activations

Simulated Annealing

Algorithm 2: Simulated Annealing

Input: T_0, T_f, X_0, *cooling*, K_b, *iter*, λ_{max}, *valset*
Output: X

T = T_0
E = energy(X_0)
E_{max} = *energy* (*ones* $(2L + 1)$); E_{min} = *energy* (*zeros* $(2L + 1)$)
while $(T \geq T_f)$ **do**
    for *i = 0; i < iter; i = i+1* **do**
        next_state = move (current_state)
        if *accuracy_drop(next_state, valset, tested)* $< \lambda_{max}$ **then**
            E_next = *energy* (next_state)
            ΔE = (E_next-E_current) / (E_{max} - E_{min})
            if *(dE < 0) or (exp[-$\Delta E/K_b \cdot T$] > random(0, 1))* **then**
                current_state = next_state
                E_current = E_new
                if *E_current < E_best* **then**
                    E_best = E_current
                    best_state = current_state
                end
        update(tested)
    end
    T = T $\cdot$ *cooling*
end
return *best_state*