Designing Low Power Computer Vision Systems for Mobile Applications

Jianping Fan
Department of Computer Science
UNC-Charlotte

Course Website:
http://webpages.uncc.edu/jfan/itcs5152.html

Motivations for Designing Low Power Computer Vision Systems

Low-Power Computer Vision

Why Low-Power? Many systems are powered by batteries

Why now? Computer vision is becoming practical

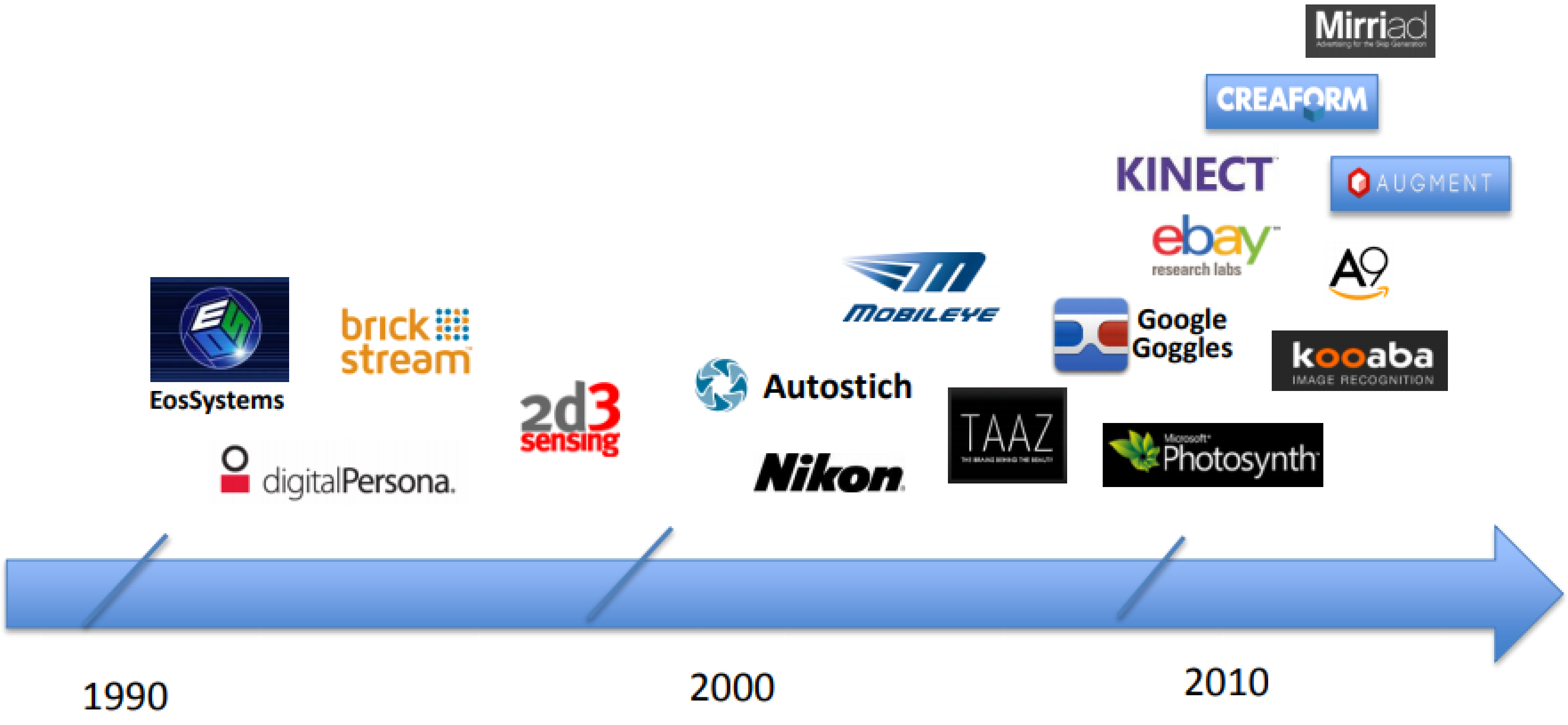
Why competition? Competitions excite people, set concrete goals, compare different solutions

Yung-Hsiang Lu, Low-power Computer Vision: Status, Challenges, Opportunities, CVPR 2019

Motivations for Designing Low Power Computer Vision Systems

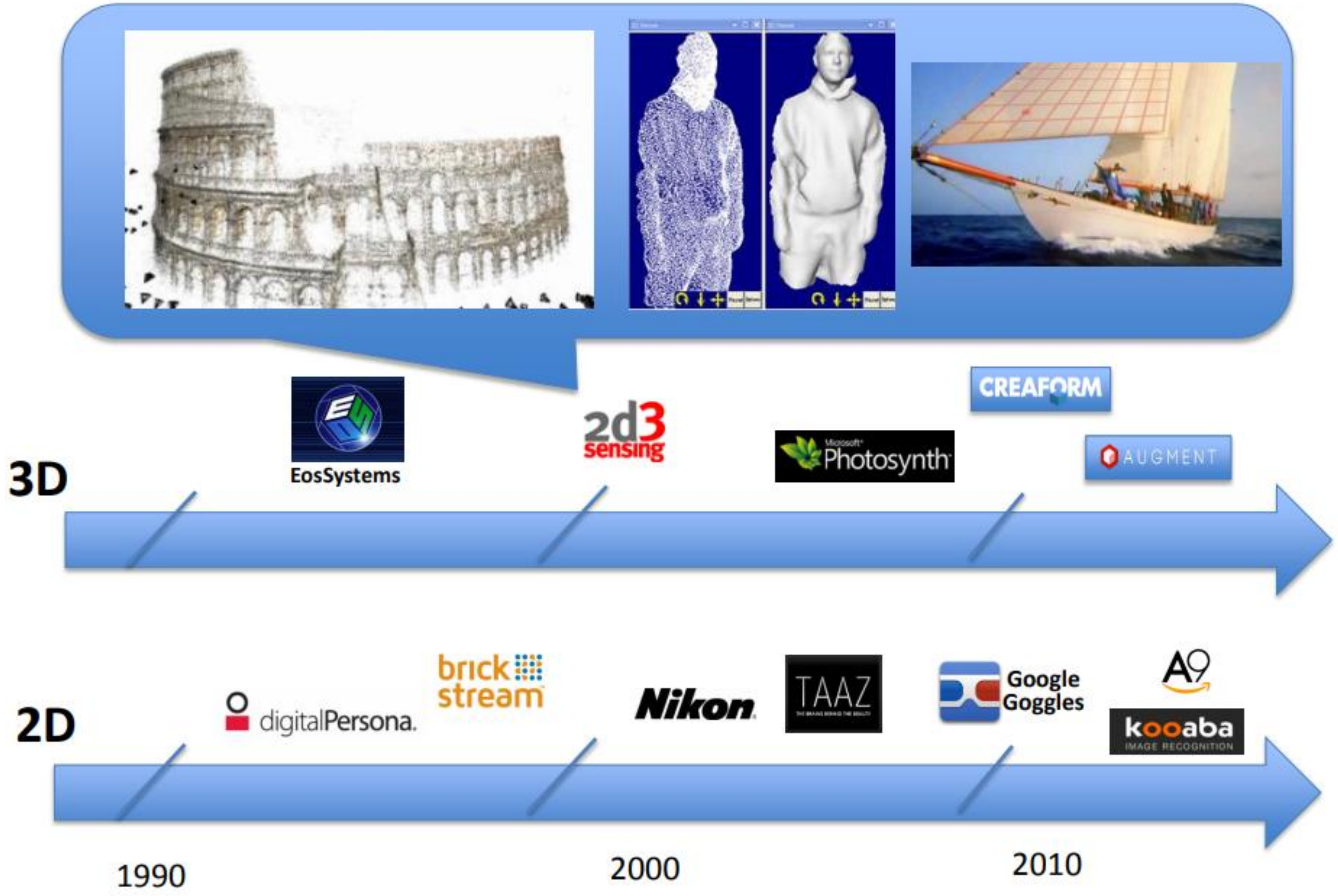
Motivations for Designing Low Power Computer Vision Systems

Computer vision and Mobile Applications



Motivations for Designing Low Power Computer Vision Systems

Motivations for Designing Low Power Computer Vision Systems



Motivations for Designing Low Power Computer Vision Systems

Current state of computer vision

3D Reconstruction

- 3D shape recovery
- 3D scene reconstruction
- Camera localization
- Pose estimation

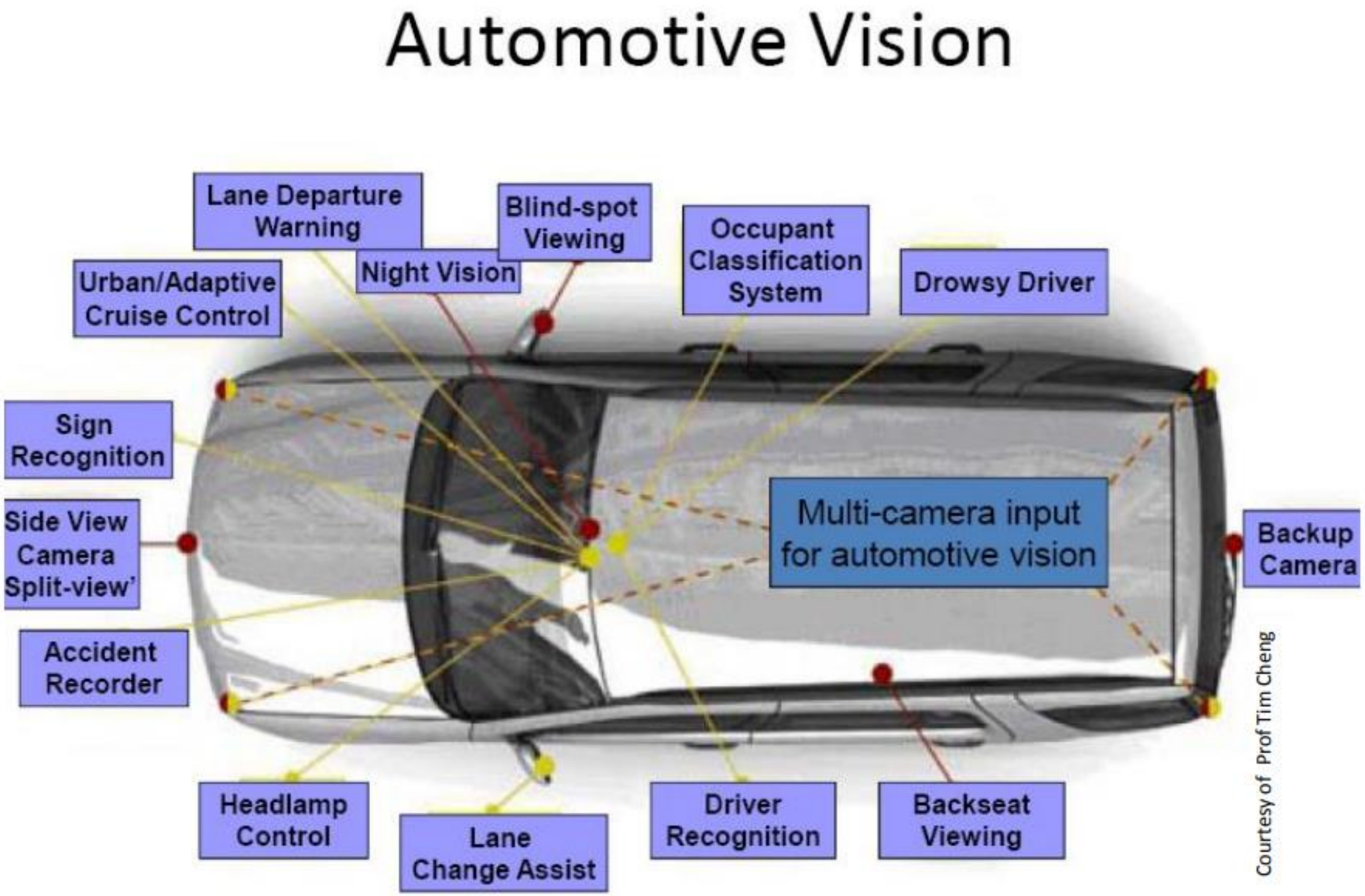
2D Recognition

- Image matching
- Object detection
- Texture classification
- Activity recognition

Mobile computer vision

40

Motivations for Designing Low Power Computer Vision Systems



Motivations for Designing Low Power Computer Vision Systems

- Video-assisted robots
- Medical imaging devices
- Autonomous cars
- Smart phones
- Tablets
- Glasses

Motivations for Designing Low Power Computer Vision Systems

Smart phones & tablets:
common characteristics

- Low cost
- Small package
- Resource constraints
- Real-time constraints (for some systems/applications)

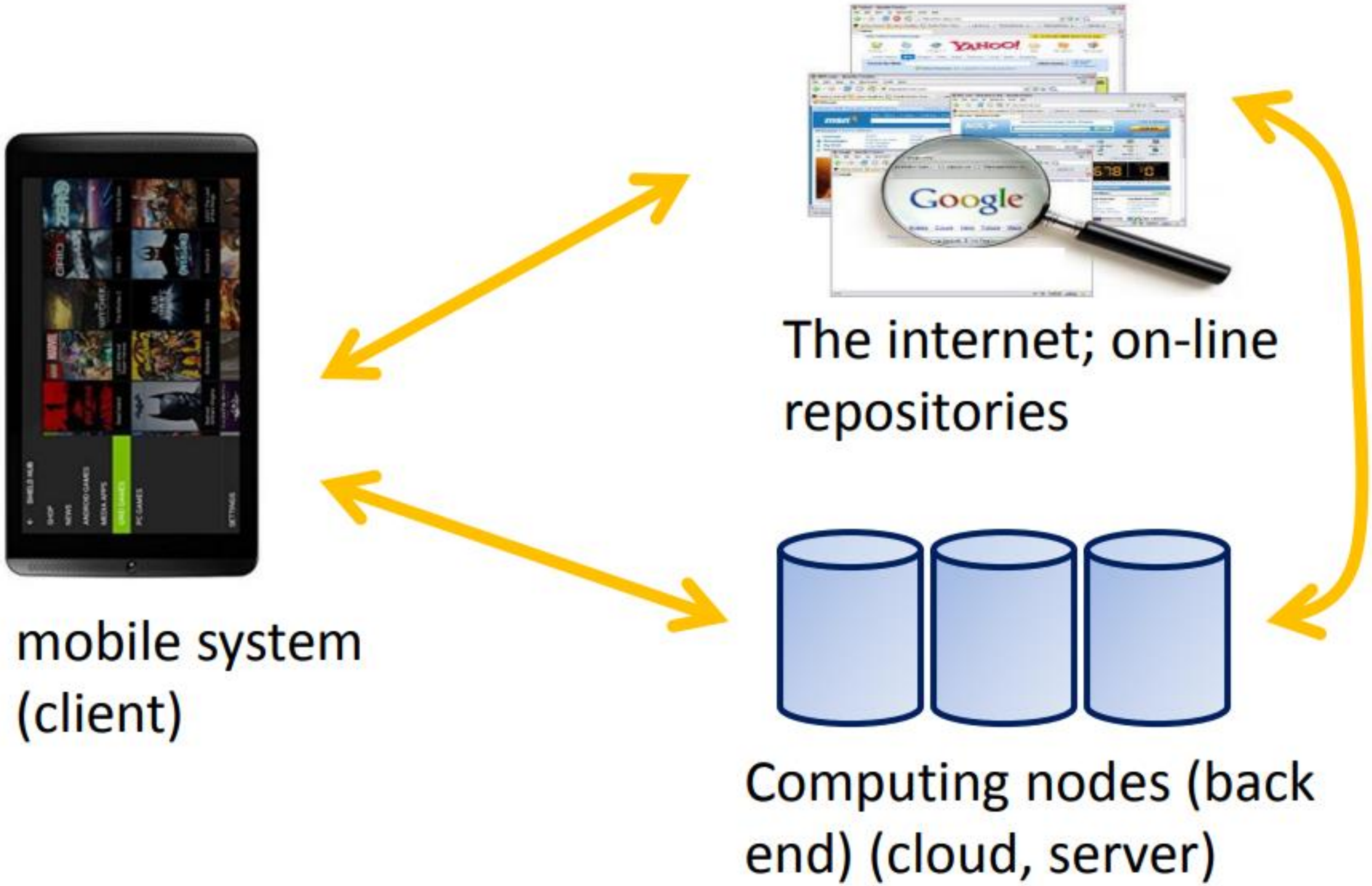
Motivations for Designing Low Power Computer Vision Systems

Challenges

- The mobile system has limited:
  computational power - bandwidth - memory
  - What to compute on the client (features, tracks)
  - How much data must be transferred to the back end
  - What to compute on the back end
  - How much data must be transferred back to the client to visualize results
- Computer vision algorithms with
  - Guaranteed (high)accuracy
  - Efficient (use little computational power/memory)
  - Fast (possibly real time)
- Many of these CV problems are still open

Motivations for Designing Low Power Computer Vision Systems

Client and Server paradigm



Requirements of computer vision systems for mobile applications:

(a) Lightweight deep networks because of limited storage and computational abilities;

(b) Real-time inferencing;

(c) Learning from few samples;

(d) Without keeping history samples: learning without using history samples;

1. Mixed Environment with Diverse Computational Abilities

Edge ← Applications → Cloud

Speed of response

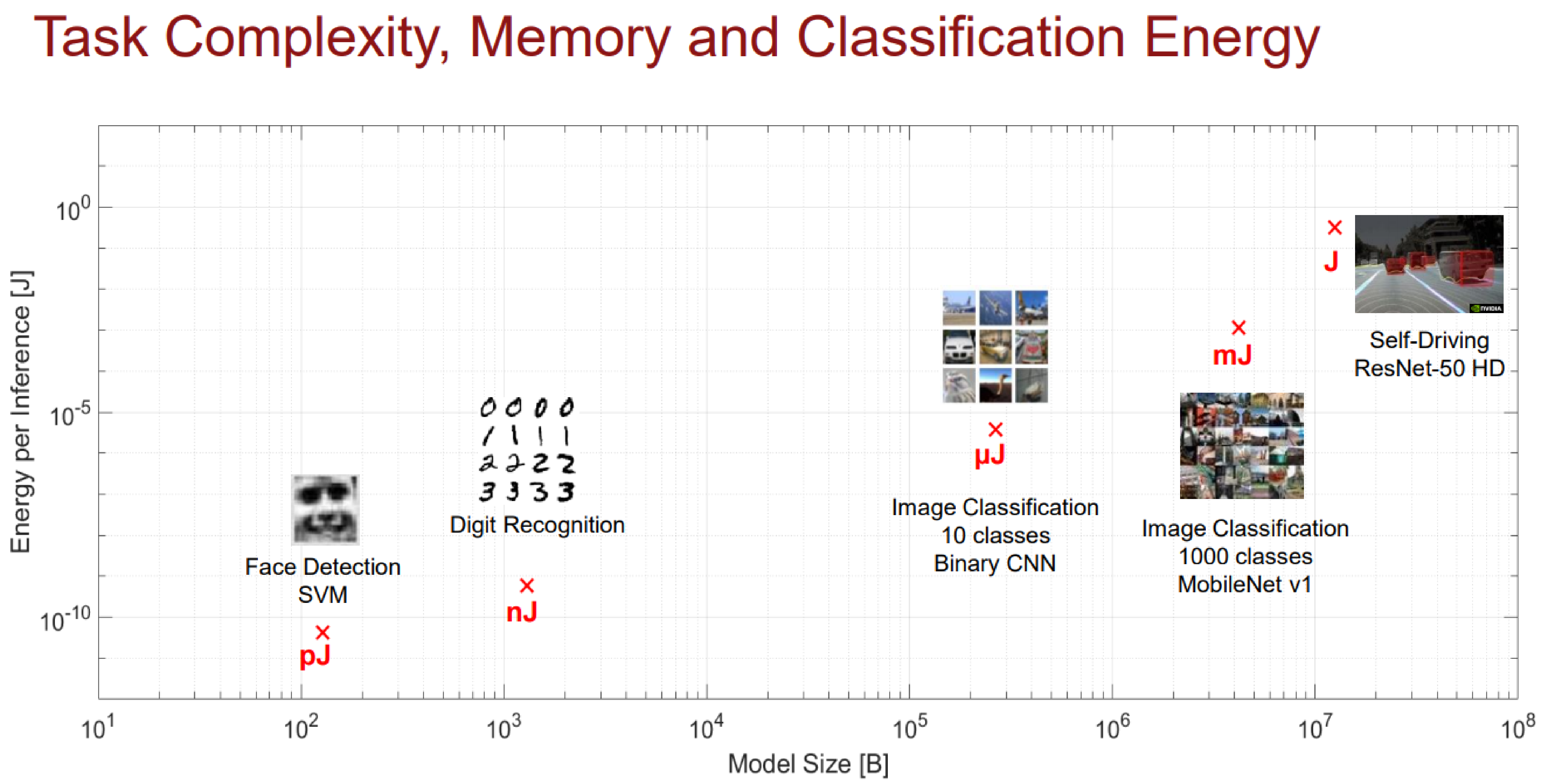
Bandwidth utilized

Privacy

Power consumed

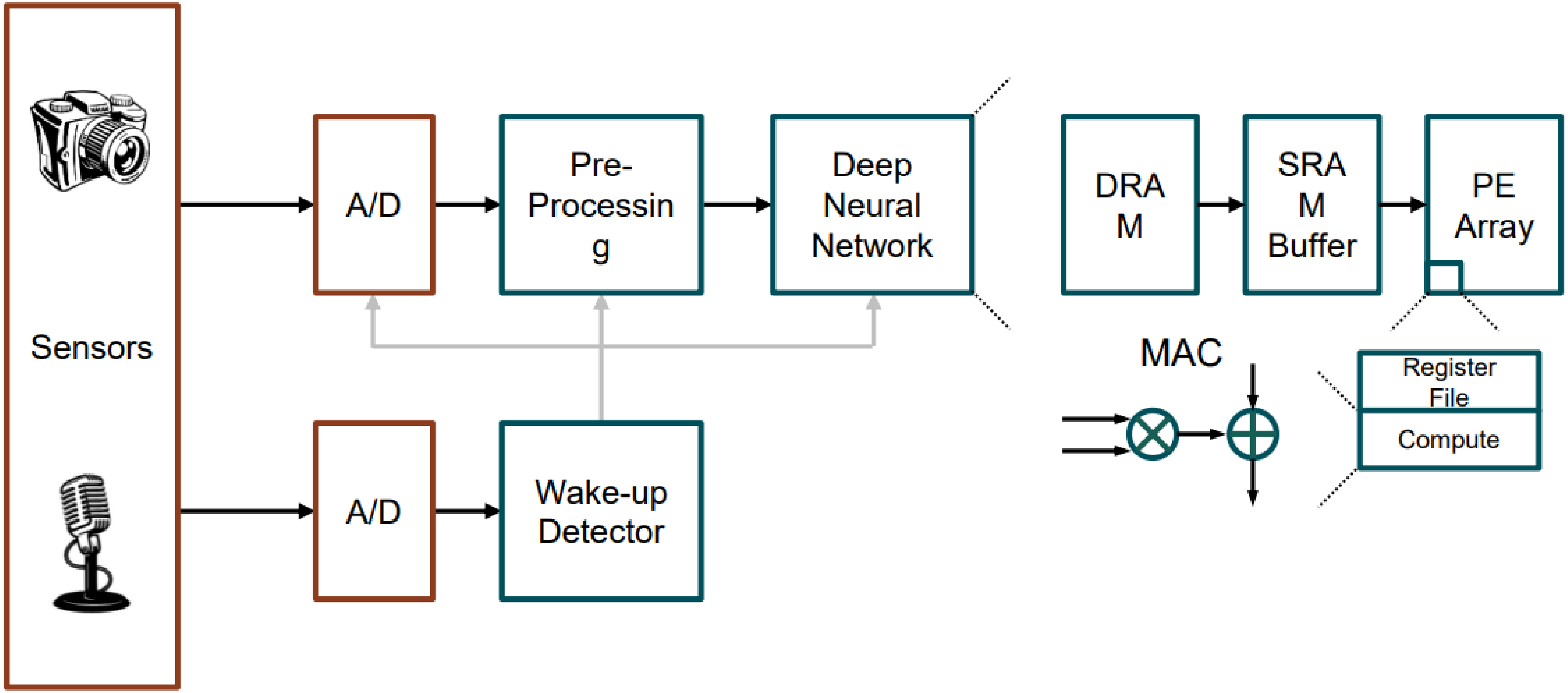
Boris Murmann, Mixed-Signal Techniques for Embedded Machine Learning Systems, CVPR 2019

1. Mixed Environment with Diverse Computational Abilities



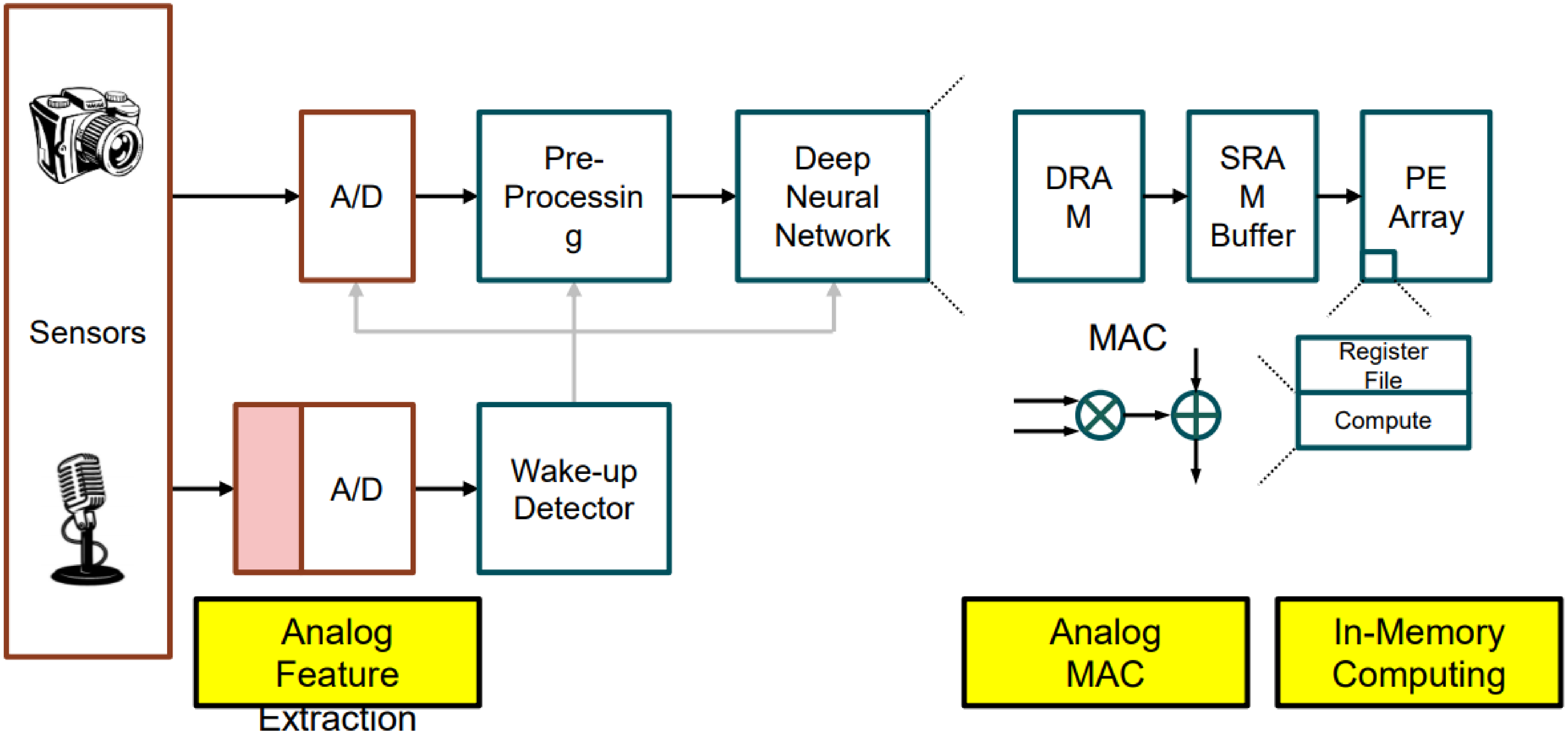
1. Mixed Environment with Diverse Computational Abilities

Edge Inference System



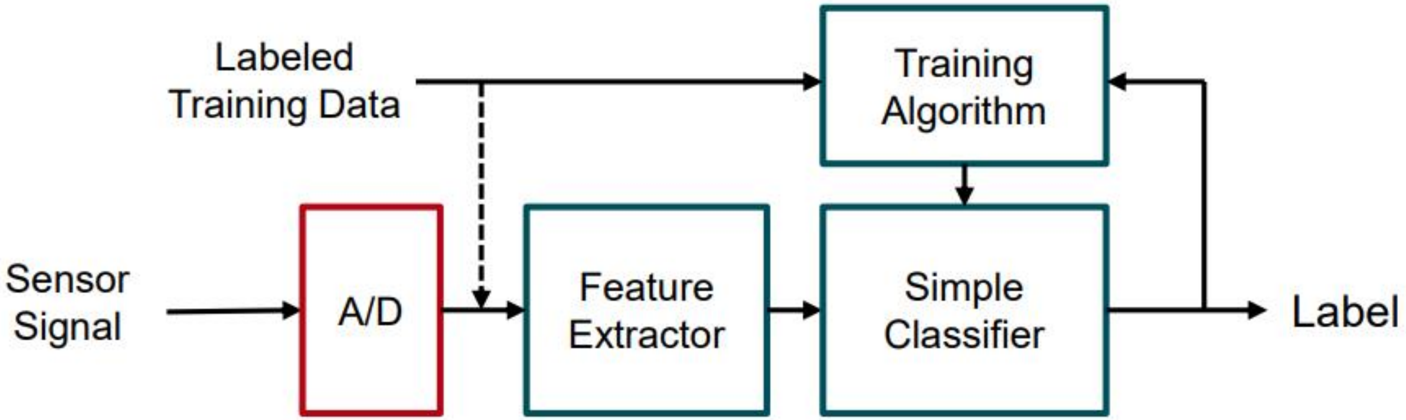
1. Mixed Environment with Diverse Computational Abilities

Opportunities for Analog/Mixed-Signal Design



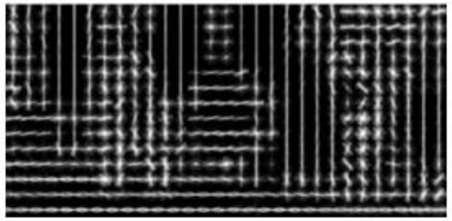
1. Mixed Environment with Diverse Computational Abilities

Wake-Up Detector with Hand-Crafted Features



Data Deluge

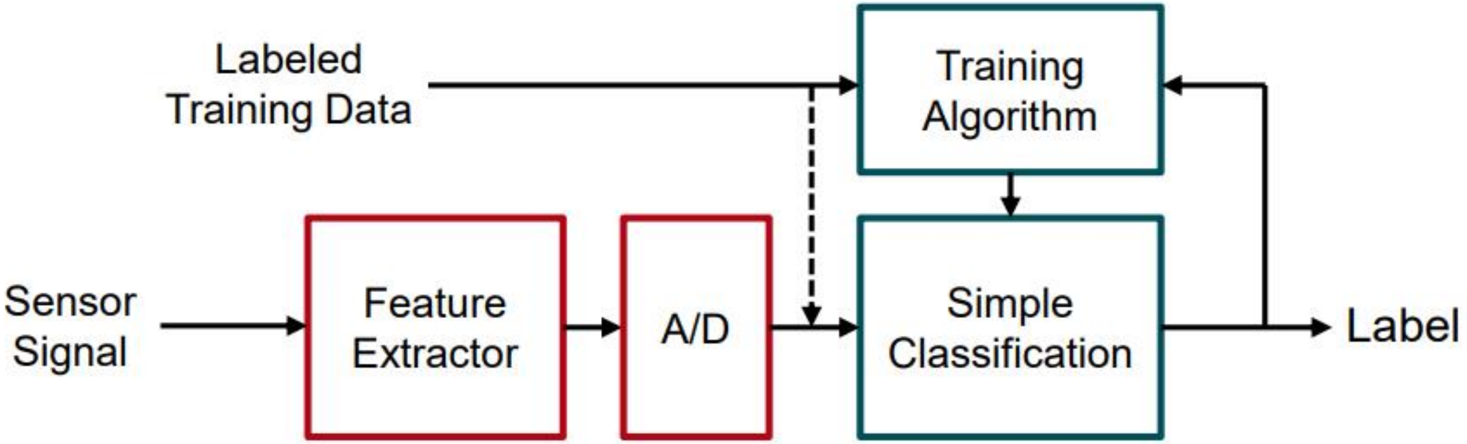
High-dimensional data



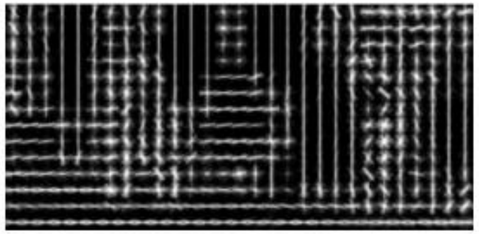
Low-dimensional representation

1. Mixed Environment with Diverse Computational Abilities

Analog Feature Extractor



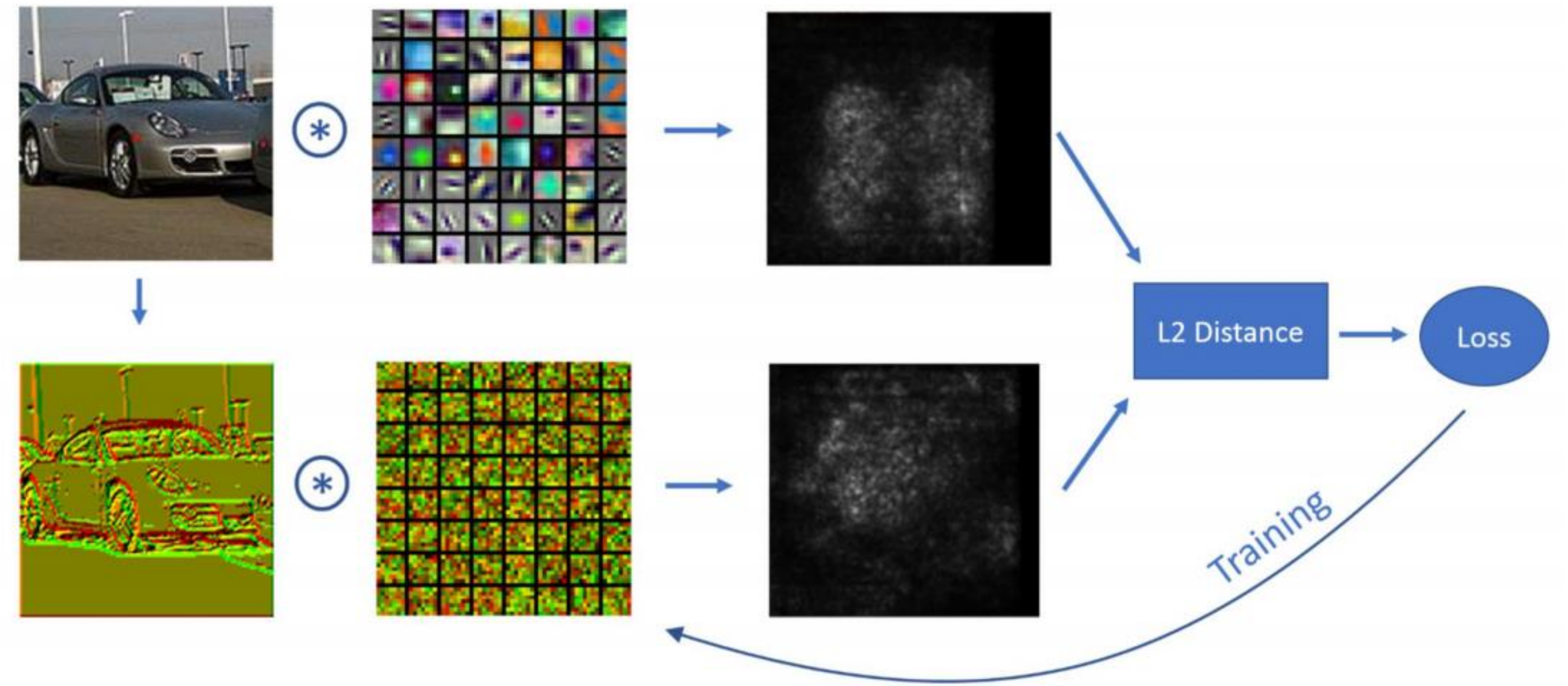
- Low-rate and/or low–resolution ADC
- Low data rate digital I/O
- Reduced memory requirements



Low-dimensional
representation

1. Mixed Environment with Diverse Computational Abilities

Use Log Gradients as ConvNet Input?



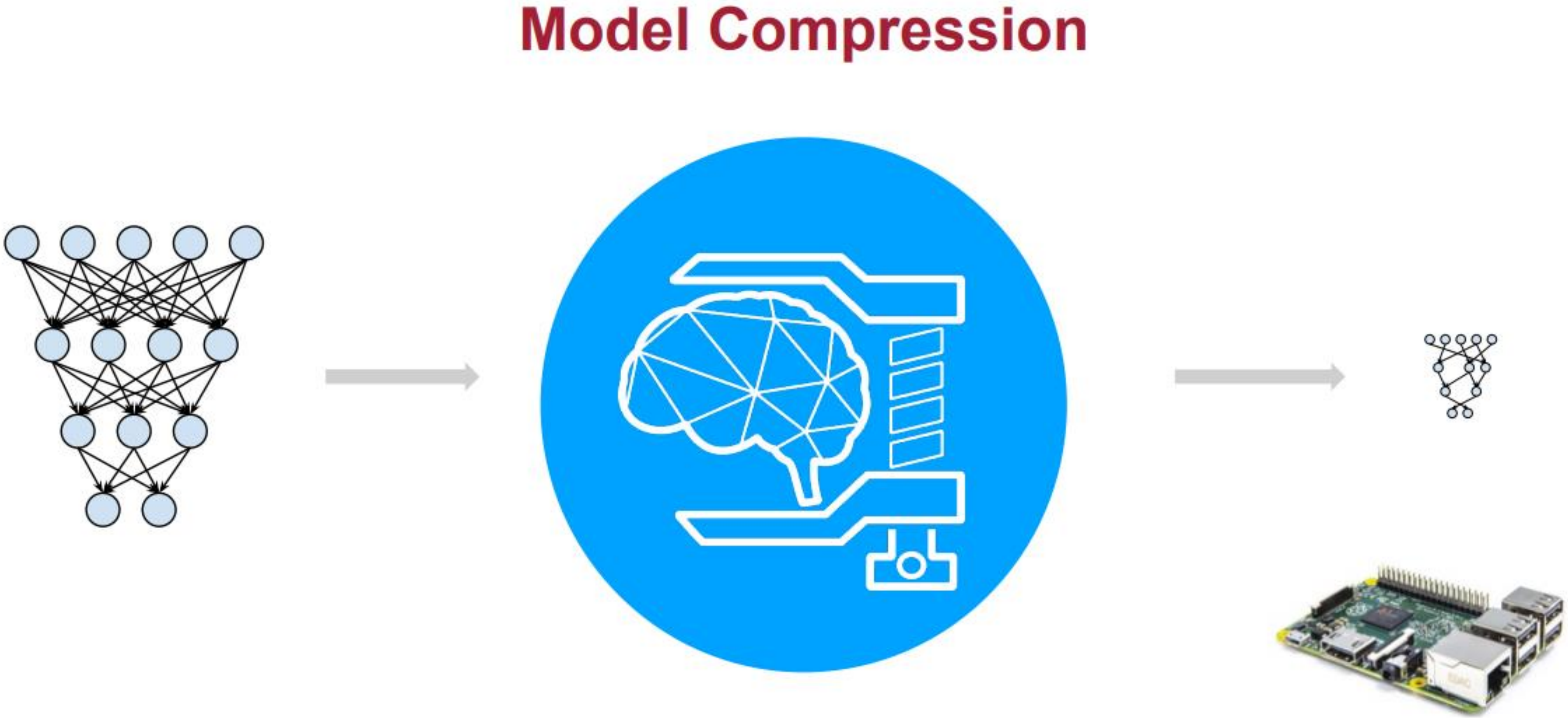
- Ongoing work; comparable performance using ResNet-10 (PascalRaw dataset)

2. Determining a lightweight network

(a) Designing a lightweight network such as MobileNet, SequeezeNet;

(b) Using network compression to learn a lightweight network from a teacher network

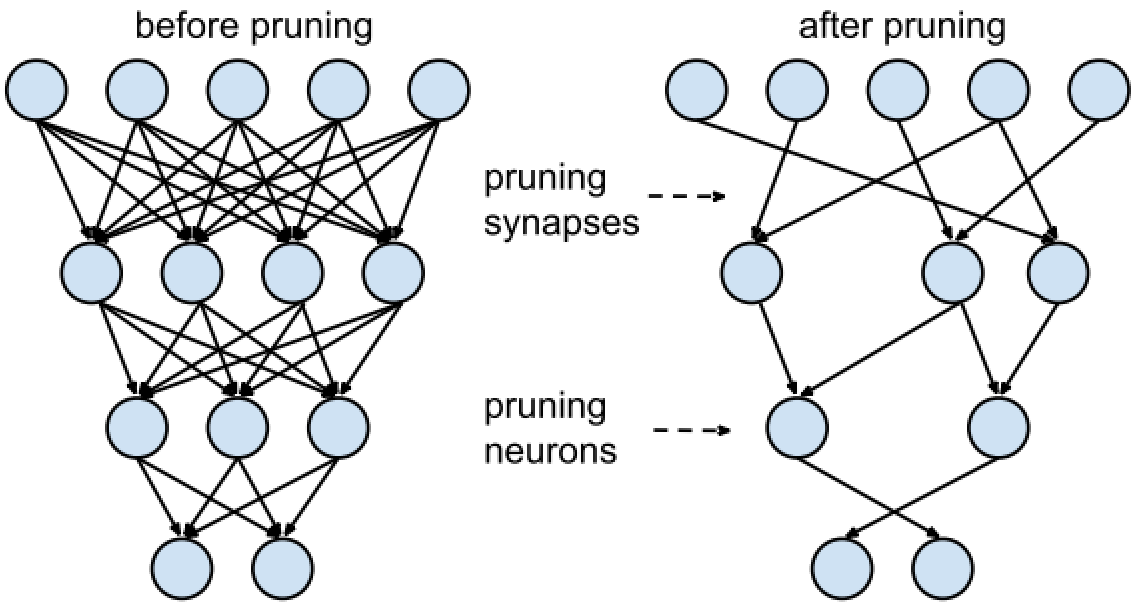
2. Determining a lightweight network



Song Han, Hardware Efficiency Aware Neural Architecture Search and Compression, CVPR 2019

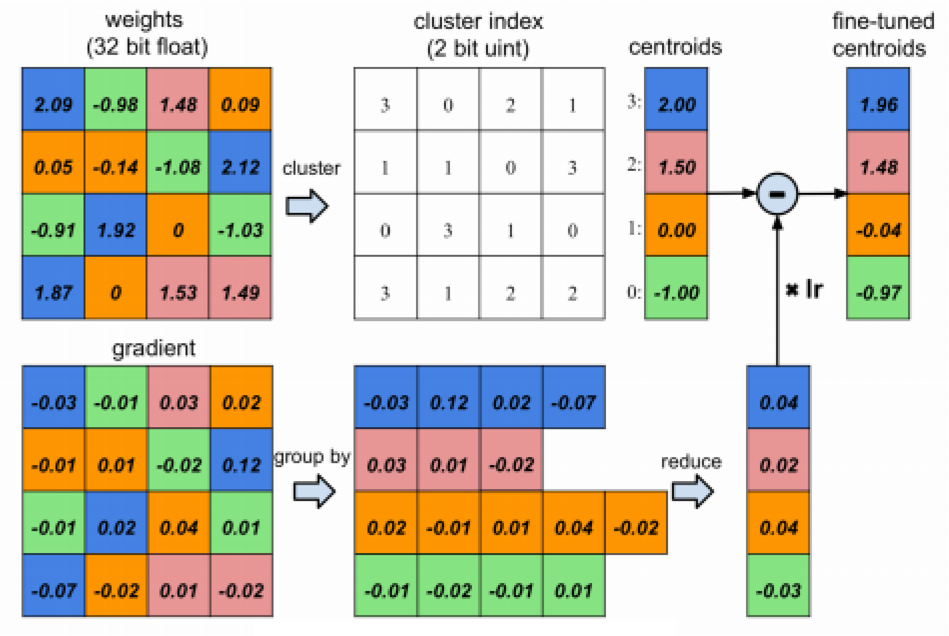
2. Determining a lightweight network

Deep Compression



Pruning

Han et al [NIPS'15]

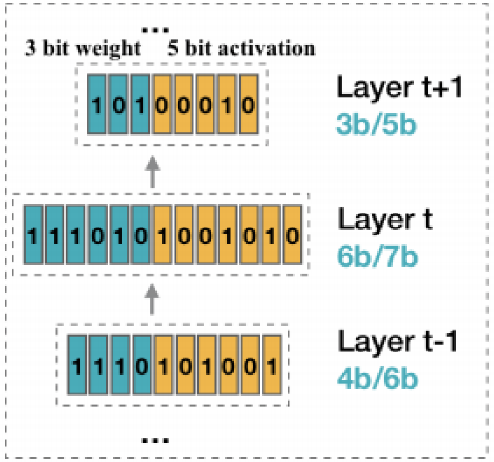


Quantization

Han et al [ICLR'16]
Best Paper Award

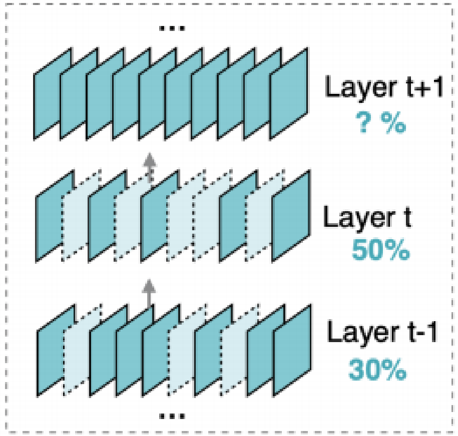
2. Determining a lightweight network

Design Automation for
Efficient Deep Learning Computing



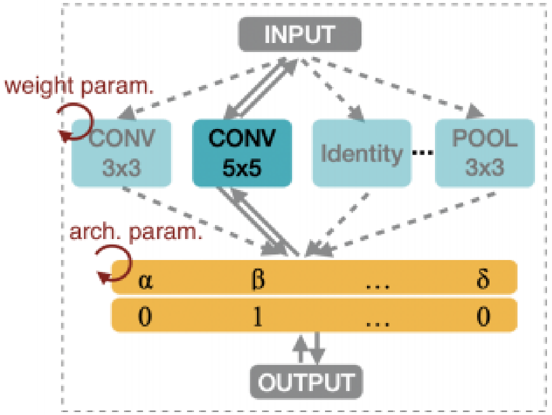
HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



AMC: AutoML for Model Compression

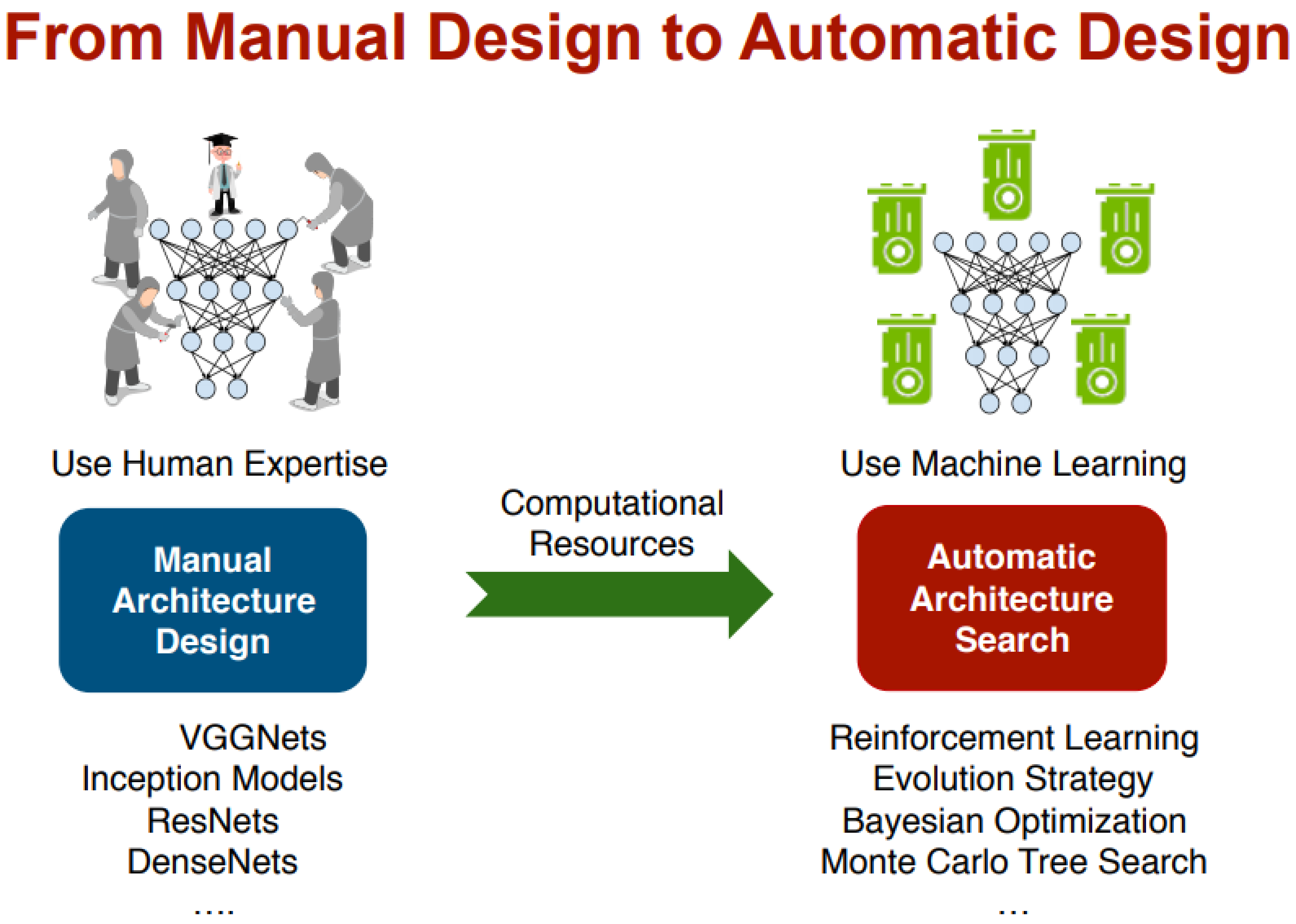
[ECCV 2018]



Proxyless Neural Architecture Search

[ICLR 2019]

2. Determining a lightweight network

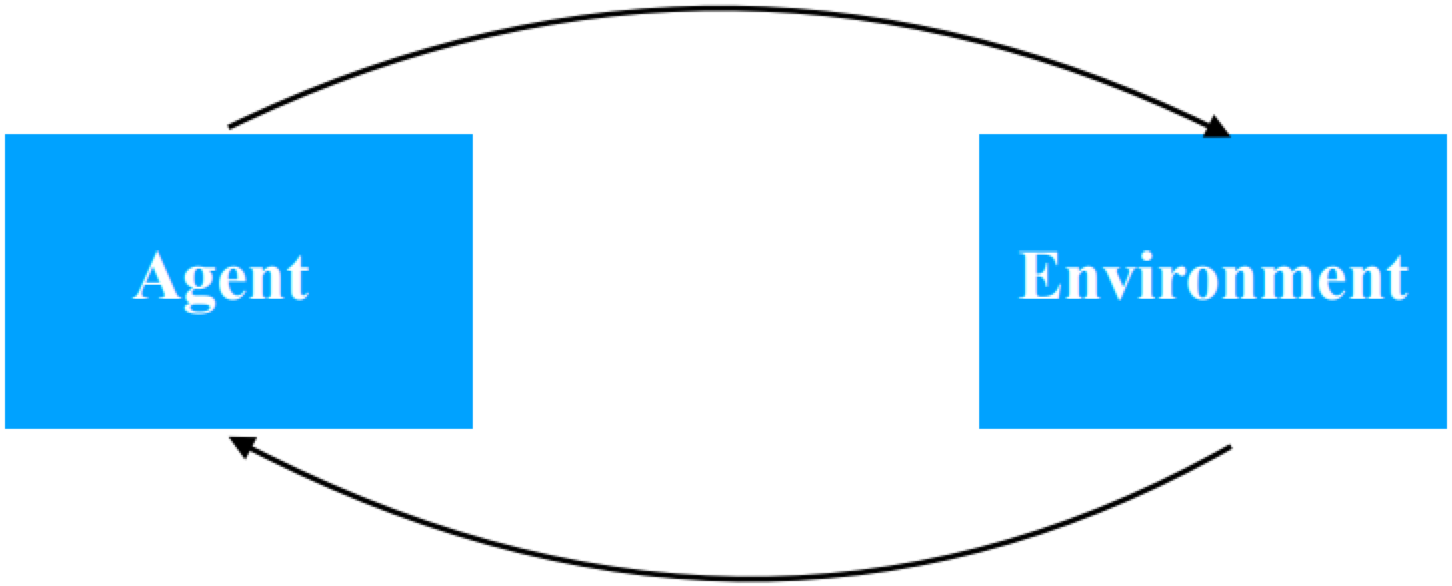


2. Determining a lightweight network

AMC: Automatic Model Compression

Actions:
sparsity for each layer

Actor Critic
DDPG



Channel
Pruning

State / Reward:
N,C,H,W, accuracy, latency…

AMC: AutoML for Model Compression and Acceleration on Mobile Devices
https://arxiv.org/pdf/1802.03494.pdf

2. Determining a lightweight network

References

- **Automated Model Architecture Tuning:**
  ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware
  Han Cai, Ligeng Zhu, Song Han
  International Conference on Learning Representations (ICLR), 2019.

- **Automated Pruning:**
  AMC: AutoML for Model Compression and Acceleration on Mobile Devices.
  Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, Song Han
  European Conference on Computer Vision (ECCV), 2018

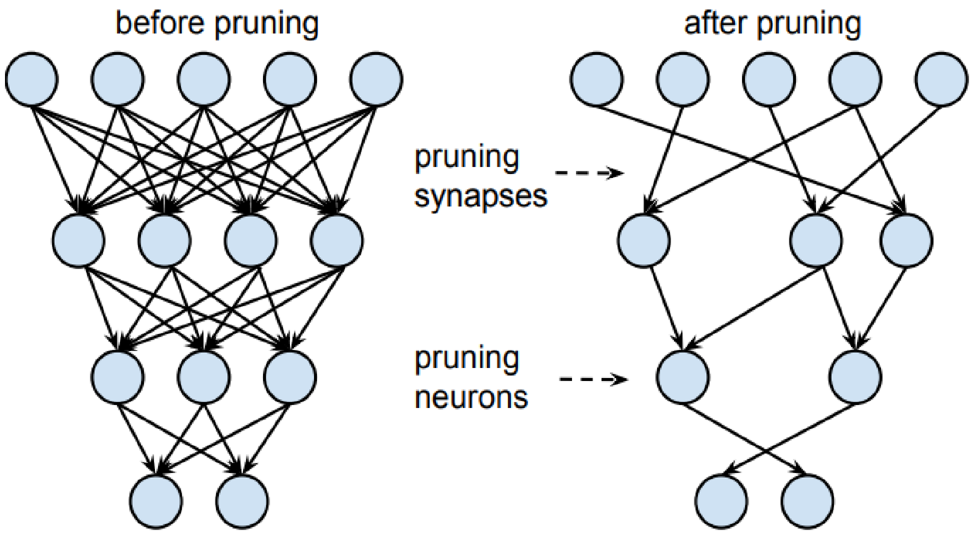
- **Automated Quantization:**
  HAQ: Hardware-Aware Automated Quantization with Mixed Precision
  Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, Song Han.
  IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019. Oral presentation.

  Defensive Quantization: When Efficiency Meets Robustness
  Ji Lin, Chuang Gan, Song Han
  International Conference on Learning Representations (ICLR), 2019.

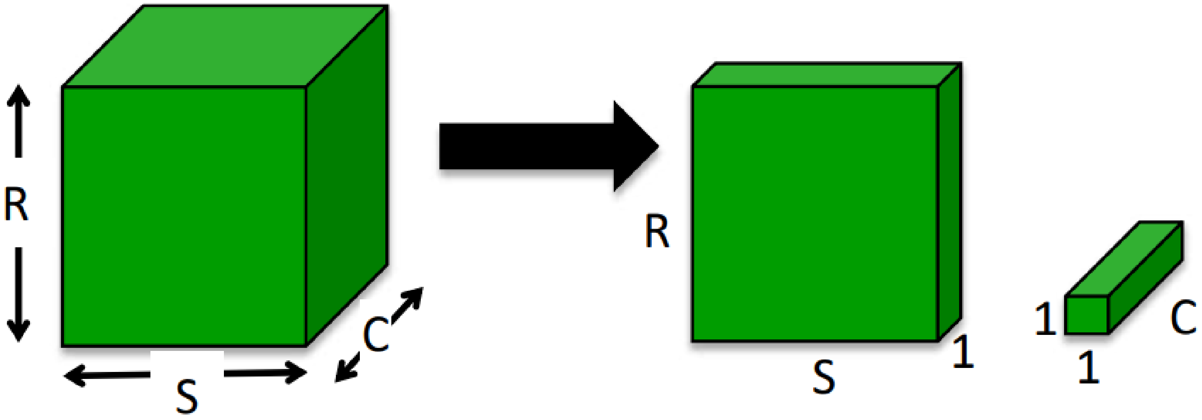
2. Determining a lightweight network

- Popular efficient DNN algorithm approaches

Network Pruning



Compact Network Architectures



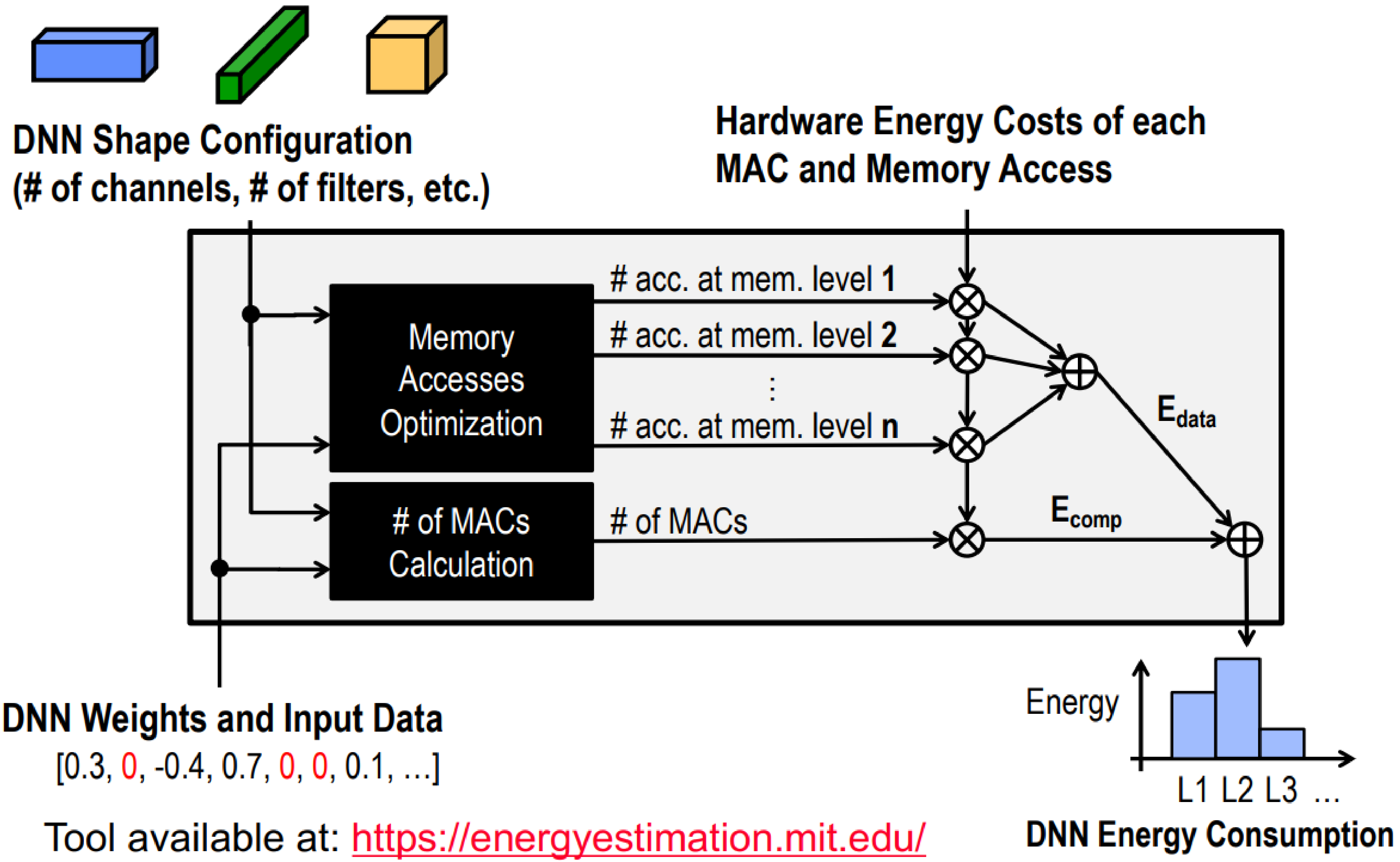
... also reduced precision

- Focus on reducing **number of MACs and weights**
- **Does it translate to energy savings and reduced latency?**

Vivienne Sze, Balancing Efficiency and Flexibility for DNN Acceleration, CVPR 2019

2. Determining a lightweight network

Energy-Evaluation Methodology



Tool available at: https://energyestimation.mit.edu/

2. Determining a lightweight network

Problem Formulation

$$\max_{Net} Accuracy(Net) \;\; subject\ to\ Resource_j(Net) \leq Budget_j, j = 1, \cdots, m$$

Break into a set of simpler problems and solve iteratively

$$\max_{Net_i} Acc(Net_i) \;\; subject\ to\ Res_j(Net_i) \leq Res_j(Net_{i-1}) - \Delta R_{i,j}, j = 1, \cdots, m$$

**Acc*: accuracy function, *Res*: resource evaluation function,
***ΔR*: resource reduction**, *Bud*: given budget
Budget incrementally tightens $\boldsymbol{Res_j(Net_{i-1}) - \Delta R_{i,j}}$

- **Advantages**
  - Supports multiple resource budgets at the same time
  - Guarantees that the budgets will be satisfied because the resource consumption decreases monotonically
  - Generates a family of networks (from each iteration) with different resource versus accuracy trade-offs
  - Intuitive and can easily set one additional hyperparameter ($\boldsymbol{\Delta R_{i,j}}$)

2. Determining a lightweight network

Metrics for DNN Hardware

- **Accuracy**
  - Quality of result for a given task
- **Throughput**
  - Analytics on high volume data
  - Real-time performance (e.g., video at 30 fps)
- **Latency**
  - For interactive applications (e.g., autonomous navigation)
- **Energy and Power**
  - Edge and embedded devices have limited battery capacity
  - Data centers have stringent power ceilings due to cooling costs
- **Hardware Cost**
  - $$$

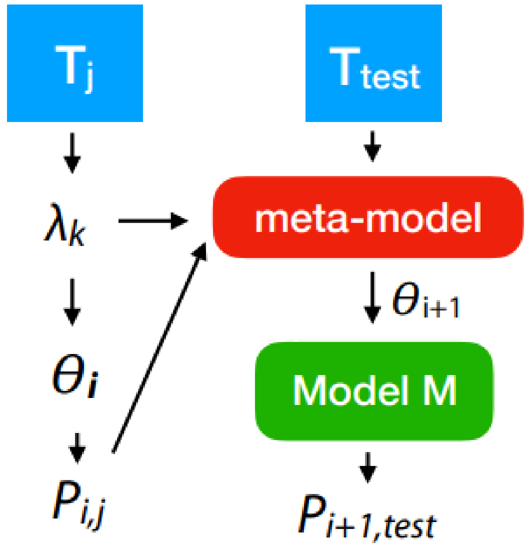
2. Determining a lightweight network

Specifications to Evaluate Metrics

- **Accuracy**
  - Difficulty of dataset and/or task should be considered
- **Throughput**
  - Number of cores (include utilization along with peak performance)
  - Runtime for running specific DNN models
- **Latency**
  - Include batch size used in evaluation
- **Energy and Power**
  - Power consumption for running specific DNN models
  - Include external memory access
- **Hardware Cost**
  - On-chip storage, number of cores, chip area + process technology

3. Learning from few samples (few-shot learning)

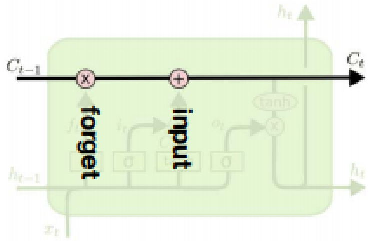
Few-shot learning: approaches



$$Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)$$

- Existing algorithm as meta-learner:
  - LSTM + gradient descent — Ravi and Larochelle 2017
  - Learn θ_{init} + gradient descent — Finn et al. 2017
  - kNN-like: Memory + similarity — Vinyals et al. 2016
  - Learn embedding + classifier — Snell et al. 2017
  - …
- Black-box meta-learner
  - Neural Turing machine (with memory) — Santoro et al. 2016
  - Neural attentive learner — Mishra et al. 2018
  - …

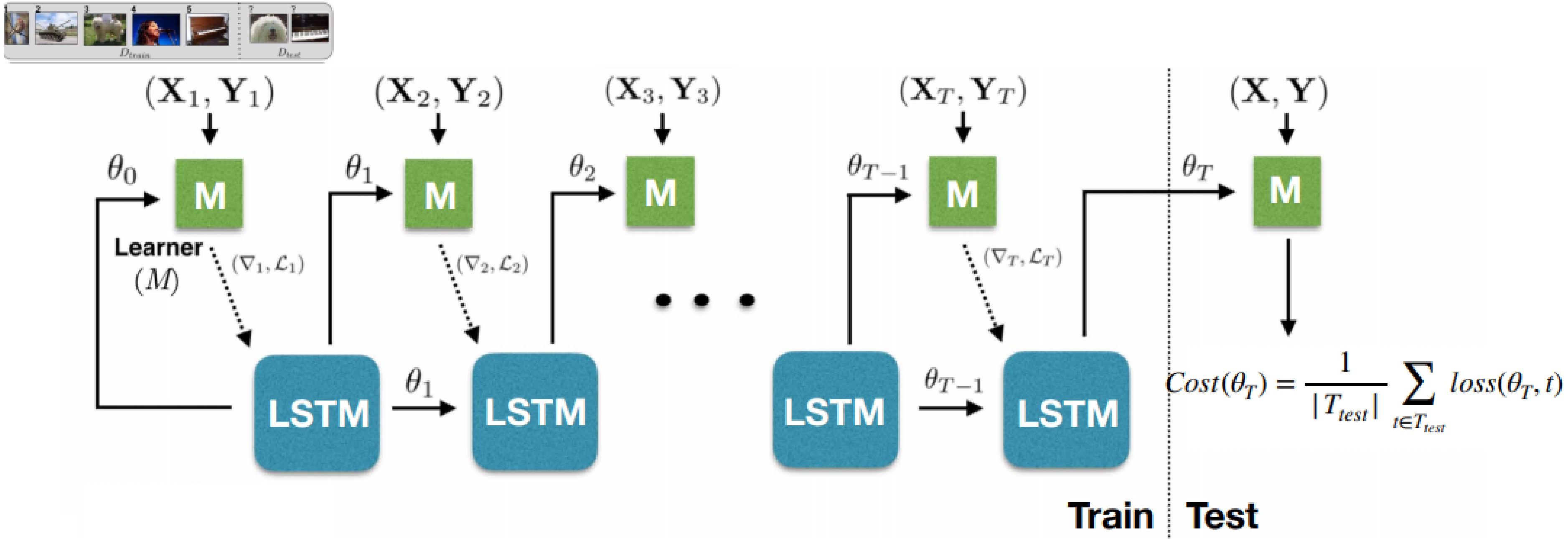
LSTM meta-learner + gradient descent



- Gradient descent update θ_t is similar to LSTM cell state update c_t

$$\boxed{\theta_t} = \boxed{\theta_{t-1}} - \alpha_t \boxed{\nabla_{\theta_{t-1}} \mathcal{L}_t} \qquad \boxed{c_t} = f_t \odot \boxed{c_{t-1}} + i_t \odot \boxed{\tilde{c}_t}$$

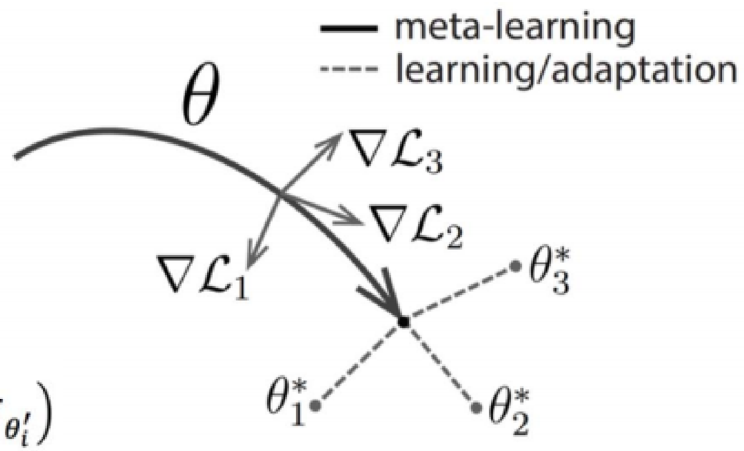
- Hence, training a meta-learner LSTM yields an update rule for training M
  - Start from initial θ_0, train model on first batch, get gradient and loss update
  - Predict θ_{t+1}, continue to t=T, get cost, backpropagate to learn LSTM weights, optimal θ_0



Model-agnostic meta-learning

- Quickly learn new skills by learning a model *initialization* that generalizes better to similar tasks
  - Current initialization θ
  - On K examples/task, evaluate $\nabla_\theta L_{T_i}(f_\theta)$
  - Update weights for $\theta_1, \theta_2, \theta_3$
  - Update θ to minimize sum of per-task losses
  - Repeat

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}\left(f_{\theta_i'}\right)$$



- More resilient to overfitting
- Generalizes better than LSTM approaches
- *Universality*: no theoretical downsides in terms of expressivity when compared to alternative meta-learning models.
- REPTILE: do SGD for k steps in one task, only then update initialization weights[3]

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks

Require: α, β: step size hyperparameters

randomly initialize θ

while not done **do**

  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$

  for all $\mathcal{T}_i$ **do**

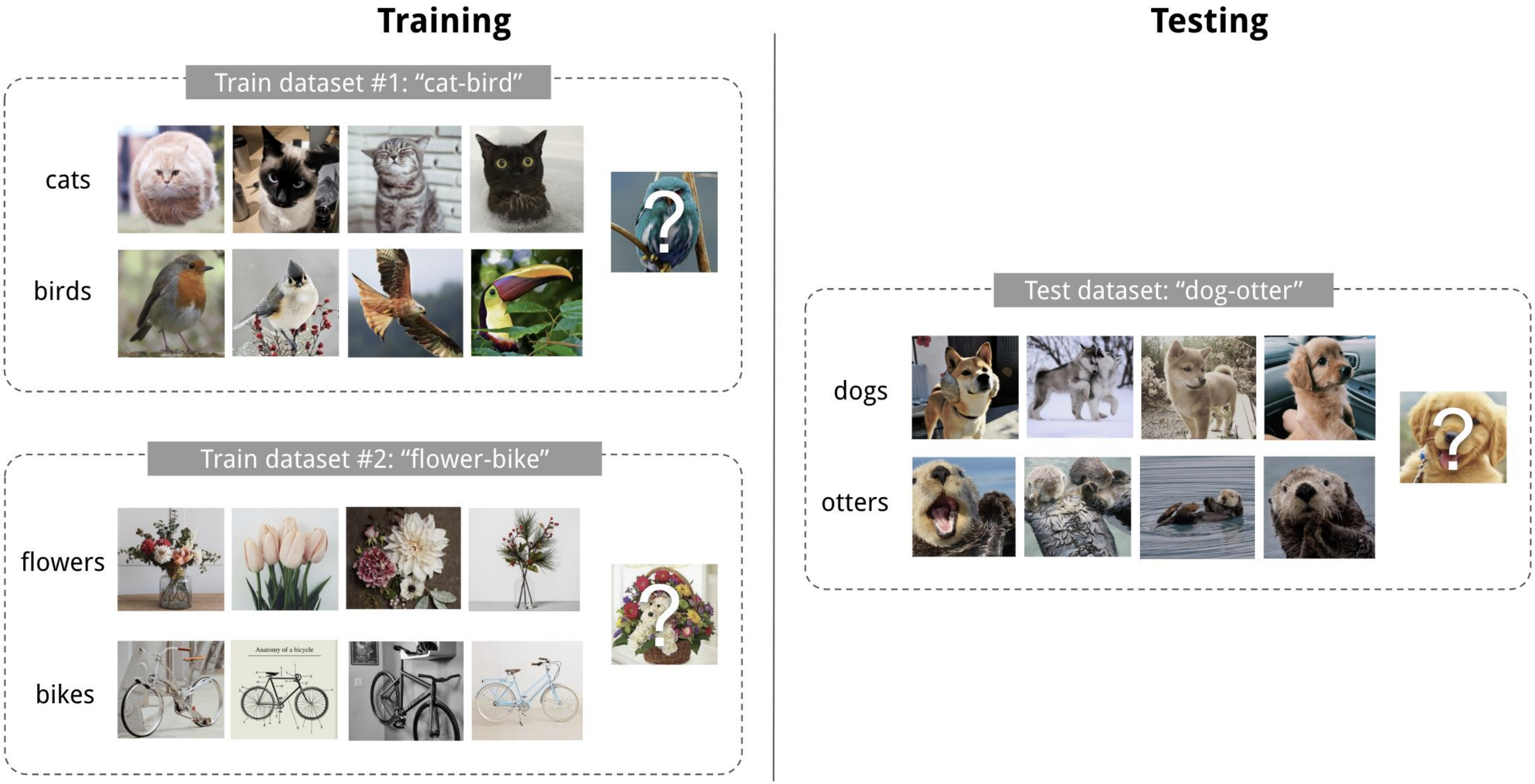
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples

    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$

  end for

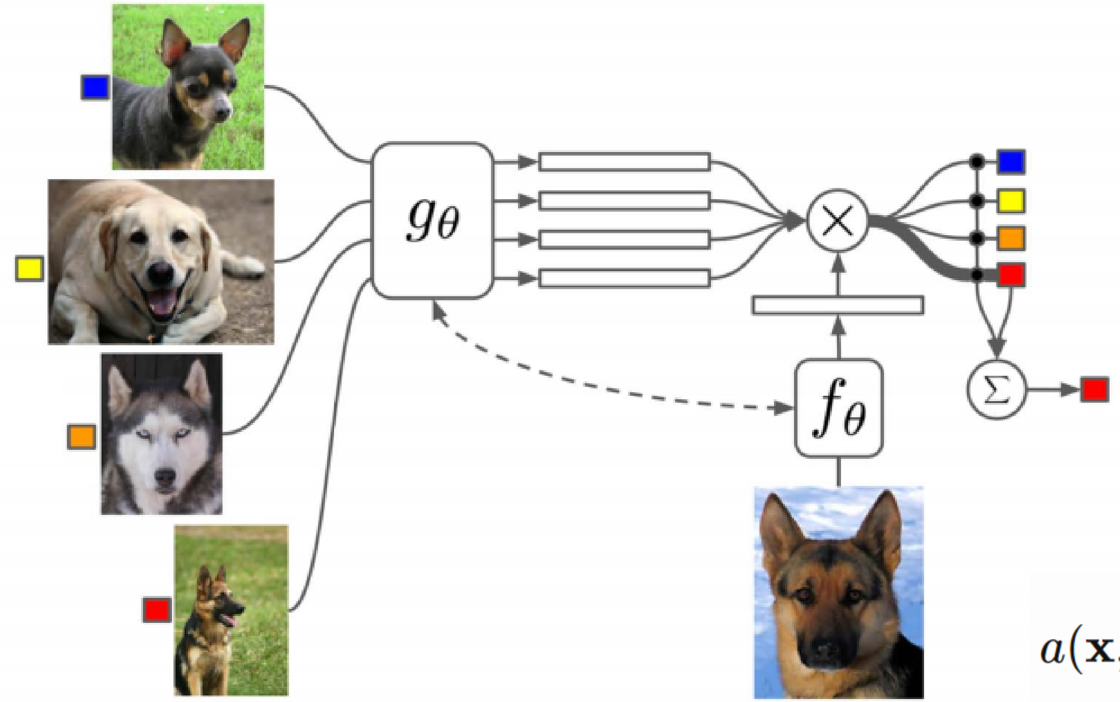
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$

end while


Training
Testing
Train dataset #1: “cat-bird”
cats
birds
?
Train dataset #2: “flower-bike”
flowers
bikes
Anatomy of a bicycle
?
Test dataset: “dog-otter”
dogs
otters
?

1-shot learning with Matching networks

- Don't learn model parameters, use non-parameters model (like kNN)
- Choose an embedding network f and g (possibly equal)
- Choose an attention kernel $a(\hat{x}, x_i)$, e.g. softmax over cosine distance
- Train complete network in minibatches with few examples per task



$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$$

$\theta = \{\text{VGG, Inception,} \ldots\}$

$$a(\mathbf{x}, \mathbf{x}_i) = \frac{\exp(\text{cosine}(f(\mathbf{x}), g(\mathbf{x}_i)))}{\sum_{j=1}^{k} \exp(\text{cosine}(f(\mathbf{x}), g(\mathbf{x}_j)))}$$

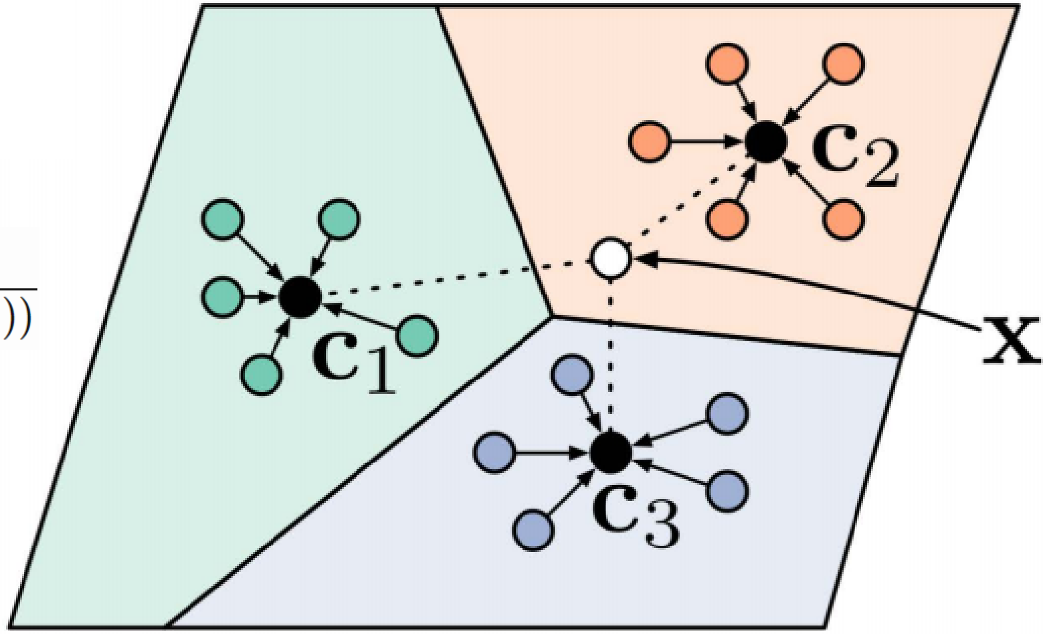
Prototypical networks

Snell et al. 2017
Ren et al. 2018

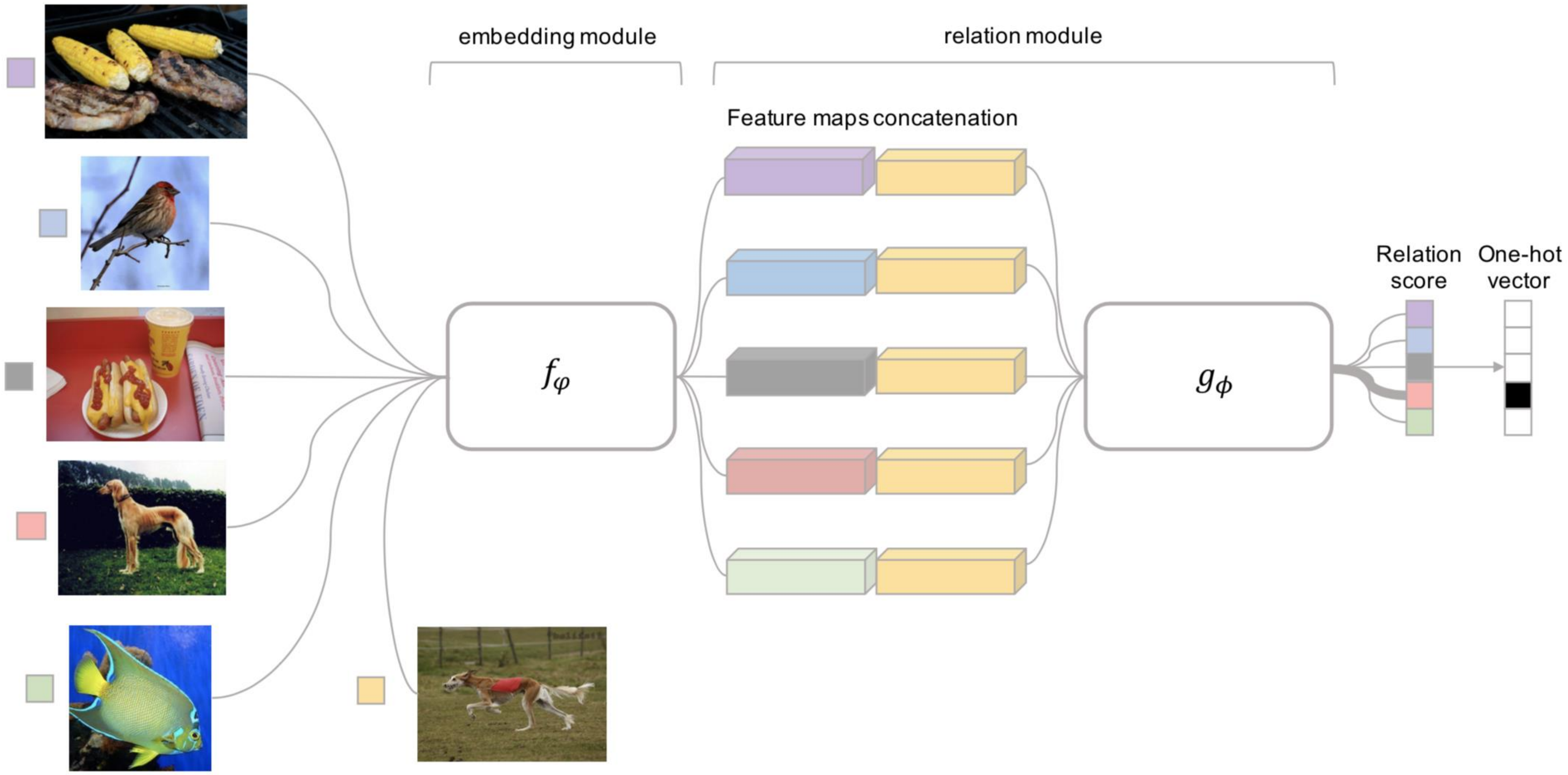
- Train a "prototype extractor" network
- Map examples to p-dimensional embedding so examples of a given class are close together
- Calculate a prototype (mean vector) for every class
- Map test instances to the same embedding, use softmax over distance to prototype
- Using more classes during meta-training works better!

$$\mathbf{v}_c = \frac{1}{|S_c|} \sum_{(\mathbf{x}_i, y_i) \in S_c} f_\theta(\mathbf{x}_i)$$

$$P(y = c|\mathbf{x}) = \text{softmax}(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_c)) = \frac{\exp(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_c))}{\sum_{c' \in \mathcal{C}} \exp(-d_\varphi(f_\theta(\mathbf{x}), \mathbf{v}_{c'}))}$$



Relation Network



Reference on few-shot learning

[1] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. "Human-level concept learning through probabilistic program induction." Science 350.6266 (2015): 1332-1338.
[2] Oriol Vinyals' talk on "Model vs Optimization Meta Learning"
[3] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition." ICML Deep Learning Workshop. 2015.
[4] Oriol Vinyals, et al. "Matching networks for one shot learning." NIPS. 2016.
[5] Flood Sung, et al. "Learning to compare: Relation network for few-shot learning." CVPR. 2018.
[6] Jake Snell, Kevin Swersky, and Richard Zemel. "Prototypical Networks for Few-shot Learning." CVPR. 2018.
[7] Adam Santoro, et al. "Meta-learning with memory-augmented neural networks." ICML. 2016.
[8] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines." arXiv preprint arXiv:1410.5401 (2014).
[9] Tsendsuren Munkhdalai and Hong Yu. "Meta Networks." ICML. 2017.
[10] Sachin Ravi and Hugo Larochelle. "Optimization as a Model for Few-Shot Learning." ICLR. 2017.
[11] Chelsea Finn's BAIR blog on "Learning to Learn".
[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." ICML 2017.
[13] Alex Nichol, Joshua Achiam, John Schulman. "On First-Order Meta-Learning Algorithms." arXiv preprint arXiv:1803.02999 (2018).