Hardware-Aware Efficient Deep Neural Network Design

Tien-Ju Yang

August 21, 2020

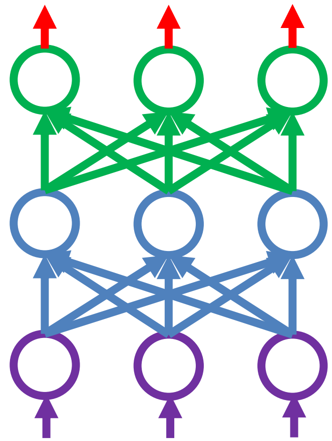
Thesis Advisor: **Prof. Vivienne Sze**

Thesis Committee: **Prof. Joel Emer, Prof. Sertac Karaman**

Motivation

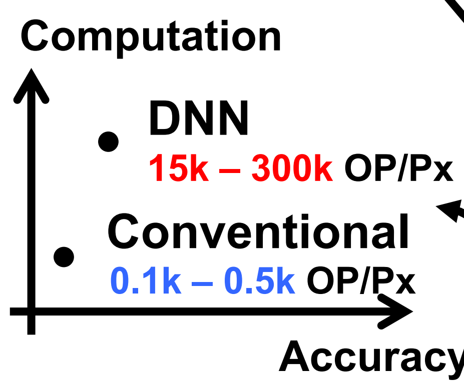
Challenge of Deep Neural Networks

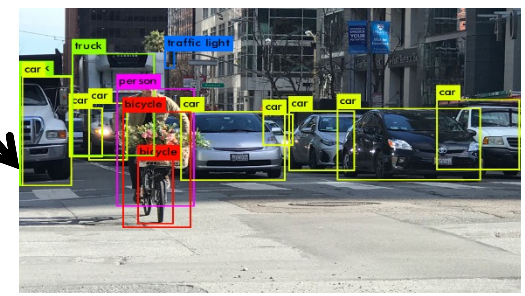
Image Classification



Speech Recognition

Machine Translation





Object Detection

The high accuracy of DNNs is at the cost of much higher computational complexity

Impact of High Complexity

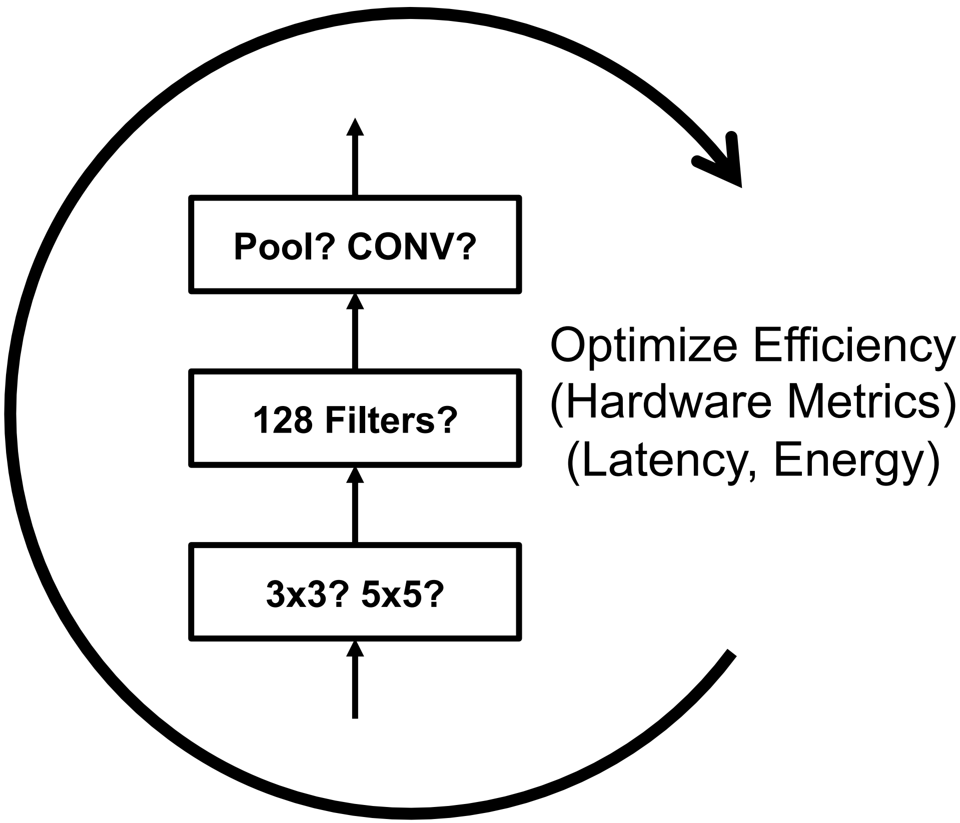
Environmentally	Financially	Functionally
		
Training Transformer emits 5x car lifetime carbon dioxide emission (high energy)	Training GPT-3 costs $4.6 million (high latency)	High accuracy networks drain battery fast or run slowly (high energy and latency)

Image sources: solarschools.net, 123rf.com

Efficient Neural Network Design

Important to design DNNs that run efficiently on various hardware

Deep Neural Network



Hardware

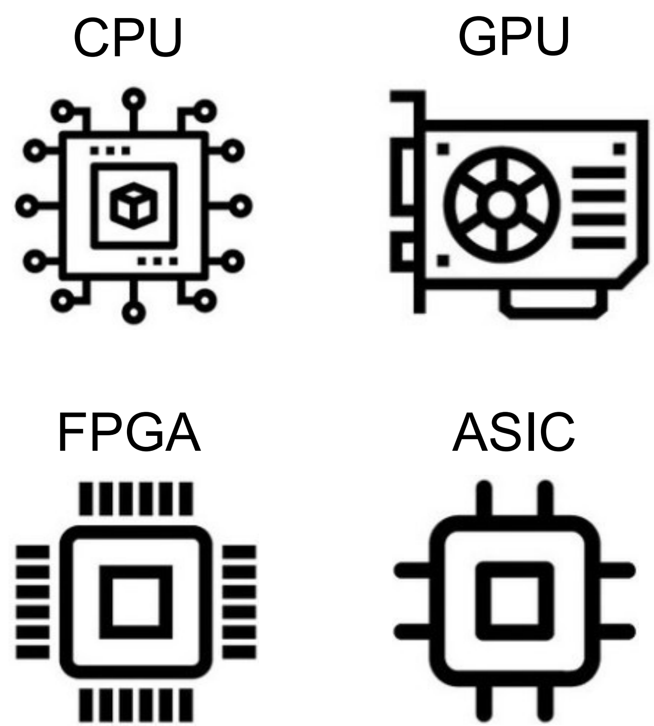
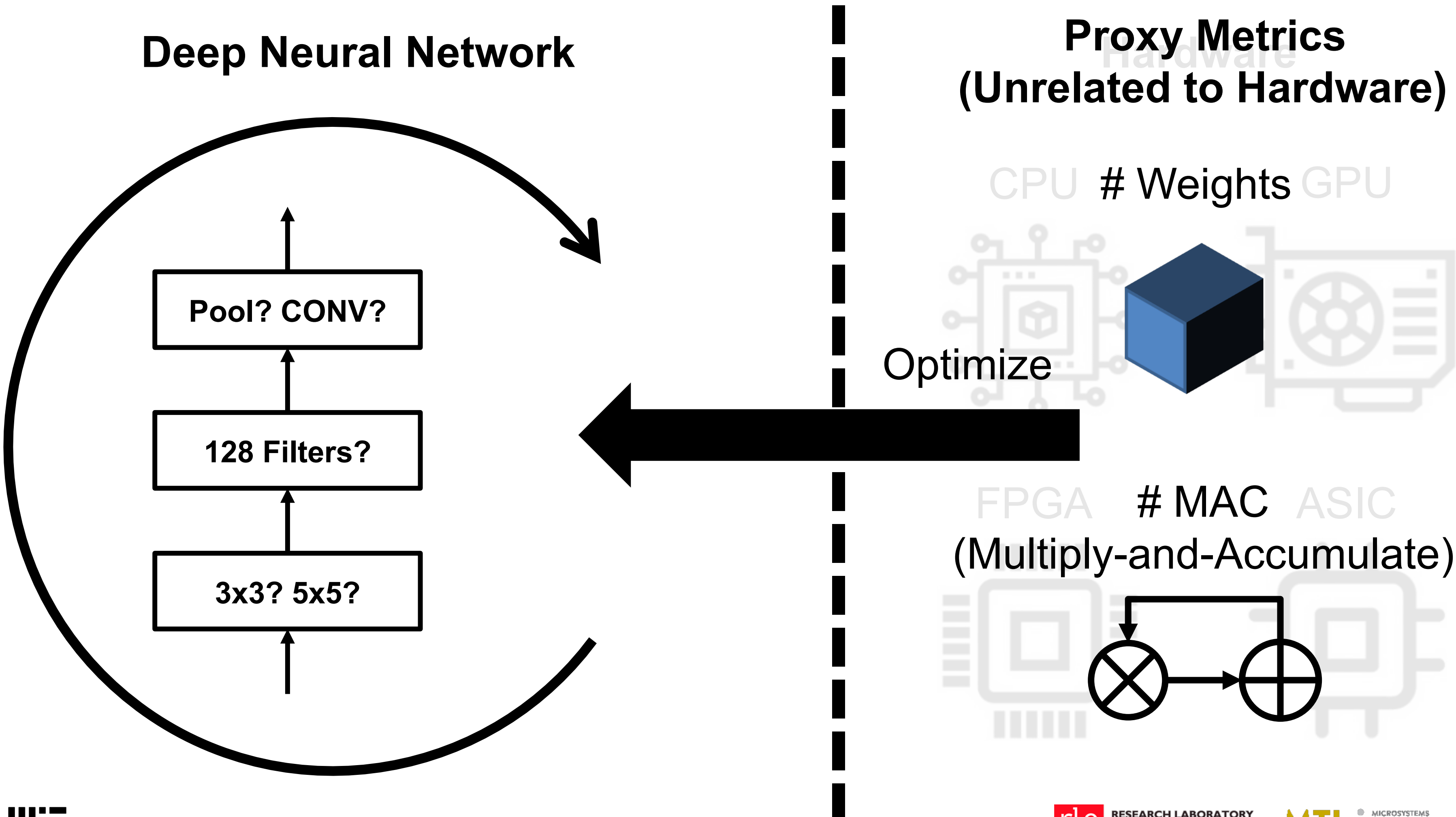


Image source: Venturebeat

DNN Design Disregards Hardware

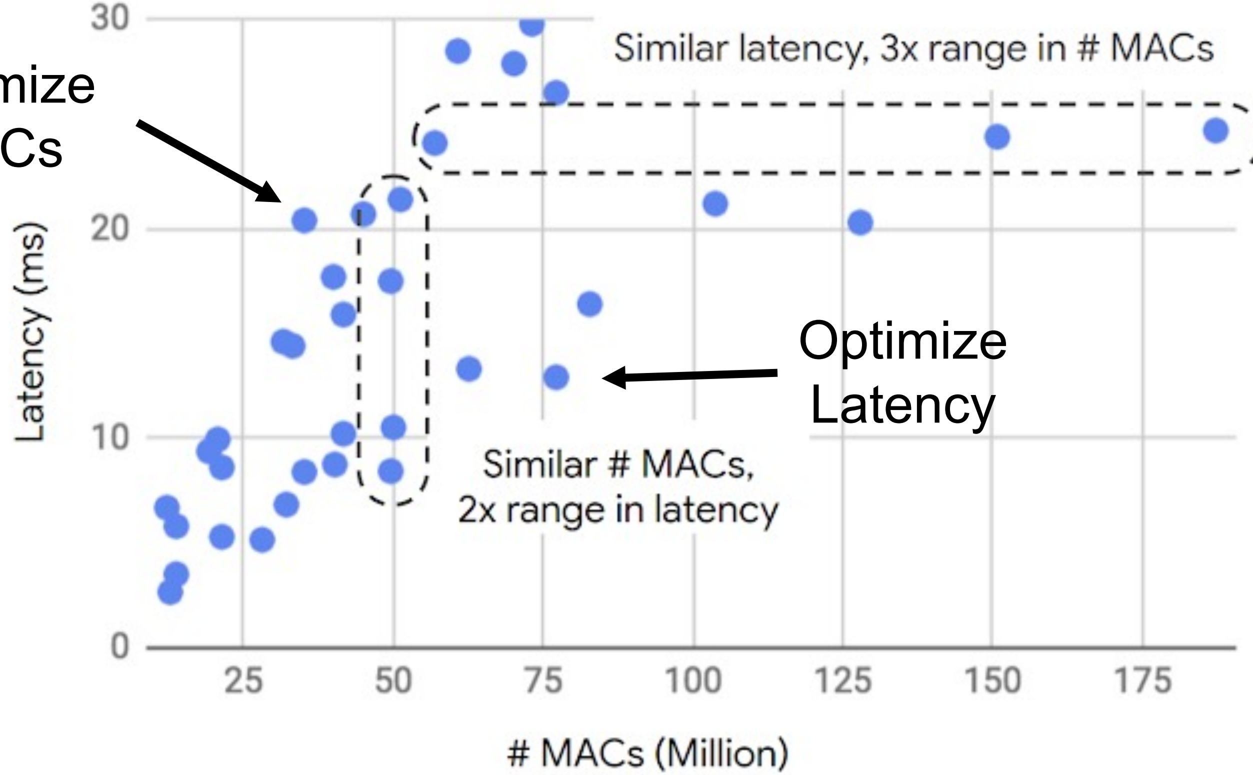
DNN design usually focuses on optimizing proxy metrics



of MACs vs. Latency

of MACs does not approximate latency well

Optimize MACs

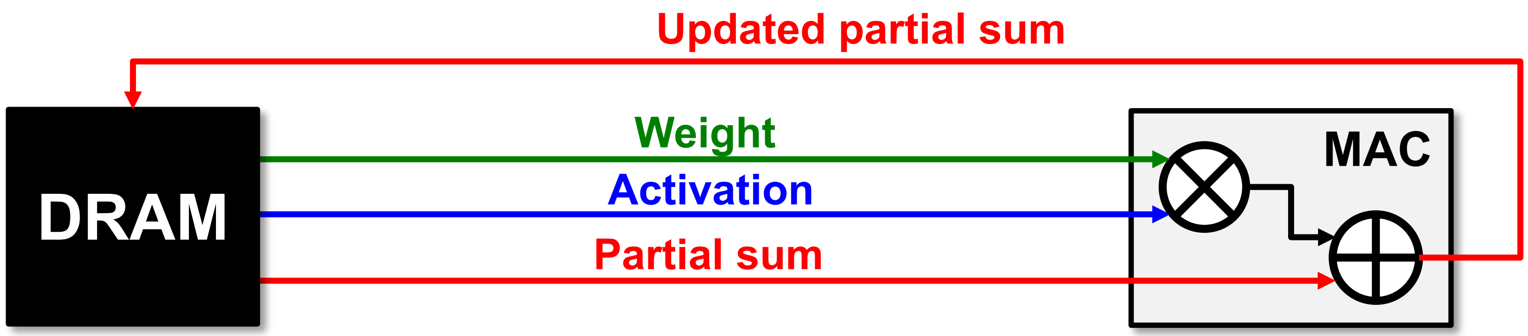


Optimize Latency

Image sources: Google

of Weights/MACs vs. Energy

of weights/MACs alone does not approximate energy well

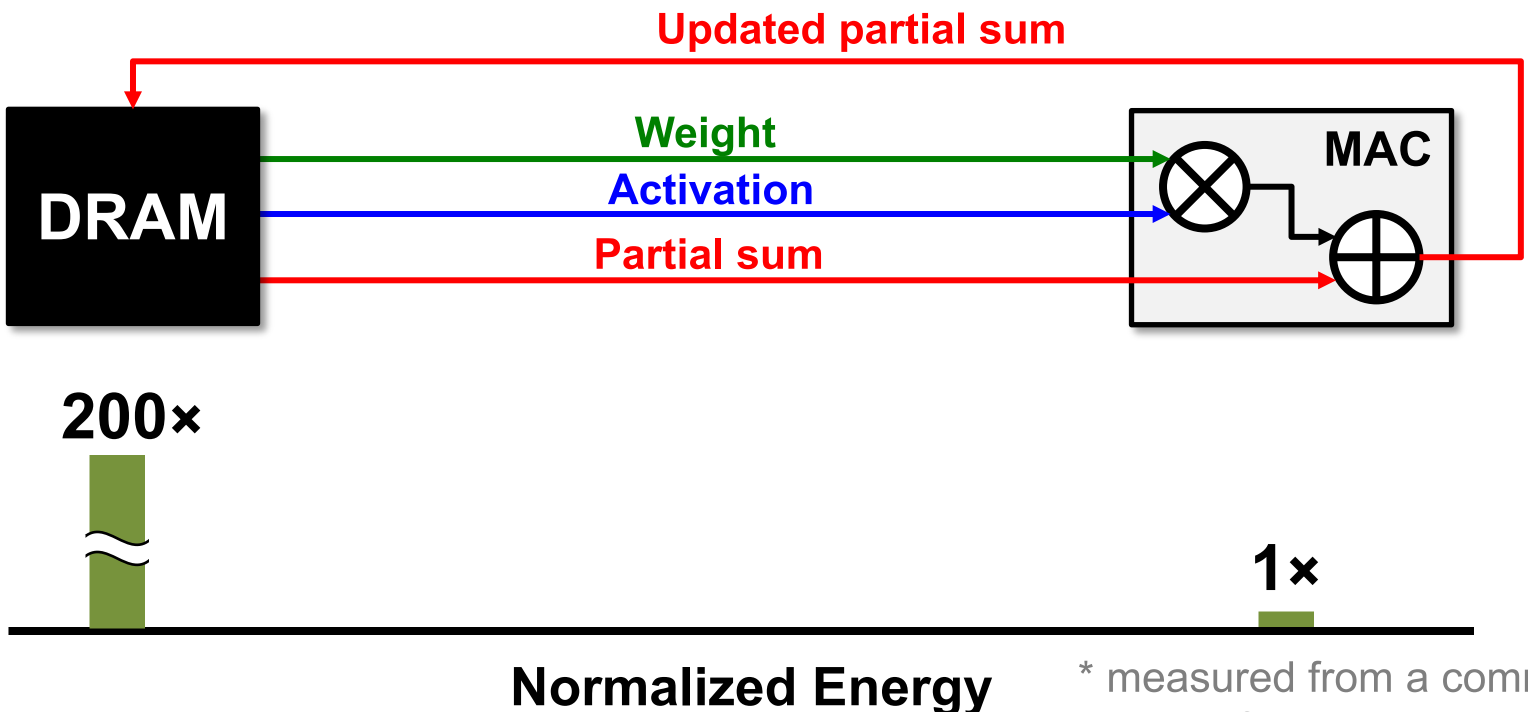


Reason 1: activations and partial sums are not considered
Reason 2:
Reason 3:

Image sources: Yu-Hsin Chen

of Weights/MACs vs. Energy

of weights/MACs does not approximate energy well



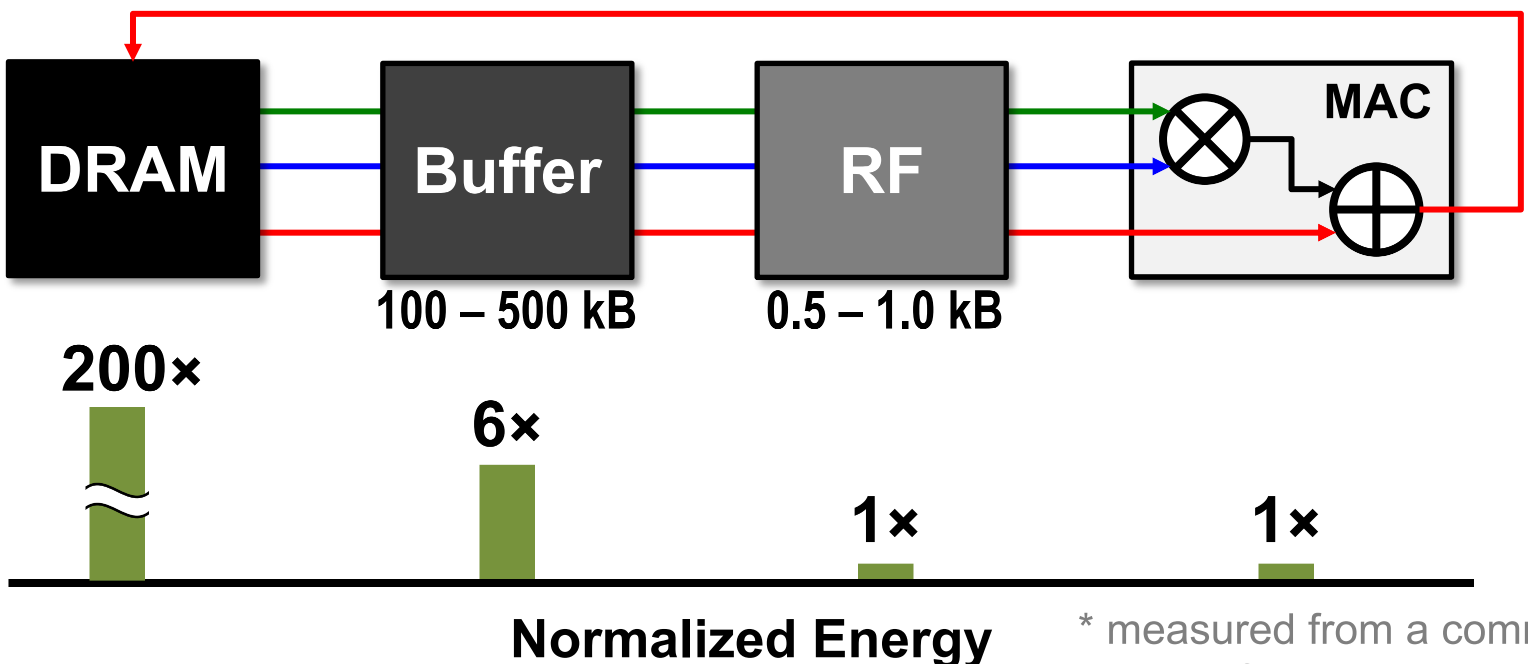
* measured from a commercial 65nm process

Reason 1: activations and partial sums are not considered
Reason 2: computation is cheap but data movement is expensive
Reason 3:

Image sources: Yu-Hsin Chen

of Weights/MACs vs. Energy

of weights/MACs does not approximate energy well



* measured from a commercial 65nm process

Reason 1: activations and partial sums are not considered
Reason 2: computation is cheap but data movement is expensive
Reason 3: where data comes from/goes to is important for energy

Hardware-Aware Efficient DNN Design

To maximize the efficiency, we need to bring **hardware in the loop** by directly **optimizing hardware metrics**

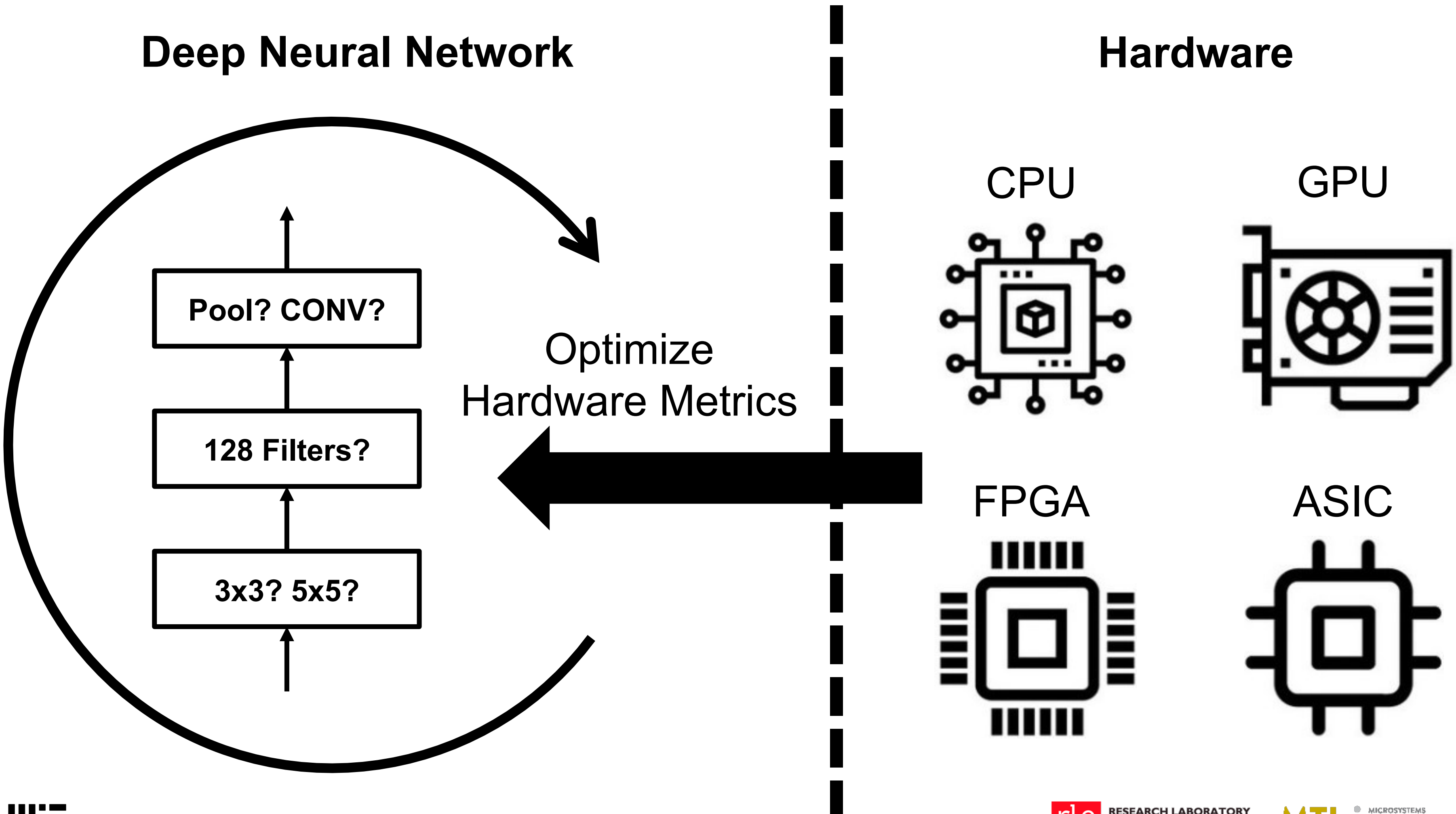


Image source: Venturebeat

Focus of Thesis

We focus on answering 3 main questions to address 3 challenges:

- Challenge 1: hardware metrics are usually not differentiable and highly depend on hardware properties
  - How to design <u>**efficient**</u> DNNs with hardware metrics?
- Challenge 2: evaluating hardware metrics on the hardware can be slow
  - How to <u>**efficiently**</u> estimate hardware metrics?
- Challenge 3: existing design approaches for efficient DNNs are mostly designed for digital accelerators and image classification
  - How to design efficient DNNs for various <u>**hardware accelerators and applications**</u>?

Our Solutions

1) Design efficient DNNs with hardware metrics

- Energy-aware pruning [CVPR 2017]
- NetAdapt V1 [ECCV 2018]
- NetAdapt V2 [Under review]

Automated algorithms optimizing hardware metrics to significantly improve the efficiency

2) Efficiently estimate hardware metrics

- Energy estimation methodology [CVPR 2017, Asilomar 2017]
- Lookup table approximation [ECCV 2018]

Fast methods for both with/without knowing how the hardware processes DNNs

3) Design efficient DNNs for various accelerators and applications

- Processing-in-memory accelerators [IEDM 2019]
- Panoptic segmentation [arXiv 2019]
- Monocular depth estimation [ICRA 2019]

Design approaches and efficient architectures for 1 hardware accelerators and 2 applications

Summary of Key Contributions

1) Design efficient DNNs with hardware metrics

- Hardware-aware DNN design strategy
- Feature-map-based network pruning
- Fast local fine-tuning
- Hardware-guided optimizers for NAS
- DNN structure for searching multiple dimensions
- Efficient search space pre-training

Energy-aware pruning

NetAdapt V1/V2

2) Efficiently estimate hardware metrics

- Fast metric estimation for white-box hardware
- Fast metric estimation for black-box hardware

Energy est. method

Lookup table approx.

3) Design efficient DNNs for various accelerators and applications

- Analysis and DNN design approach for PIM accelerators
- Novel single-shot, bottom-up architecture for panoptic segmentation
- Design approaches for efficient dense prediction applications
- New accuracy metric for panoptic segmentation
- Efficient architecture for depth estimation with hardware-oriented design

PIM accelerator

Panoptic segmentation

Depth estimation

What We Will Cover Today

1) Design efficient DNNs with hardware metrics

- Hardware-aware DNN design strategy
- Feature-map-based network pruning
- Fast local fine-tuning
- Hardware-guided optimizers for NAS
- DNN structure for searching multiple dimensions
- Efficient search space pre-training

Energy-aware pruning

NetAdapt V1/V2

2) Efficiently estimate hardware metrics

- Fast metric estimation for white-box hardware
- Fast metric estimation for black-box hardware

Energy est. method

Lookup table approx.

3) Design efficient DNNs for various accelerators and applications

- Analysis and DNN design approach for PIM accelerators
- Novel single-shot, bottom-up architecture for panoptic segmentation
- Design approaches for efficient dense prediction applications
- New accuracy metric for panoptic segmentation
- Efficient architecture for depth estimation with hardware-oriented design

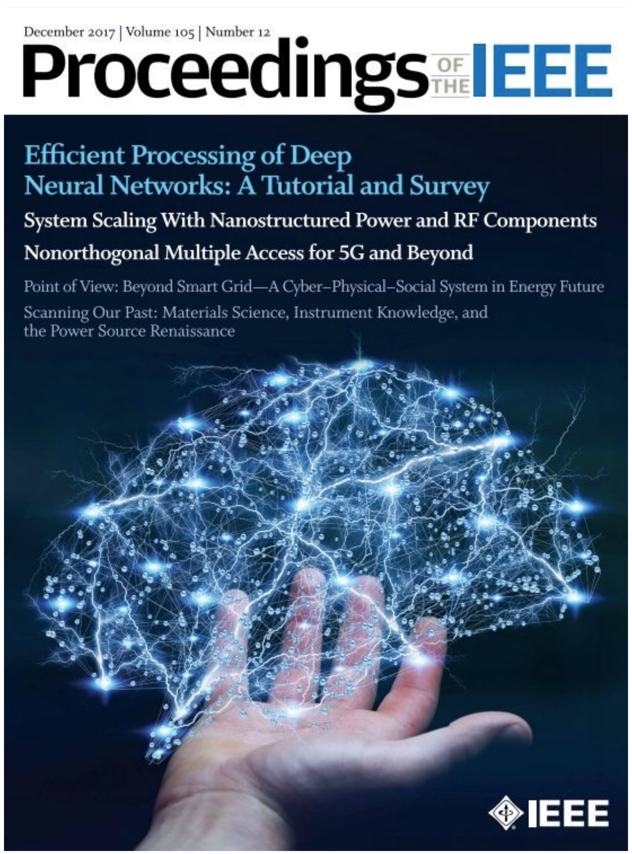
PIM accelerator

Panoptic segmentation

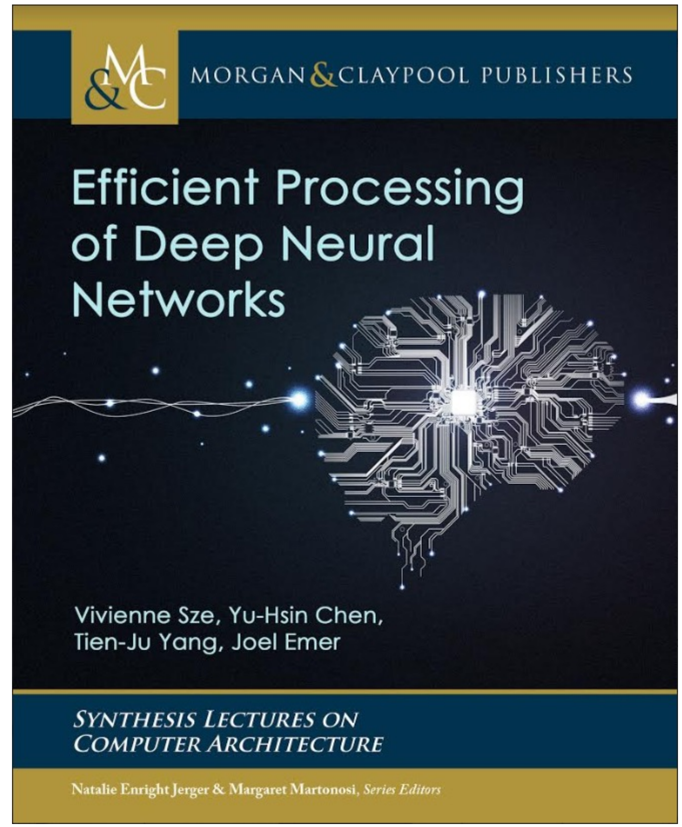
Depth estimation

Summary of Key Contributions

A survey and a book provide a structured treatment of the key principles and techniques for enabling efficient processing of DNNs



[V. Sze, Y.-H. Chen, **T.-J. Yang**, J. Emer, PIEEE 2017]



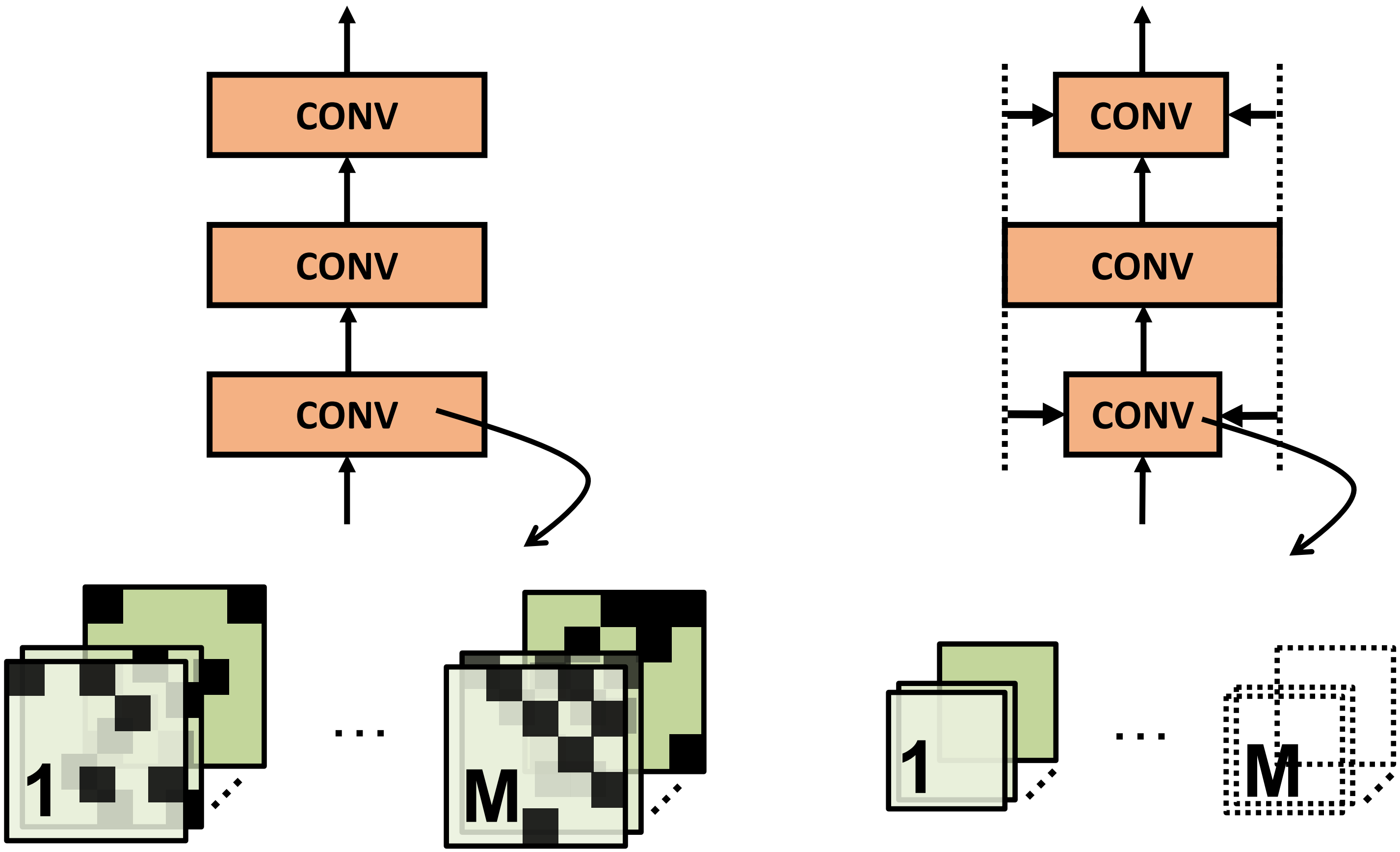
[V. Sze, Y.-H. Chen, **T.-J. Yang**, J. Emer, Morgan & Claypool 2020]

Designing Efficient DNNs with Hardware Metrics

Two Classes of Approaches

Network Pruning
(Change Weights)

Neural Architecture Search (NAS)
(Change Architecture)

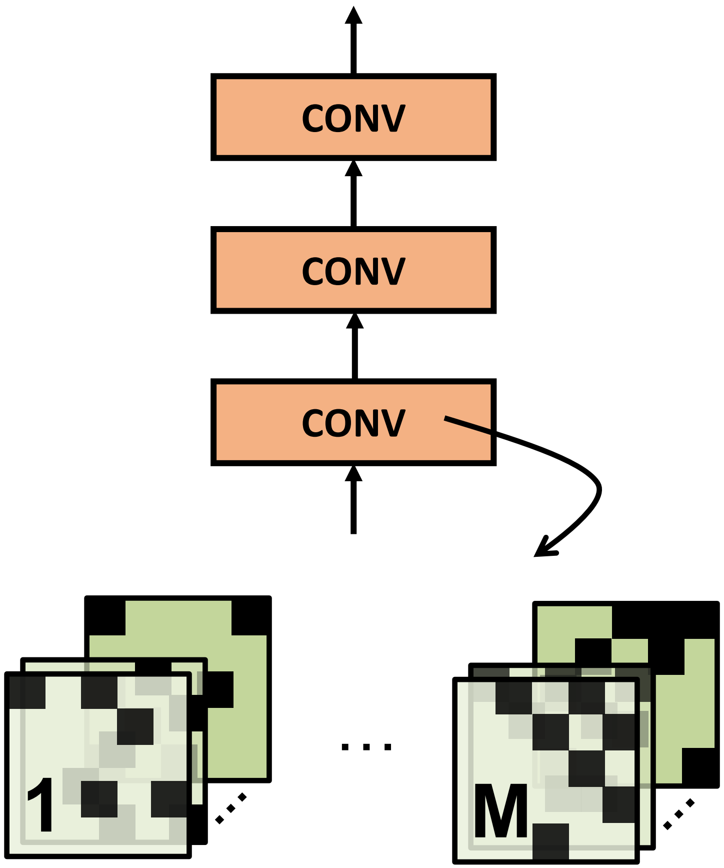


Energy-Aware Pruning

NetAdapt V1

Two Classes of Approaches

Network Pruning
(Change Weights)



Energy-Aware Pruning

Neural Architecture Search (NAS)
(Change Architecture)

NetAdapt V1

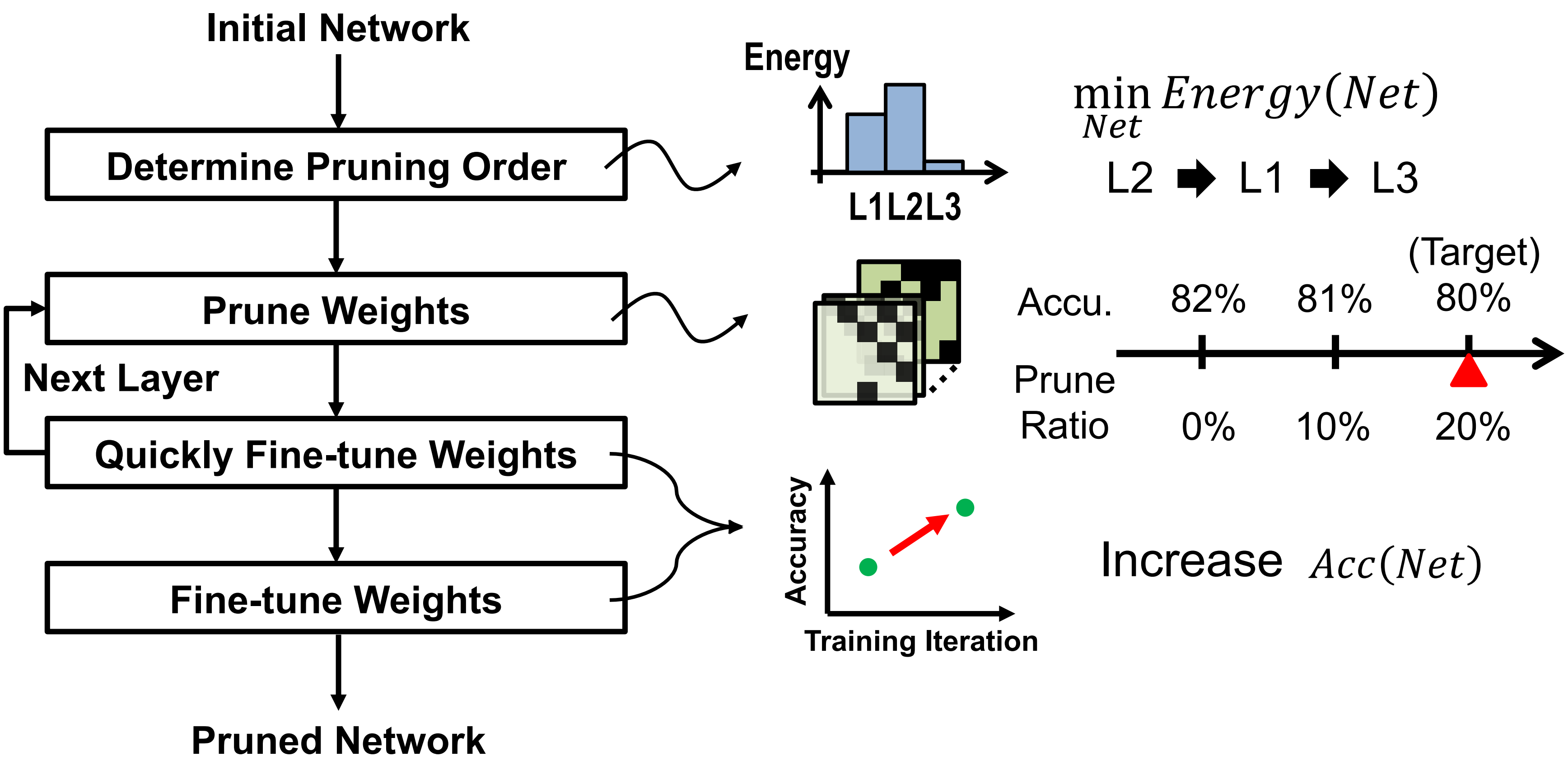
The Proposed Methods

1) Energy-Aware Pruning [CVPR 2017]
    - A hardware-aware network pruning method guided by energy

2) NetAdapt V1 [ECCV 2018]

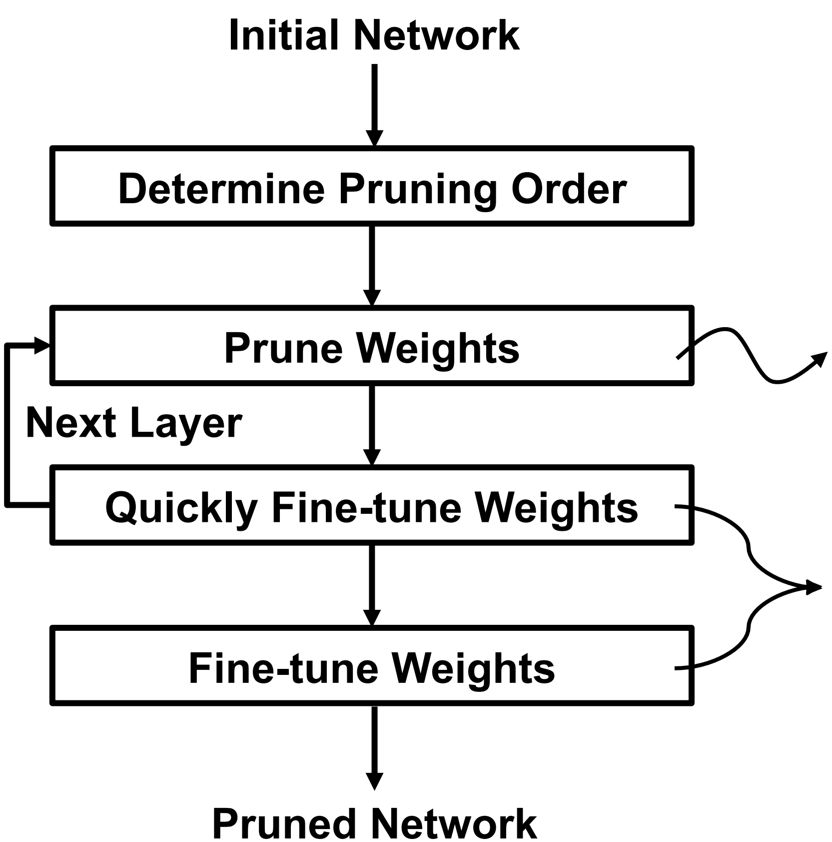
Energy-Aware Pruning (EAP)

- Problem formulation: $\min_{Net} Energy(Net)\ \ subject\ to\ Acc(Net) \geq Target$
- Reduces energy by **pruning redundant weights**
- Layer-by-layer pruning algorithm guided by **per-layer energy**



Energy-Aware Pruning (EAP)

- The two main questions to answer:

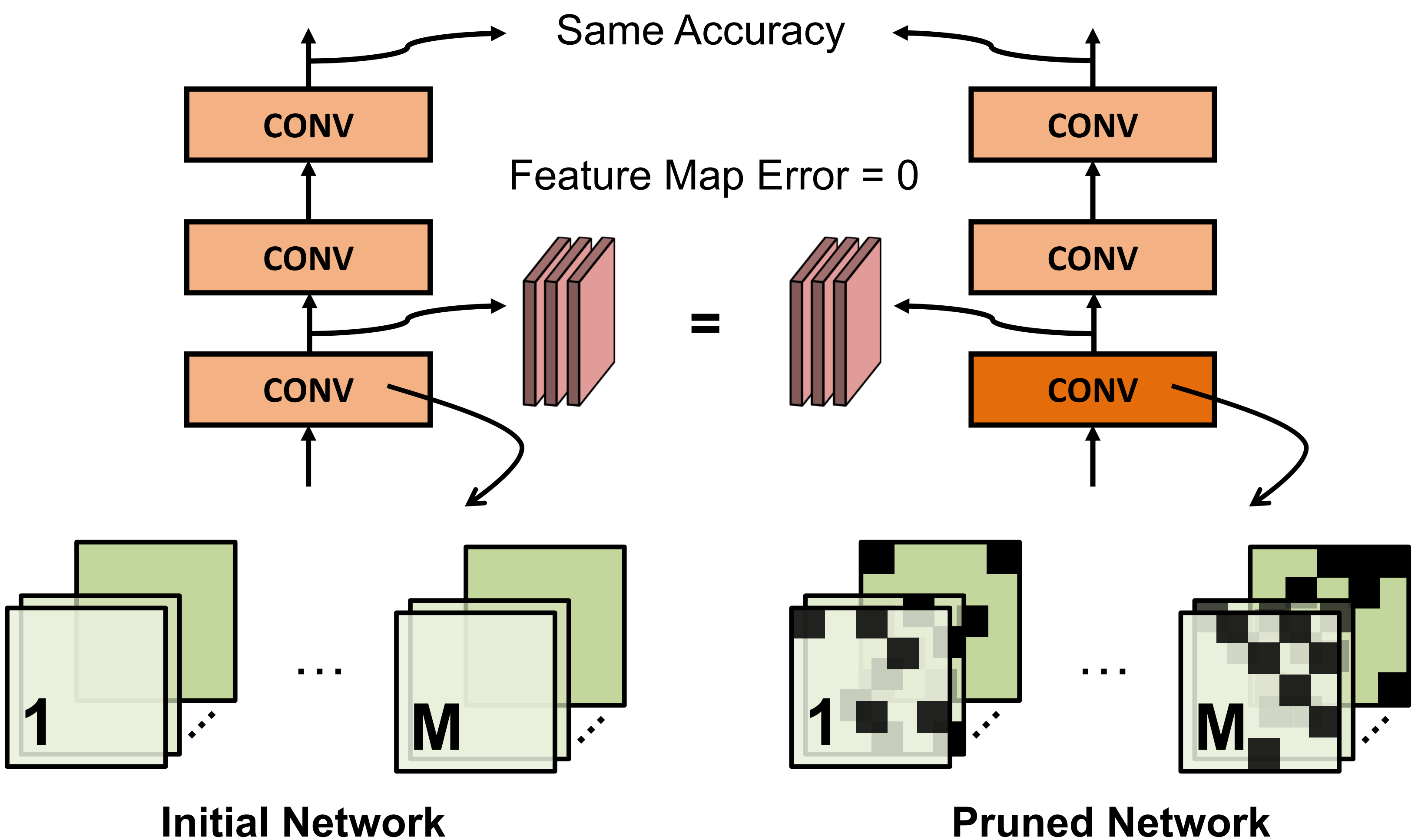


Q1) Which weights to prune?
A: Use the proposed **feature-map-based pruning**

Q2) How to speed up fine-tuning?
A: Use the proposed **local fine-tuning**

Which Weights to Prune?

If a pruned layer can generate the same feature map as that before pruning, the accuracy will be maintained



Filter-Based Method

The **filter-based** method focuses on minimizing the error in the **filters** by pruning small magnitude weights

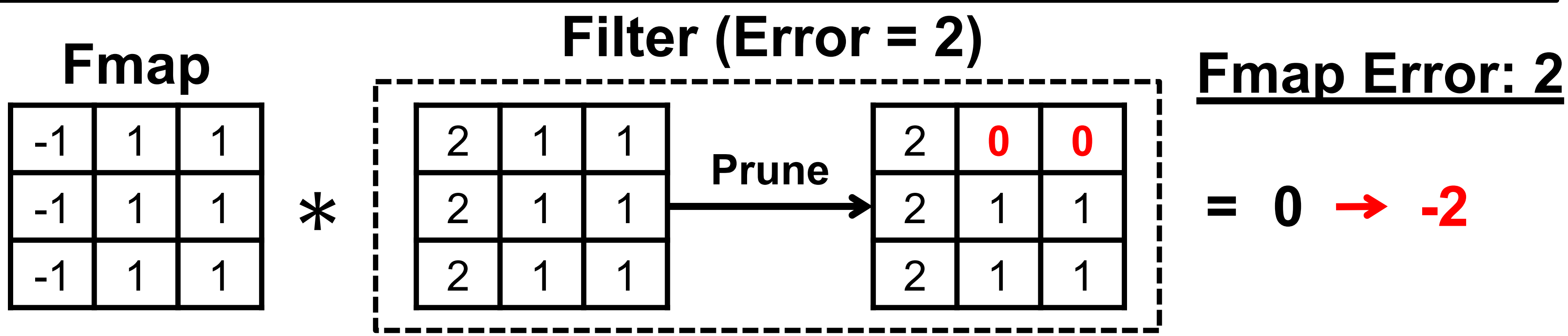
Prior work (filter-based): minimize filter error

$$\underset{\textbf{Fmap}}{\begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}} * \underset{\textbf{Filter}}{\begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}} = \mathbf{0}$$

Filter-Based Method

The **filter-based** method focuses on minimizing the error in the **filters** by pruning small magnitude weights

Prior work (filter-based): minimize filter error



Feature-Map-Based Method

The proposed **feature-map-based** method focuses on minimizing the error in the **output feature maps**

Prior work (filter-based): minimize filter error

Fmap | **Filter (Error = 2)** | **Fmap Error: 2**

$$\begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} * \left(\begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} \xrightarrow{\text{Prune}} \begin{bmatrix} 2 & 0 & 0 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} \right) = 0 \rightarrow -2$$

Our method (feature-map-based): minimize feature map error

Fmap | **Filter**

$$\begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} = 0$$

Feature-Map-Based Method

The proposed **feature-map-based** method focuses on minimizing the error in the **output feature maps**

Prior work (filter-based): minimize filter error

Fmap   **Filter (Error = 2)**   **Fmap Error: 2**

$$\begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} \xrightarrow{\text{Prune}} \begin{bmatrix} 2 & \color{red}{0} & \color{red}{0} \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} = 0 \rightarrow \color{red}{-2}$$

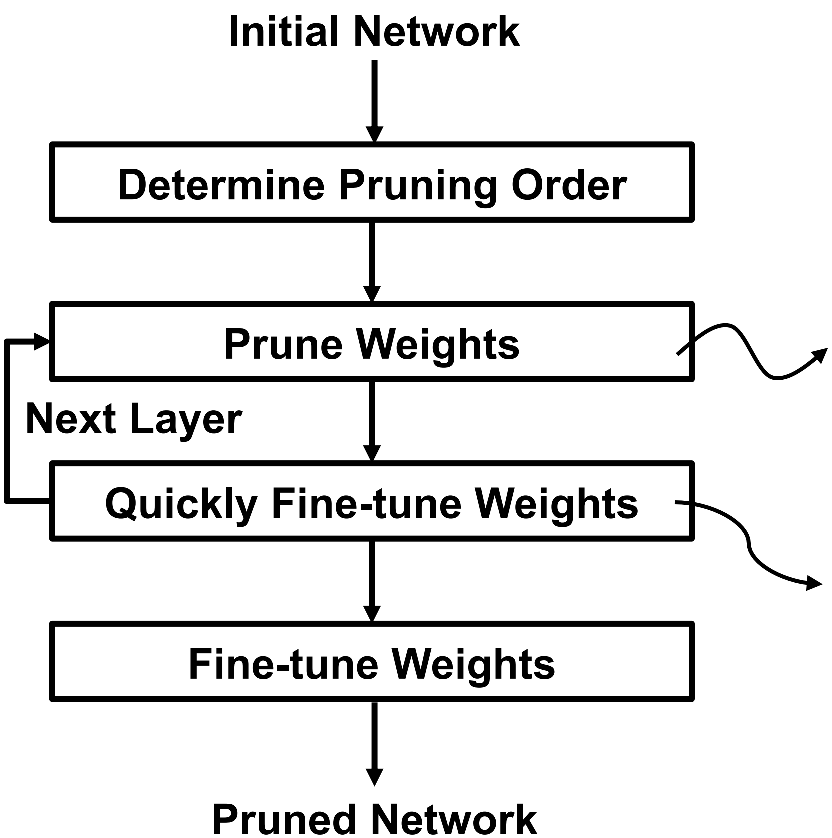
Our method (feature-map-based): minimize feature map error

Fmap   **Filter (Error = 3)**   **Fmap Error: 1**

$$\begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} \xrightarrow{\text{Prune}} \begin{bmatrix} \color{red}{0} & \color{red}{0} & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} = 0 \rightarrow \color{red}{1}$$

Energy-Aware Pruning (EAP)

- The two main questions to answer:



Q1) How to determine which weights to prune?
A: Use the proposed **feature-map-based pruning**

Q2) How to speed up fine-tuning?
A: Use the proposed **local fine-tuning**

Local Fine-Tuning (LFT)

LFT minimizes the **feature map error** by fine-tuning the non-pruned weights, which has a closed-form solution and is fast

After Pruning

$$\underset{\textbf{fmap}}{\begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}} * \underset{\textbf{filter}}{\boxed{\begin{bmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} \xrightarrow{\textbf{Prune}} \begin{bmatrix} \color{red}{0} & \color{red}{0} & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix}}}$$

Fmap Error: 1

$= 0 \rightarrow \color{red}{1}$

After Local Fine-Tuning

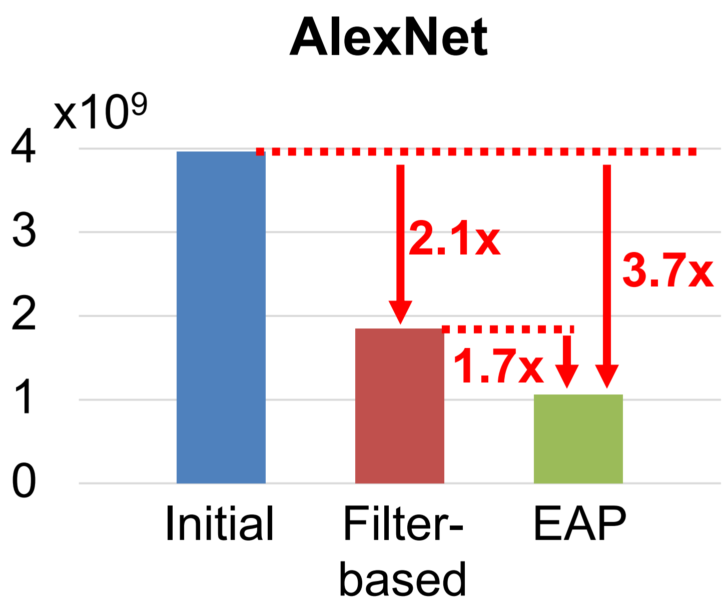
$$\underset{\textbf{fmap}}{\begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}} * \underset{\textbf{filter}}{\boxed{\begin{bmatrix} 0 & 0 & 1 \\ 2 & 1 & 1 \\ 2 & 1 & 1 \end{bmatrix} \xrightarrow{\textbf{Fine-tune}} \begin{bmatrix} 0 & 0 & 1 \\ 2 & 1 & 1 \\ \color{red}{3} & 1 & 1 \end{bmatrix}}}$$

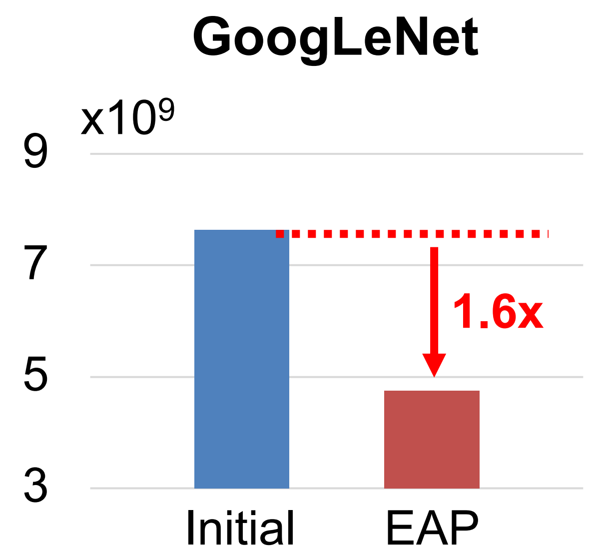
Fmap Error: 0

$= 1 \rightarrow \color{red}{0}$

Results of EAP

- EAP achieves **3.7x (1.6x)** energy reduction for AlexNet (GoogLeNet) with comparable accuracy
- EAP outperforms the filter-based method by **1.7x** with comparable accuracy





Normalized Energy

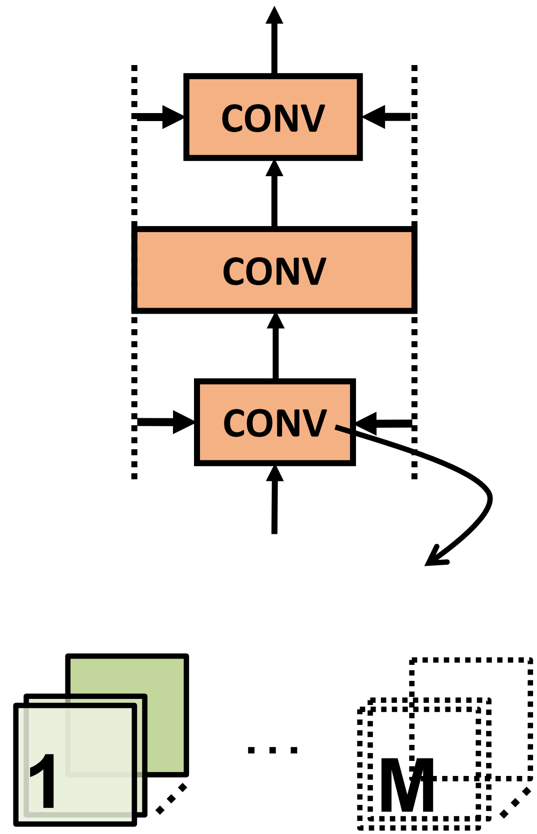
Dataset: ImageNet

Two Classes of Approaches

Network Pruning
(Change Weights)

Energy-Aware Pruning

Neural Architecture Search (NAS)
(Change Architecture)



NetAdapt V1

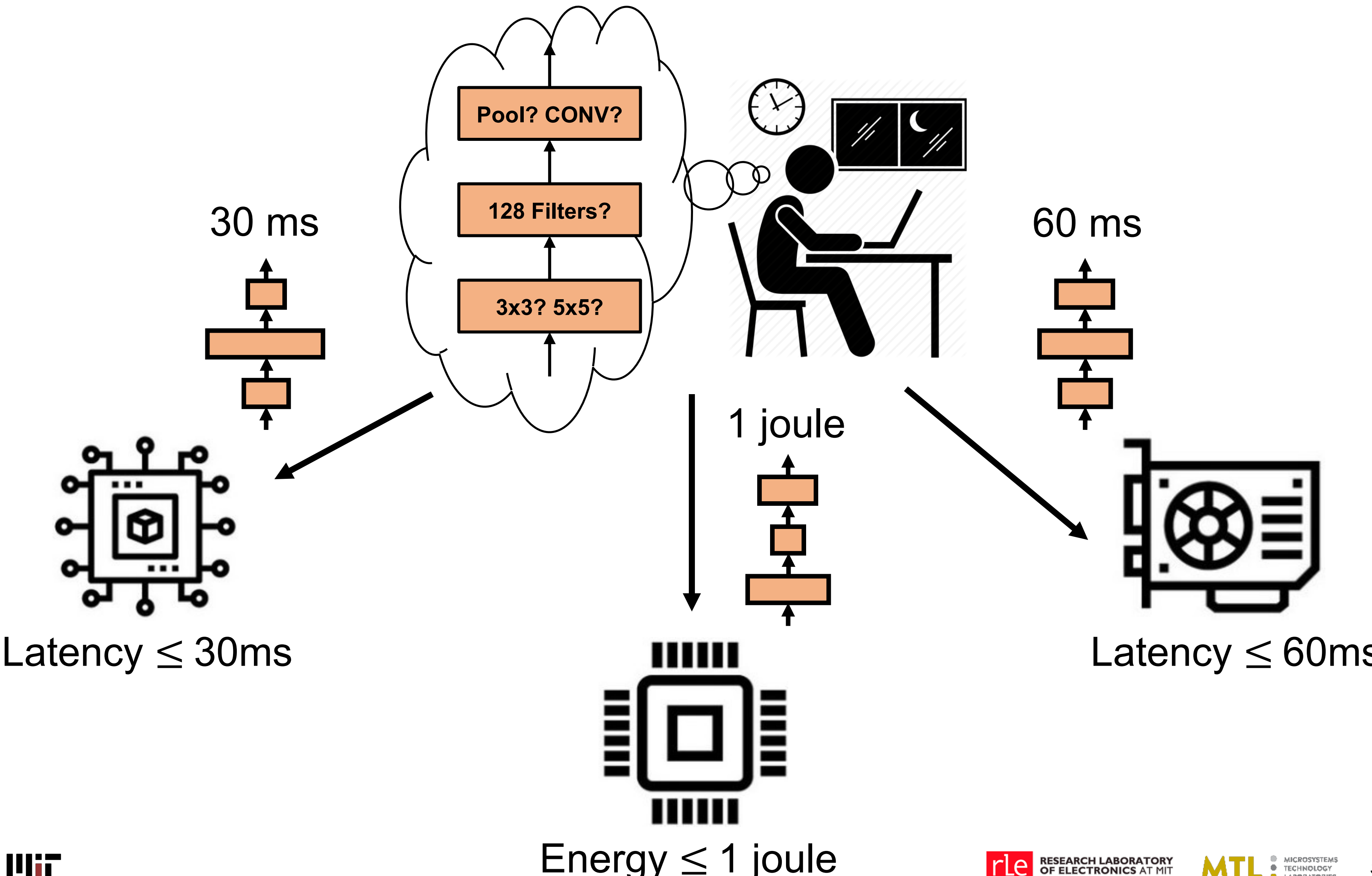
The Proposed Methods

1) Energy-Aware Pruning [CVPR 2017]

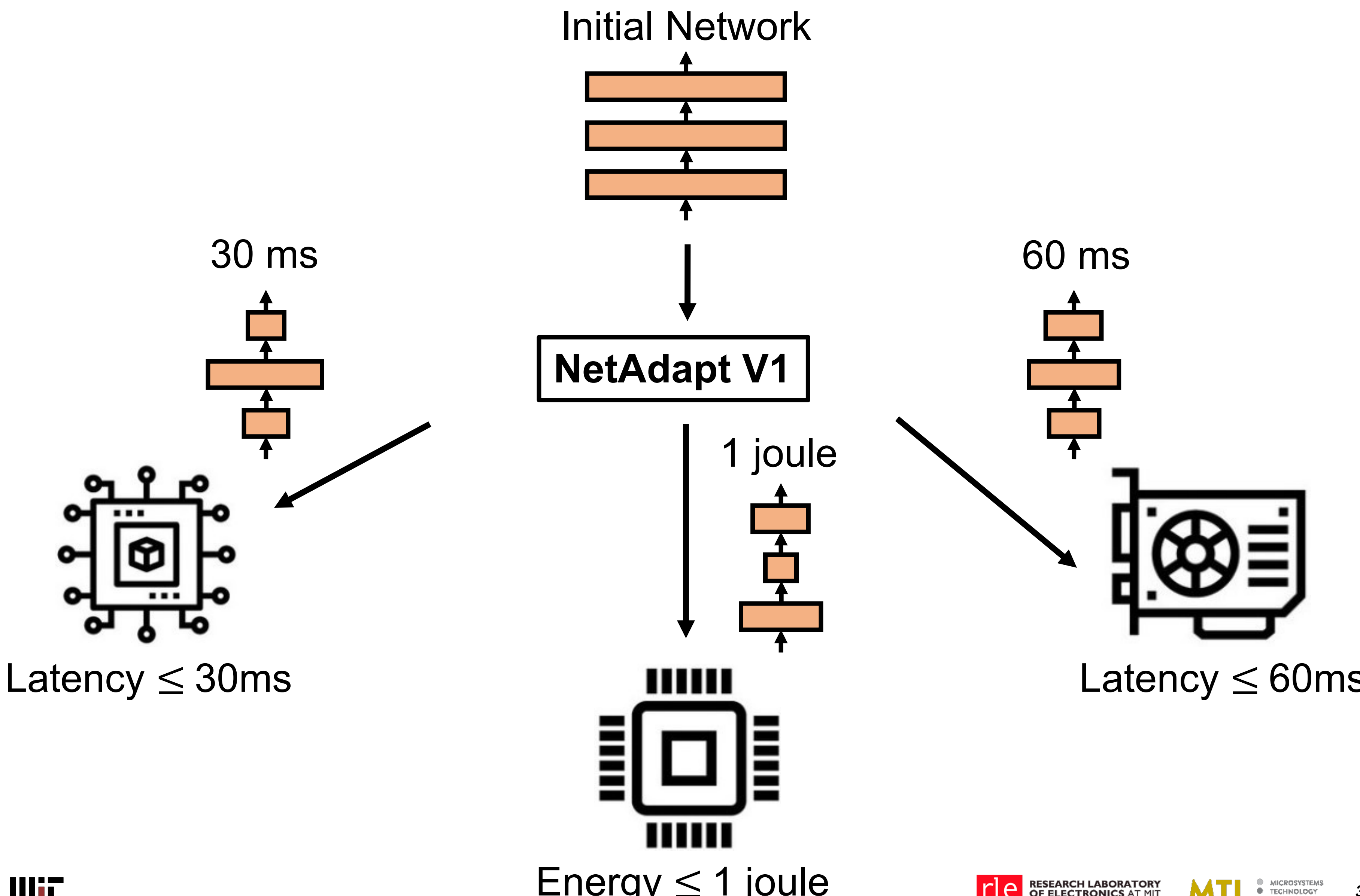
2) NetAdapt V1 [ECCV 2018]

- A hardware-aware neural architecture search method guided by latency

DNN Design with Resource Constraints



Automatic Design with NetAdapt V1



Formulation of NetAdapt V1

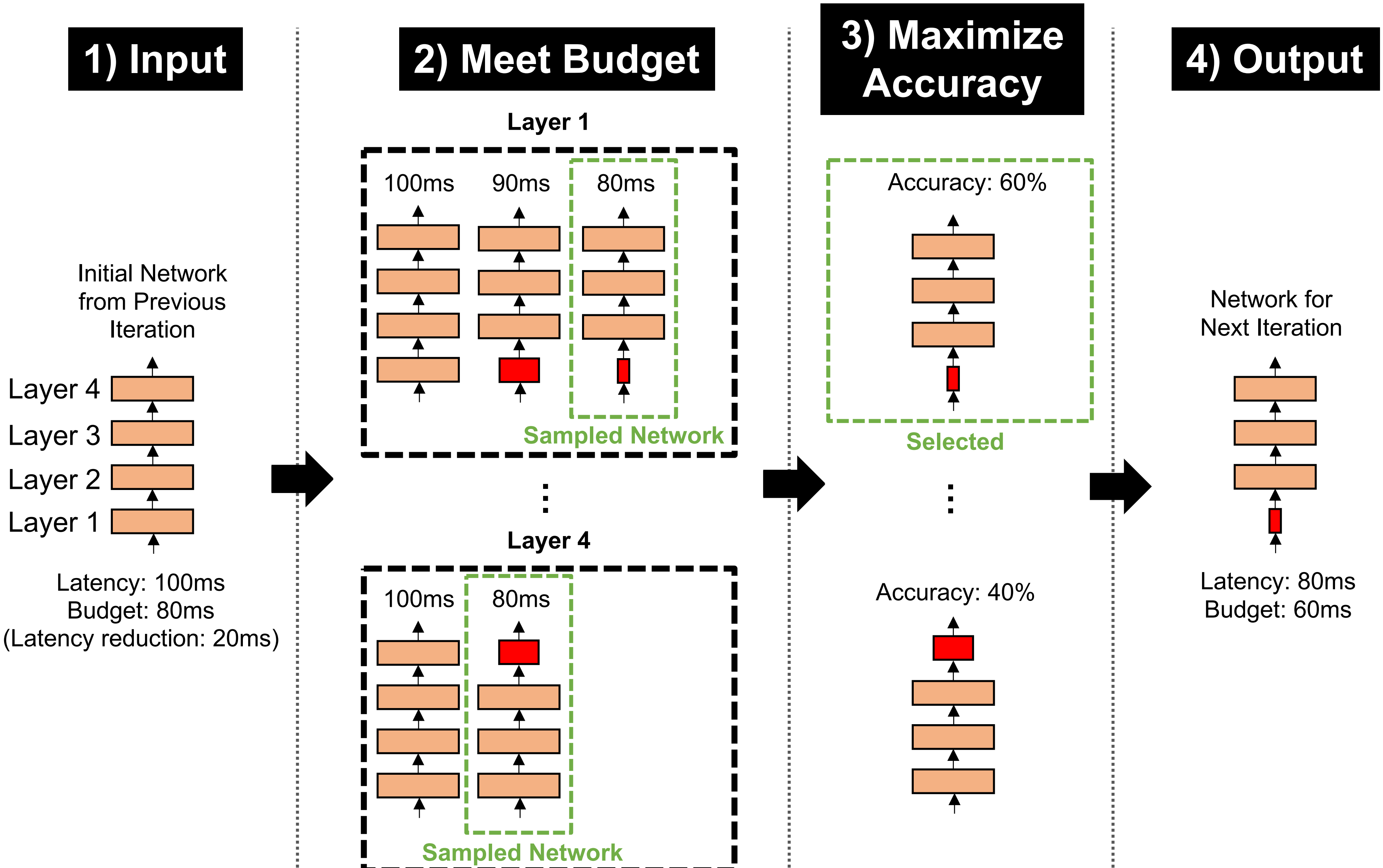
$$\max_{Net} Acc(Net) \;\; subject\ to\ Res_j(Net) \leq Bud_j, j = 1, \cdots, m$$

- Break into a set of simpler problems and solve iteratively
- Remove filters from a single layer per iteration

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

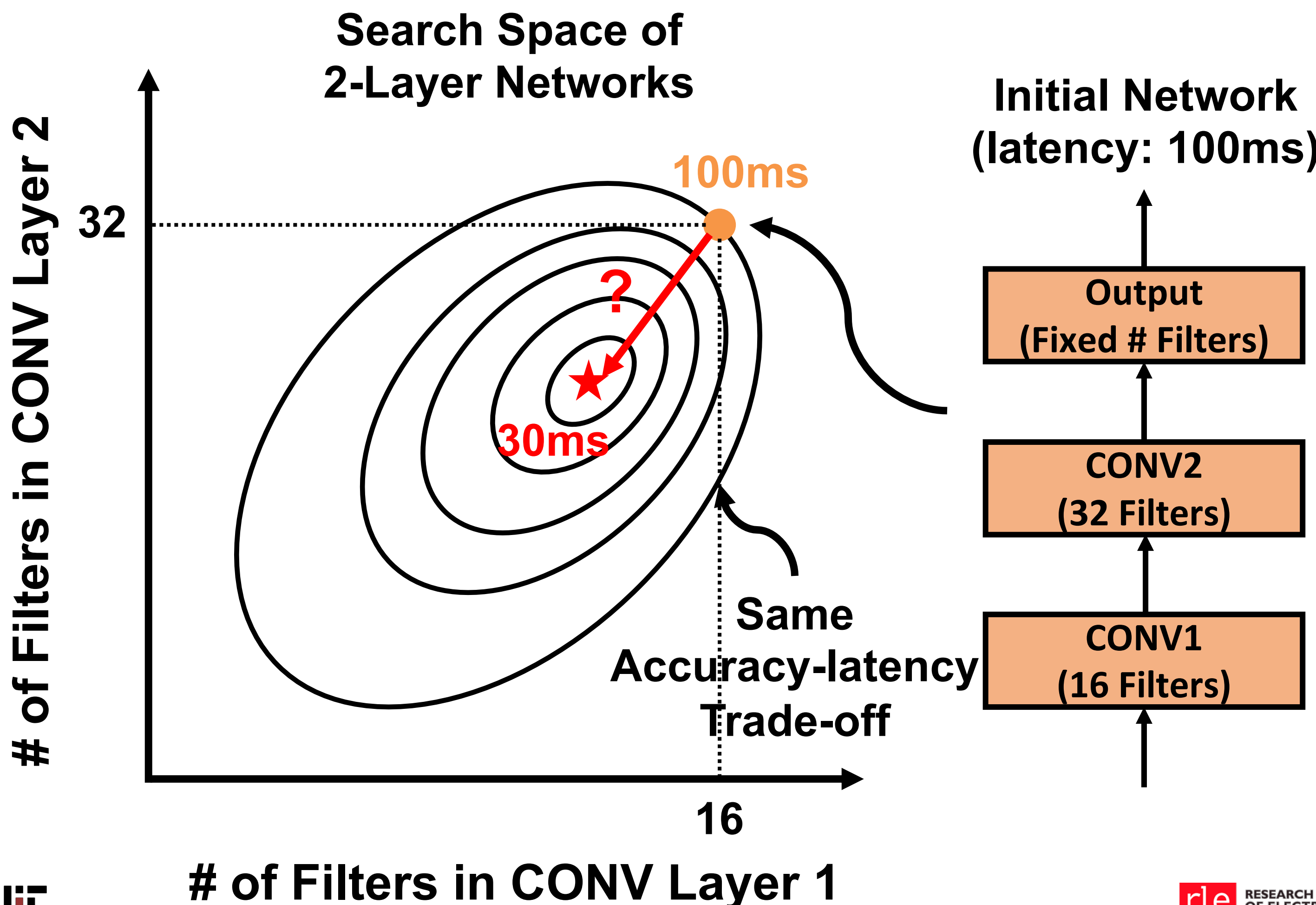
**Acc*: accuracy function, *Res*: resource evaluation function,
***ΔR*: resource reduction**, *Bud*: given budget

Simplified Example of One Iteration



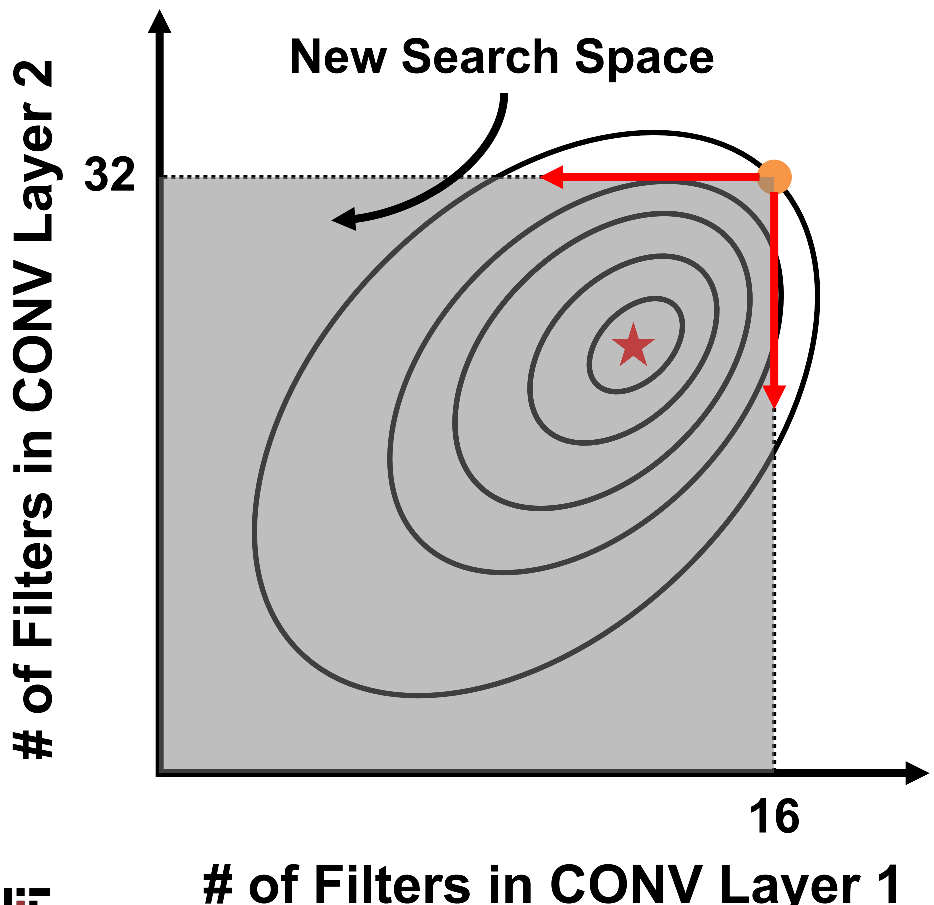
Single-Layer Coordinate Descent

- We can view this process as performing single-layer coordinate descent



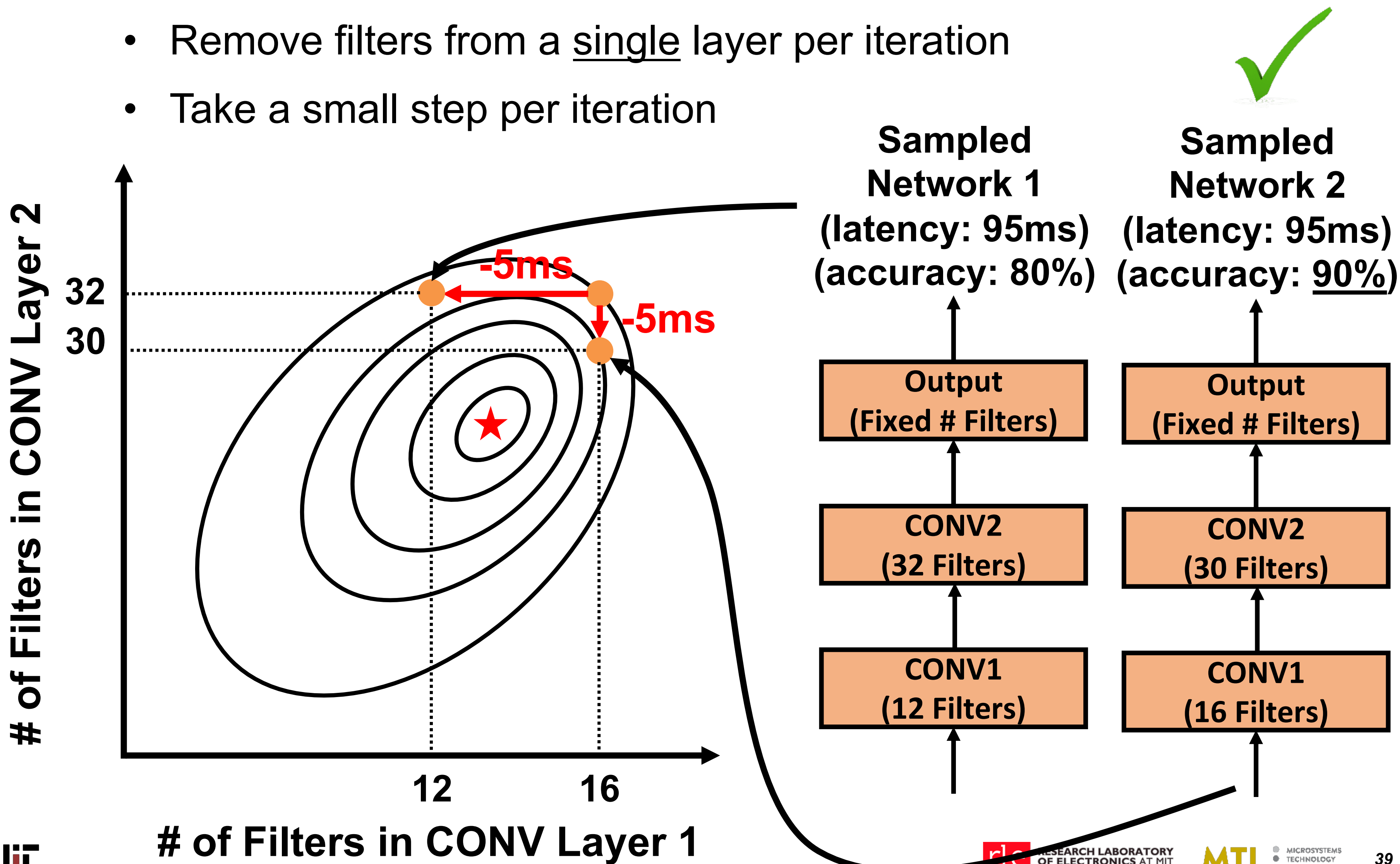
Single-Layer Coordinate Descent

- Two constraints of this iterative optimizer:
  - Remove filters from a single layer per iteration



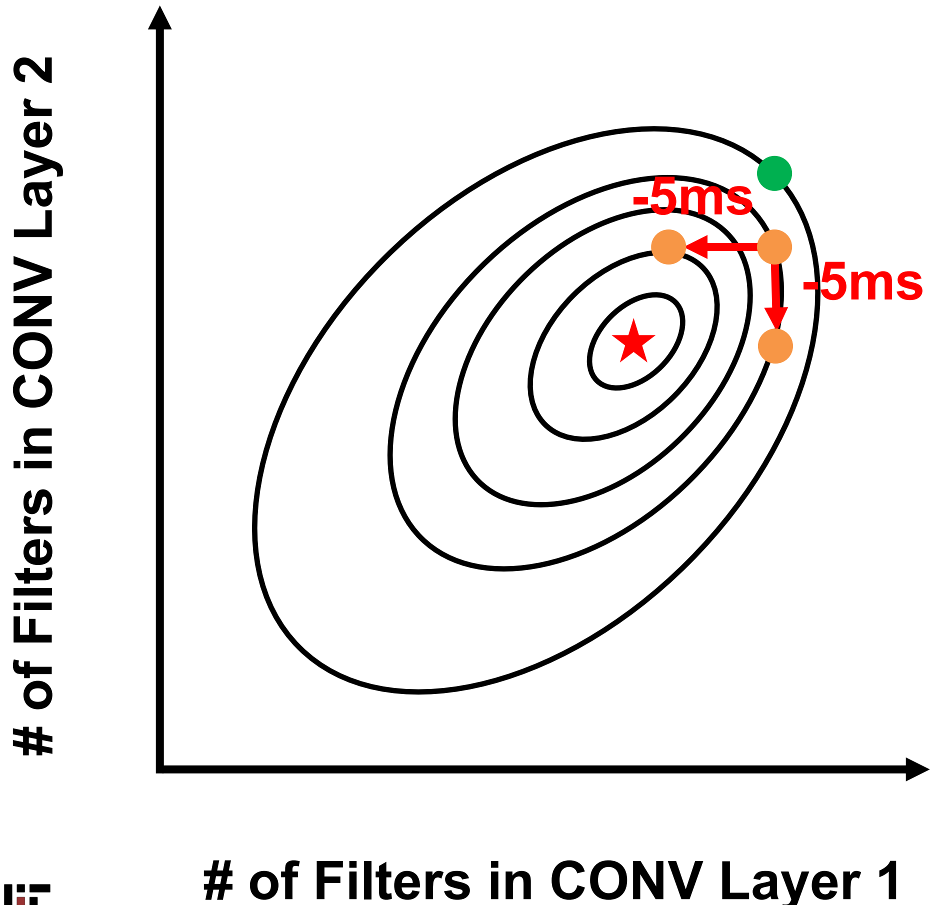
Single-Layer Coordinate Descent

- Two constraints of this iterative optimizer:
  - Remove filters from a single layer per iteration
  - Take a small step per iteration



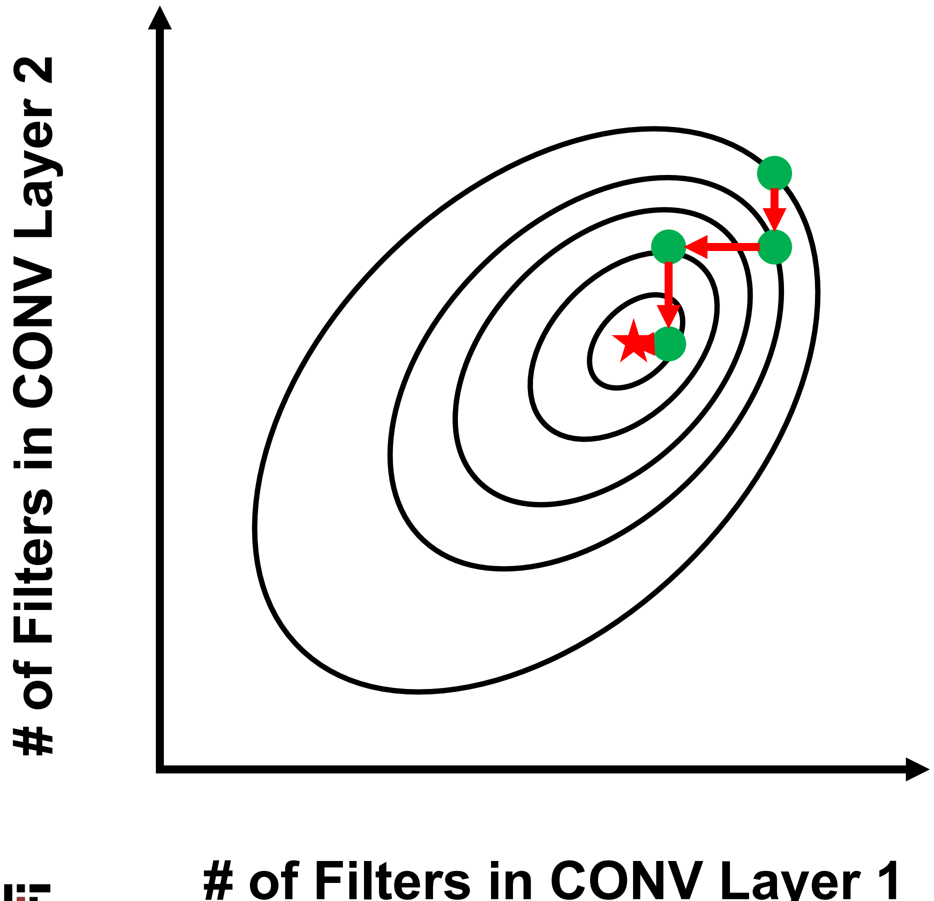
Single-Layer Coordinate Descent

- Move to the next location in the search space and perform the same process again



Single-Layer Coordinate Descent

- This process continues until the given resource budget (30 ms in this example) is satisfied

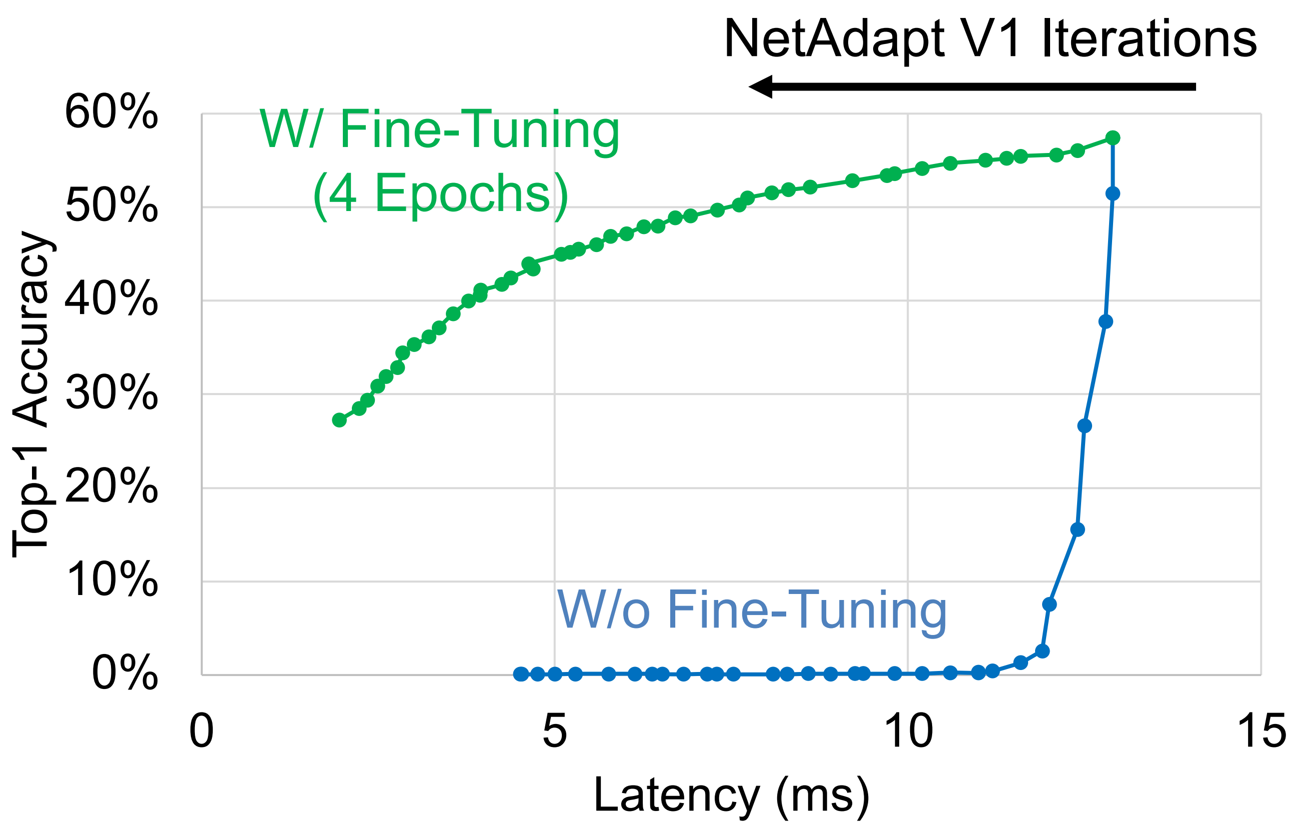


NetAdapt V1

- **Advantages of NetAdapt V1**
  - Supports non-differentiable metrics
  - 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 accuracy versus resource trade-offs
  - Adds only a few extra hyper-parameters

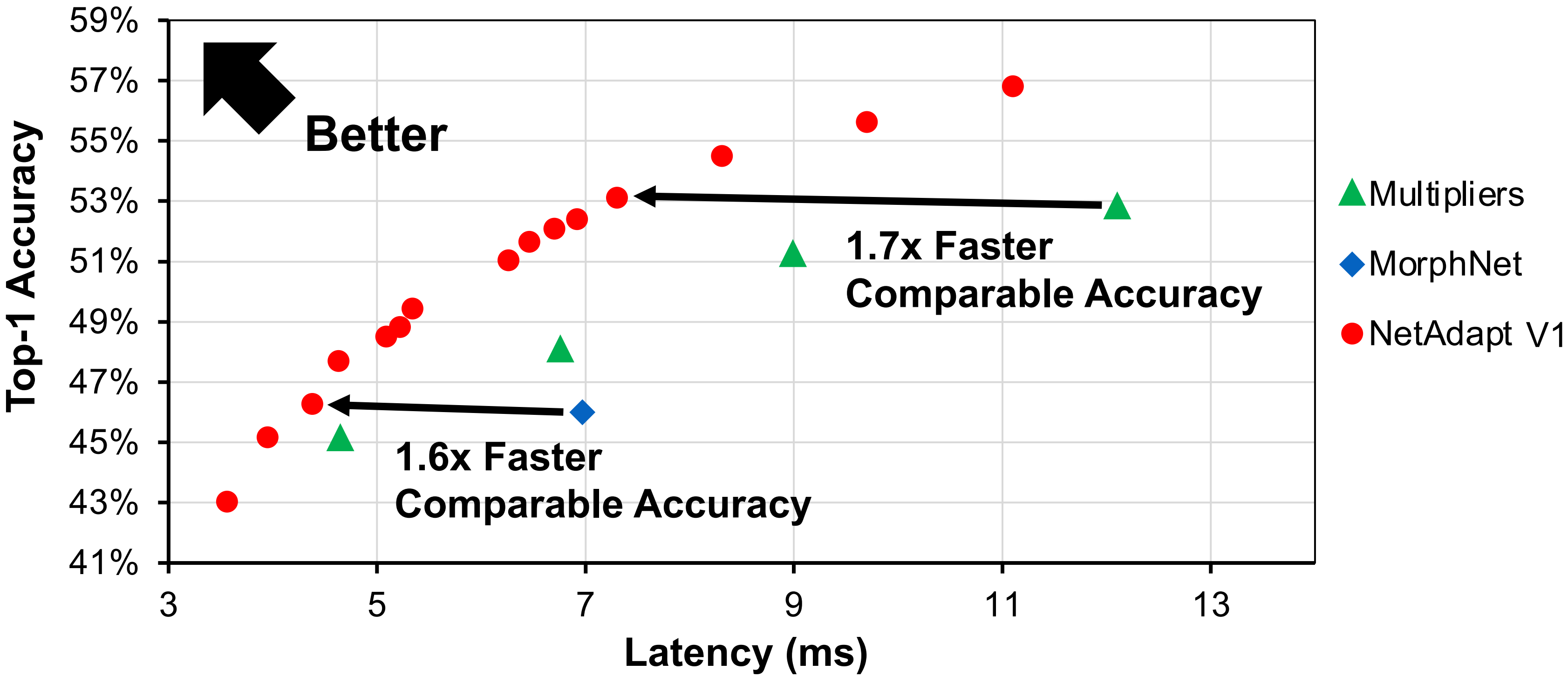
Short-Term Fine-Tuning

- NetAdapt V1 also fine-tunes every sampled network for a few epochs (i.e., short-term) to restore the accuracy
- Otherwise, the accuracy will quickly drop to zero and lead to wrong network selection



Results on Image Classification

NetAdapt V1 achieves up to **1.7x** faster with comparable accuracy than previous works



- Initial network: MobileNet V1
- Dataset: ImageNet
- Multipliers: [Howard, arXiv 2017]; MorphNet: [Gordon, CVPR 2018]
- Single large core of Pixel 1 CPU

Using Hardware Metrics is Critical

- If NetAdapt V1 was guided by the number of MACs, it would also achieve a better accuracy-MAC trade-off

Network	Top-1 Accuracy	# of MACs (M)
MobileNet V1	45.1% (+0%)	13.6 (100%)
NetAdapt V1	**46.3% (+1.2%)**	**11.0 (81%)**

Using Hardware Metrics is Critical

- If NetAdapt V1 was guided by the number of MACs, it would also achieve a better accuracy-MAC trade-off
- However, it does not mean lower latency
- It is important to incorporate hardware metrics rather than proxy metrics into the design of DNNs

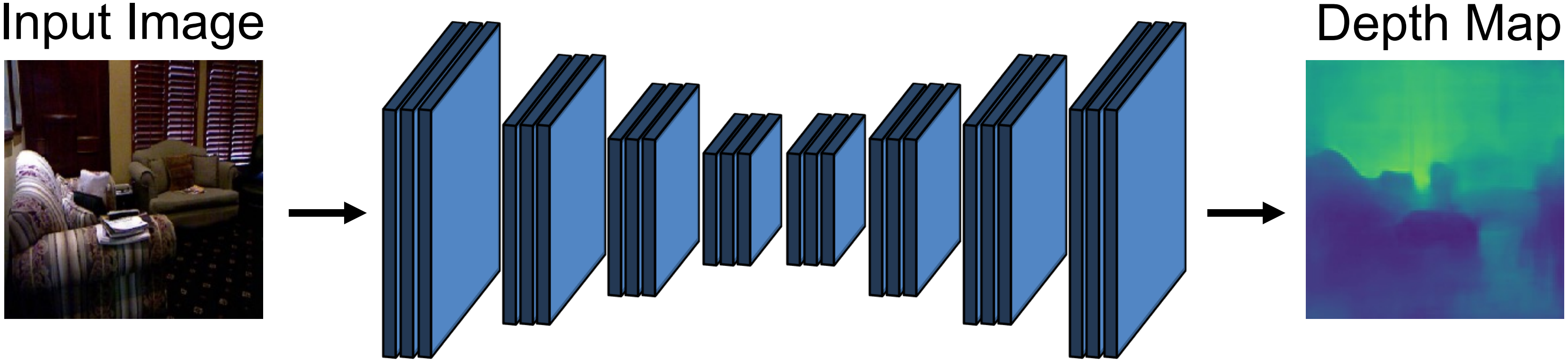
Network	Top-1 Accuracy	# of MACs (M)	Latency (ms)
MobileNet V1	45.1% (+0%)	13.6 (100%)	4.65 (100%)
NetAdapt V1	**46.3% (+1.2%)**	**11.0 (81%)**	**6.01 (129%)**

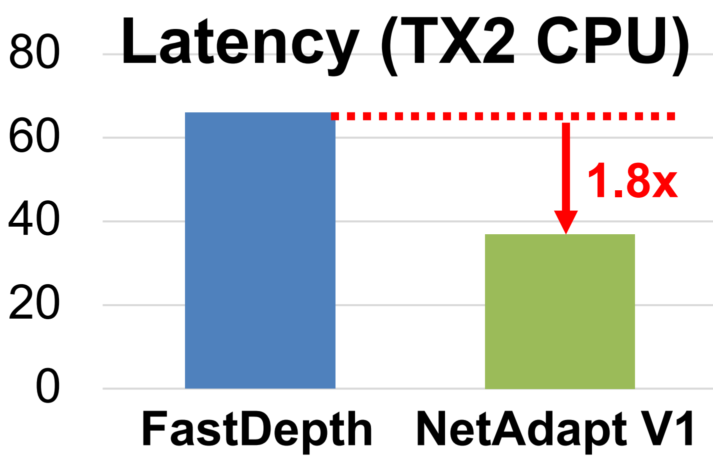
Results on Depth Estimation

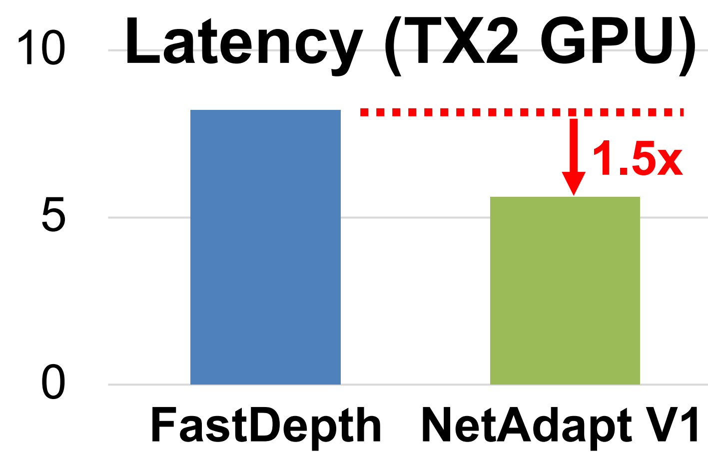
NetAdapt V1 reduces the latency of FastDepth by **1.8x** on Jetson TX2 CPU and **1.5x** on Jetson TX2 GPU

Input Image

Depth Map



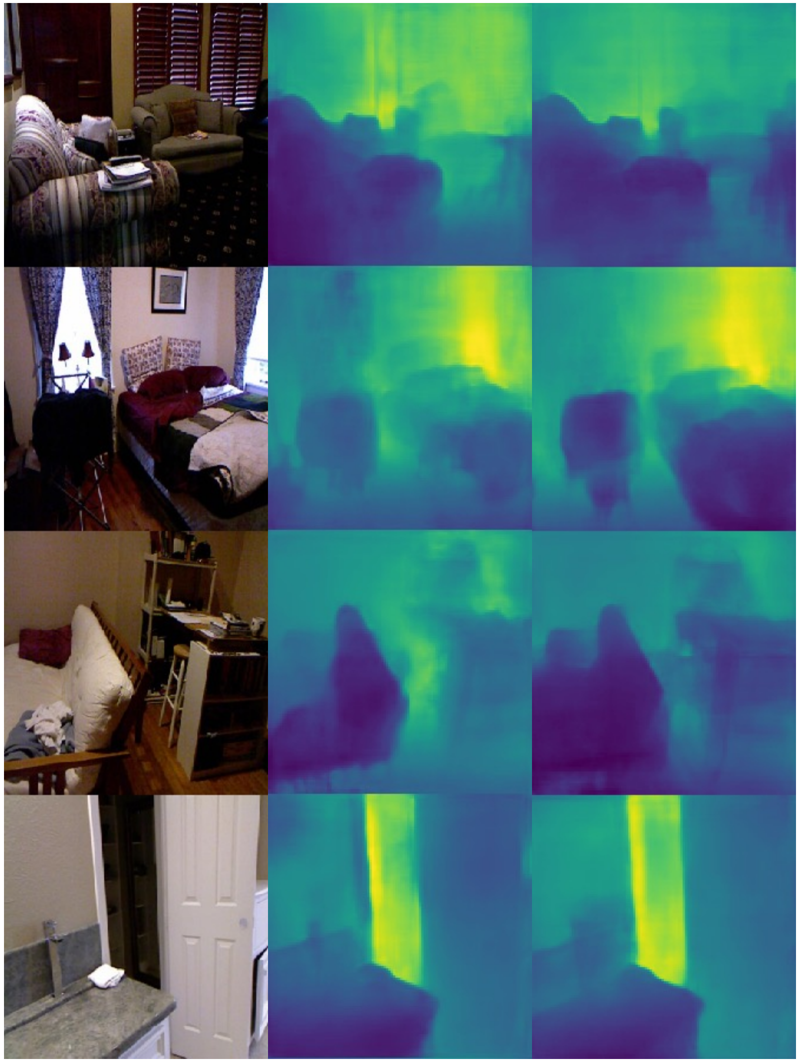




FastDepth: [Wofk*, Ma* ICRA 2019]

Results on Depth Estimation

NetAdapt V1 preserves the sharpness and visual clarity of the output depth maps



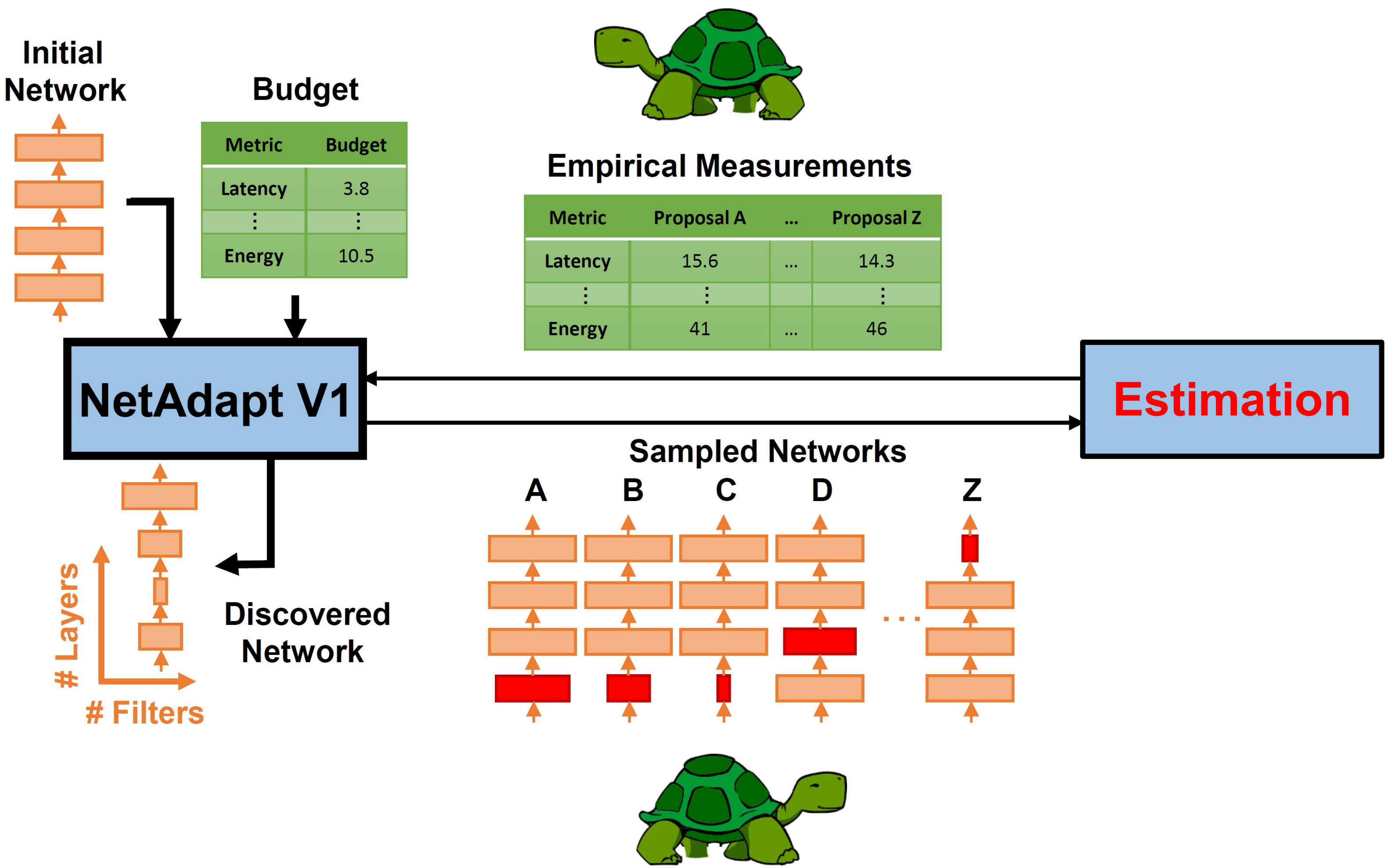
Summary

- Using hardware metrics is the key to obtaining better accuracy-efficiency trade-off

- We proposed two methods guided by hardware metrics to significantly improve the efficiency of DNNs
  - **Energy-aware pruning**
    - A network pruning method guided by energy
    - It targets at minimizing the difference in the feature maps rather than that in filters to improve accuracy-efficiency trade-off
  - **NetAdapt V1**
    - A NAS method guided by latency
    - It uses the simple-yet-effective coordinate descent optimizer to automatically and progressively search for networks with better accuracy-efficiency trade-off

Efficient Methods for Estimating Hardware Metrics

Metric Evaluation can be Slow



Metric	Budget
Latency	3.8
⋮	⋮
Energy	10.5

Metric	Proposal A	...	Proposal Z
Latency	15.6	...	14.3
⋮	⋮		⋮
Energy	41	...	46

Two Use Cases

Know how the target hardware processes DNNs?

1) **Yes:** energy estimation methodology [CVPR 2017, Asilomar 2017]

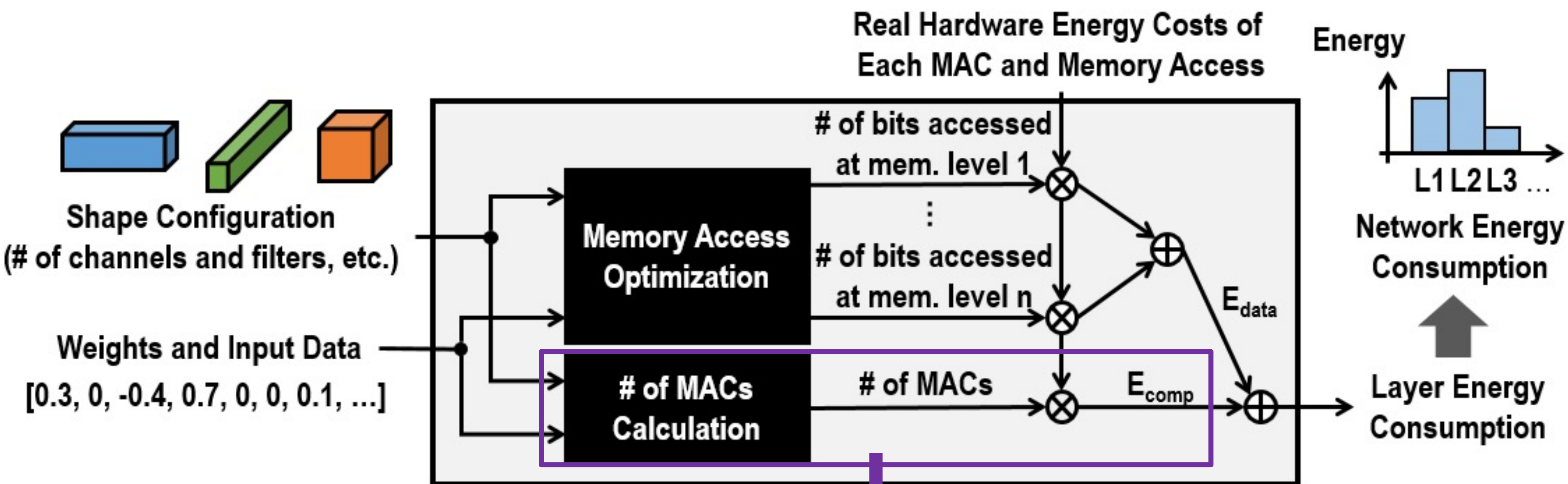
- Can be used for hardware that is still in the early design phase and has not been fabricated yet

2) **No:** lookup-table approximation [ECCV 2018]

- Can be used for proprietary, off-the-shelf hardware

Energy Estimation Methodology

- Estimate the energy consumption of each layer separately
- For each layer, $E_{layer} = \boxed{E_{comp}} + E_{data}$

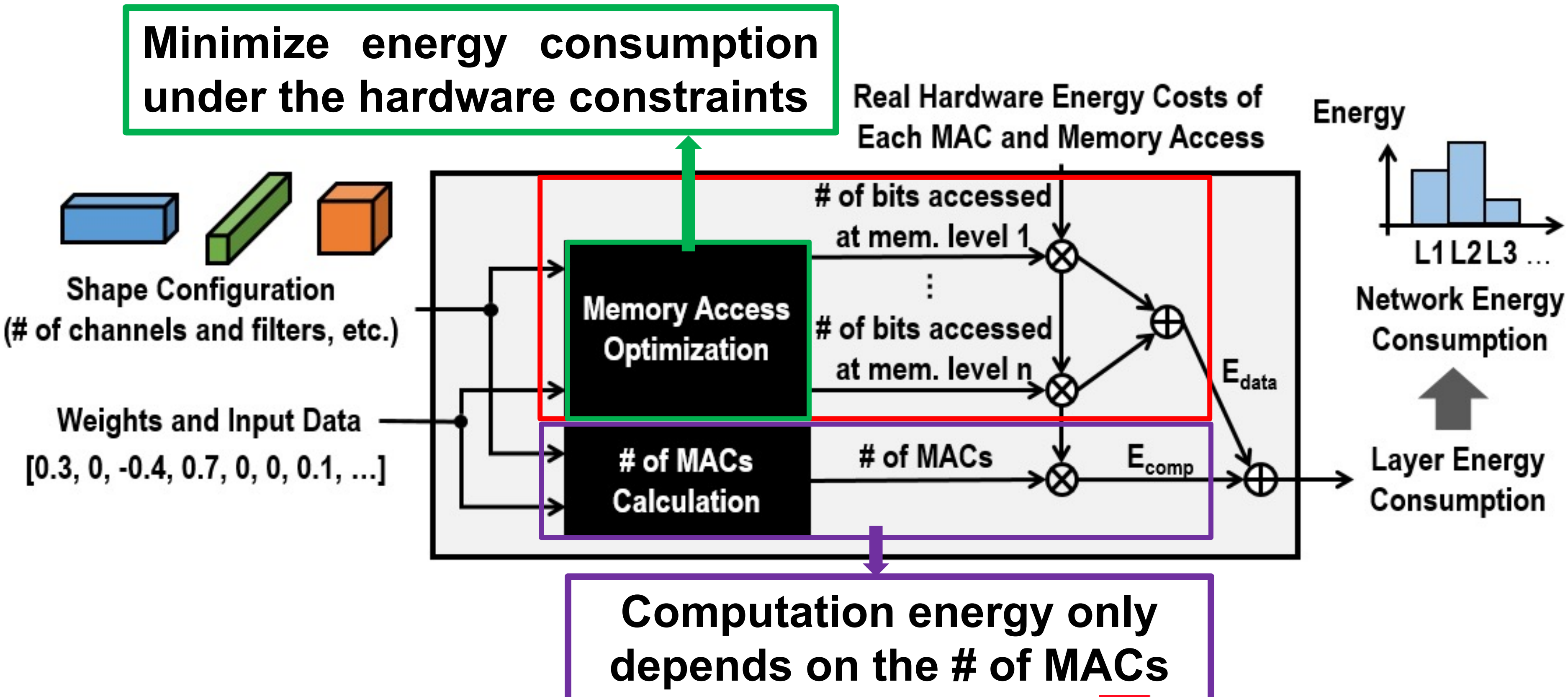


Computation energy only depends on the # of MACs

[Collaborate with Yu-Hsin Chen]

Energy Estimation Methodology

- Estimate the energy consumption of each layer separately
- For each layer, $E_{layer} = \boxed{E_{comp}} + \boxed{E_{data}}$



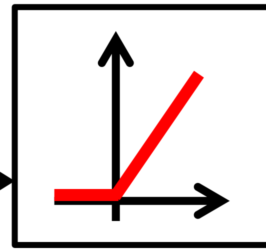
Factor in Sparsity

Apply Non-Linearity **ReLU** on Feature Maps

Fmap

9	-1	-3
1	-5	5
-2	6	-1

ReLU



Fmap

9	**0**	**0**
1	**0**	5
0	6	**0**

Pruned Network Filters

Fmap

-1	1	1
-1	1	1
-1	1	1

*

Filter

2	1	1
2	1	1
2	1	1

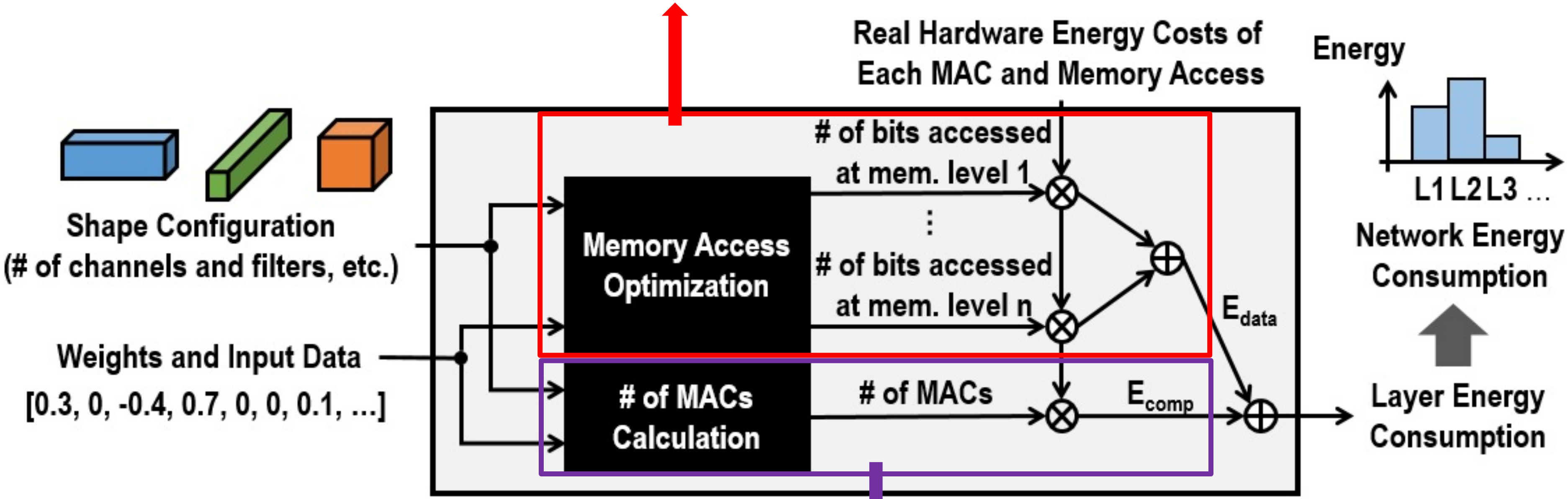
Prune →

0	**0**	1
2	1	1
2	1	1

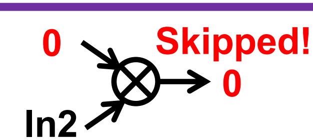
= **1**

Factor in Sparsity

- Use data compression to reduce the # of bits accessed
- Consider sparsity in the memory access optimization

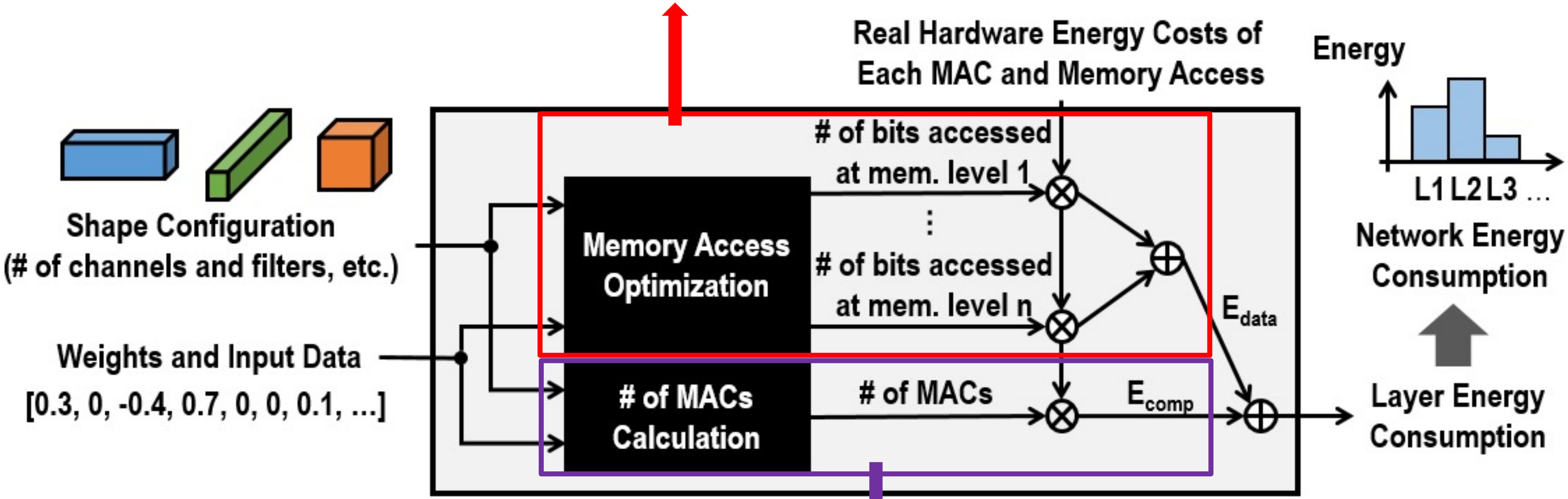


Skip the MAC when at least one input is zero

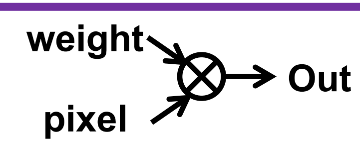


Factor in Bitwidth

- Scale # of bits accessed linearly with the bitwidth
- Consider bitwidths in the memory access optimization

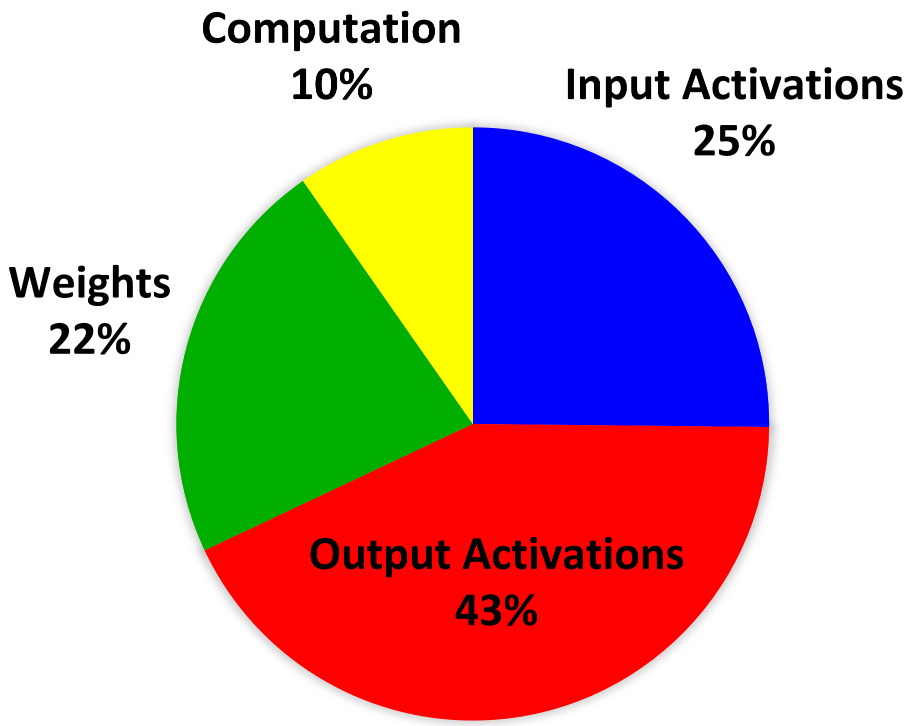


Computation energy scales linearly with the bitwidth of each input



Estimated Energy

- Data movement, not computation, dominates the energy
- The movement of activations needs to be considered



GoogLeNet

On Eyeriss V1 [ISSCC 2016]

Two Use Cases

Know how the target hardware processes DNNs?

1) **Yes:** energy estimation methodology [CVPR 2017, Asilomar 2017]

- Can be used for hardware that is still in the early design phase and has not been fabricated yet

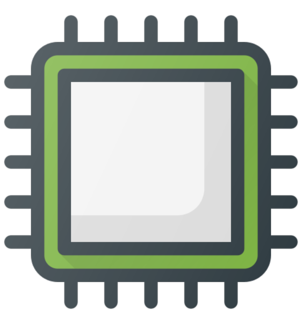
2) **No:** lookup-table approximation [ECCV 2018]

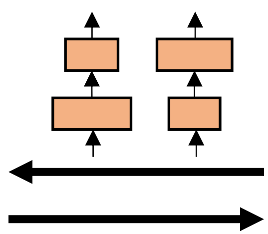
- A common case when using proprietary, off-the-shelf hardware

Lookup-Table Approximation

- We propose using <u>per-layer lookup tables</u>
- Estimate the network latency by the sum of per-layer latency
- The lookup tables only need to be built once and can be used multiple times
- Why per-layer instead of per-network?
  - The size of the per-network table grows exponentially with # of layers
    - 10 layers + 10 shapes/layer → per-network: 10^{10} entries, per-layer: 100 entries
  - The layers with the same shape only need to be measured once

Target Hardware



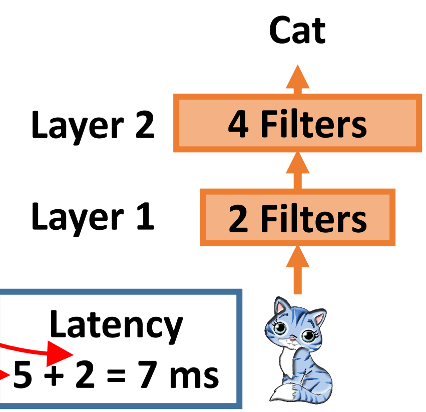


Layer 1

# Filters \ # Channels	1	2	3
2	1	3	5
4	2	4	6
6	3	5	7
8	4	6	8

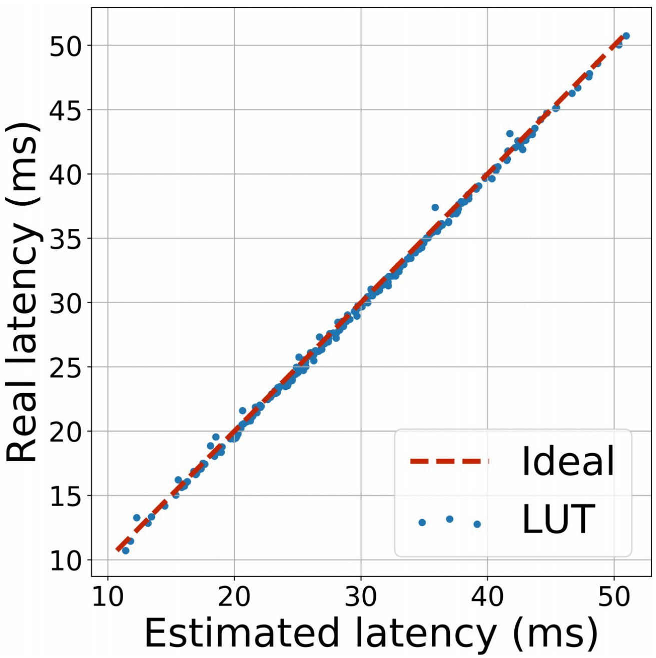
Layer 2

# Filters \ # Channels	2	4	6	8
2	1	2	3	4
4	2	3	4	5
6	3	4	5	6
8	4	5	6	7

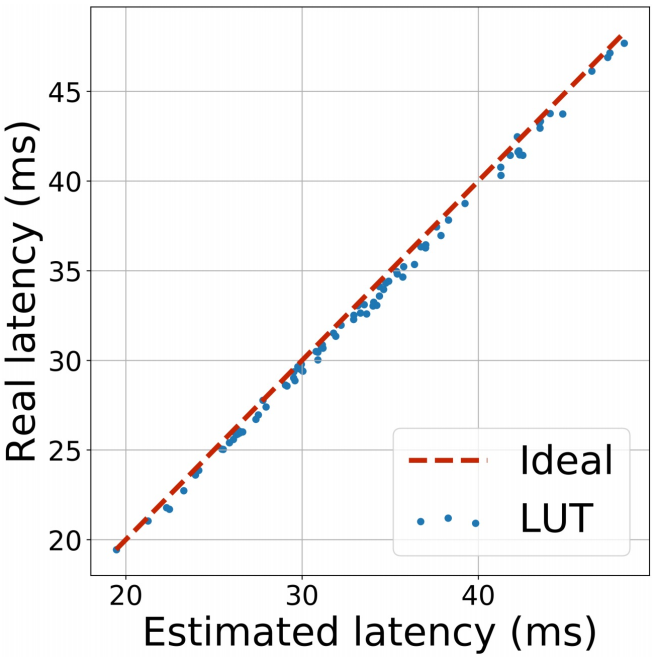


Results of Per-Layer Lookup Table

- Real latency vs. estimated latency on Google Pixel 1 CPU
- The proposed per-layer lookup table has been widely used in various works for neural architecture search



MobileNet V1



MobileNet V3

Summary

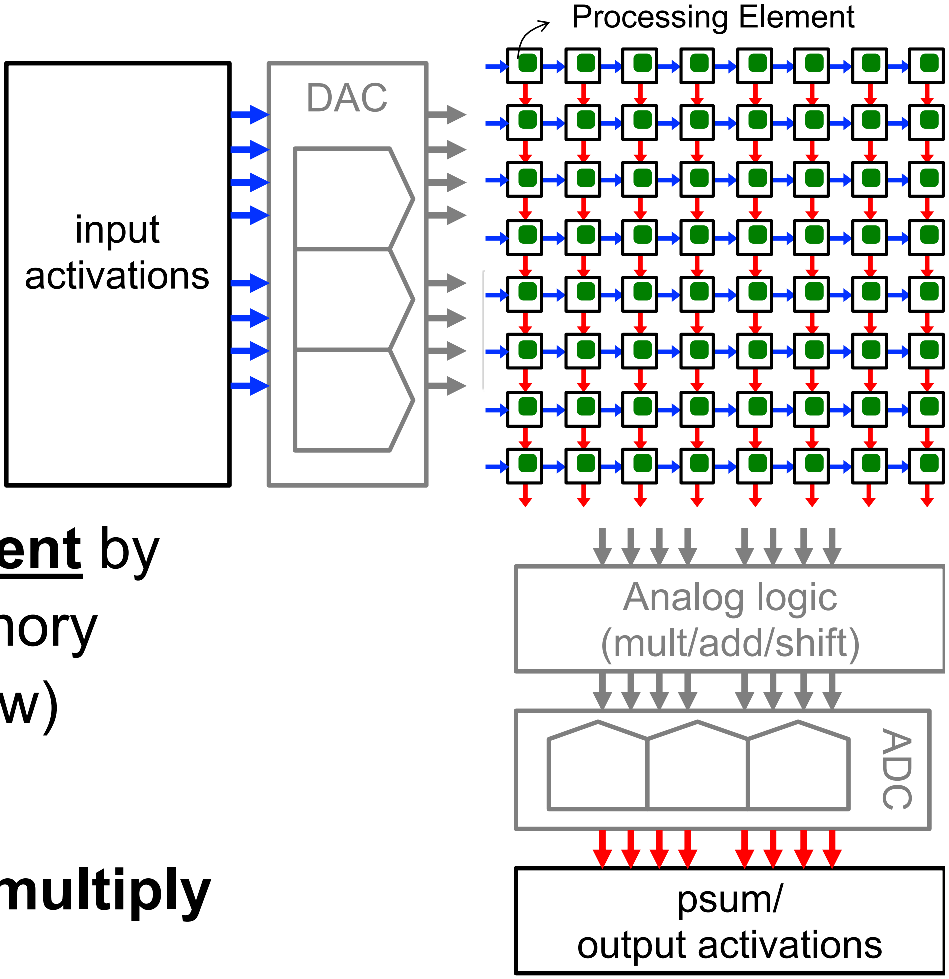
- Proxy metrics may not well approximate hardware metrics because they fail to capture some important factors, such as memory hierarchy and data movement

- We proposed two efficient methods for estimating hardware metrics for two use cases
  - With knowledge of hardware: **energy estimation methodology**
    - Considers the two main sources of energy: computation and data movement
    - Provides insights for improving the system
  - Without knowledge of hardware: **lookup-table approximation**
    - Uses pre-layer lookup tables that capture the properties of hardware
    - Builds the tables once and uses them multiple times

Beyond Current Digital Accelerators

Processing-in-Memory (PIM) Accelerators

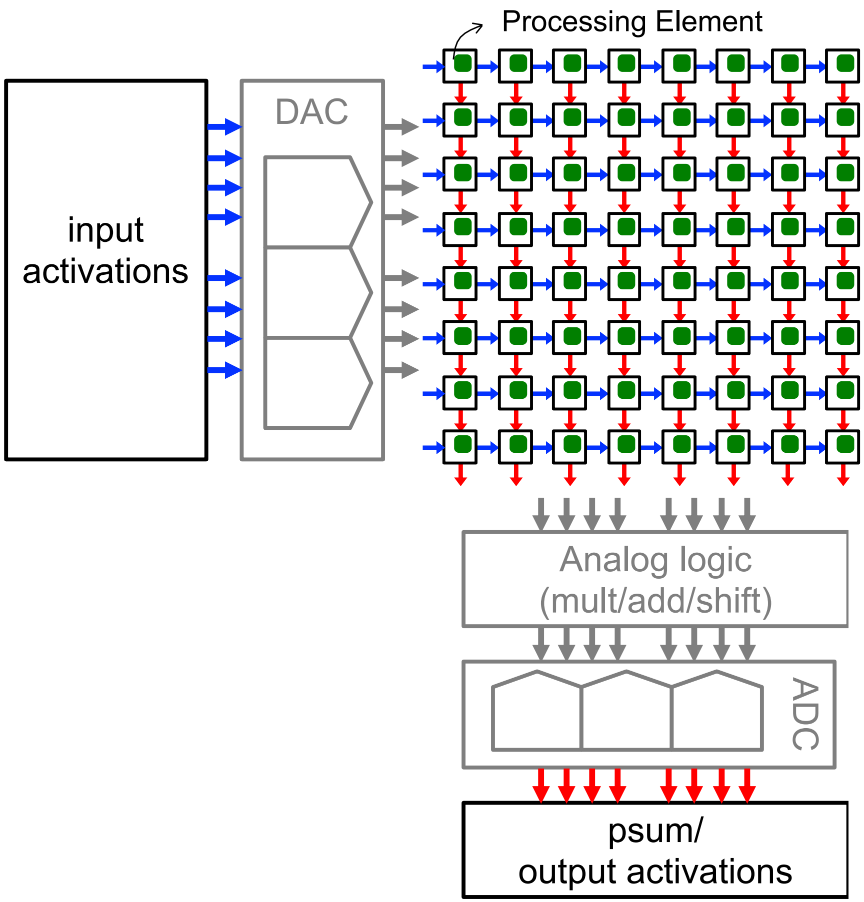
- Emerging approach for processing DNNs
- Reduce **weight data movement** by moving compute into the memory (i.e., weight-stationary dataflow)
- Implement as **matrix-vector multiply** in the **analog domain**



[Collaborated with IBM]

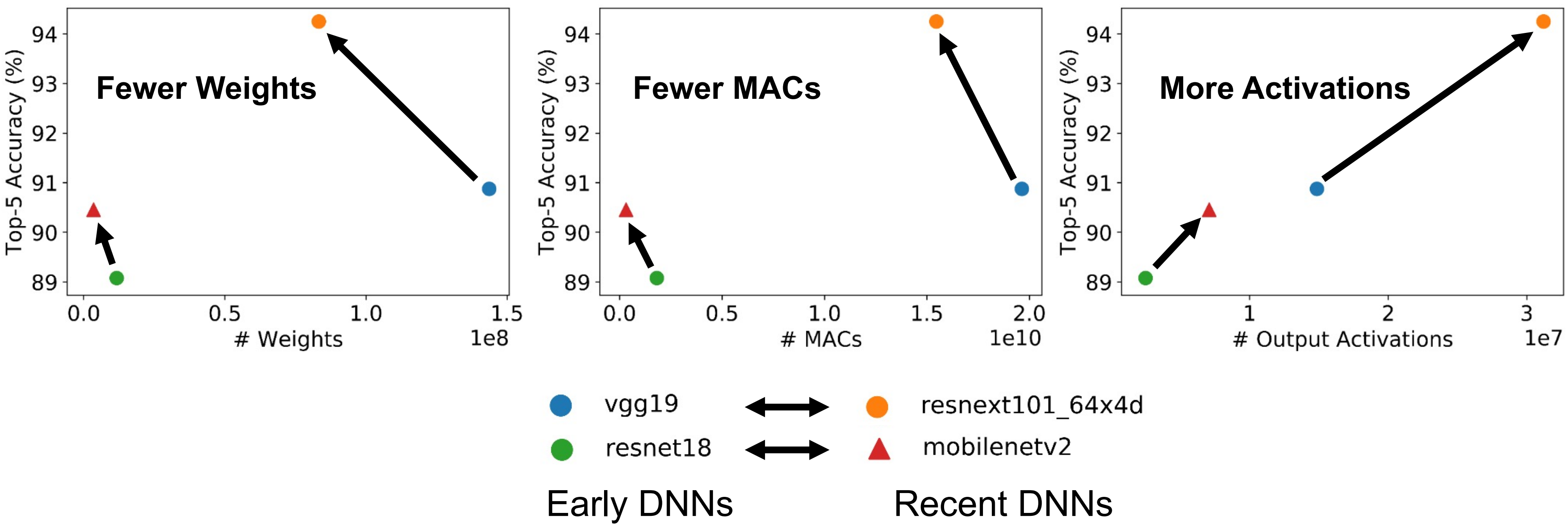
Data Movement of Activations

- Weight-stationary dataflow trades the movement of weights for the movement of activations
- Movement of activations can dominate energy consumption of PIM accelerators due to the **costly peripheral circuits**



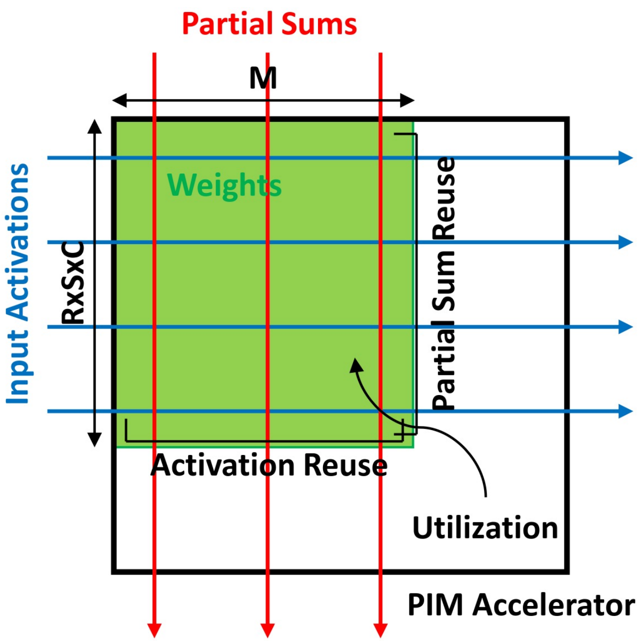
Data Movement of Activations

- Recent DNN design for digital accelerators tends to make network deeper with smaller layers
  - Achieves higher accuracy with fewer weights and MACs
- However, the decrease in MACs and weights can be accompanied by an increase in the number of activations
  - Activations are much more expensive than weights and MACs in PIM!



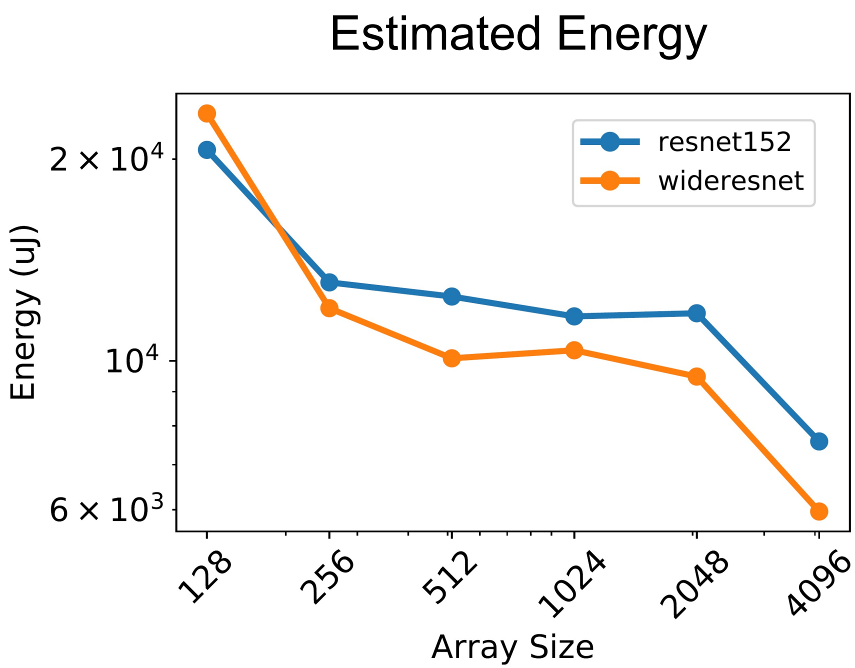
Impact of Array Size on Utilization

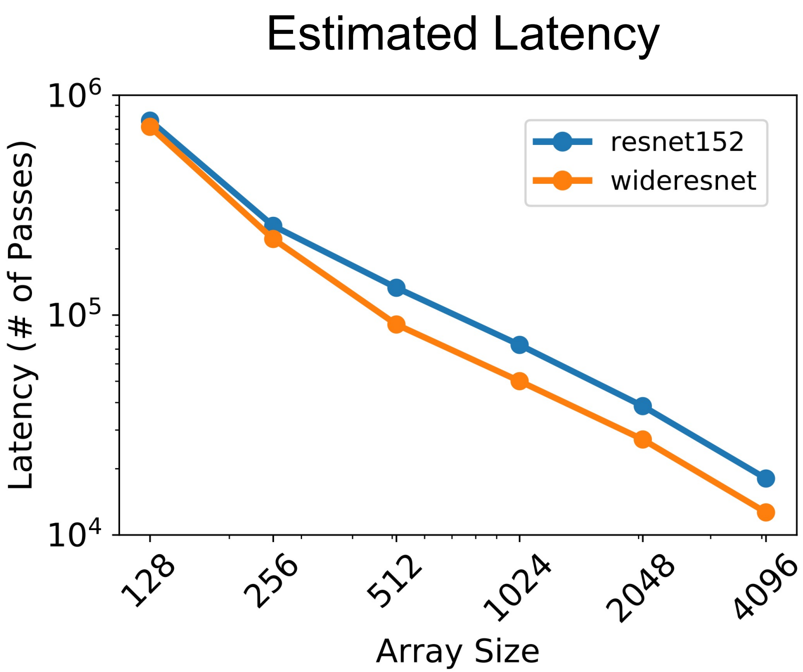
- PIM accelerators often have a large array size to amortize the cost of the peripheral circuits
  - Digital: 16x16 → 128x128
  - PIM: 128x128 → 4096x4096
- Array utilization depends on filter size
  - Recent DNNs have smaller filters
  - However, smaller filter causes lower utilization!
- Lower utilization means
  - Fewer MACs are processed in parallel → Increased latency
  - Reduced data reuse of activations → Increased energy consumption



Hardware Efficiency – Trade-Off

- Shallower DNNs with larger layers may benefit more from PIM accelerators, going against the design approach for digital hardware
- Examples with comparable accuracy:
  - Deep network with small layers: ResNet152 [He, CVPR 2016]
  - Shallow network with large layers: Wide ResNet [Zagoruyko, BMVC 2017]





Summary

Important to consider the hardware while designing DNNs

Design approaches that achieve high efficiency on digital accelerators do NOT necessarily translate to PIM accelerators

Conclusion

Conclusion

Considering hardware is the key to achieving efficient DNN design

- Designing DNN architecture with hardware metrics can improve the accuracy-efficiency trade-off
- Efficient methods for estimating hardware metrics provide insights into the bottleneck of the system and accelerate hardware-aware DNN design
- Different hardware may require different design approaches because of the distinct hardware properties

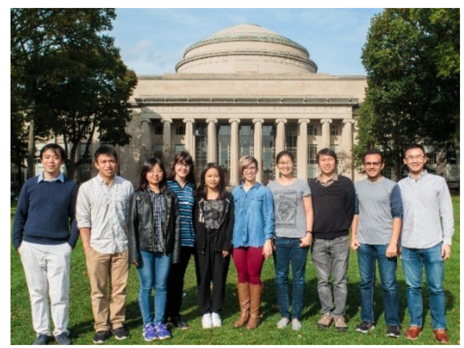
Solution to High Complexity

Environmentally | **Financially** | **Functionally**

Image sources: solarschools.net, 123rf.com

Acknowledgment

- **Advisor:**
  - Prof. Vivienne Sze
- **Thesis Committee:**
  - Prof. Joel Emer
  - Prof. Sertac Karaman
- **EEMS Group:**
  - Yu-Hsin Chen, Amr Suleiman, Mehul Tikekar, Zhengdong Zhang, James Noraky, Hsin-Yu Lai, Gladynel Peña , Yannan Wu, Peter Li, Soumya Sudhakar, Yi-Lun Liao, Diana Wofk
- **Collaborators:**
  - Google Mobile Vision: Hartwig Adam, Andrew Howard, Liang-Chieh Chen, Bo Chen, Xiao Zhang
  - IBM: Tayfun Gokmen, Wilfried Haensch
- **My family**
- **Friends**



Thanks!