Accelerator Architectures for Machine Learning

Lecture 12: TinyML Basics

Tuesday: 3:30 – 6:20 pm

Classroom: ED-302

Acknowledgements and Disclaimer

- Slides was developed in the reference with
  Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019 tutorial
  Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
  Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC Berkeley, 2020
  CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2020
- CS7960, Neuromorphic Accelerator, University of Utah
  https://www.cs.utah.edu/~rajeev/cs7960

Outline

- TinyML and EdgeAI
  - Edge AI hardware
- MCUNet TinyML Inference
  - Neuromorphic accelerator

TinyML and Edge AI

Deep Learning is Everywhere

Vision

Language

Speech

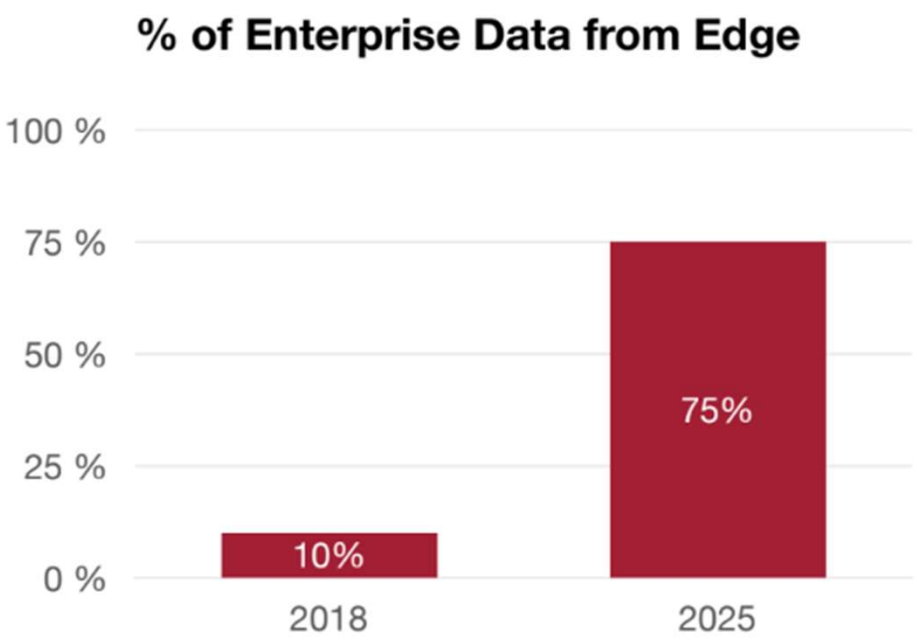
AI is Coming to the Edge Quickly

Privacy

Latency

Cost

The Market of Edge Device is Growing

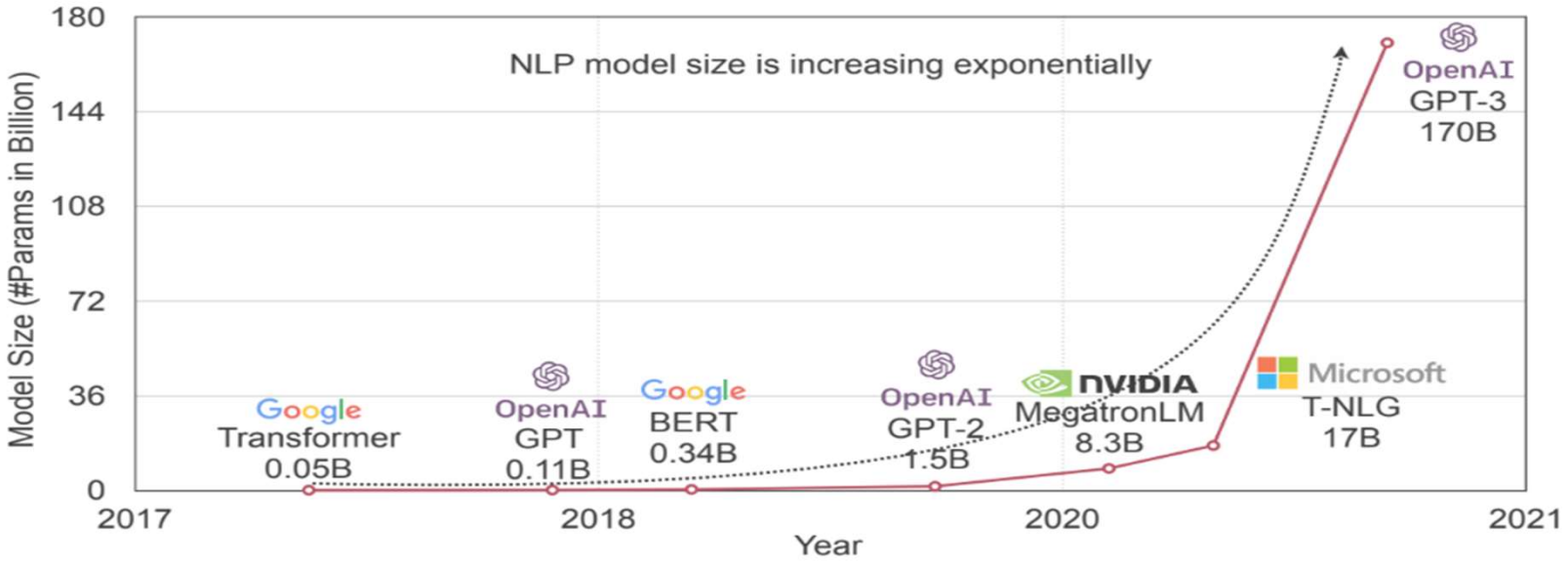


Number of Edge Devices 2021

- 15 Billion Mobile phones
- 1.4 Billion Cars
- 770 Million Security cameras
- 15 Million Robots

Today's AI is too Big

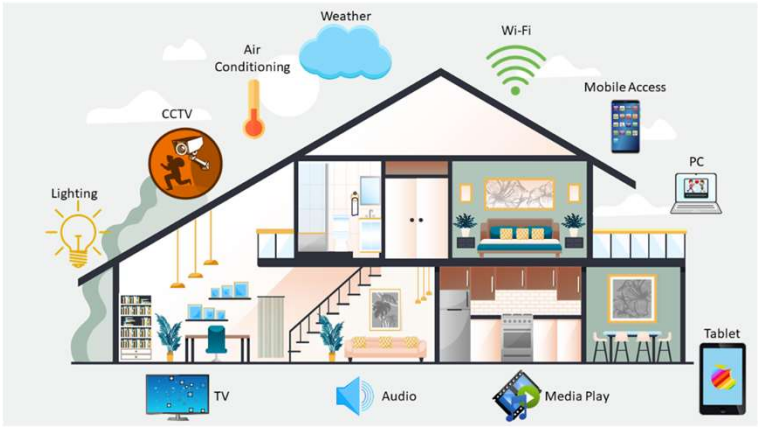
- We need new algorithms and hardware for **TinyML** and **Green AI** – Low Energy, Latency, Cost, Better Privacy



GPT-3: 355 GPU years to train and cost $4.6M
AlphaGo: 1920 CPUs and 280 GPUs, $3000 per game for electric bill

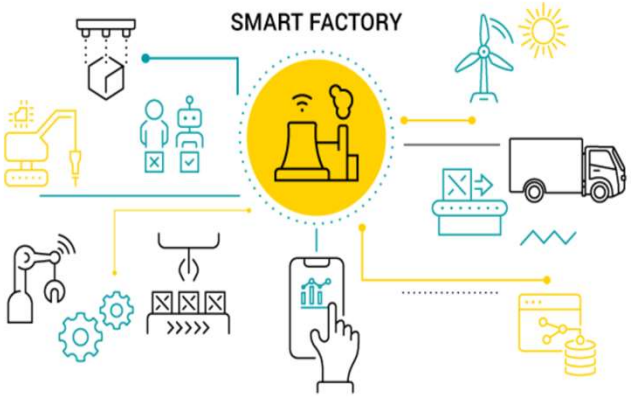
What is TinyML ?

- Squeezing deep learning into IoT devices
  - Billions of IoT devices around the world based on microcontrollers
  - Low-cost, low-power (reduce carbon)
  - A variety of domain applications



Smart Home

Smart City



Smart Factory

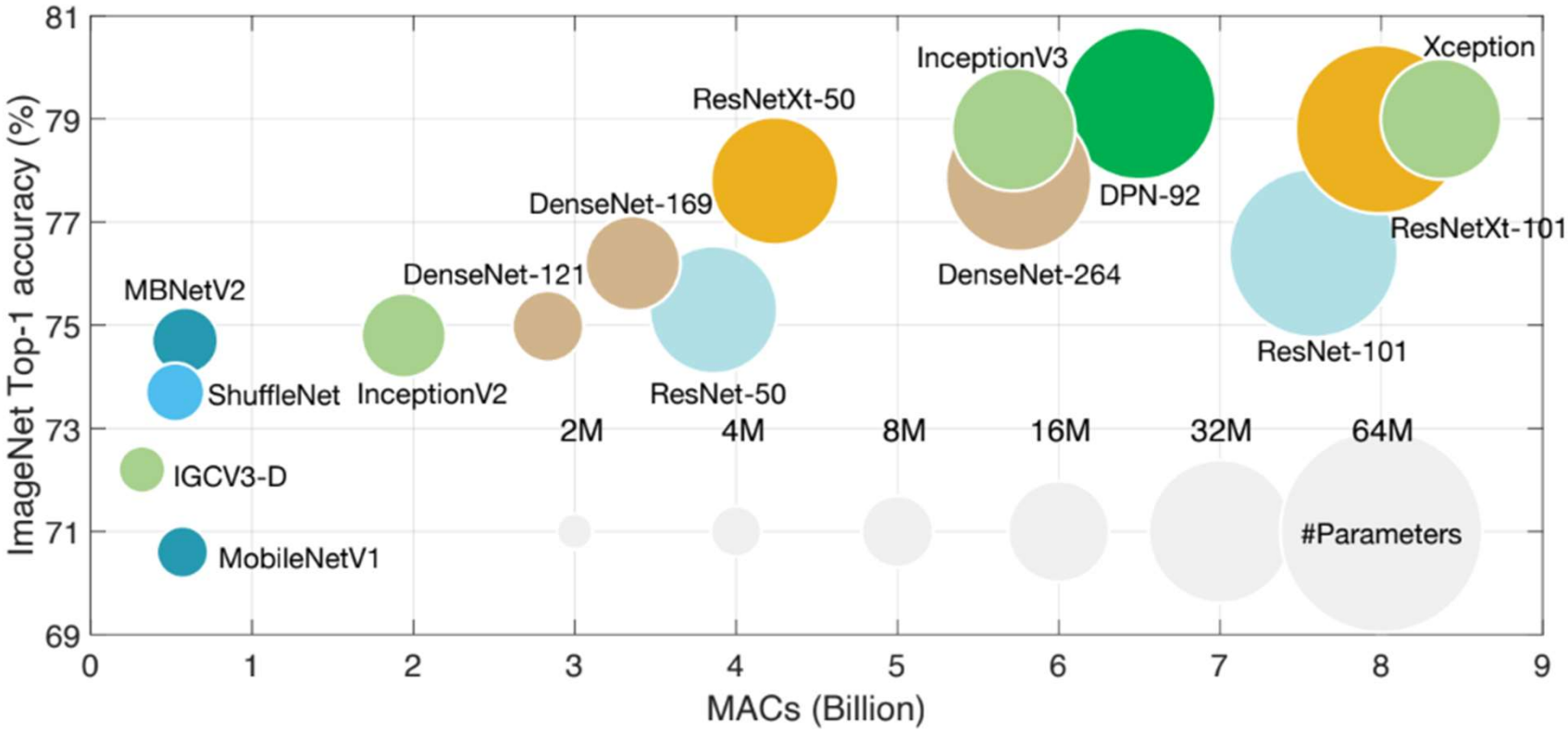
TinyML is challenging

- Memory size is too small to hold DNNs
- Latency, energy, memory constraints

	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	32GB	4GB	320kB
Storage (Weights)	~TB/PB	256GB	1MB

13,000X Smaller

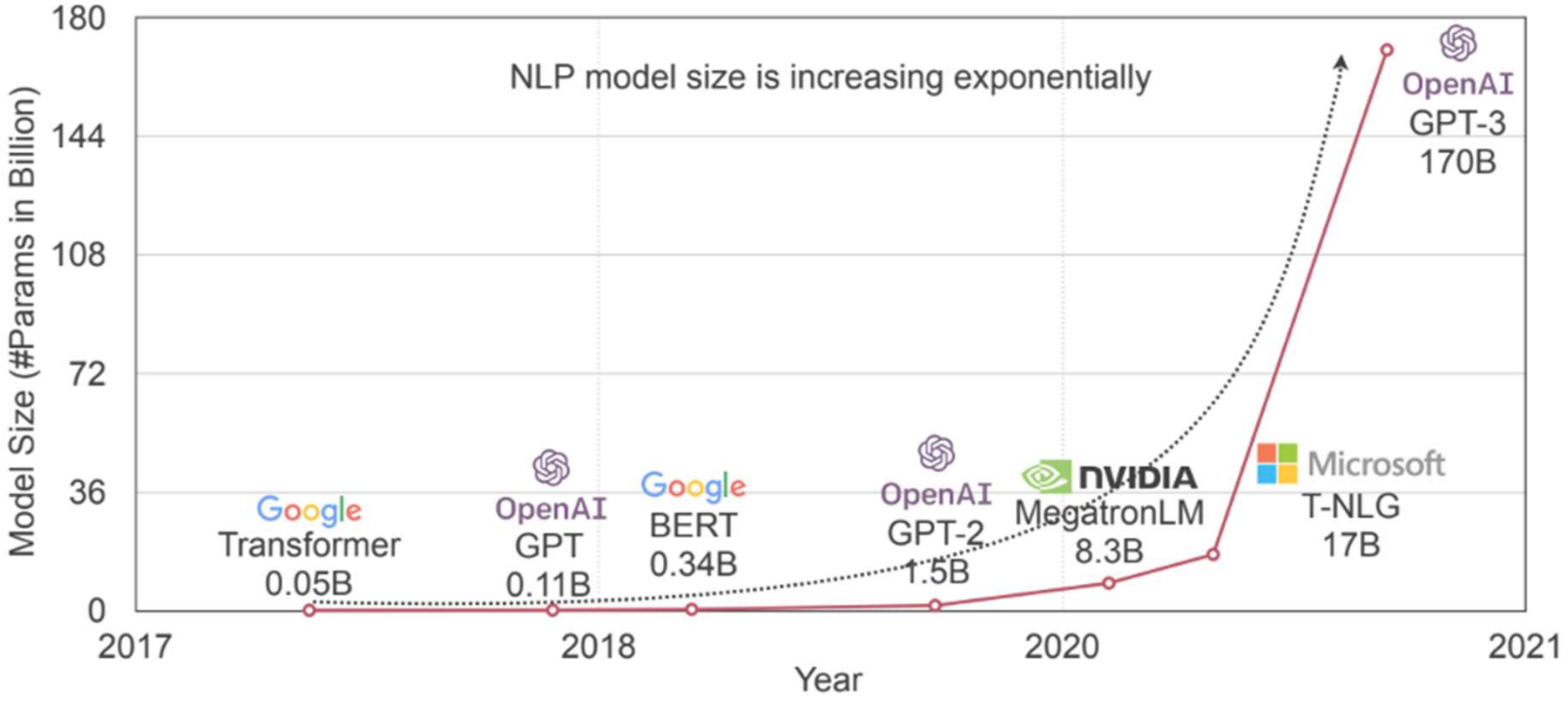
Computational Cost of DNN is Growing



Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey [Deng et al., IEE 2020]

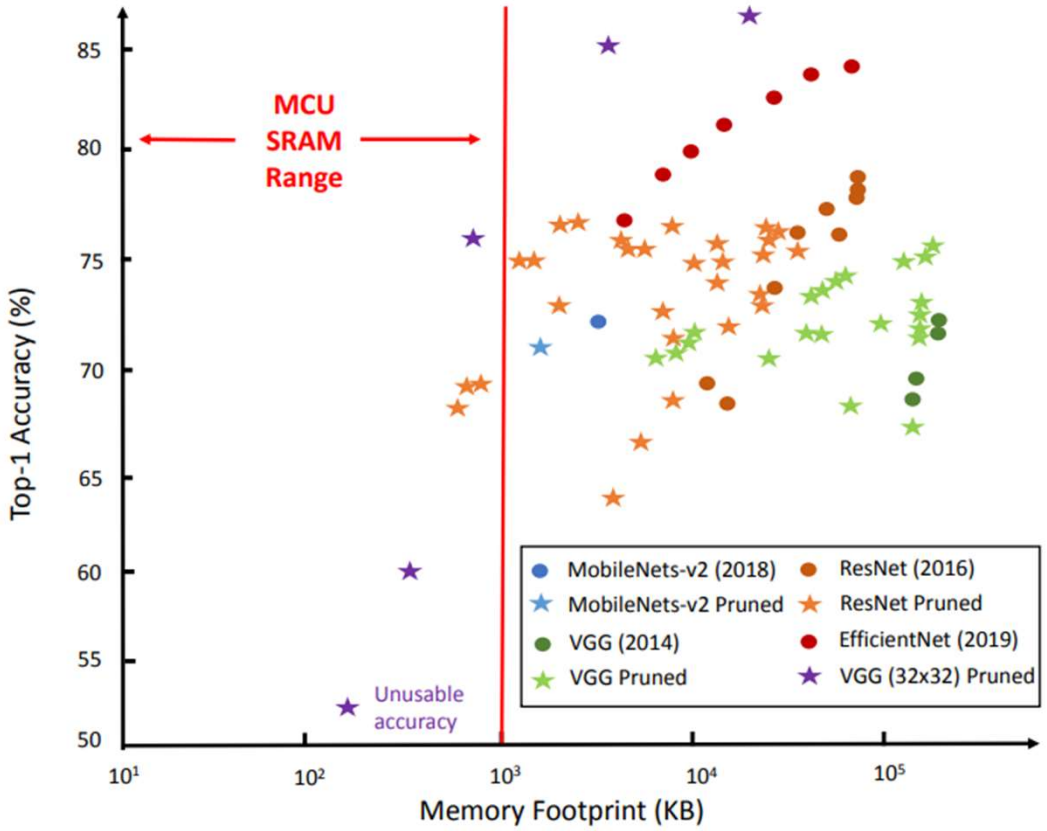
Deep Learning for Language Modeling

- Model size of language is growing exponentially



Challenges on DNN Models

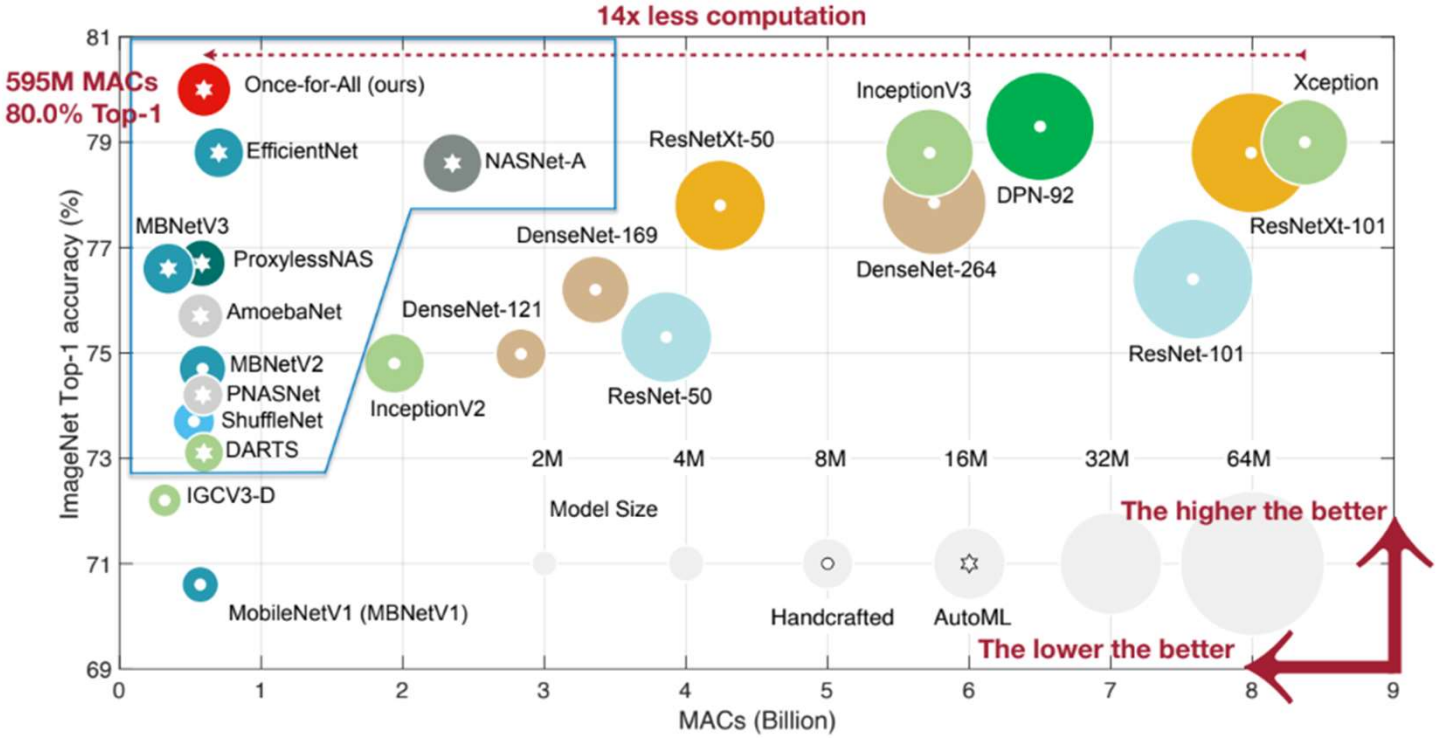
- **Large model size**
  - Model compression
  - Model Pruning
  - Quantization
  - Network Architecture Search
- **Large input/output tensors**
  - Input tensor size is associated to the ratio of Top-1 accuracy
  - MCU SRAM range is constrained



Miao et al. Arxiv, 2021

Neural Architecture Search (NAS)

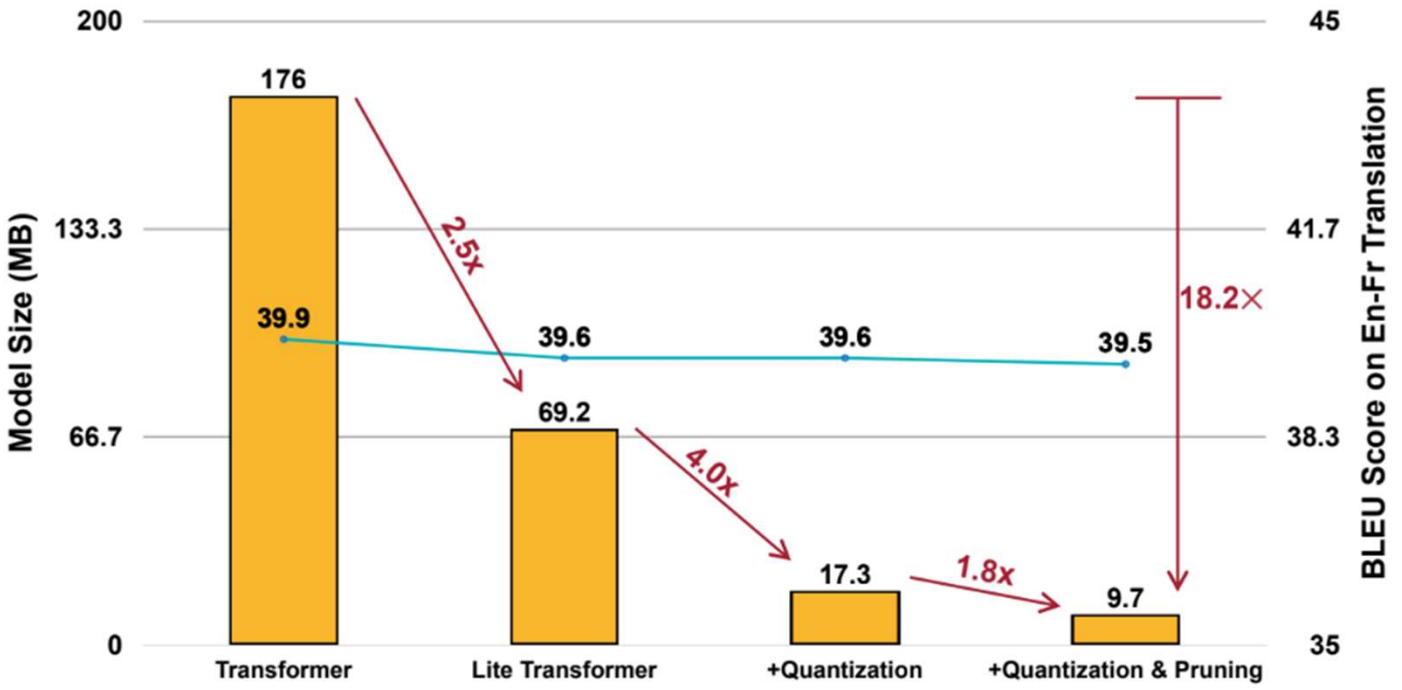
- NAS reduces the computational cost



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

TinyML for Text Translation

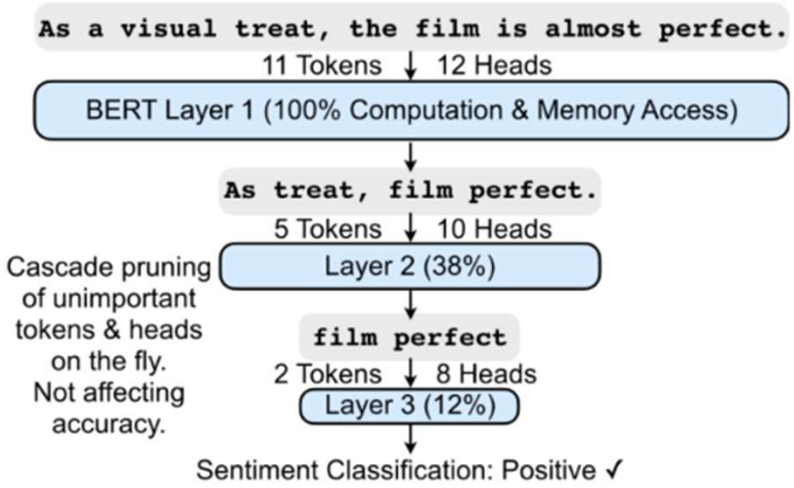
- Lite Transformer reduces the model size with pruning and quantization

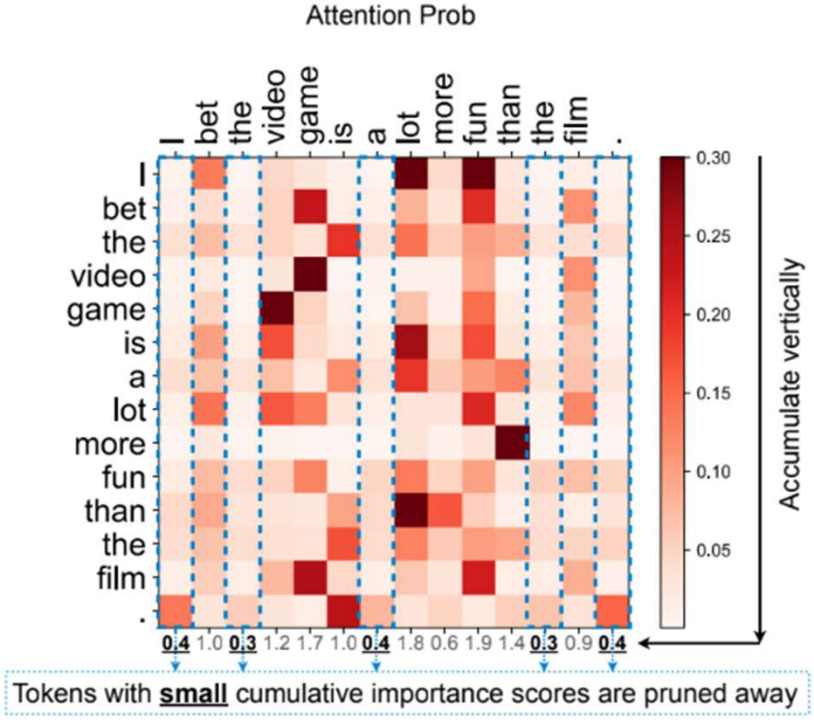


Lite Transformer with Long-Short Range Attention [Wu et al., ICLR 2020]

Efficient Language Modeling

- SpAtten accelerates language models by pruning redundant tokens

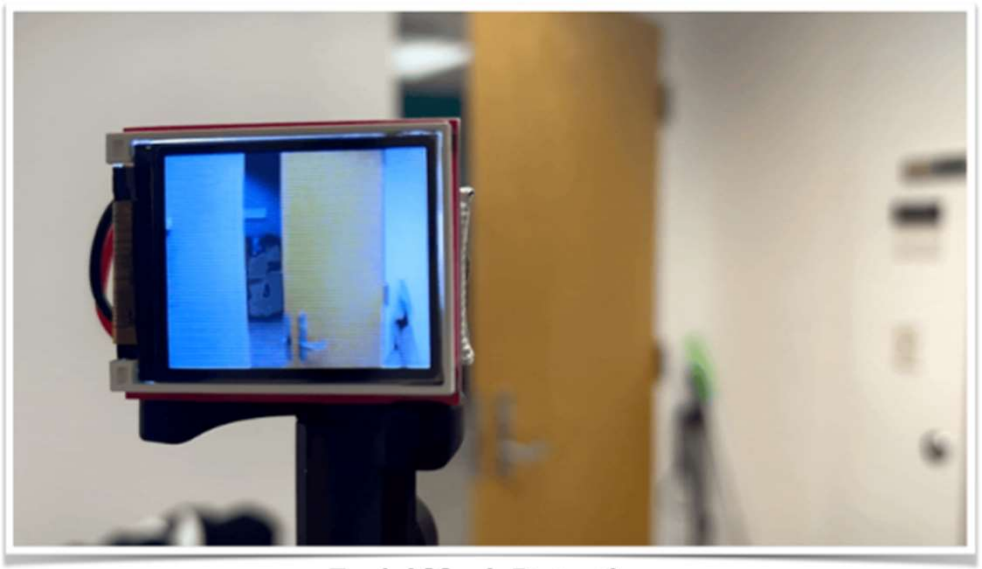




SpAtten: Efficient Natural Language Processing [Wang et al., HPCA 2021]

TinyML for Image Recognition

- MCUNet: Tiny Machine Learning on IoT devices



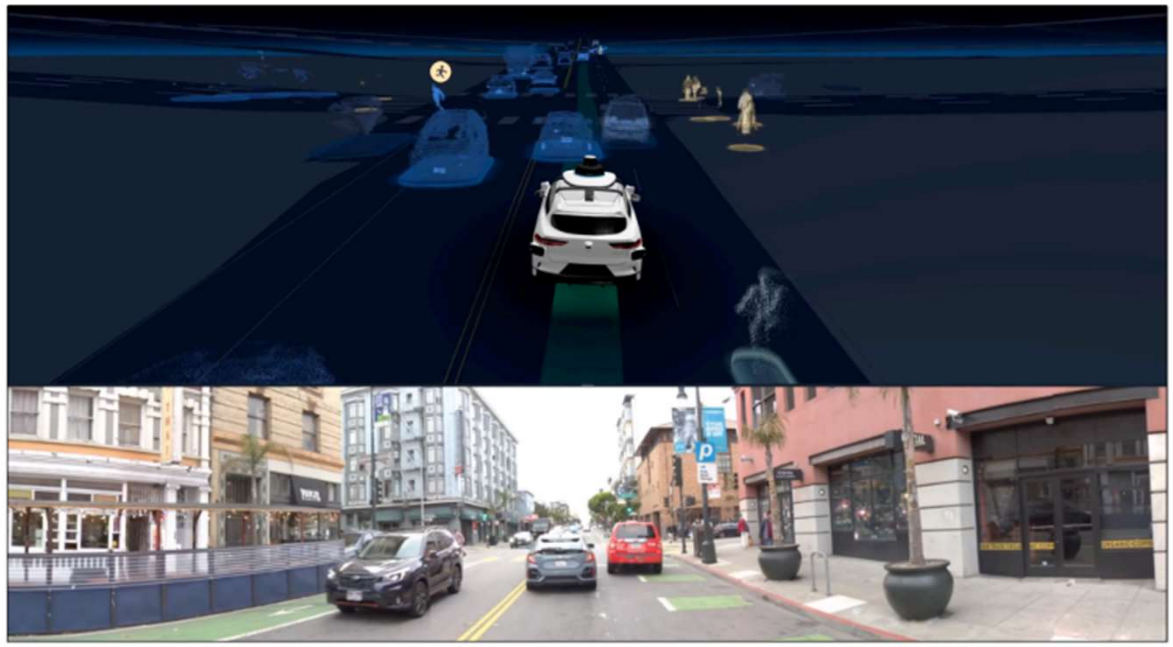
Facial Mask Detection

Person Detection

MCUNet: Tiny Deep Learning on IoT Devices [Lin et al., NeurIPS 2020]

TinyML on Autonomous Driving

- Deep learning helps machine perceive the surrounding environment

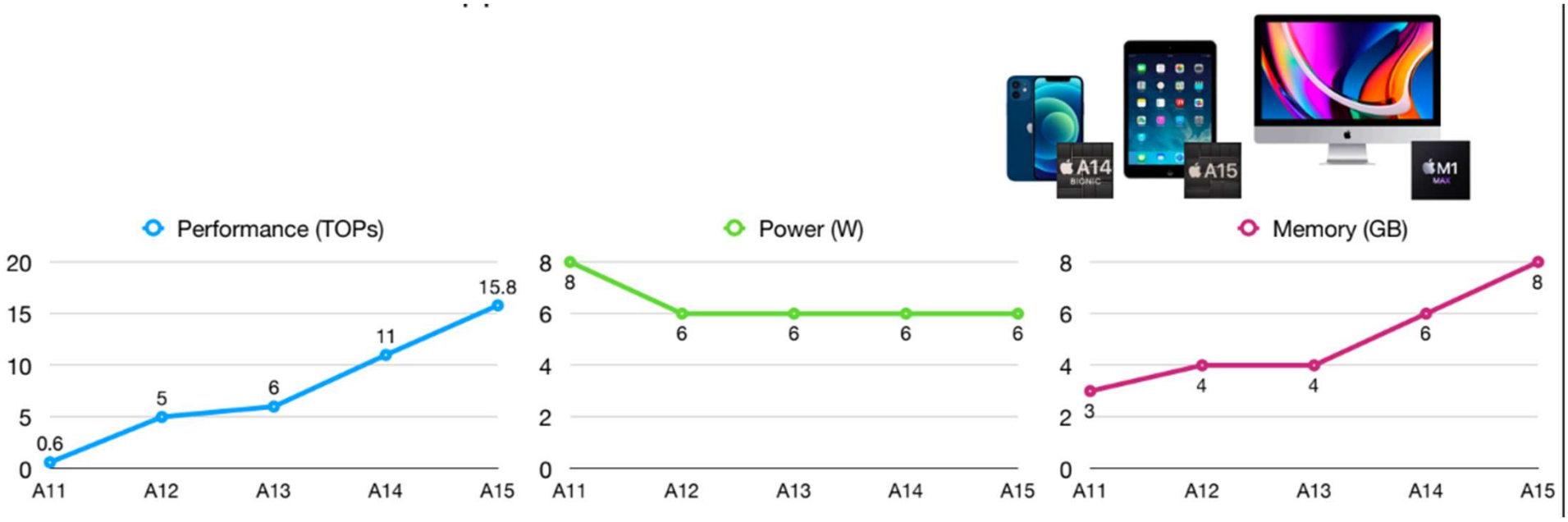


Waymo Driver

A whole trunk of workstation

Edge AI Hardware

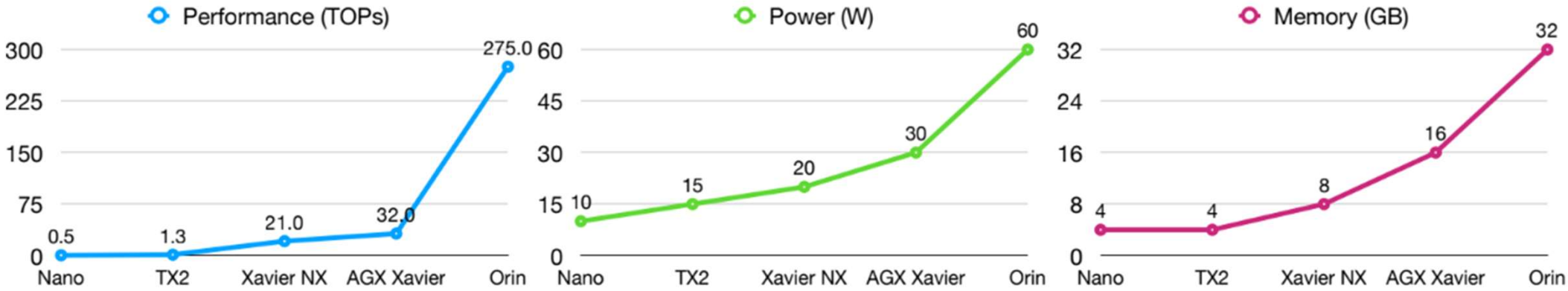
- Apple Neural Engine
  - An energy-efficient and high-throughput engine for ML inference on Apple silicon



NanoReview. https://nanoreview.net/en

Edge AI Hardware

- Nvidia Jetson
  - A complete system on module that includes a GPU, CPU, memory, …..



NanoReview. https://connecttech.com/jetson/jetson-module-comparison/

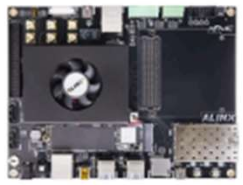
Edge AI Hardware

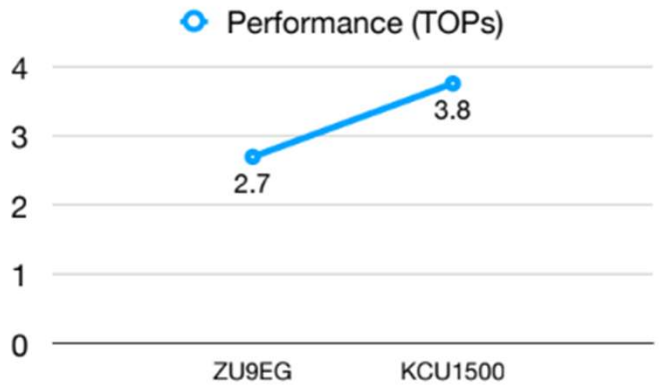
- Tensor Processing Unit (TPU)
  - An AI accelerator ASIC developed by Google

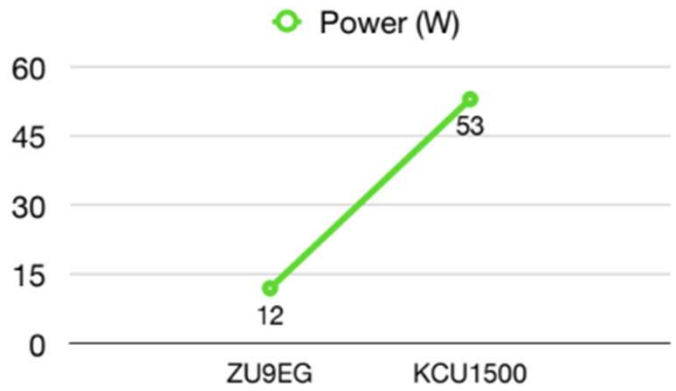
Tensor Processing Unit. https://en.wikipedia.org/wiki/Tensor_Processing_Unit

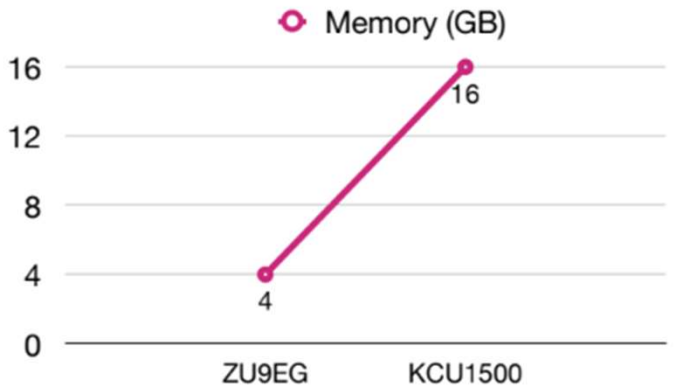
Edge AI Hardware

- FPGA-based Accelerators
  - Efficiency of custom hardware acceleration





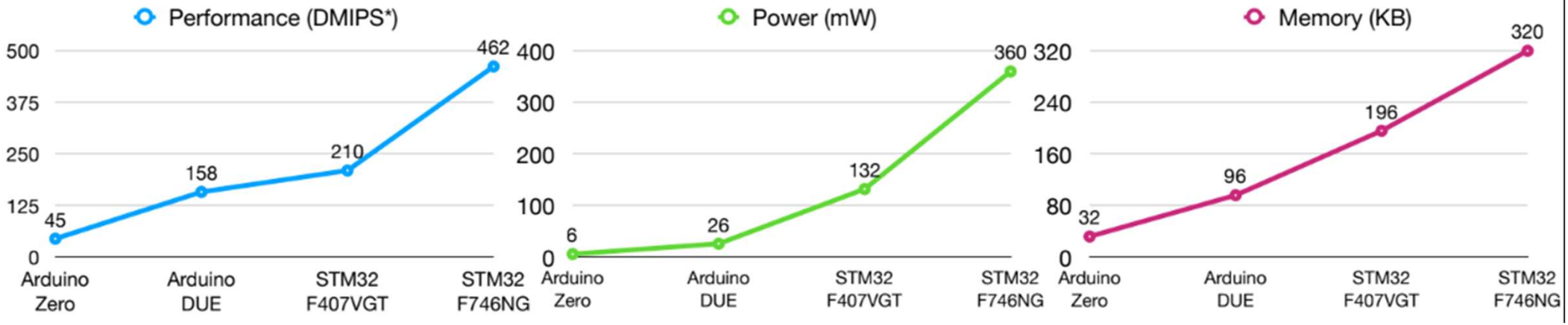




Neural Network Accelerator Comparison. https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator.html

Edge AI Hardware

- Microcontrollers (MCU)
  - Includes a processor, memory and I/O peripherals on a single chip



* Dhrystone Million Instructions Per Second (DMIPs) is an index for integer computation

Summary

- TinyML and Edge AI applications is emerging
- TinyML and Edge AI are challenging
  - Memory size is too small to hold DNNs
  - Latency, energy, memory constraints
- How to create a small DNN model
  - Model pruning/compression
  - NAS
  - How about large intermediate tensors?
- Edge AI hardware

Takeaway Questions

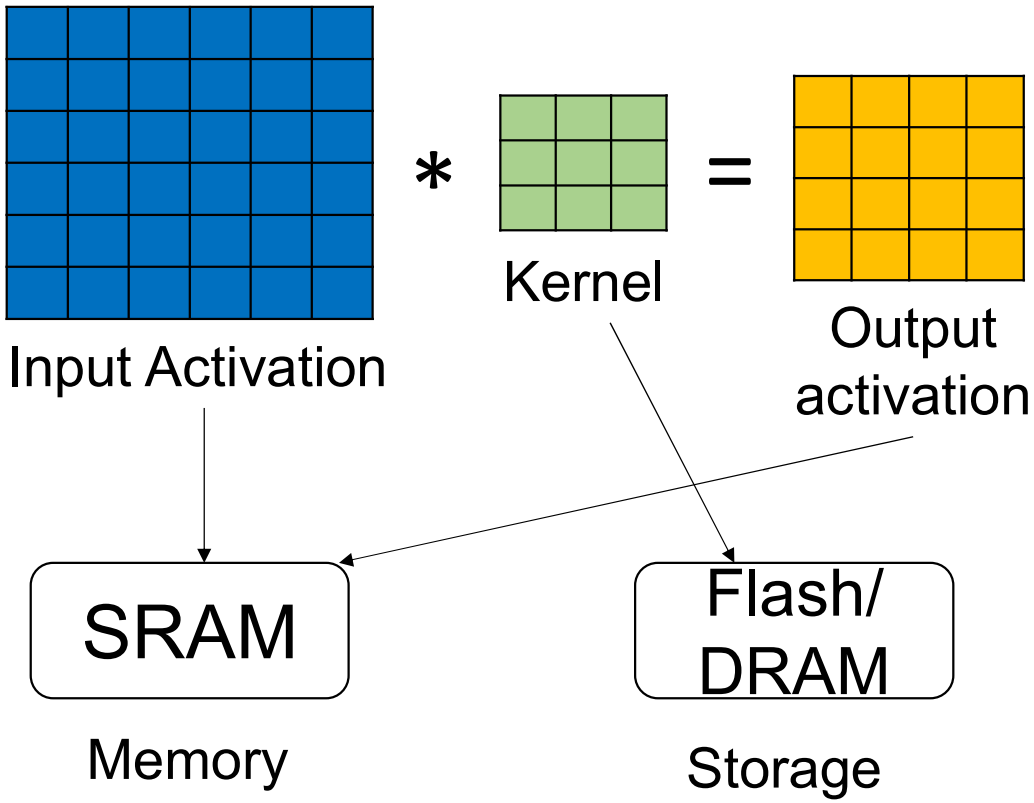
- What are advantages of TinyML ?
  - (A) Low latency
  - (B) Low cost
  - (C) Good privacy
- What are challenges of TinyML on MCUs?
  - (A) Memory size of MCUs is too small to hold DNNs
  - (B) Slow CPU processing speed
  - (C) Sparse input activations

Takeaway Questions

- What are possible ways to reduce model size?
  - (A) Data augmentation
  - (B) Model pruning
  - (C) Quantization

MCUNet TinyML Inference

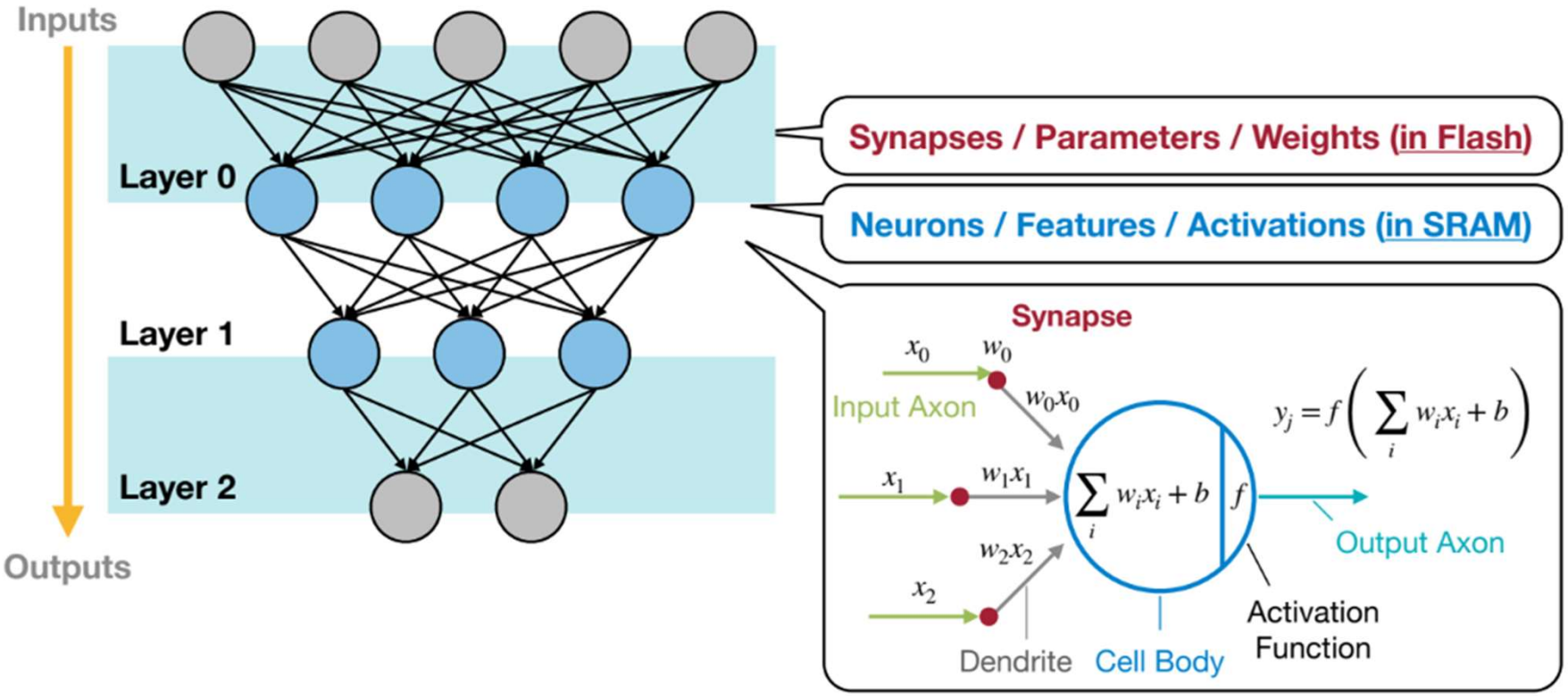
Running CNNs on Microcontrollers



Arduino nano 33 BLE sense
SRAM: 256KB
Flash: 1MB

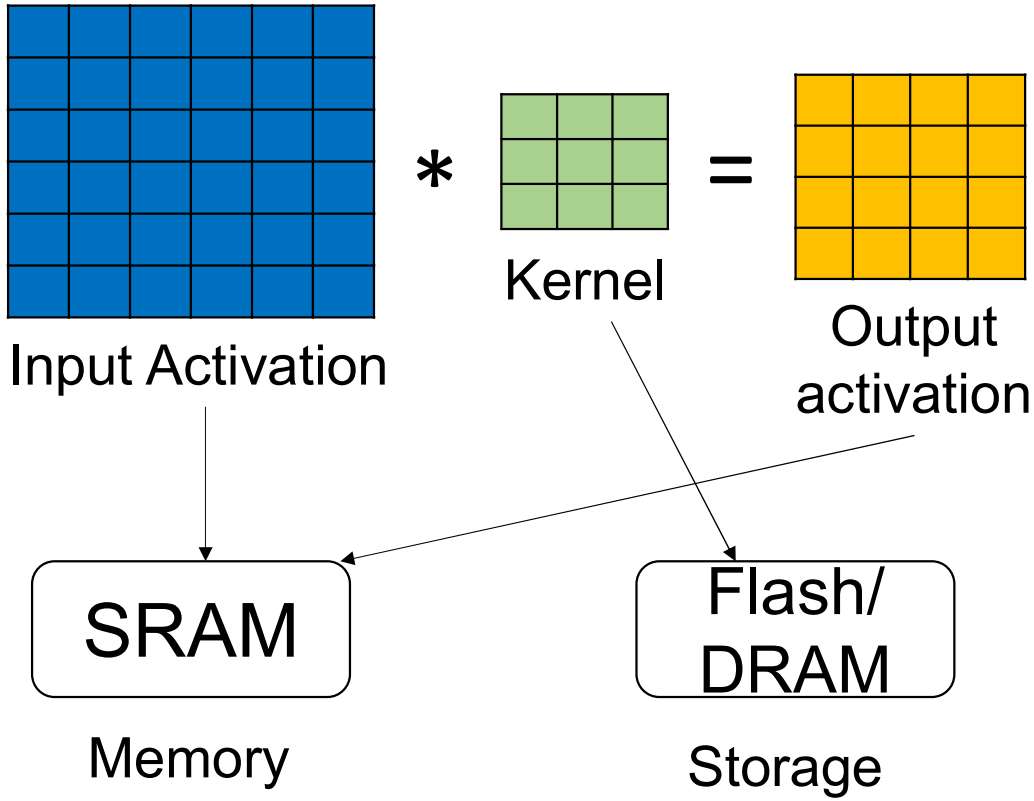
Running CNNs on Microcontrollers

- Activations and Weights on MCUs



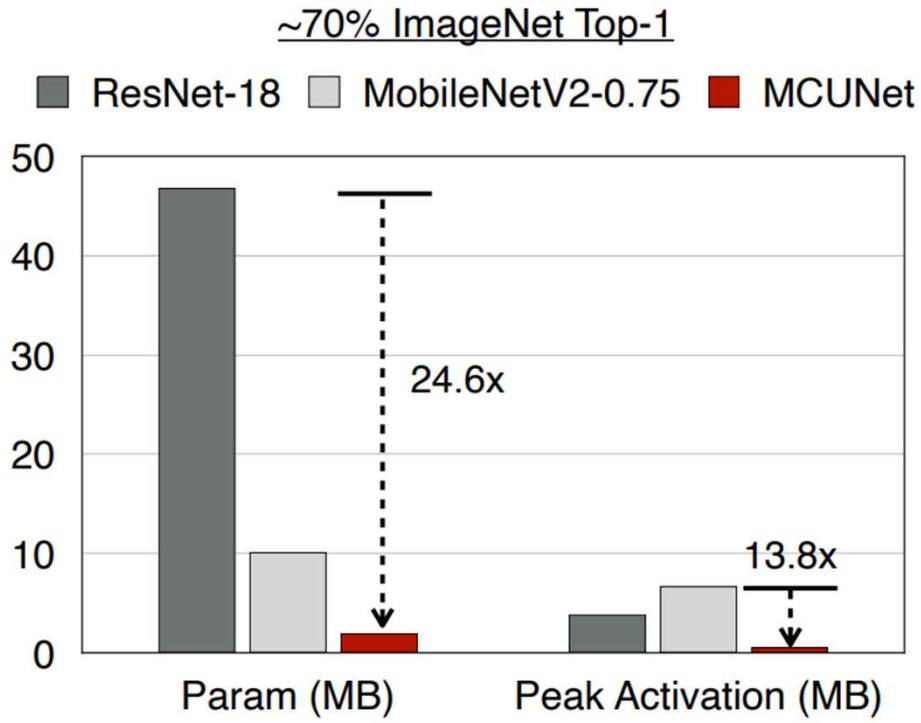
Running CNNs on Microcontrollers

- Flash Usage
  - Store model parameters
  - Static, need to hold the entire model
- SRAM usage
  - Input + output activation
  - Dynamic, different for each layer
  - We care about peak SRAM usage
  - Weights are not counted

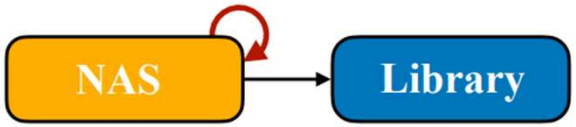


Today's CNN are Too Big for TinyML

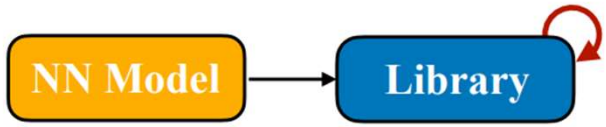
- We need to reduce model and activation size
  - MobileNetV2 reduces only model size but not peak activation size



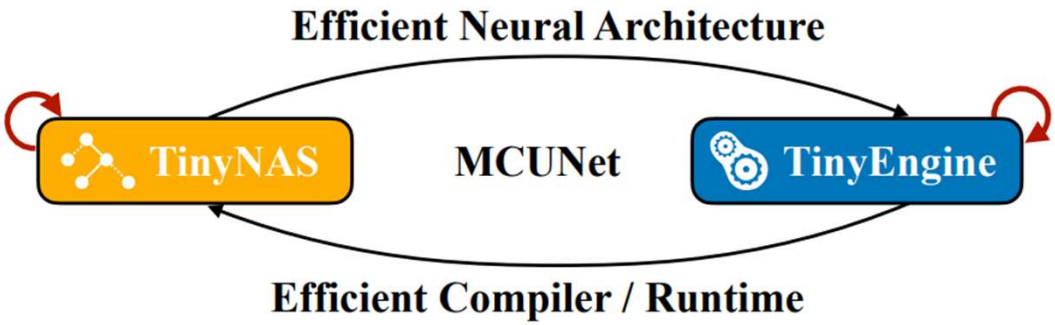
MCUNet: System-Algorithm Co-design



(a) Search NN model on an existing library
e.g., *ProxylessNAS, MnasNet*



(b) Tune deep learning library given a NN model
e.g., *TVM*

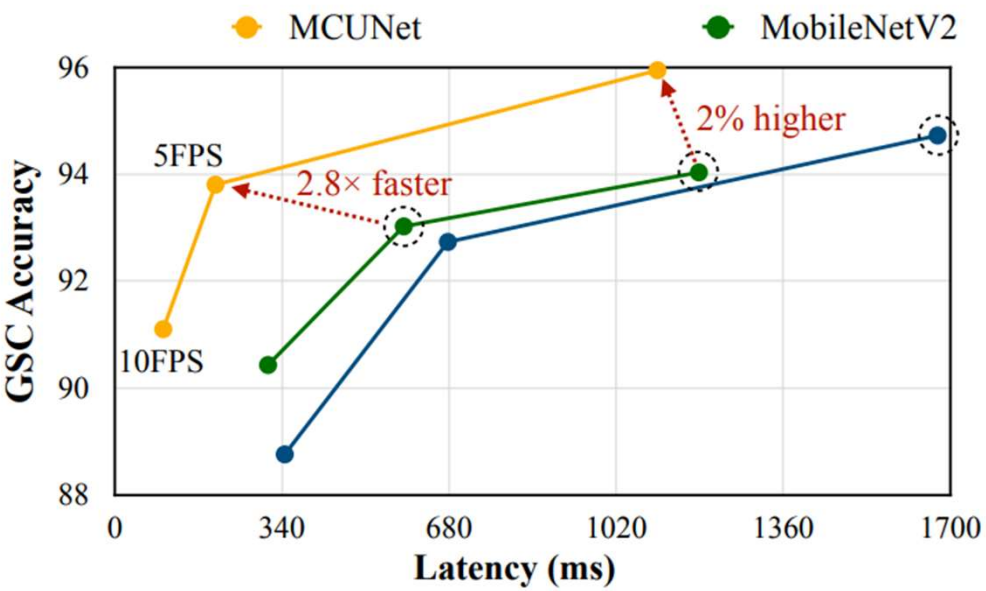


(c) *MCUNet*: system-algorithm co-design

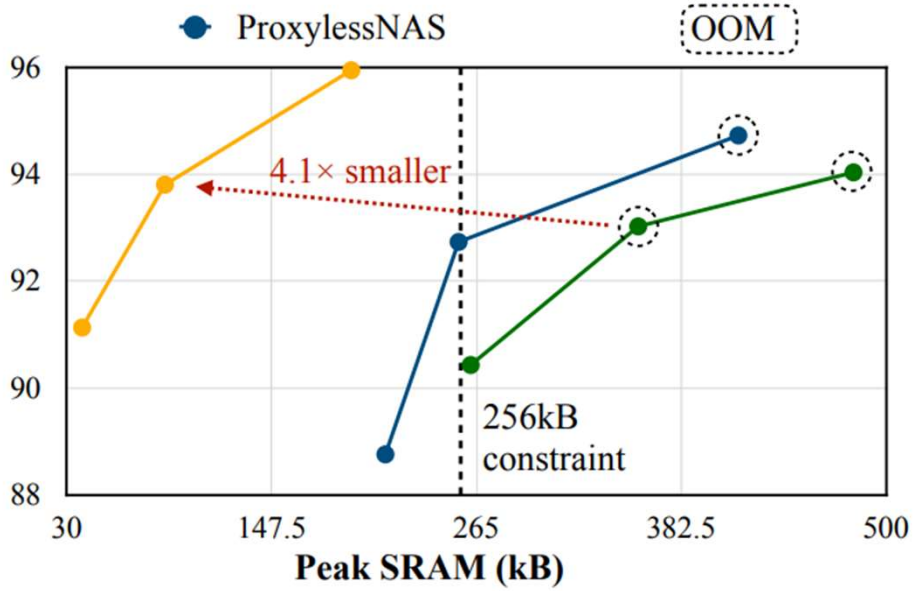
```
# TinyNAS: sample a DNN arch
for arch in arch_space:
  # TinyEngine: find a good schedule
  for schedule in schedule_space:
    # check if satisfy mem. constraints
    if can_fit_memory(arch, schedule):
      # eval acc. and update best arch
      acc = get_valid_acc(arch)
      best_acc = max(best_acc, acc)
      break
```

Outperforming Manual & NAS Models

- MCUNet achieves higher accuracy at lower memory
- Audio wake words (speech commands)



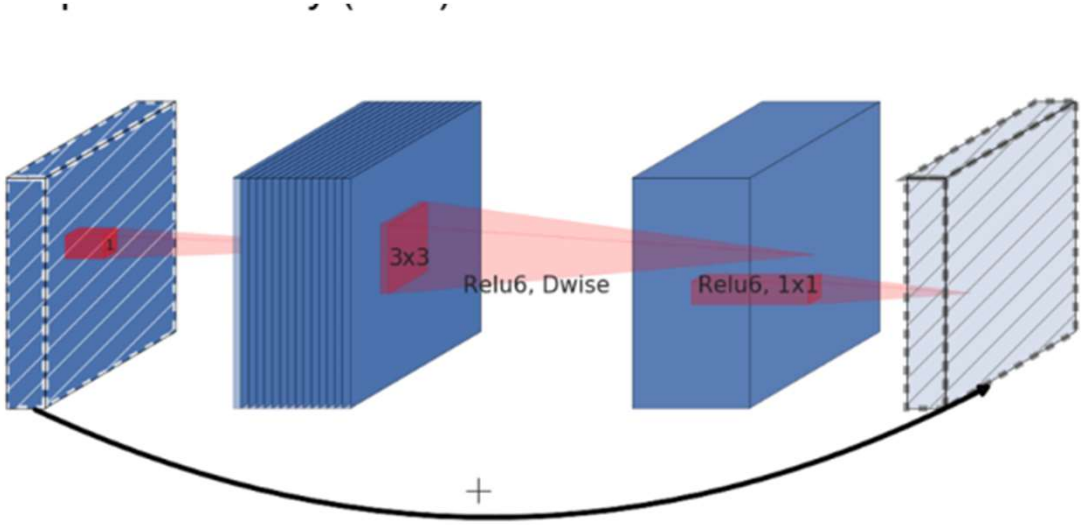
(a) Trade-off: accuracy *vs.* measured latency

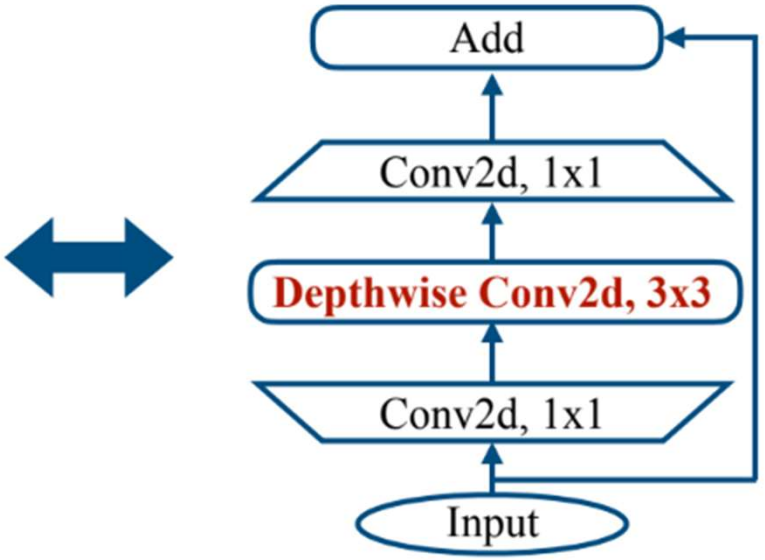


(b) Trade-off: accuracy *vs.* peak memory

MCUNet-V1: In-place Depthwise Convolution

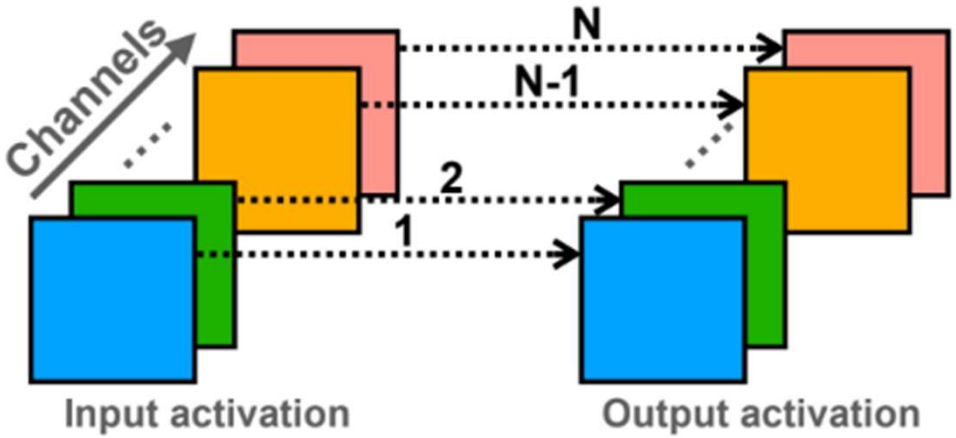
- Inverted Residual Block
  - MobileNet-V2 have “inverted residual blocks” with depth-wise convolutions which reduce model size and FLOPs, but increases peak memory





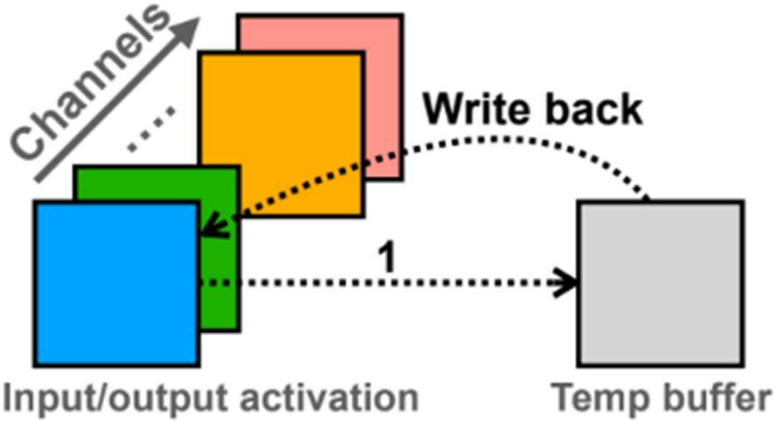
MCUNet-V1: In-place Depthwise Convolution

- In-place depthwise convolution
  - Store the input/output activation of a channel in a temp buffer



General depth-wise convolution

Peak Memory: 2xCxHxW

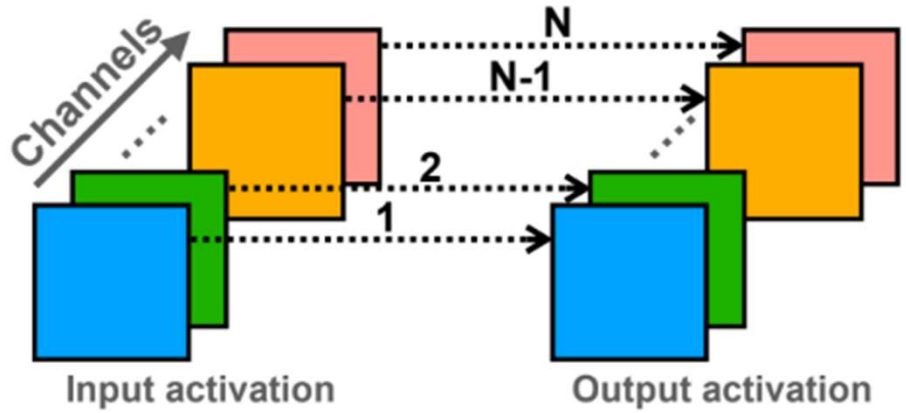


In-place depth-wise convolution

Peak Memory: (1+C)xHxW

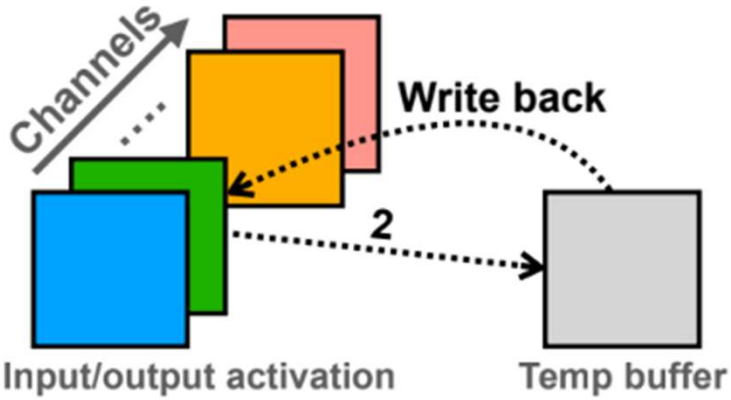
MCUNet-V1: In-place Depthwise Convolution

- Using the “in-place” updating policy with a temporary buffer



General depth-wise convolution

Peak Memory: 2xCxHxW

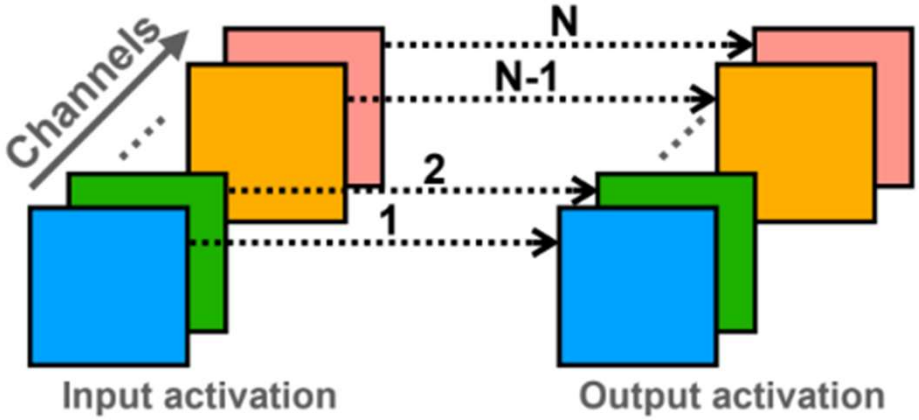


In-place depth-wise convolution

Peak Memory: (1+C)xHxW

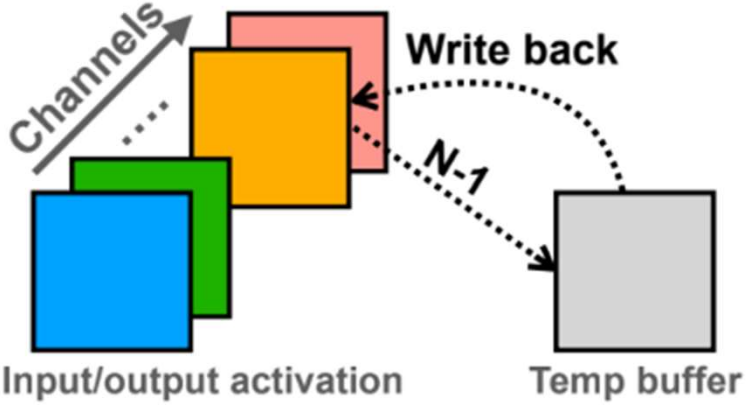
MCUNet-V1: In-place Depthwise Convolution

- Using the “in-place” updating policy with a temporary buffer



General depth-wise convolution

Peak Memory: 2xCxHxW

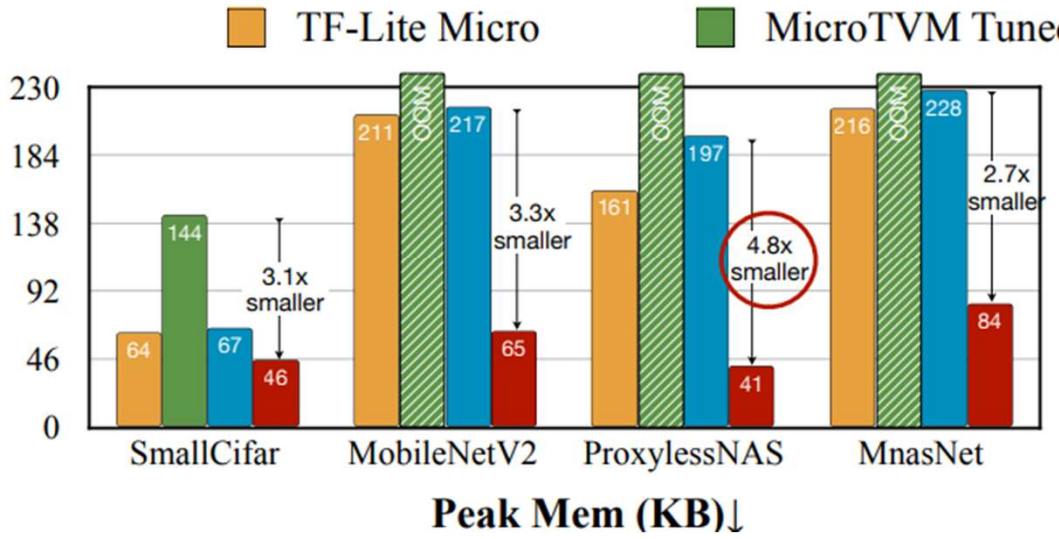


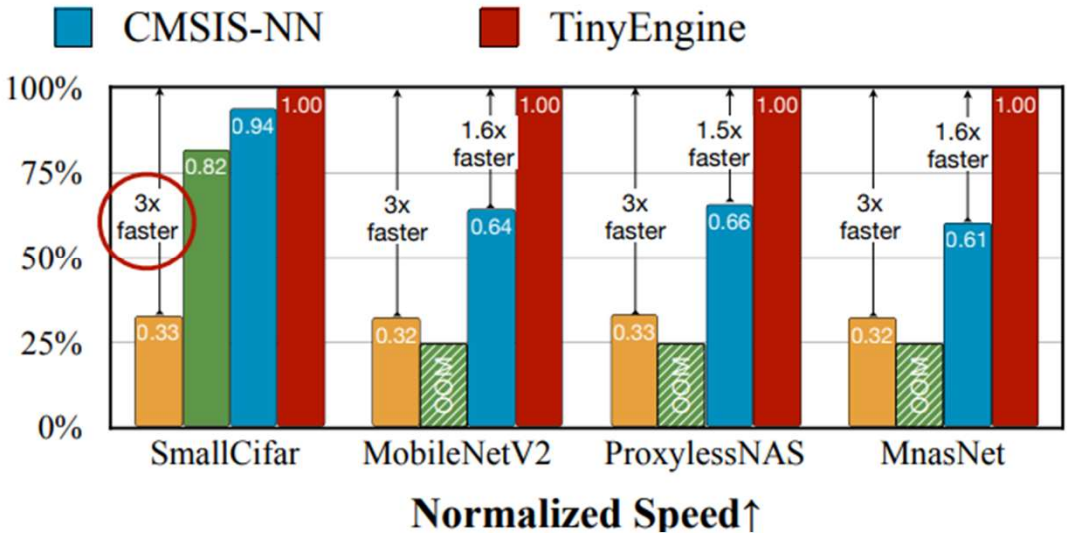
In-place depth-wise convolution

Peak Memory: (1+C)xHxW

MCUNet-V1 Results

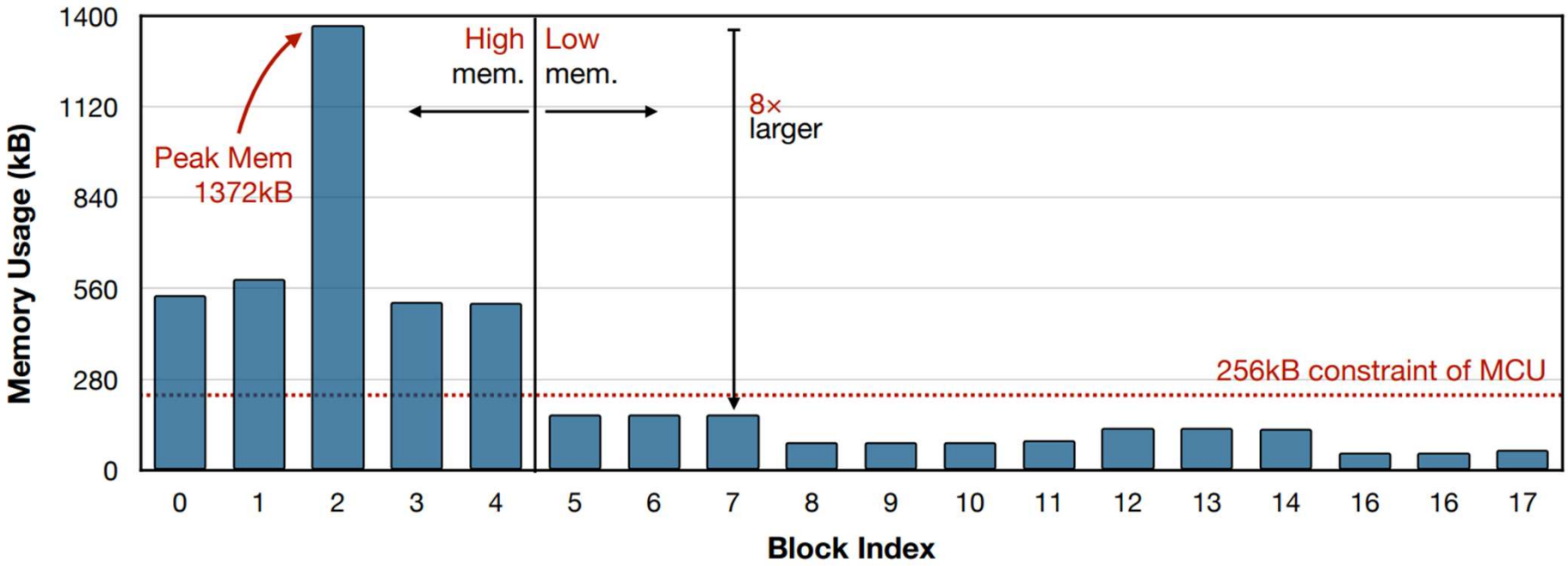
- TinyEngine (MCUNet-v1)
  - reduces the peak memory usage across different TinyML models
  - Speedup comes from the optimization of operator kernels





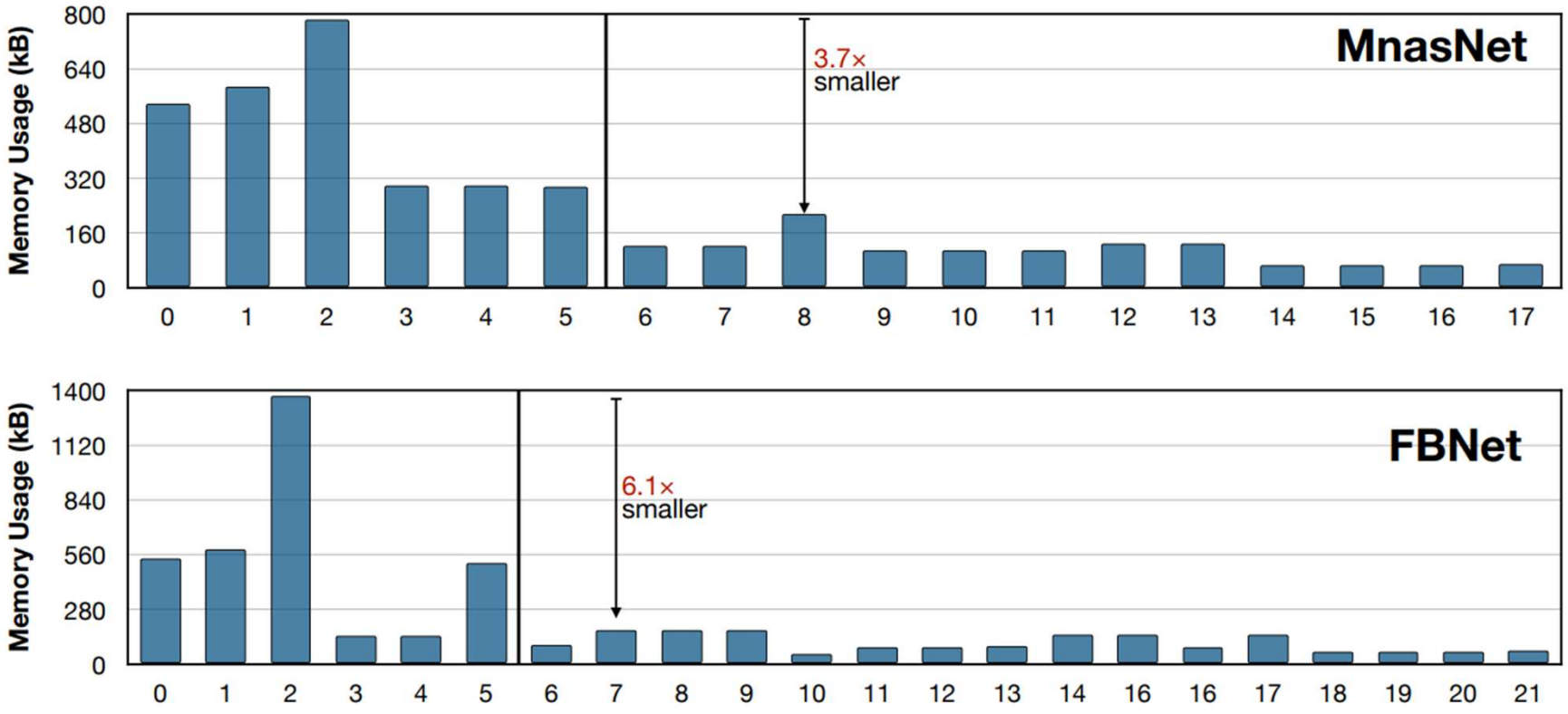
MCUNet-V2: Patch-based Inference

- Imbalanced memory distribution of CNNs
  - The SRAM usage of each layer in MobileNet-V2



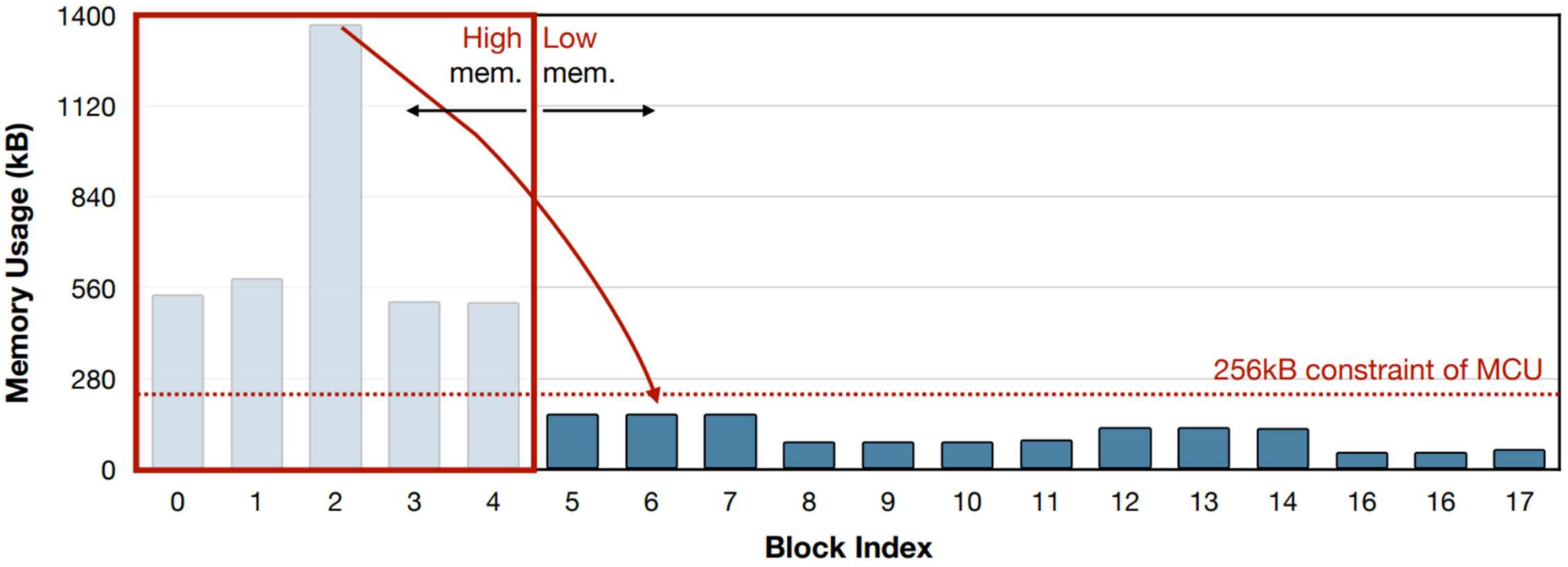
Imbalanced Memory Distribution of CNNs

- Common case in efficient CNN design



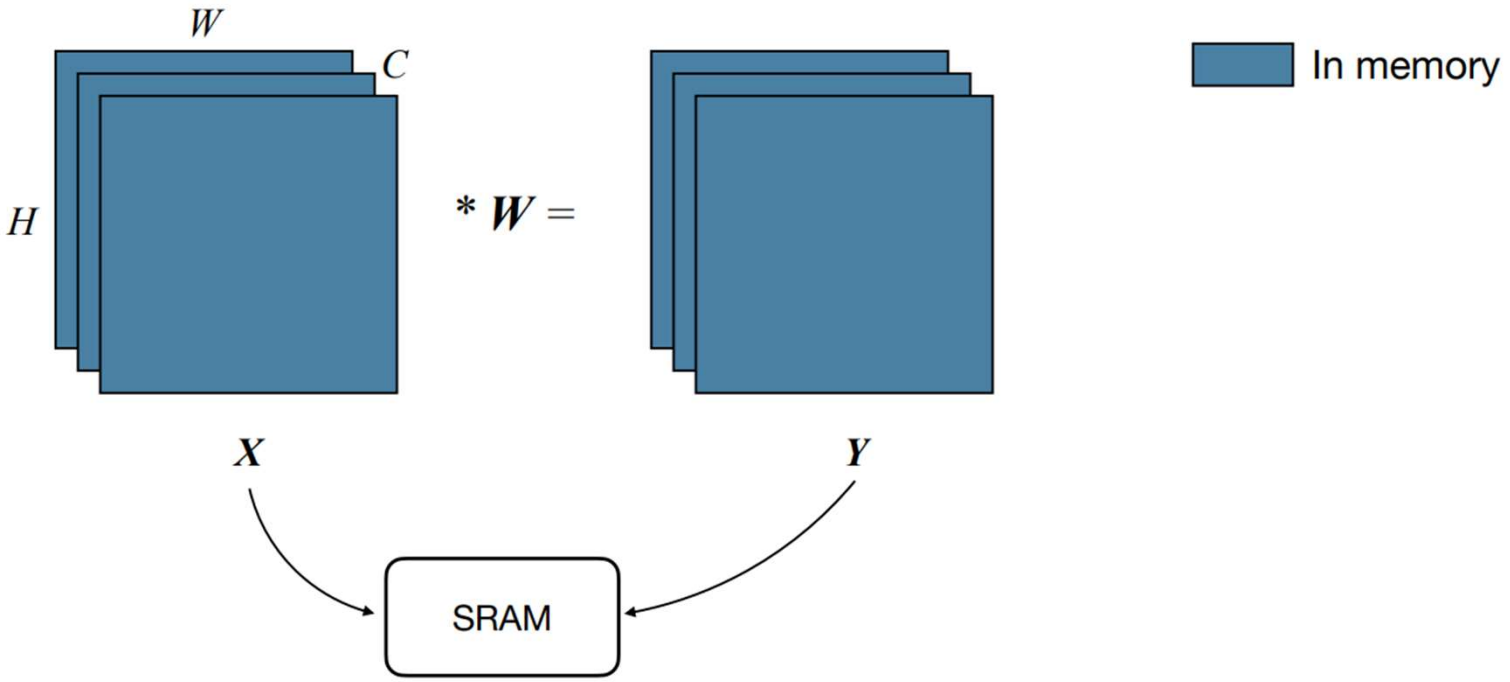
Imbalanced Memory Distribution of CNNs

- Reduce memory usage of the initial stage
  - Reduce the overall peak SRAM memory usage



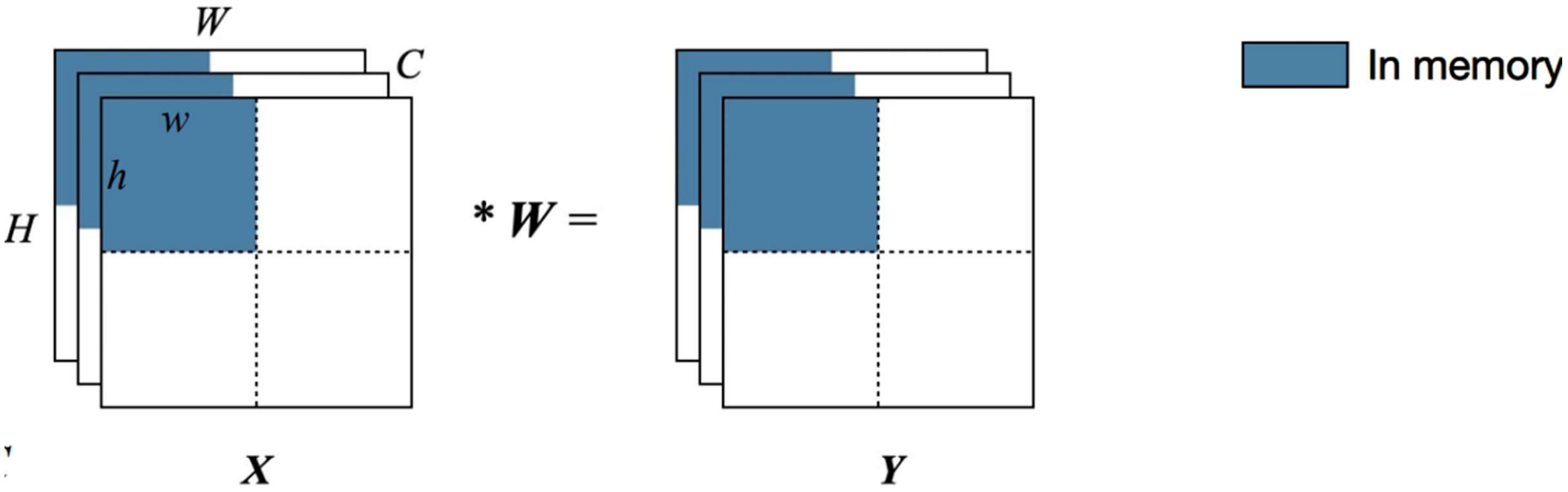
Per-Layer Inference

- Peak memory = 2 WHC



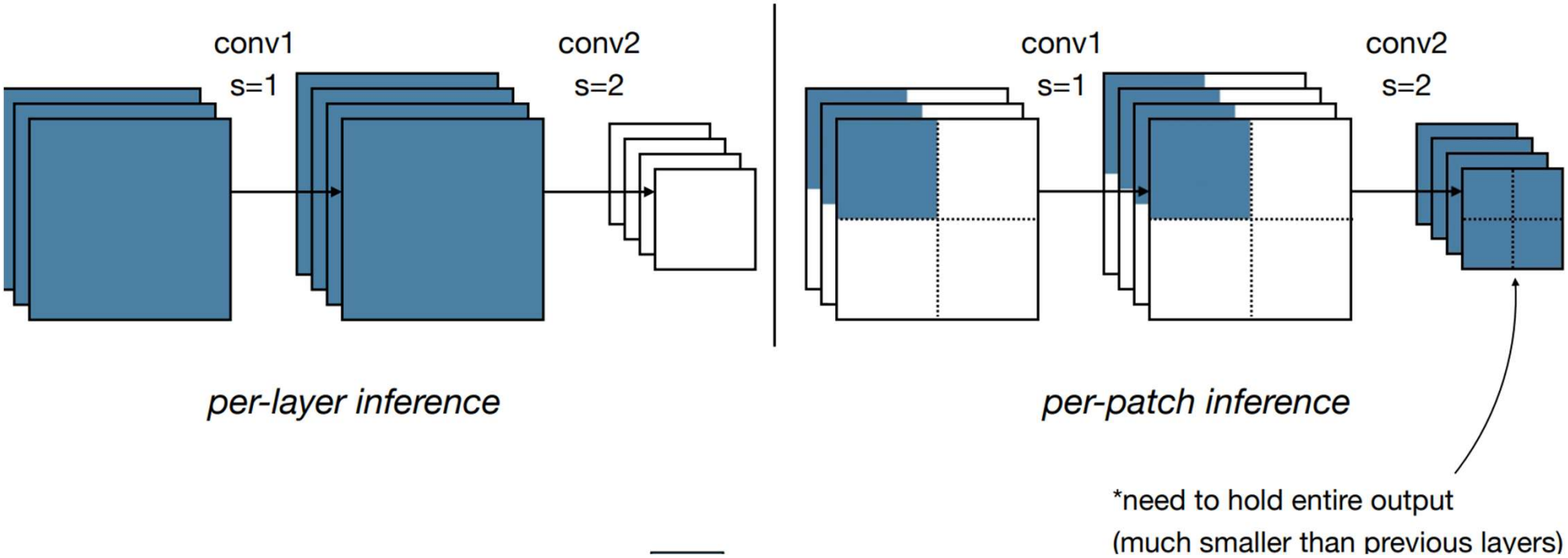
Per-patch Inference

- Peak Memory = 2whC << 2WHC



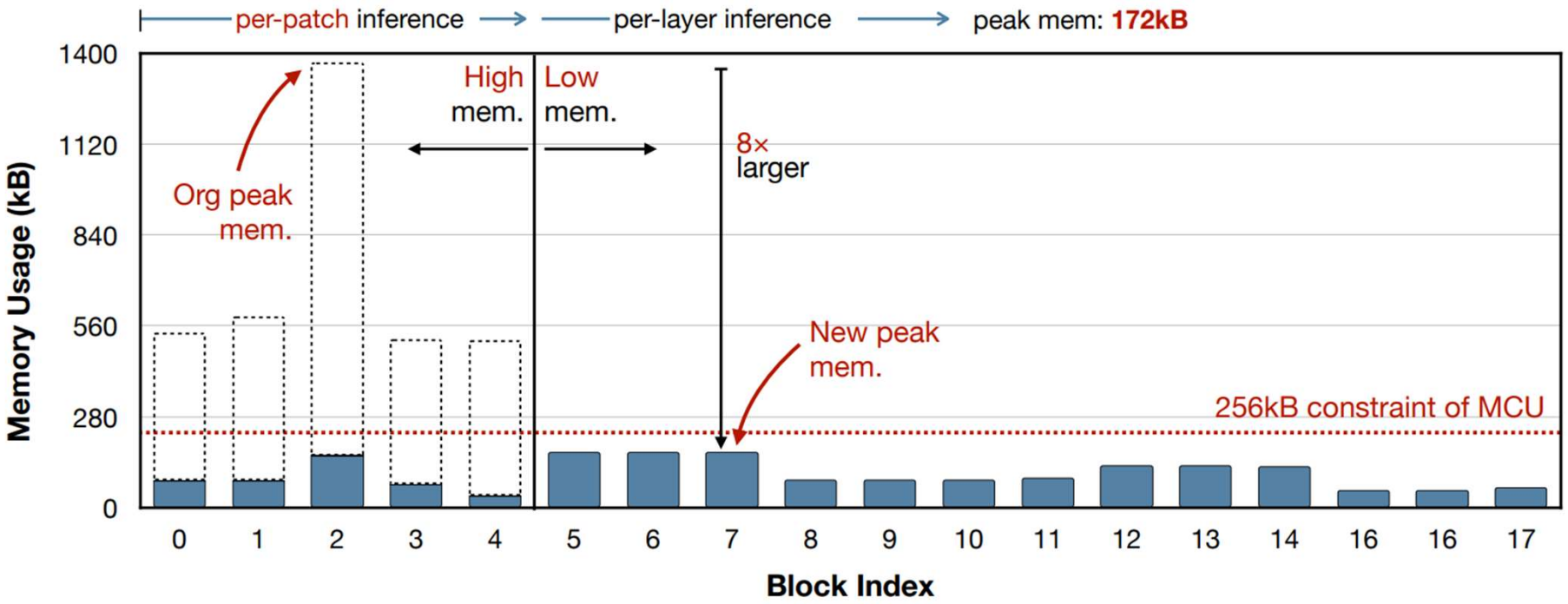
Saving Memory with Patch-based Inference

- a practical 2-layer example



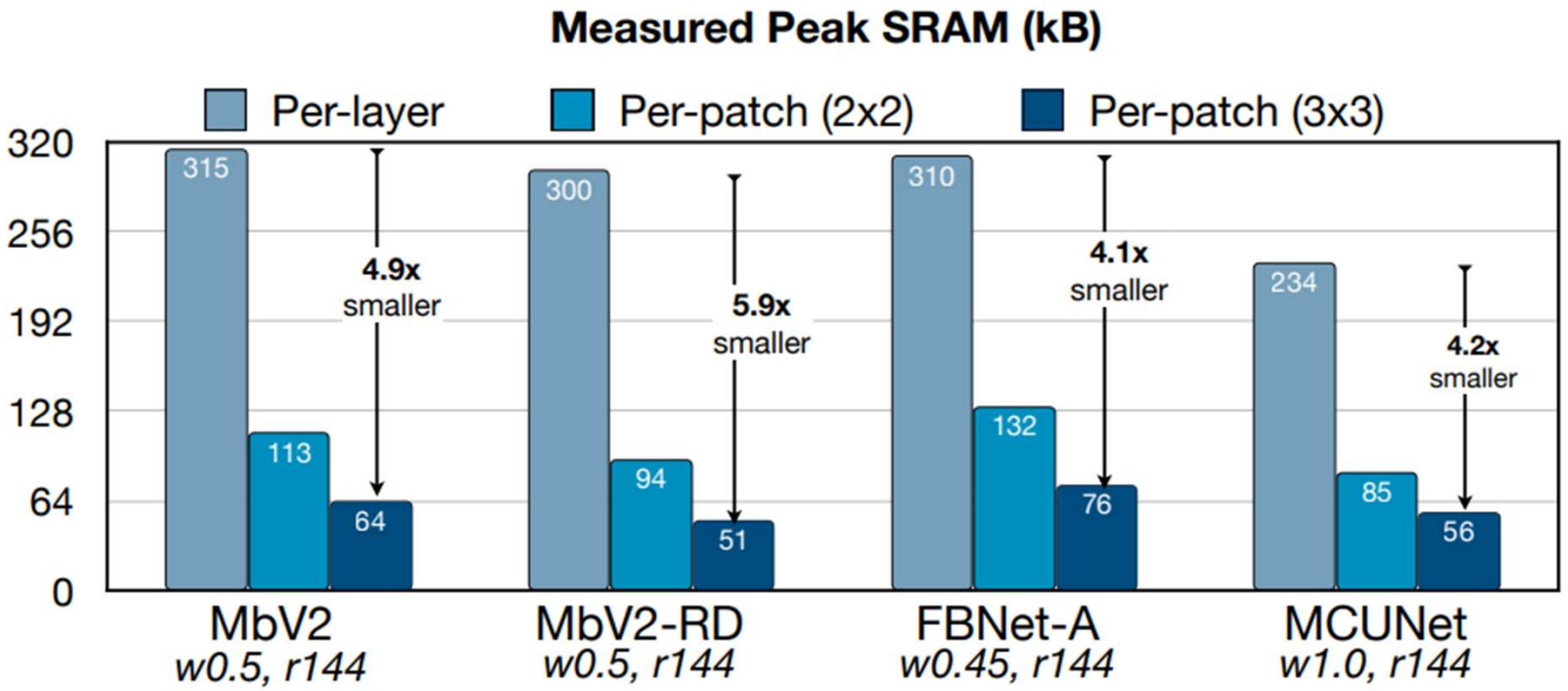
Saving Memory with Patch-based Inference

- Applying to MobileNet-V2



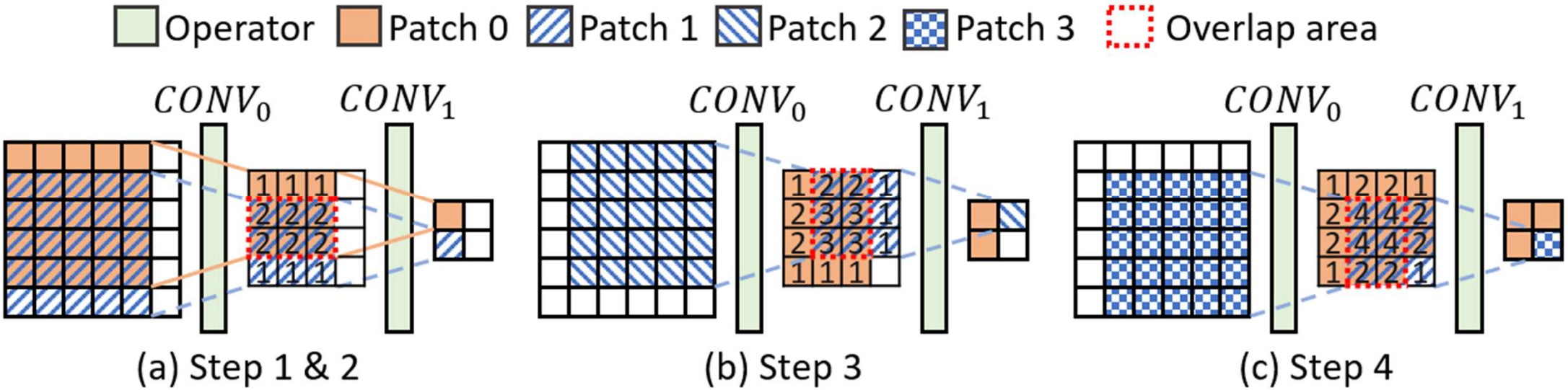
Reducing the Peak Memory of CNNs

- Baseline: MCUNet-v1 on STM32F746 MCU



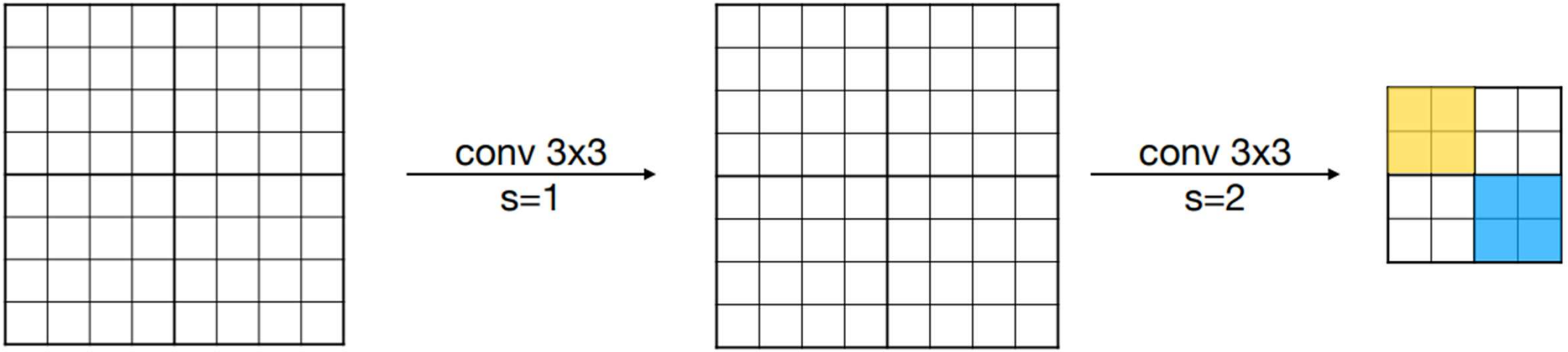
Patch-based Inference Operation

- Store each patch of activations in the SRAM memory
  - Computation overhead from overlapping



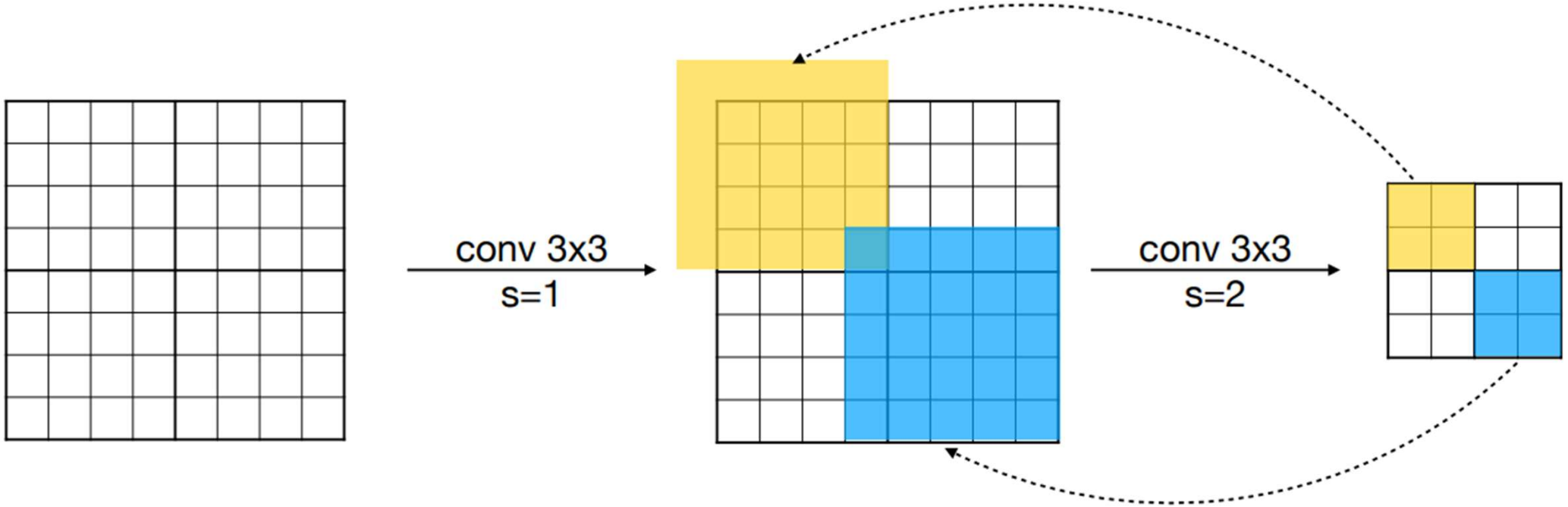
Problem: Computation overhead from Overlapping

- Using 2 x 2 patches



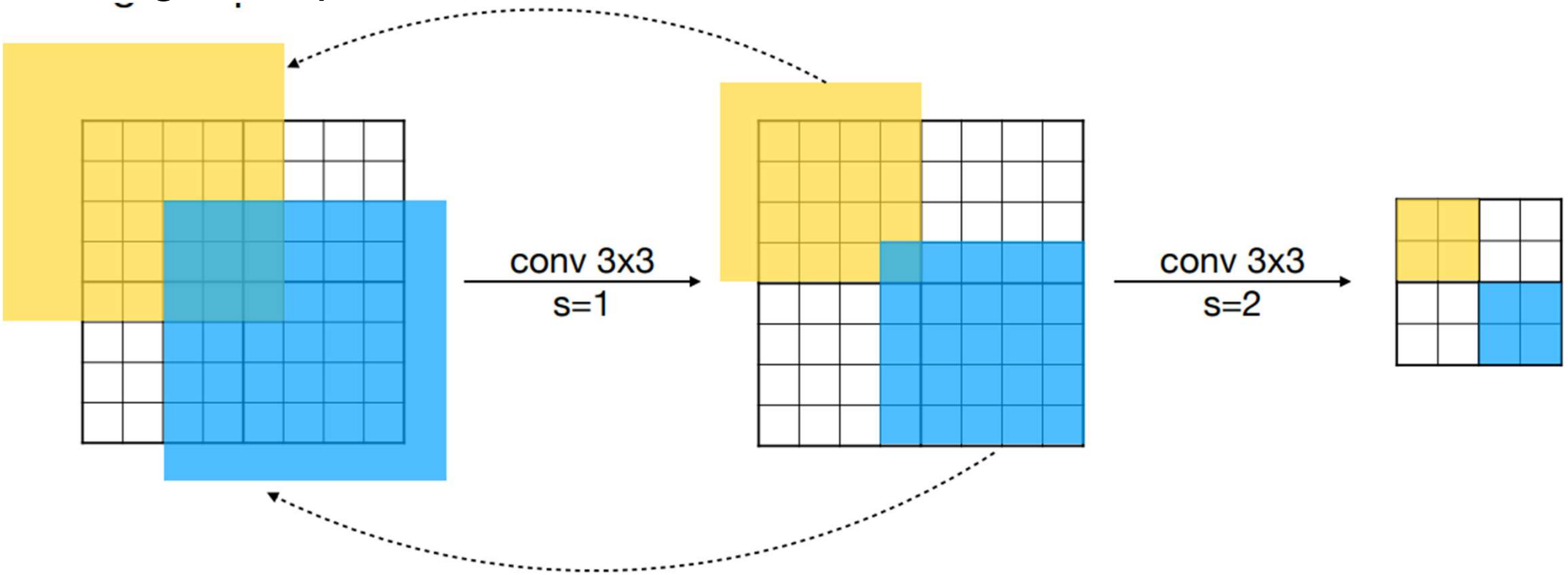
Problem: Computation overhead from Overlapping

- Using 2 x 2 patches



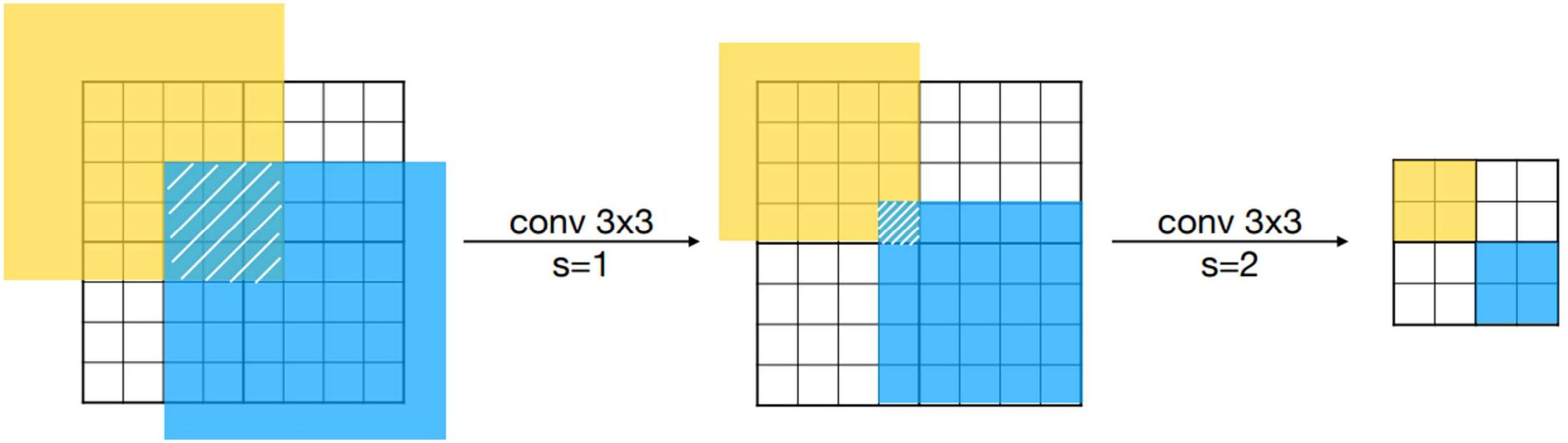
Problem: Computation overhead from Overlapping

- Using 2 x 2 patches



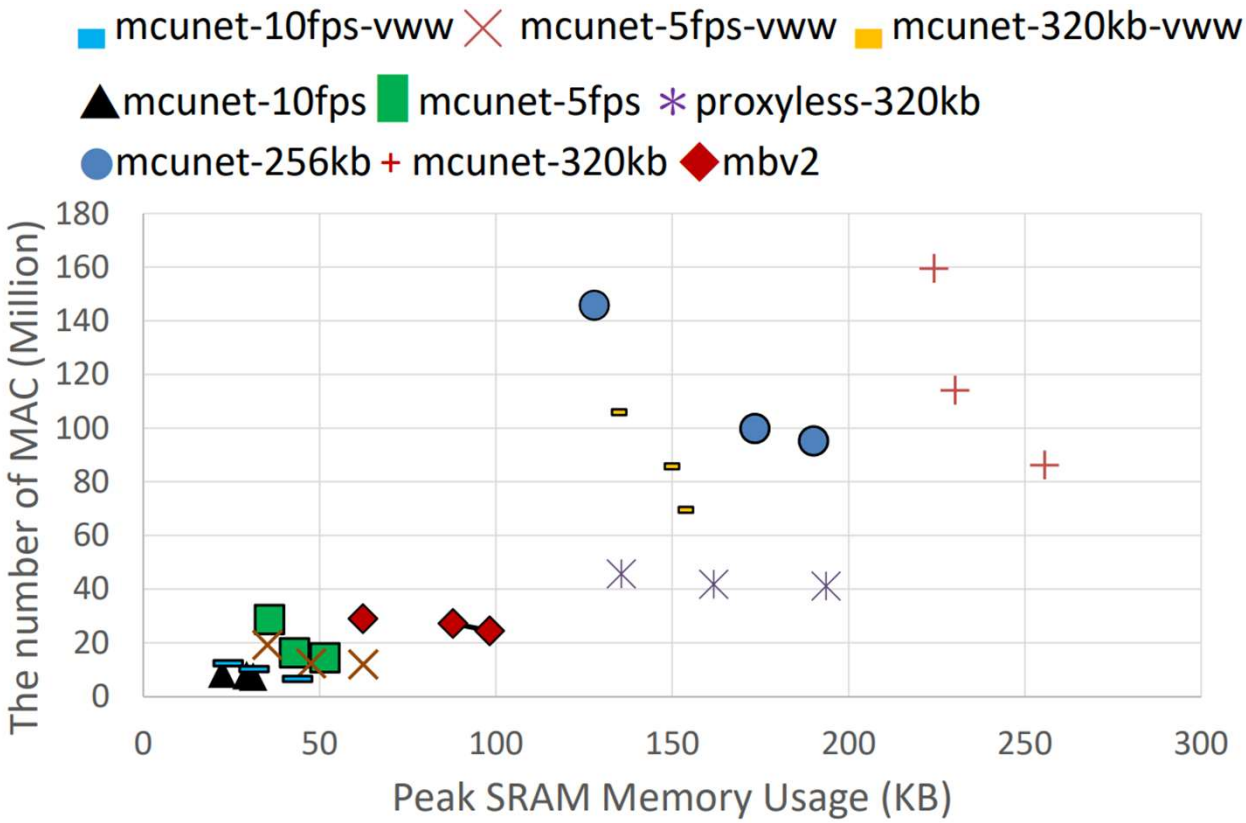
Problem: Computation overhead from Overlapping

- Spatial overlapping gets larger as receptive field grows



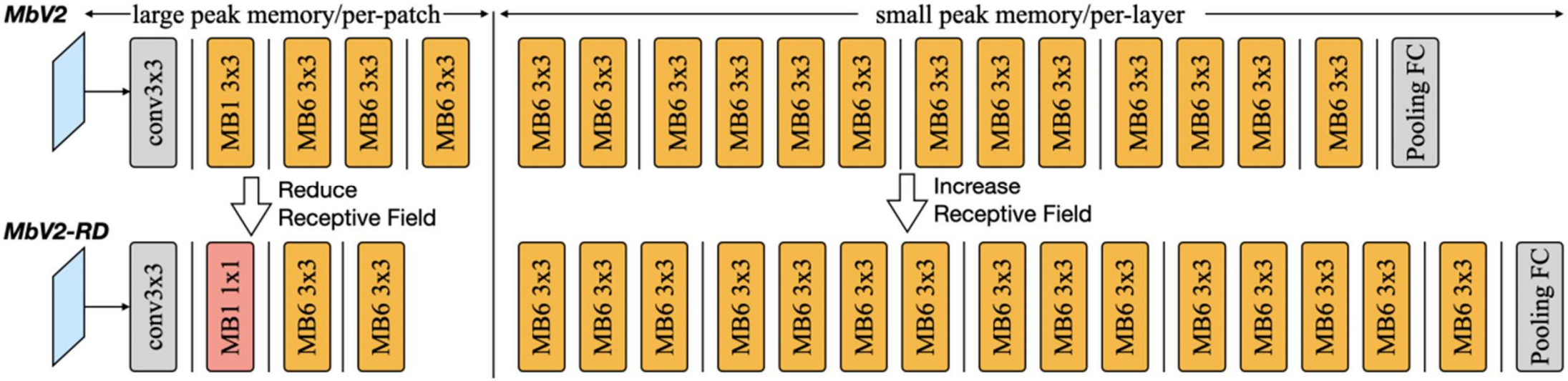
Computation overhead from overlapping

- Patch-based inference overhead
  - The number of MACs increases with the reduction of peak memory



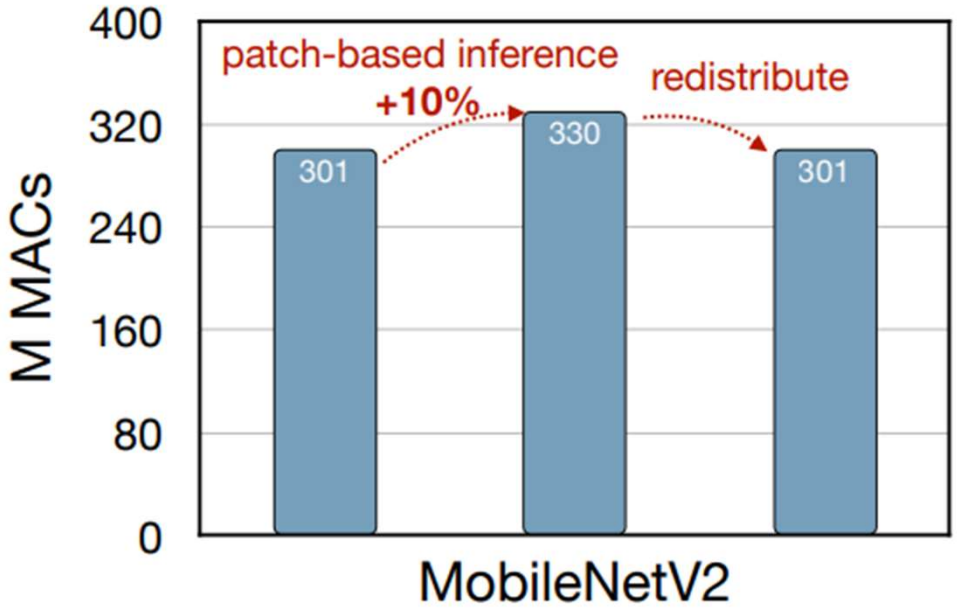
Network Redistribution to Reduce Overhead

- Reduce the size of receptive fields in certain layers



Network Redistribution to Reduce Overhead

- Using receptive fields on different layers can reduce the overhead



Summary

- Running CNNs on Microcontrollers
  - Flash usage: store model parameters
  - SRAM usage: input/output activations
- Today's CNN are Too Big for TinyML
  - Reduce model size and MACs – MobileNet
  - Reduce the intermediate tensor size - MCUNet

Takeaway Questions

- Why does TinyML put input/output activations on SRAM memory?
  - (A) Low latency
  - (B) The speed of data written on SRAM is higher than Flash
  - (C) The SRAM memory density is high
- What are advantages of depth-wise separable convolution?
  - (A) Reduce the size of input/output activation
  - (B) Decrease the number of layers
  - (C) Shrink the model size

Takeaway Questions

- Why are correct descriptions for MCUNet-v1?
  - (A) Using in-place depthwise convolution to reduce peak memory usage
  - (B) Small peak memory contributes the speedup
  - (C) Store activations on a temp buffer
- How does MCUNet-v2 save peak SRAM memory usage?
  - (A) Only store the small patches on SRAM memory
  - (B) Overlapping patch computation
  - (C) Regularization