Advanced Deep Learning Architectures

COMP 5214 & ELEC 5680

Instructor: Dr. Qifeng Chen
https://cqf.io

Efficient Deep Network Architectures

MobileNets:
Efficient Convolutional Neural Networks for Mobile Vision Applications

MobileNets

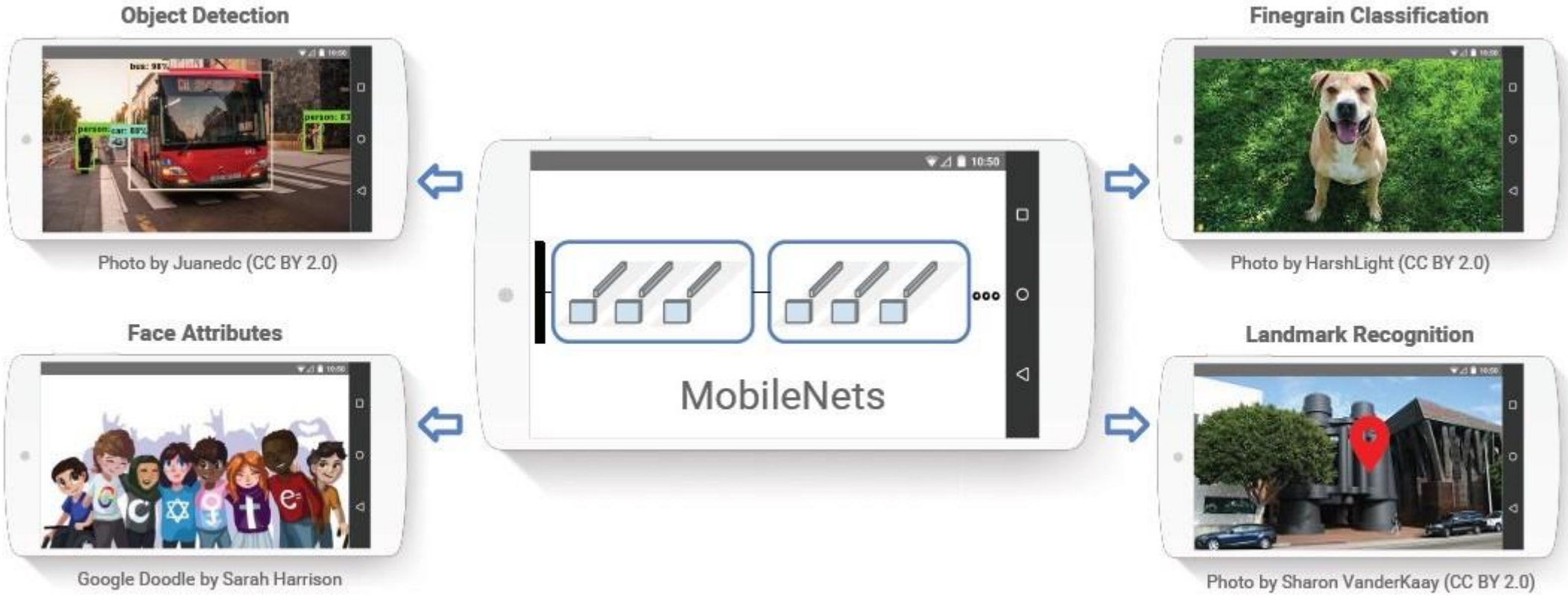


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

Key Requirements for Commercial Computer Vision Usage

- Data-centers(Clouds)
  - Rarely safety-critical
  - Low power is nice to have
  - Real-time is preferable
- Gadgets – Smartphones, Self-driving cars, Drones, etc.
  - Usually safety-critical(except smartphones)
  - Low power is must-have
  - Real-time is required

Slide Credit : Small Deep Neural Networks - Their Advantages, and Their Design by Forrest Iandola

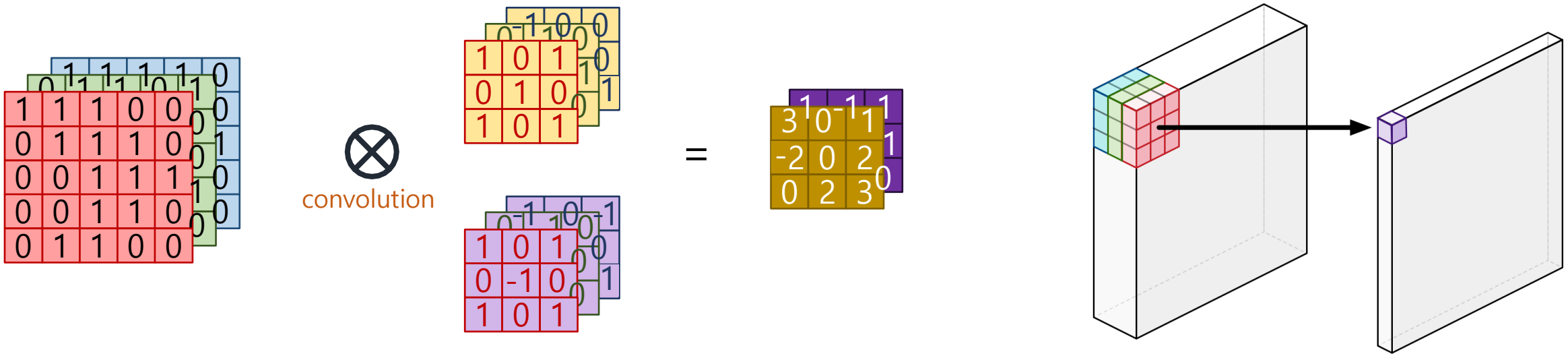
Techniques for Small Deep Neural Networks

- Remove Fully-Connected Layers
- Kernel Reduction (3x3 → 1x1)
- Channel Reduction
- Evenly Spaced Downsampling
- Depthwise Separable Convolutions
- Shuffle Operations
- Distillation & Compression

Slide Credit : Small Deep Neural Networks - Their Advantages, and Their Design by Forrest Iandola

Key Idea : Depthwise Separable Convolution!

Recap – Convolution Operation



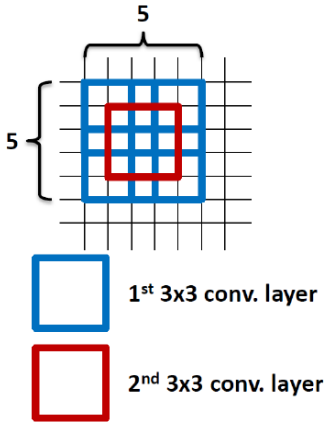
Input channel : 3

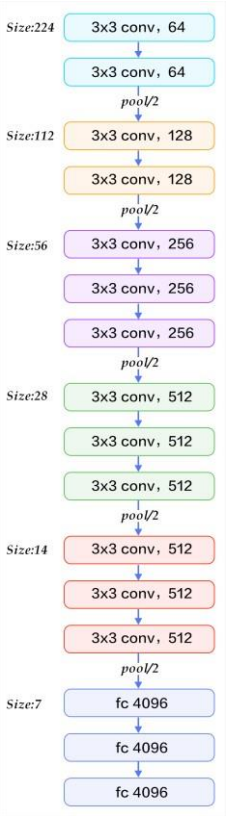
# of filters : 2

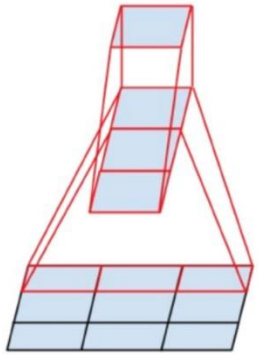
Output channel : 2

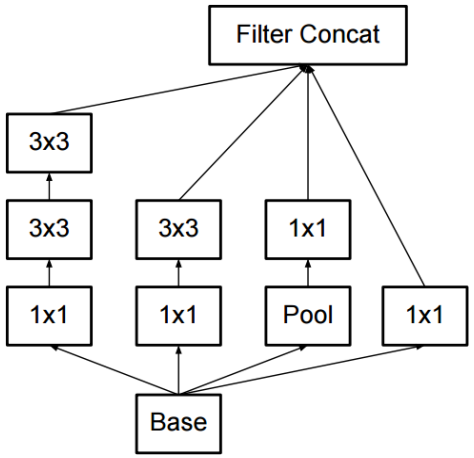
Recap – VGG, Inception-v3

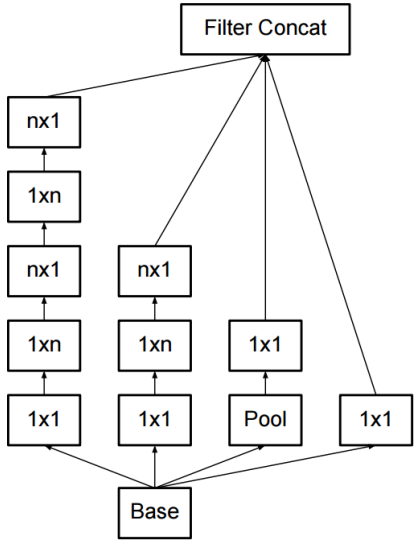
- VGG – use only 3x3 convolution
  - Stack of 3x3 conv layers has same effective receptive field as 5x5 or 7x7 conv layer
  - Deeper means more non-linearities
  - Fewer parameters: 2 x (3 x 3 x C) vs (5 x 5 x C)
    → regularization effect
- Inception-v3
  - Factorization of filters

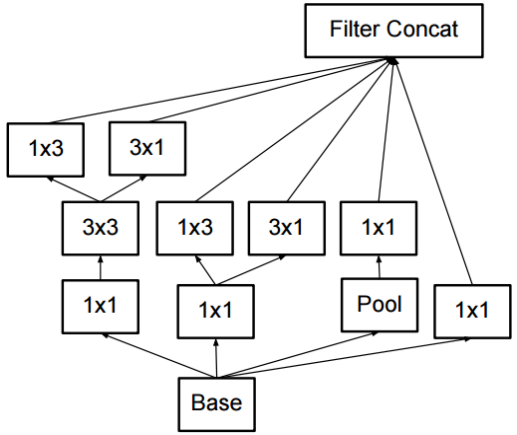






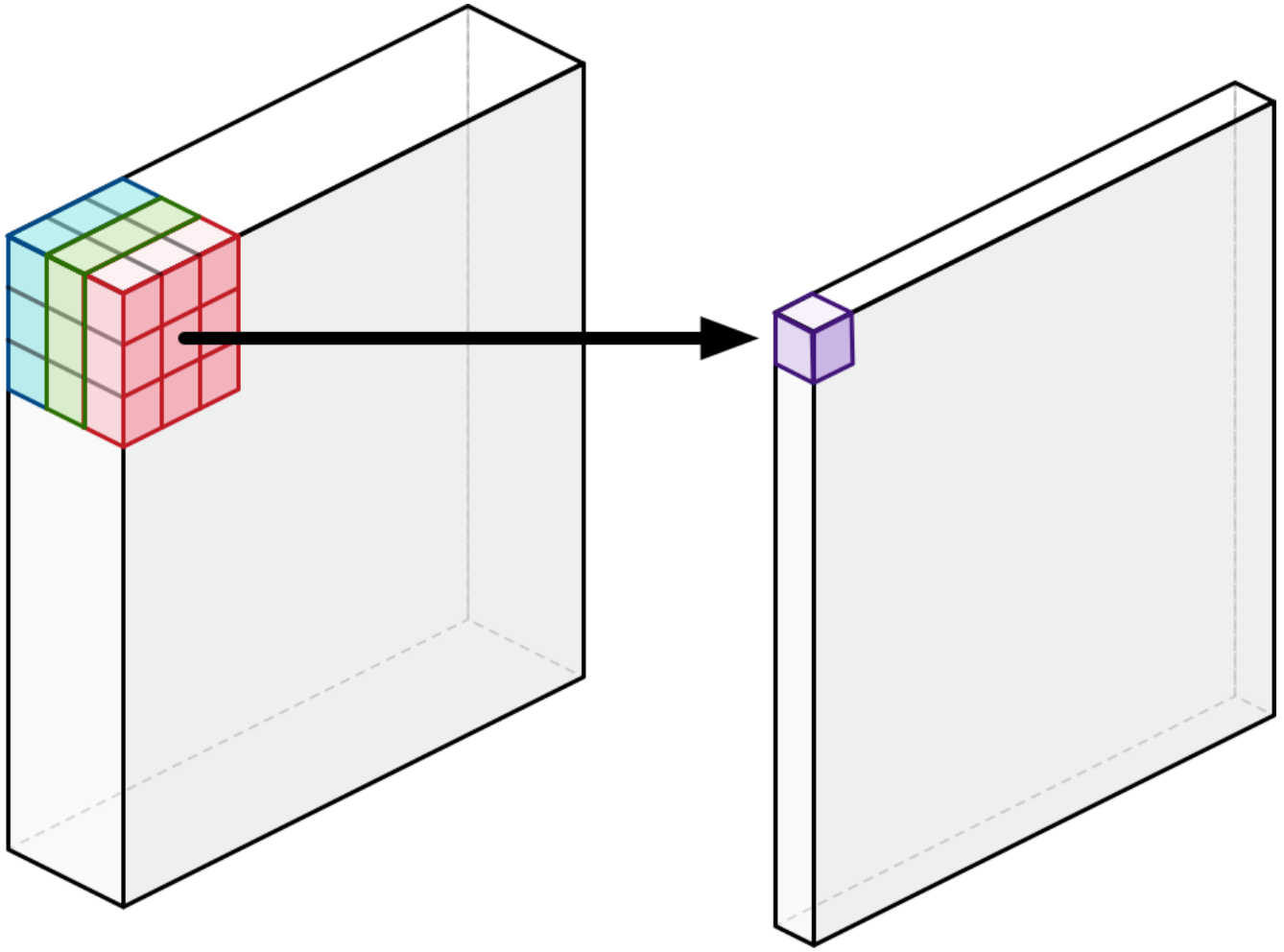






Why should we always consider all channels?

Standard Convolution

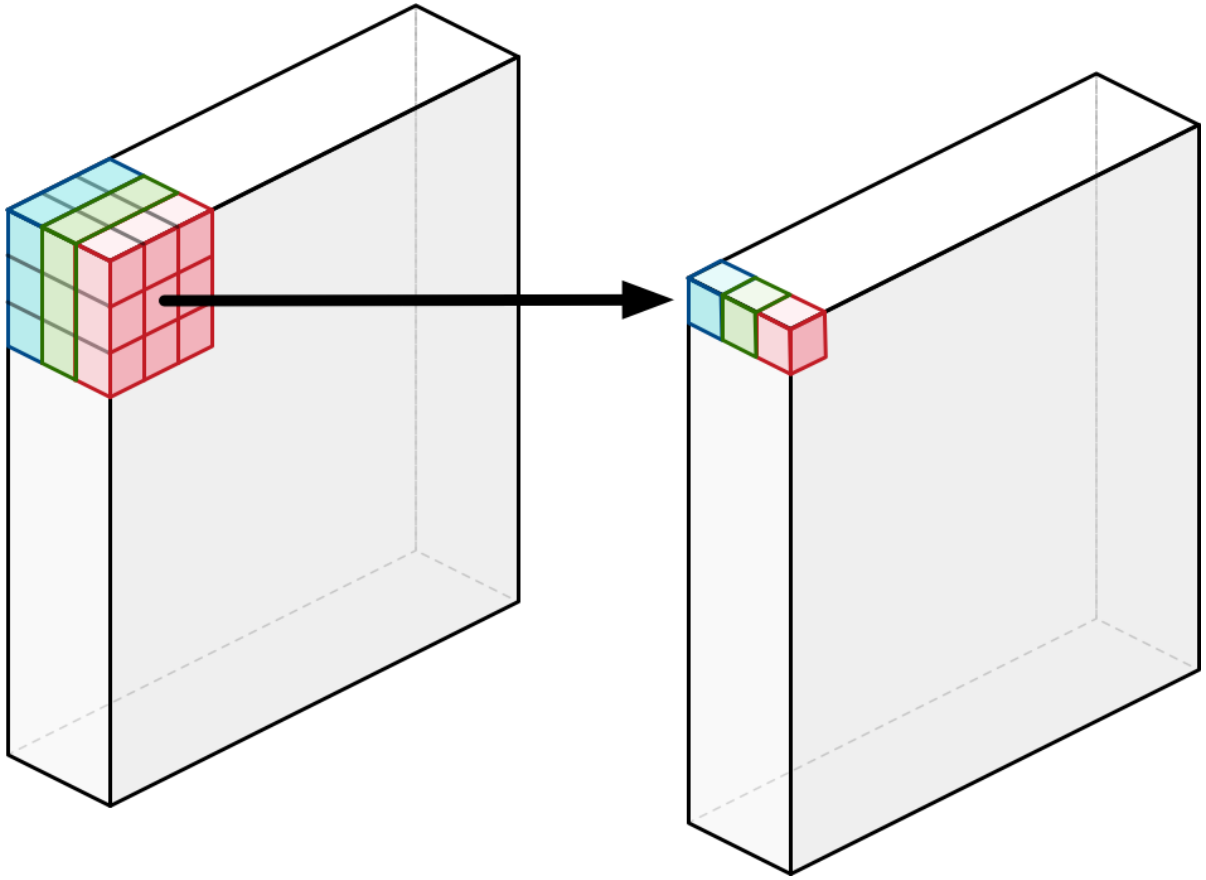


Depthwise convolution

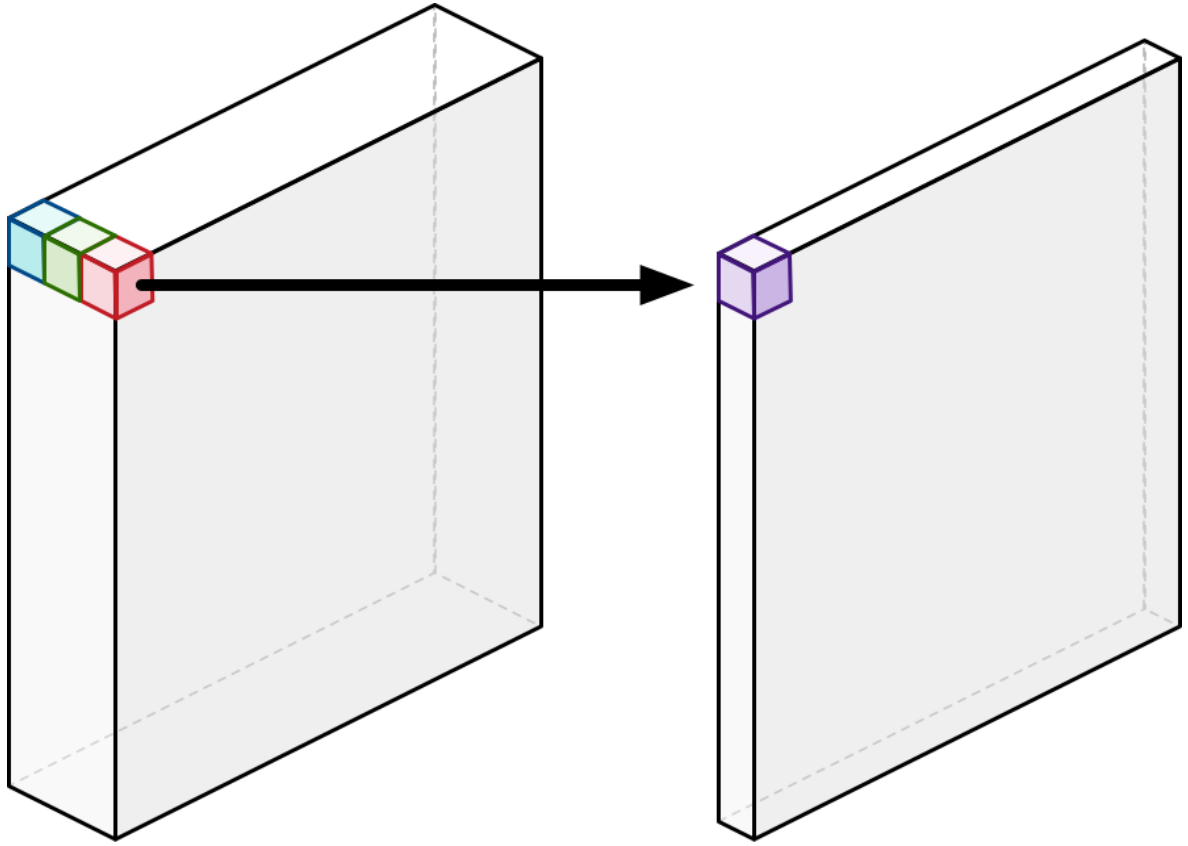
Figures from http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

Depthwise Separable Convolution

- Depthwise Convolution + Pointwise Convolution(1x1 convolution)



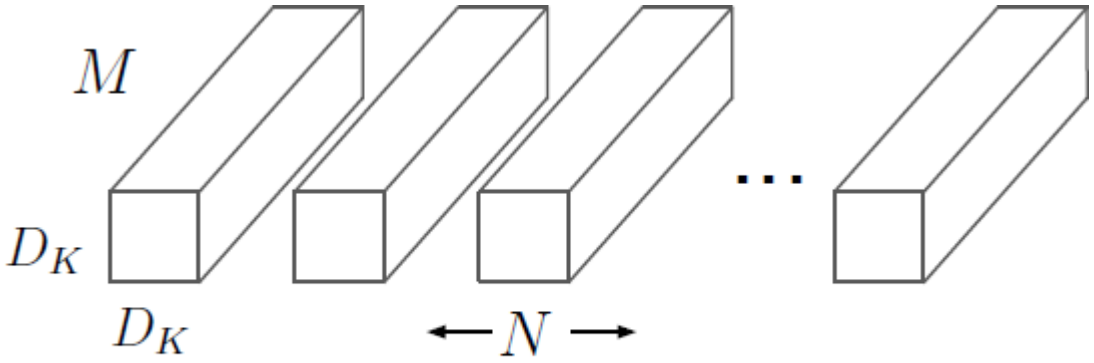
Depthwise convolution



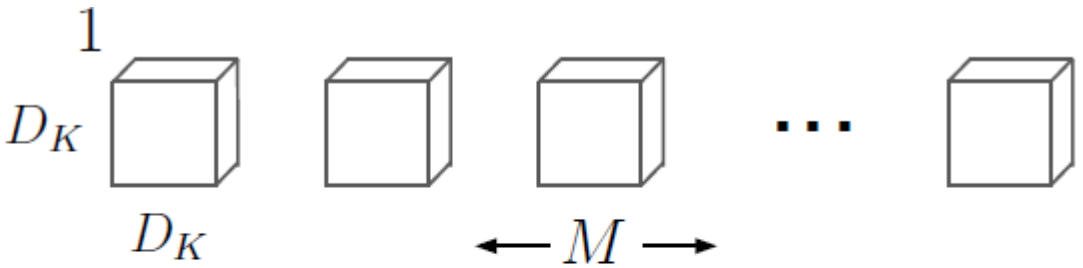
Pointwise convolution

Figures from http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

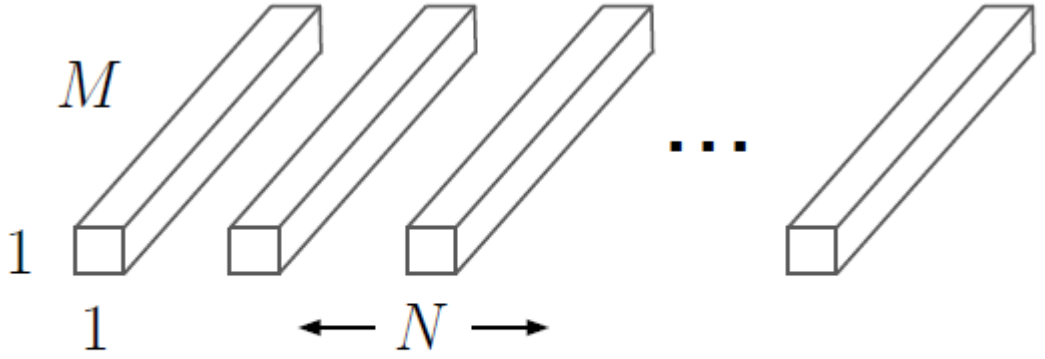
Standard Convolution vs Depthwise Separable Convolution



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Standard Convolution vs Depthwise Separable Convolution

- Standard convolutions have the computational cost of
  - $D_K \times D_K \times M \times N \times D_F \times D_F$
- Depthwise separable convolutions cost
  - $D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F$
- Reduction in computations
  - $1 / N + 1 / D_K^2$
  - If we use 3x3 depthwise separable convolutions, we get between 8 to 9 times less computations

D_K : width/height of filters
D_F : width/height of feature maps
M : number of input channels
N : number of output channels(number of filters)

Depthwise Separable Convolutions

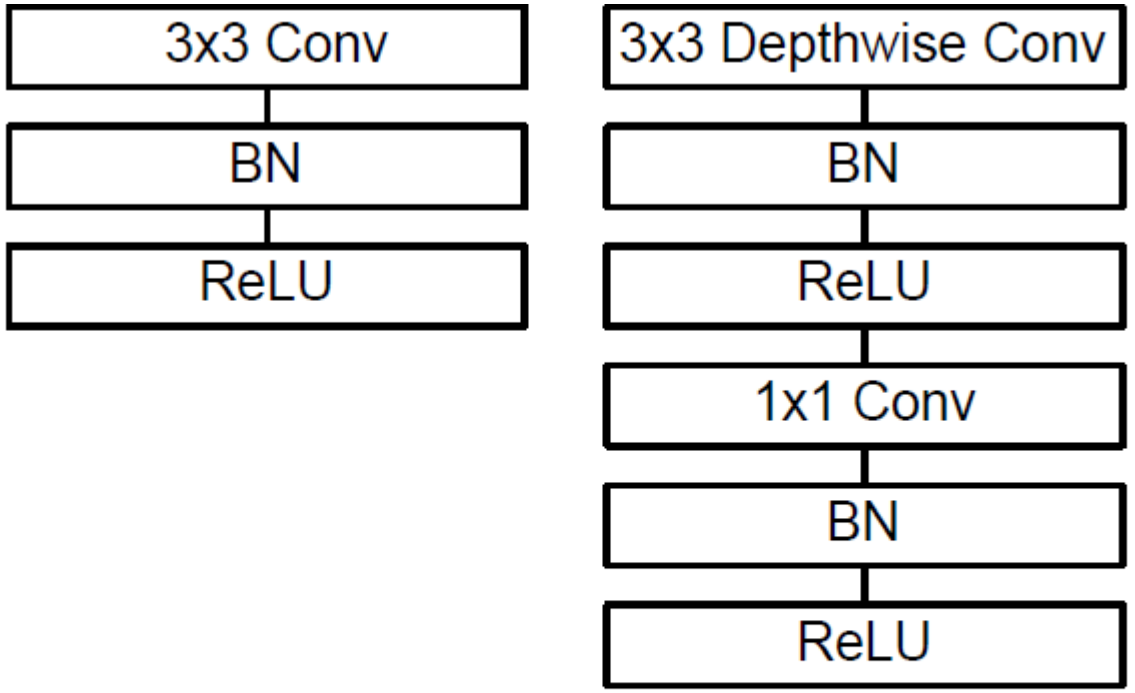


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Width Multiplier & Resolution Multiplier

- Width Multiplier –Thinner Models
  - For a given layer and width multiplier α, the number of input channels M becomes αM and the number of output channels N becomes αN – where α with typical settings of 1, 0.75, 0.6 and 0.25
- Resolution Multiplier – Reduced Representation
  - The second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier ρ
  - $0<\rho\leq 1$, which is typically set of implicitly so that input resolution of network is 224, 192, 160 or 128($\rho = 1,\ 0.857, 0.714, 0.571$)
- Computational cost:

$$D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_F \times \rho D_F$$

Width Multiplier & Resolution Multiplier

Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the previous row. This example is for an internal MobileNet layer with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

Experiments – Model Choices

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

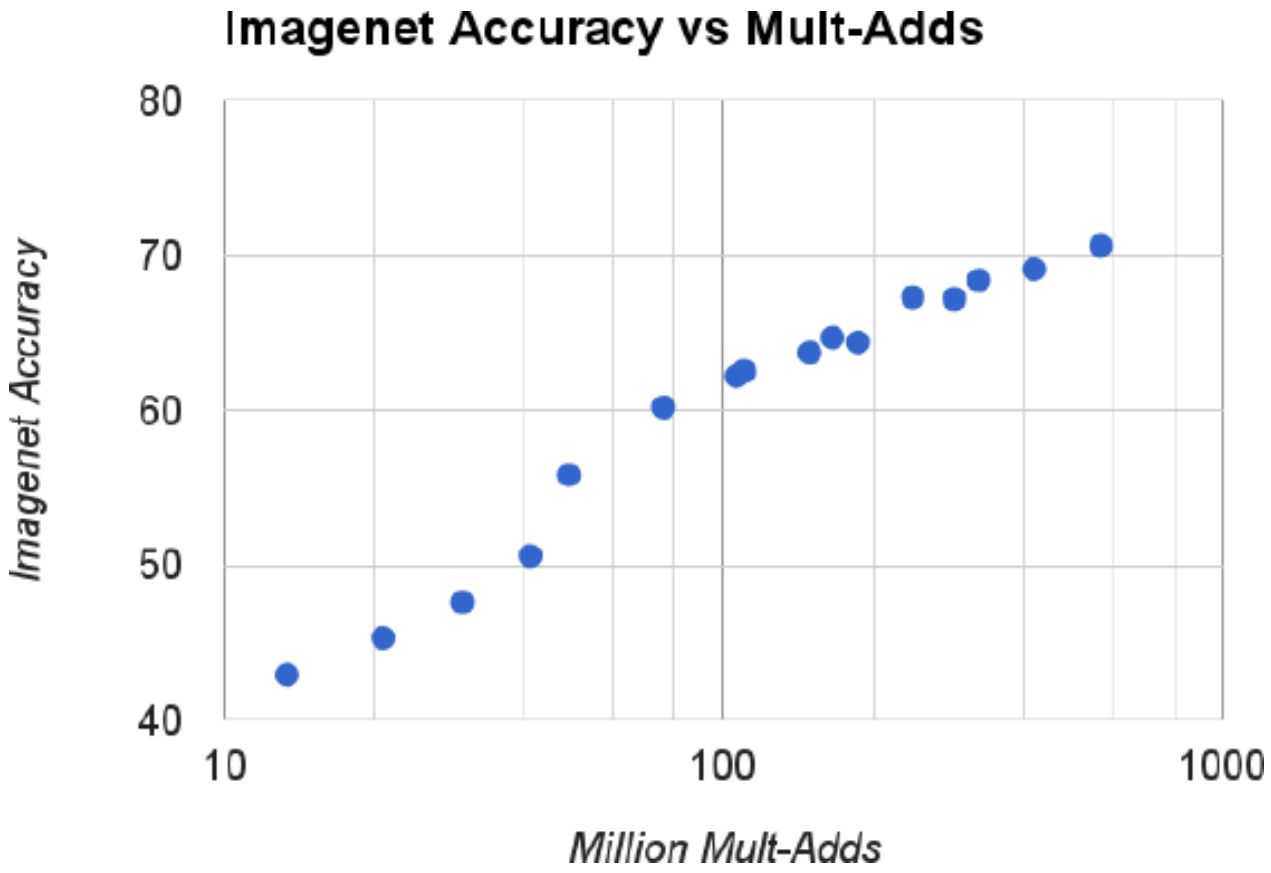
Table 6. MobileNet Width Multiplier

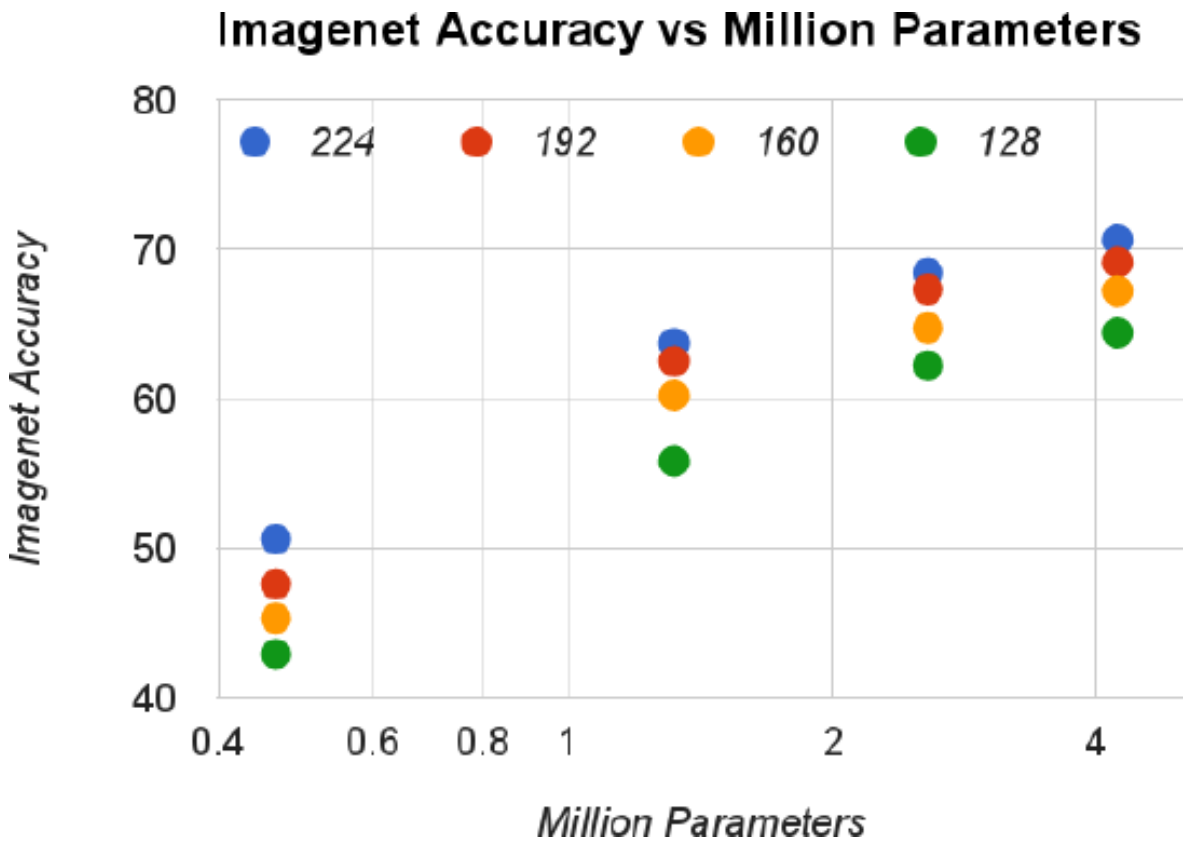
Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

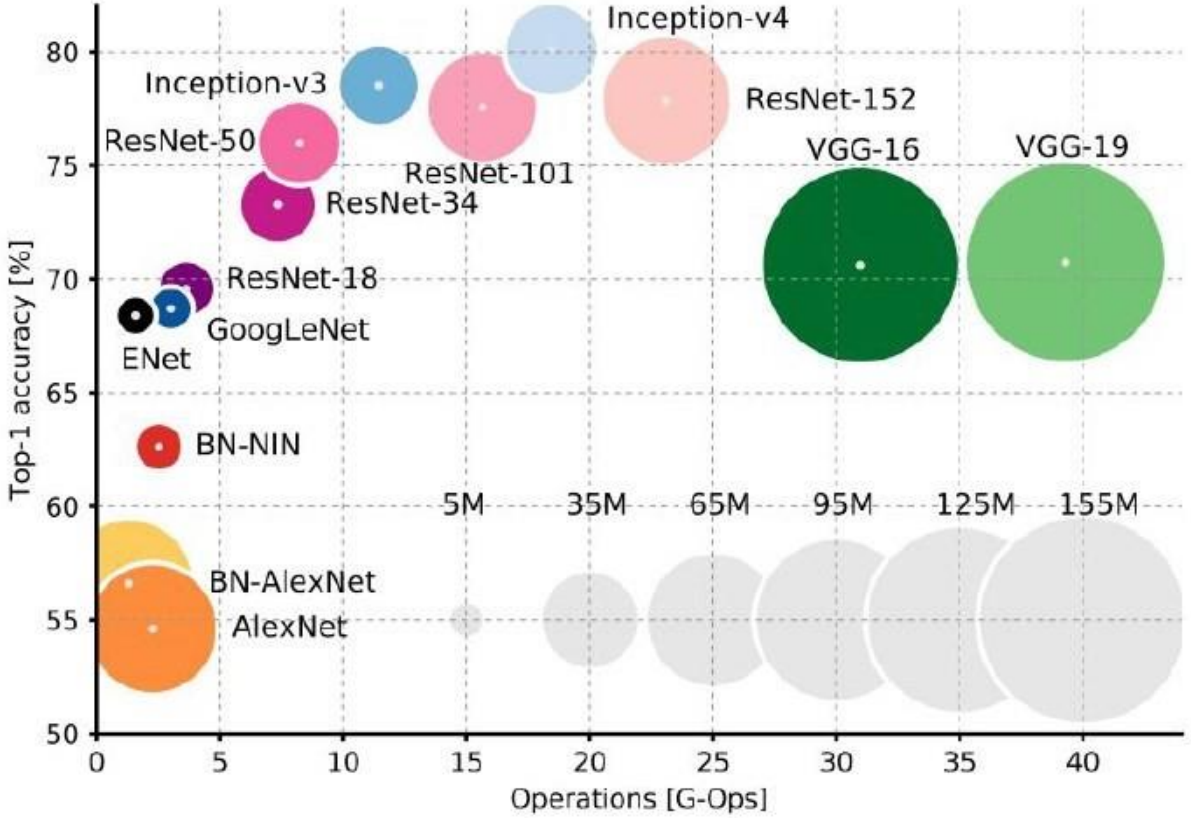
Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

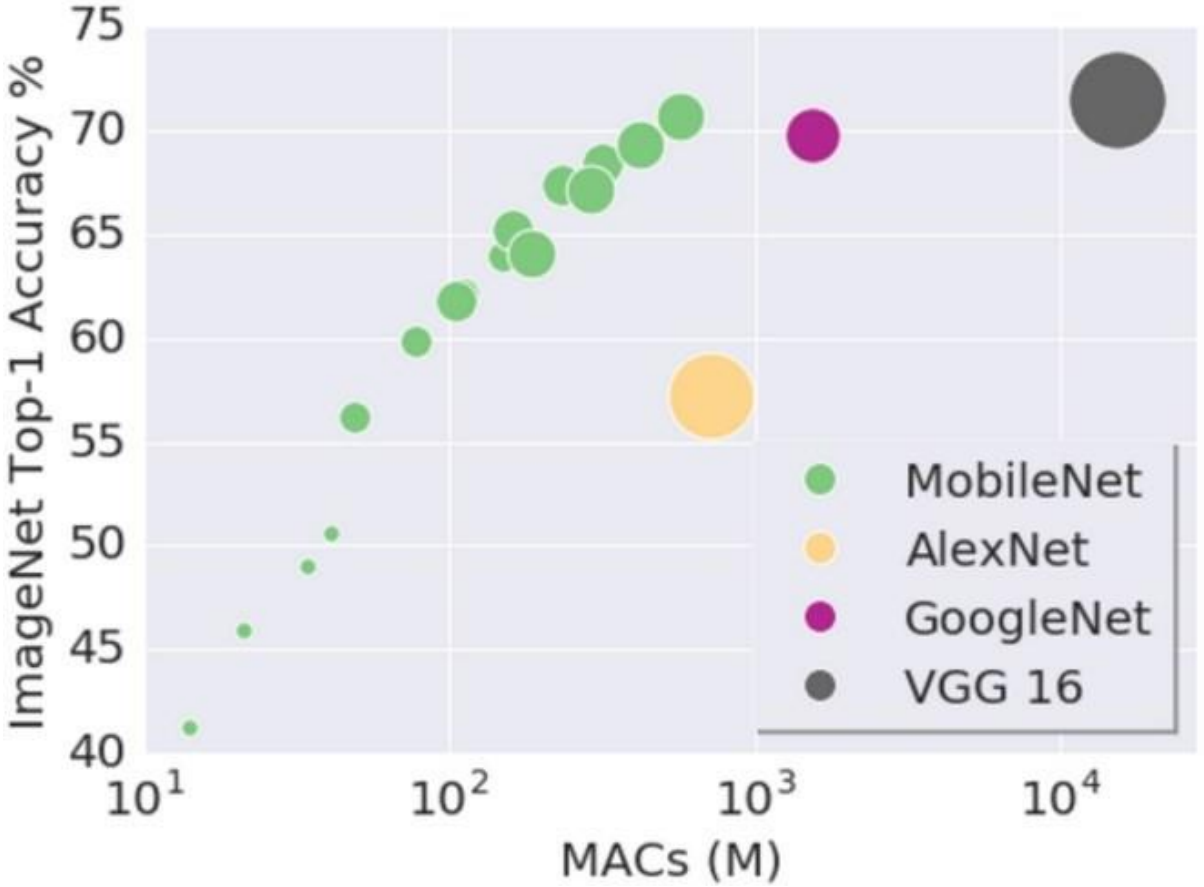
Model Shrinking Hyperparameters





Model Shrinking Hyperparameters





Results

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Results

Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

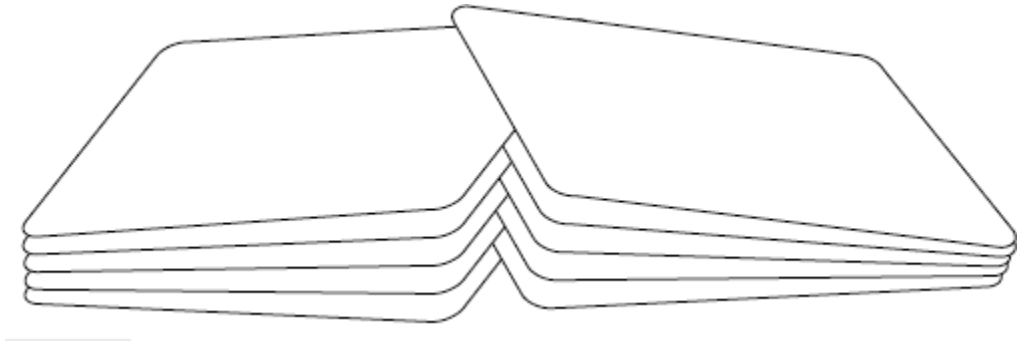
Table 11. Performance of PlaNet using the MobileNet architecture. Percentages are the fraction of the Im2GPS test dataset that were localized within a certain distance from the ground truth. The numbers for the original PlaNet model are based on an updated version that has an improved architecture and training dataset.

Scale	Im2GPS [7]	PlaNet [35]	PlaNet MobileNet
Continent (2500 km)	51.9%	77.6%	79.3%
Country (750 km)	35.4%	64.0%	60.3%
Region (200 km)	32.1%	51.1%	45.2%
City (25 km)	21.9%	31.7%	31.7%
Street (1 km)	2.5%	11.0%	11.4%

PlaNet : 52M parameters, 5.74B mult-adds
MobilNet : 13M parameters, 0.58M mult-adds

ShuffleNet:

An Extremely Efficient Convolutional Neural Network for Mobile Devices



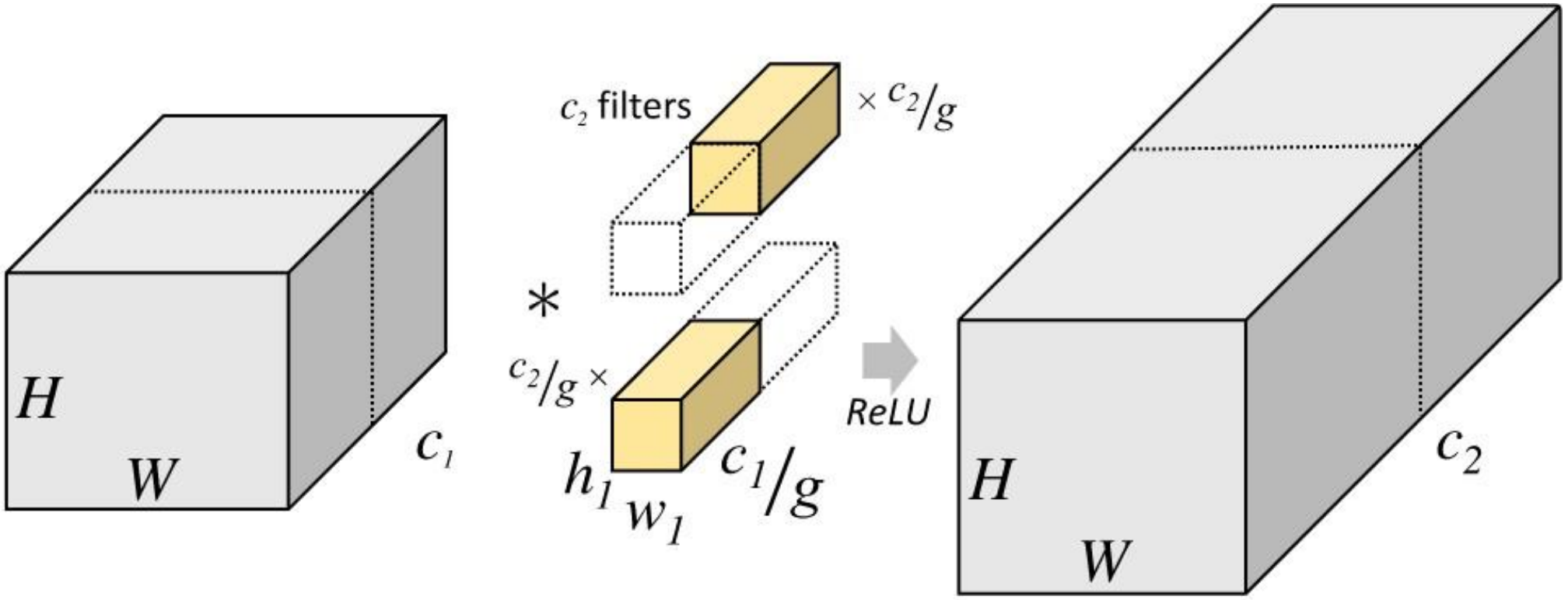
Xiangyu Zhang, et al. "ShuffleNet: an extremely efficient convolutional neural network for mobile devices.", arXiv:1707.01083

Credit: Jinwon Lee
Samsung Electronics

Main Ideas of ShuffleNet

- (Use depthwise separable convolution)
- Grouped convolution on 1x1 convolution layers – pointwise group convolution
- Channel shuffle operation after pointwise group convolution

Grouped Convolution

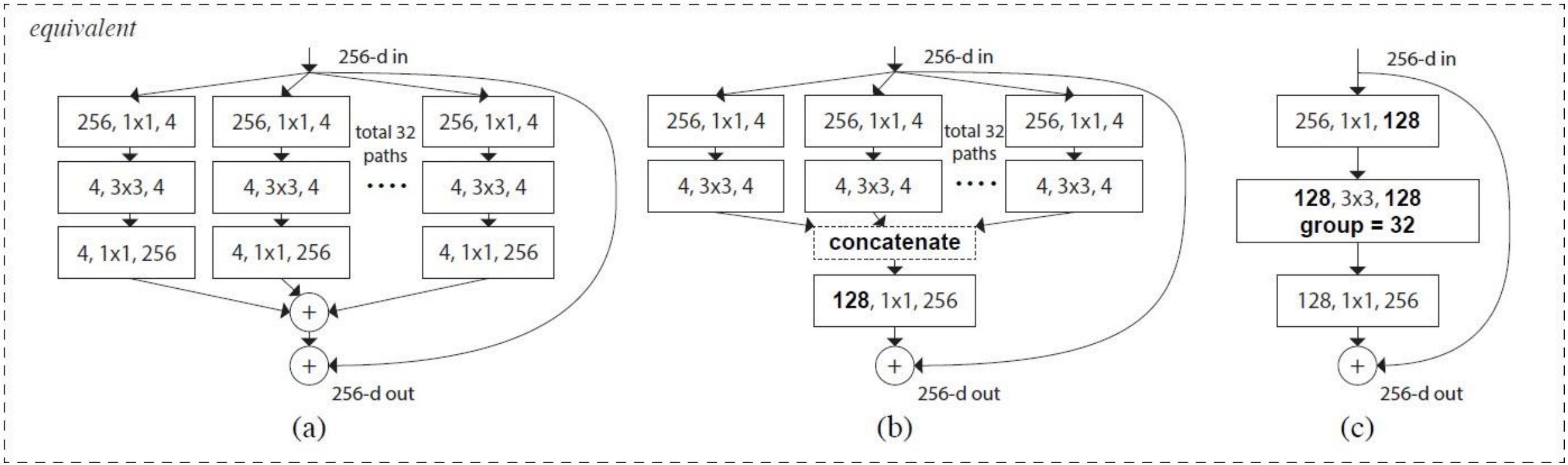


A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convolutional layer is now exactly half the depth, i.e. half the parameters and half the compute as the original filter.

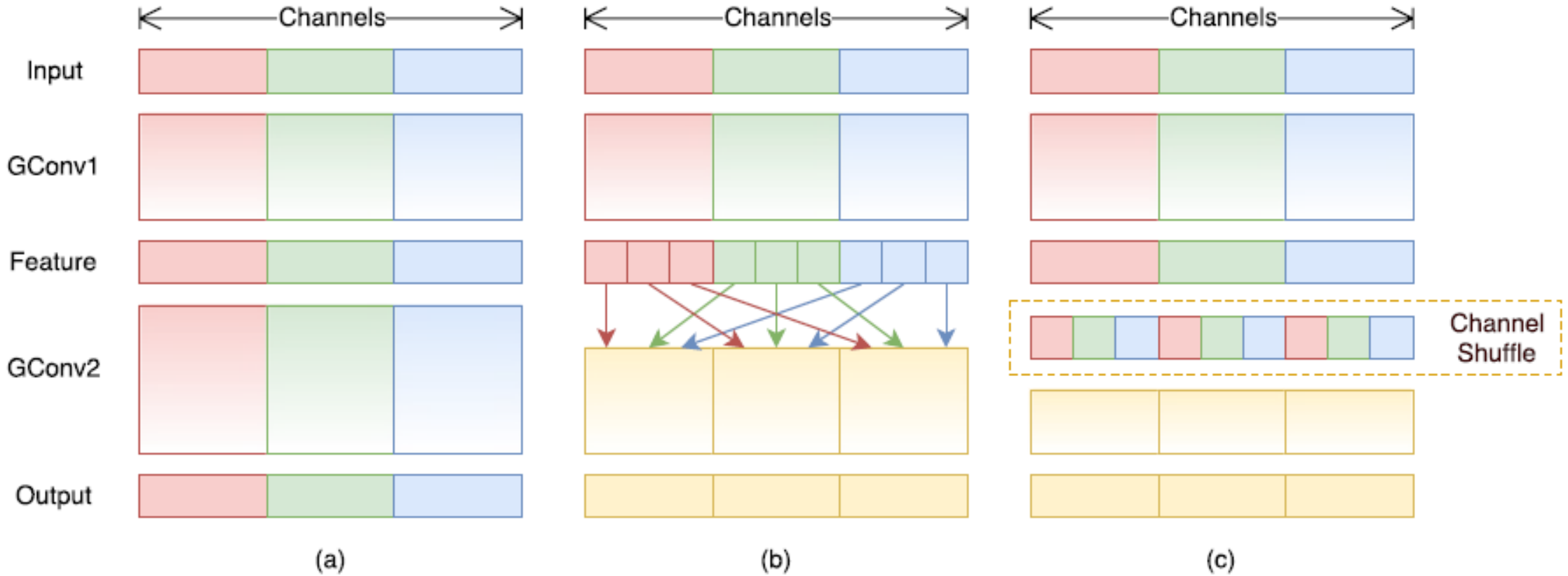
Figure from https://blog.yani.io/filter-group-tutorial/

ResNext

- Equivalent building blocks of RexNext (cardinality=32)



1x1 Grouped Convolution with Channel



- If multiple group convolutions stack together, there is one side effect(a)
  - Outputs from a certain channel are only derived from a small fraction of input channels
- If we allow group convolution to obtain input data from different groups, the input and outputchannels will be fully related.

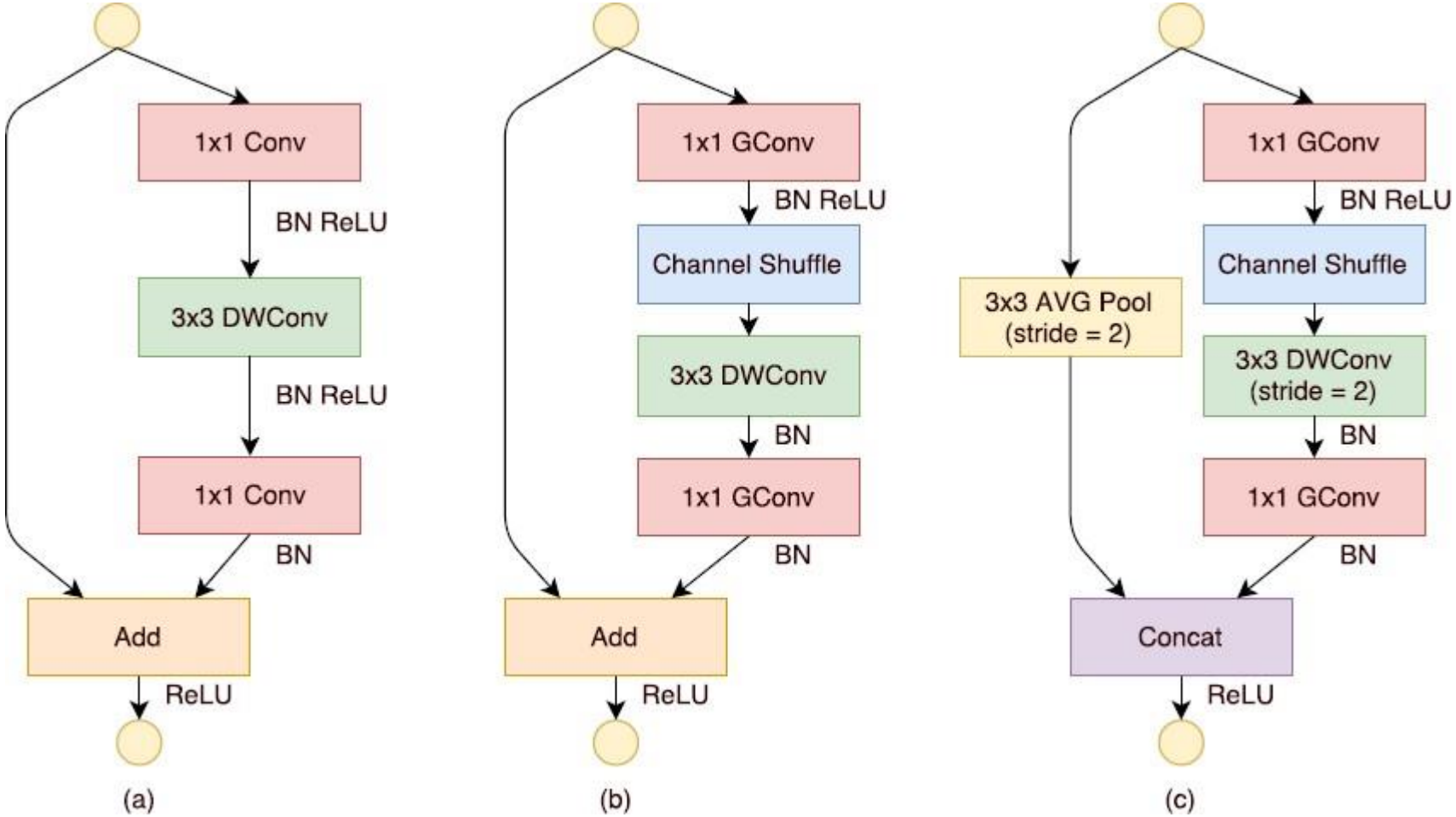
Channel Shuffle Operation

- Suppose a convolutional layer with g groups whose output has g x n channels; we first reshape the output channel dimension into (g, n), transposing and then flattening it back as the input of next layer.
- Channel shuffle operation is also differentiable

```
def channel_shuffle(name, x, num_groups):
    with tf.variable_scope(name) as scope:
        n, h, w, c = x.shape.as_list()
        x_reshaped = tf.reshape(x, [-1, h, w, num_groups, c // num_groups])
        x_transposed = tf.transpose(x_reshaped, [0, 1, 2, 4, 3])
        output = tf.reshape(x_transposed, [-1, h, w, c])
        return output
```

Tensorflow Implementation from https://github.com/MG2033/ShuffleNet

ShuffleNet Units



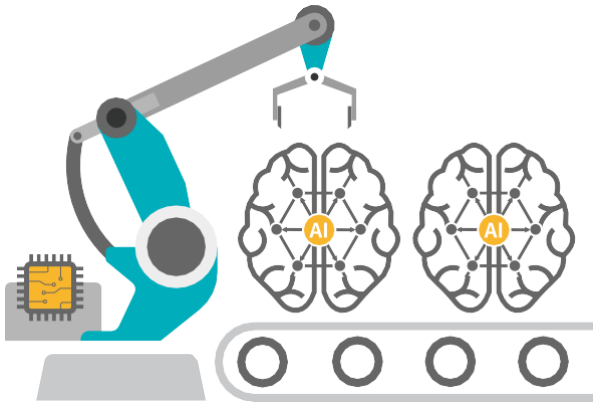
Experimental Results

- It is clear that ShuffleNet models are superior to MobileNet for all the complexities though ShuffleNet network is specially designed for small models (< 150 MFLOPs)

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet 2× ($g = 3$)	524	**26.3**	3.1
ShuffleNet 2× (with *SE*[13], $g = 3$)	527	**24.7**	4.7
0.75 MobileNet-224	325	31.6	-
ShuffleNet 1.5× ($g = 3$)	292	**28.5**	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet 1× ($g = 8$)	140	**32.4**	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet 0.5× ($g = 4$)	38	**41.6**	7.8
ShuffleNet 0.5× (shallow, $g = 3$)	40	42.8	6.6

Table 5. ShuffleNet vs. MobileNet [12] on ImageNet Classification

Hardware Efficiency Aware Neural Architecture Search and Compression



Song Han
Assistant Professor
Massachusetts Institute of Technology

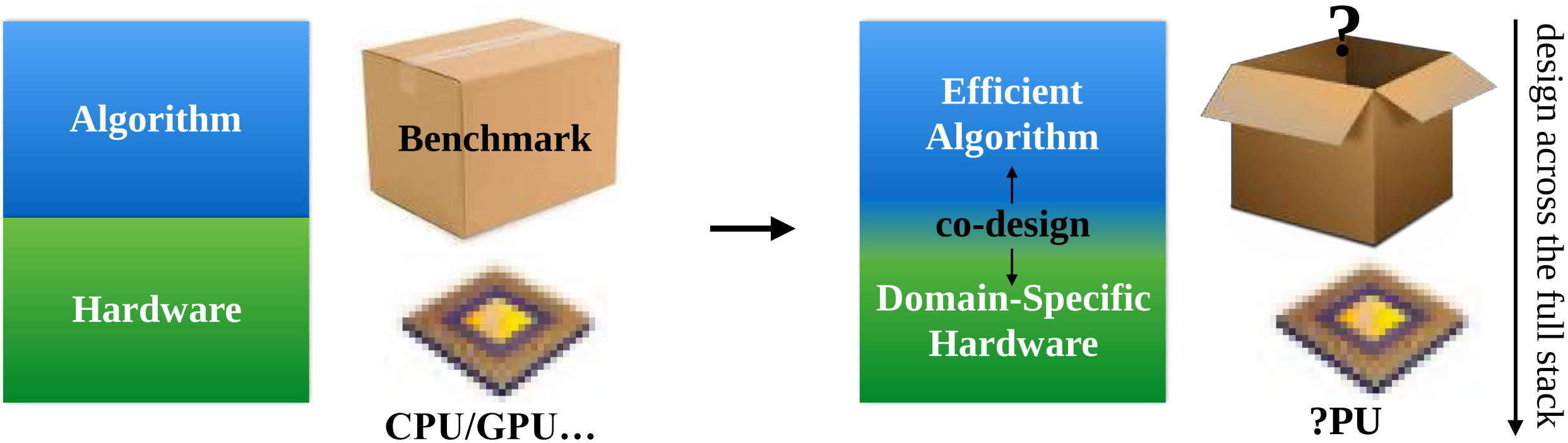
https://songhan.mit.edu

A Challenge for Deep Learning Computing

- We are solving more complicated AI problems with larger datasets, which **requires more computation**.

- However, Moore's Law is slowing down; the amount of computation per unit cost is **no longer increasing** at its historic rate.

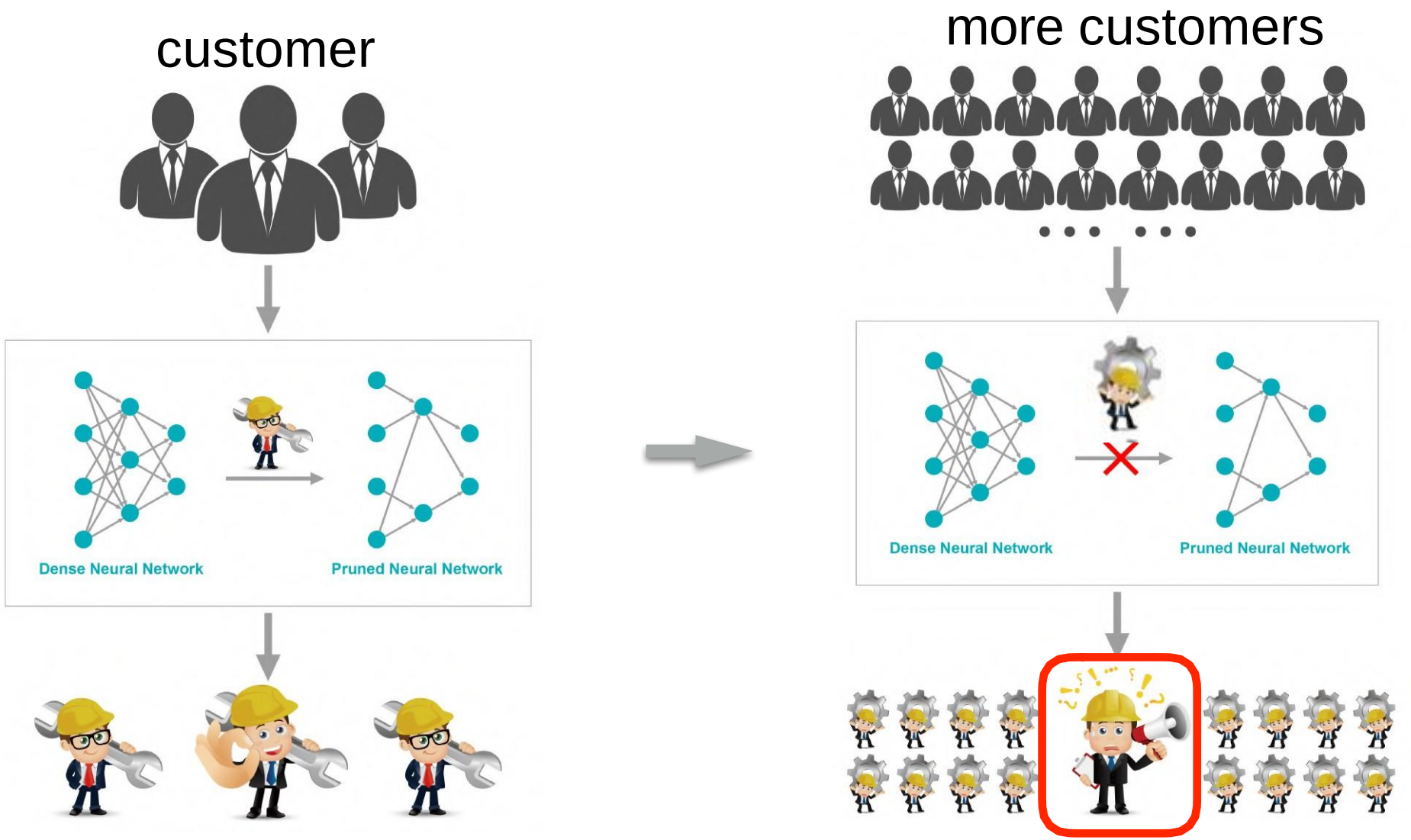
We Need Algorithm and Hardware Co-Design



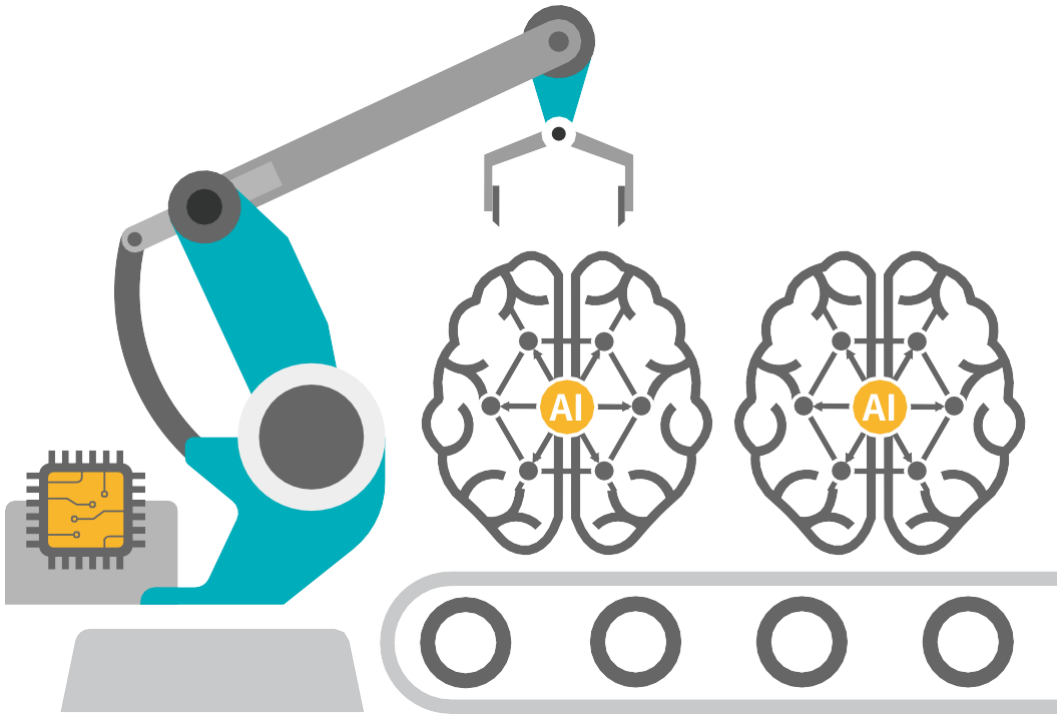
"**There is plenty of room at the top** by optimizing the algorithm. We found that DNN models can be significantly compressed and simplified"

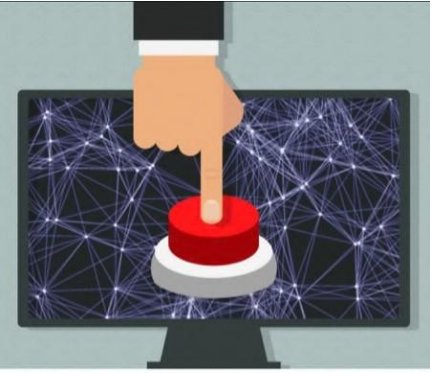
Song Han, Stanford PhD thesis, 2017. https://purl.stanford.edu/qf934gh3708

There's a Shortage of Deep Learning Engineers

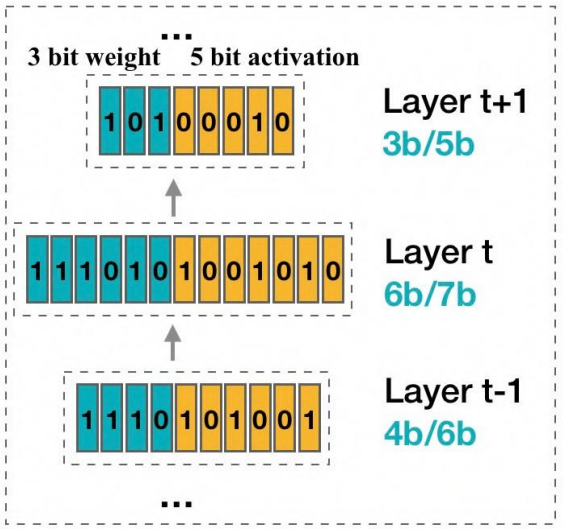


Design Automation for NN



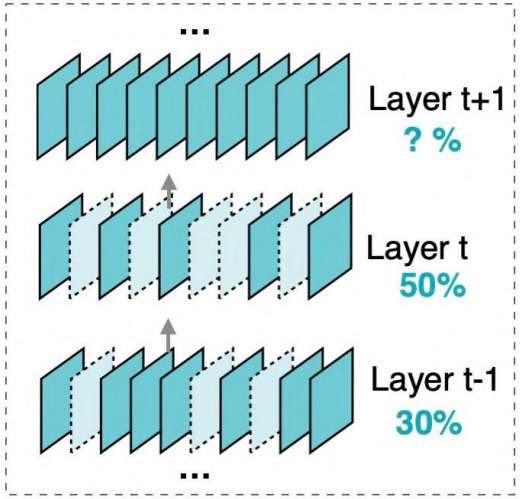


Design Automation for Efficient Deep Learning Computing



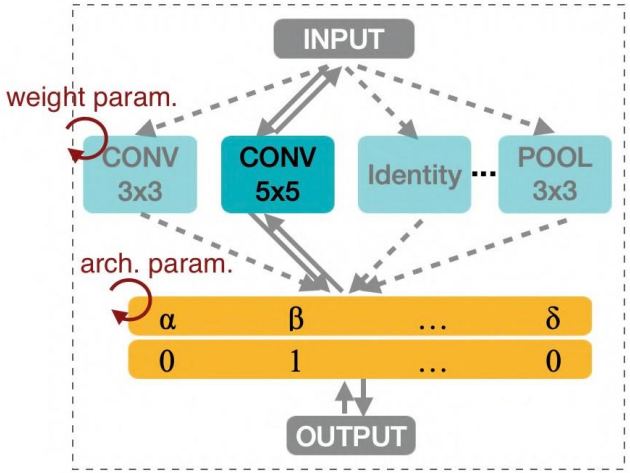
HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



AMC: AutoML for Model Compression

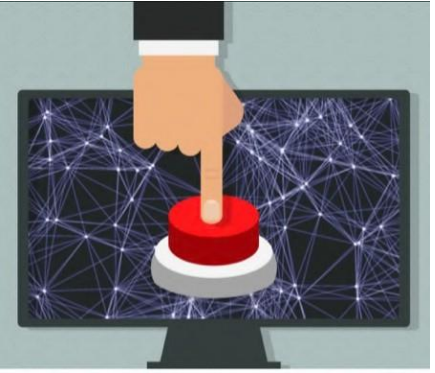
[ECCV 2018]



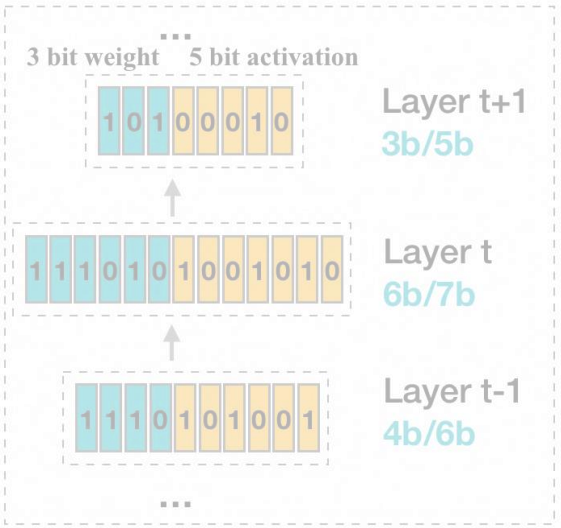
Proxyless Neural Architecture Search

[ICLR 2019]

HAN LAB

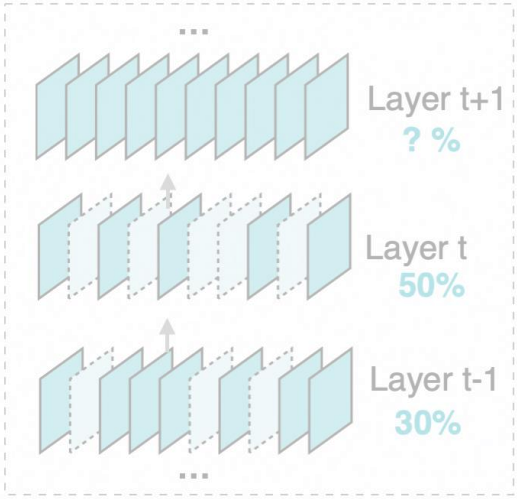


Design Automation for Efficient Deep Learning Computing



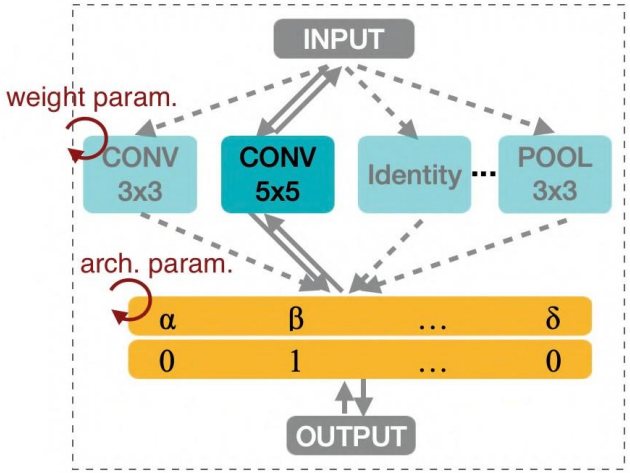
HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



AMC: AutoML for Model Compression

[ECCV 2018]



Proxyless Neural Architecture Search

[ICLR 2019]

HAN LAB

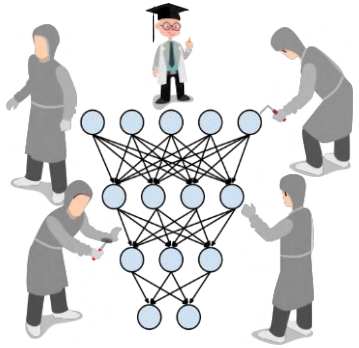
ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware

Han Cai, Ligeng Zhu, Song Han

Massachusetts Institute of Technology

ICLR'19

From Manual Design to Automatic Design

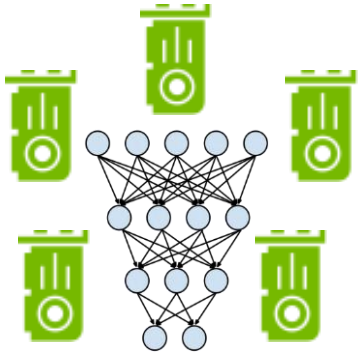


Use Human Expertise

Manual Architecture Design

VGGNets
Inception Models
ResNets
DenseNets
….

Computational Resources

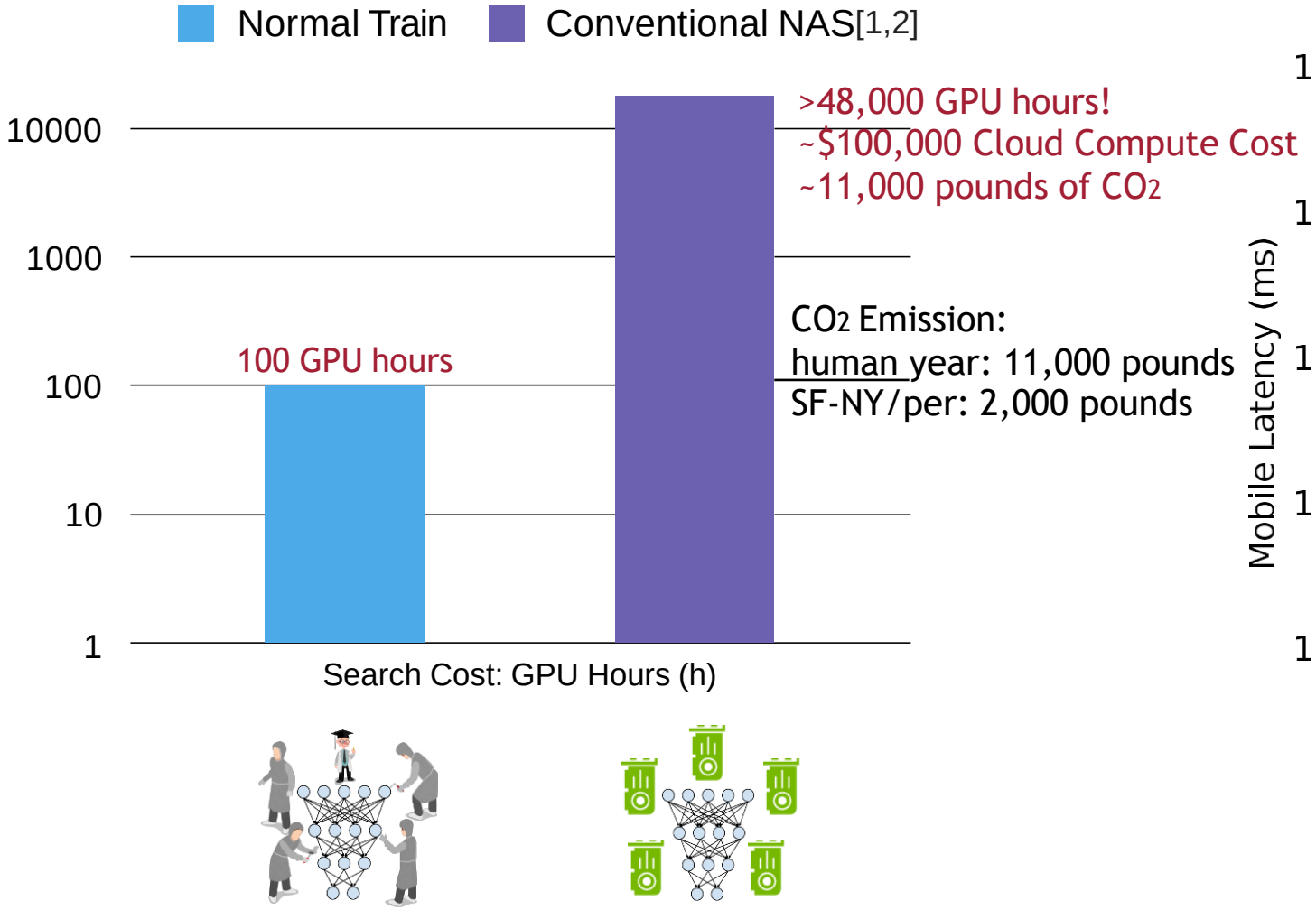


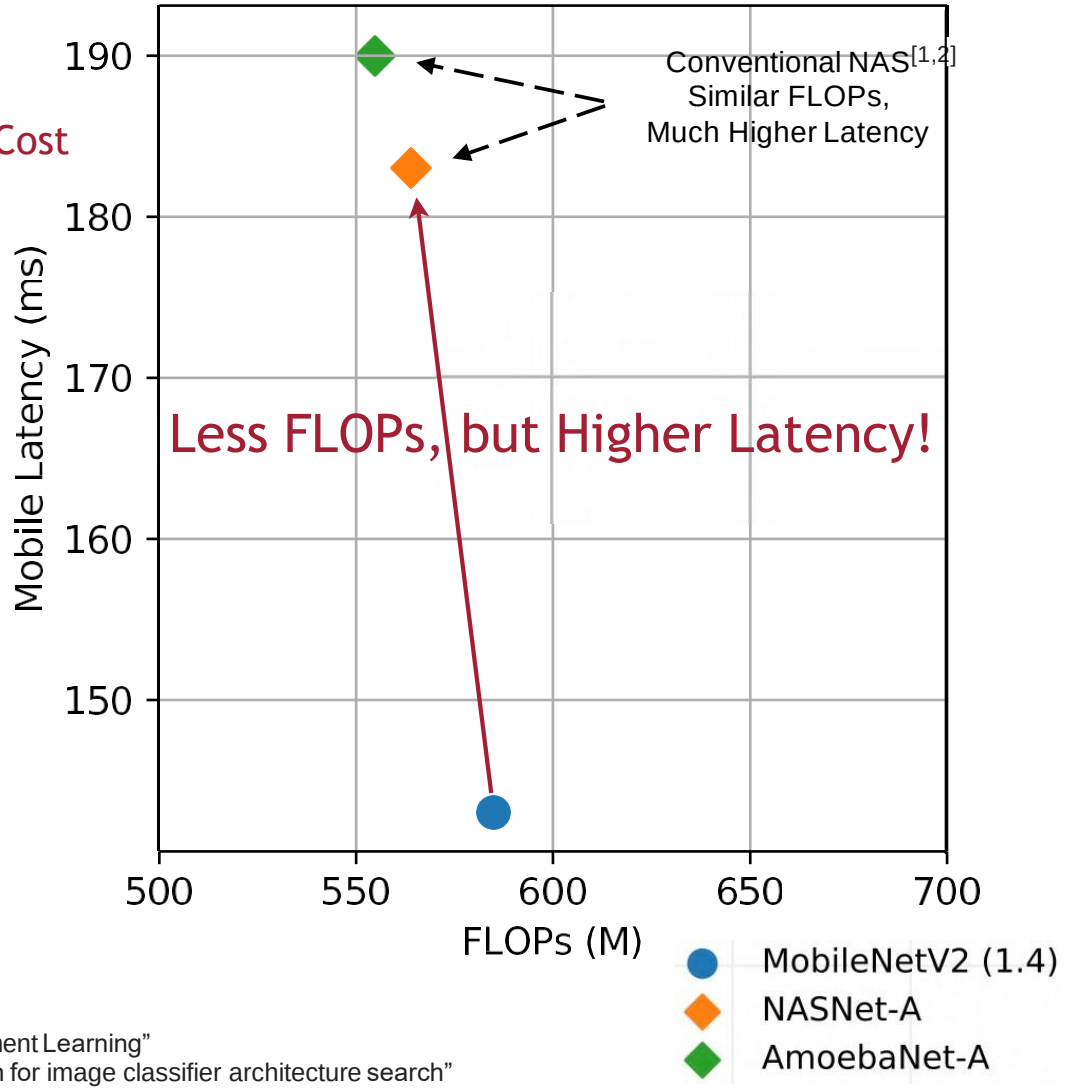
Use Machine Learning

Automatic Architecture Search

Reinforcement Learning
Evolution Strategy
Bayesian Optimization
Monte Carlo Tree Search
…

Conventional NAS: High Search Cost, High Inference Latency





1 B Zoph, QV Le, "Neural Architecture Search with Reinforcement Learning"
2 E Real, A Aggarwal, Y Huang, QV Le, "Regularized evolution for image classifier architecture search"

Model Compression

Neural Architecture Search

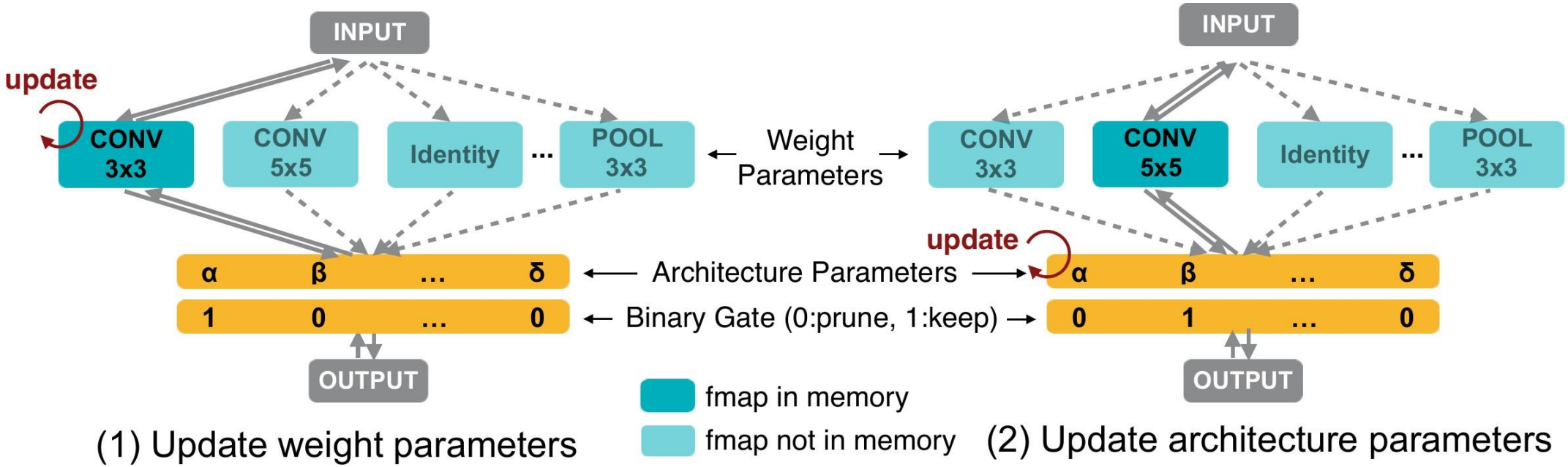
Pruning

Binarization

Save GPU hours

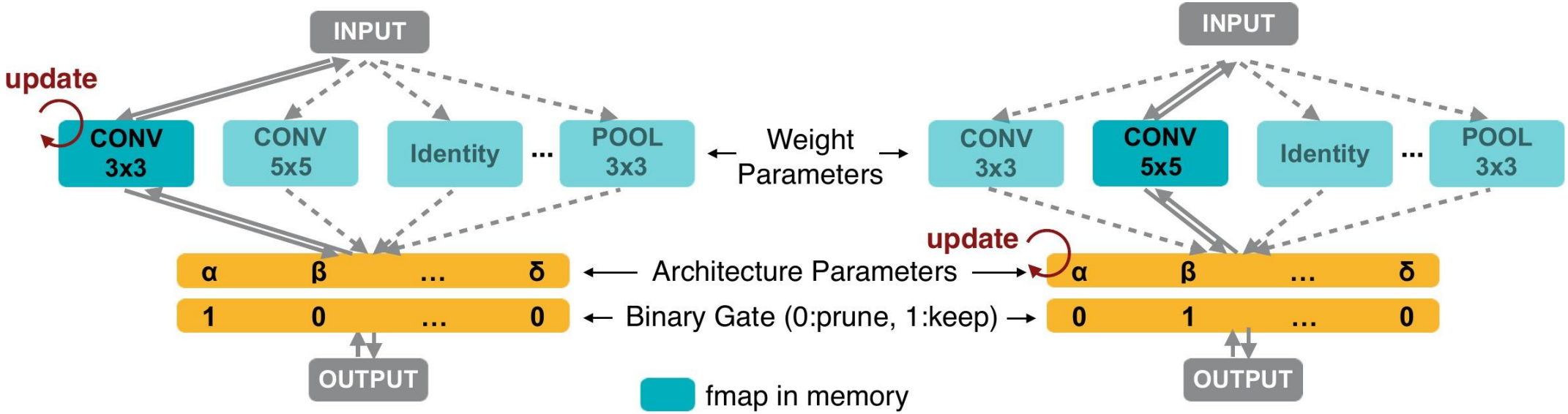
Save GPU Memory

ProxylessNAS: Implementation



Only one path in GPU memory. Scalable to a large design space.

ProxylessNAS: Implementation



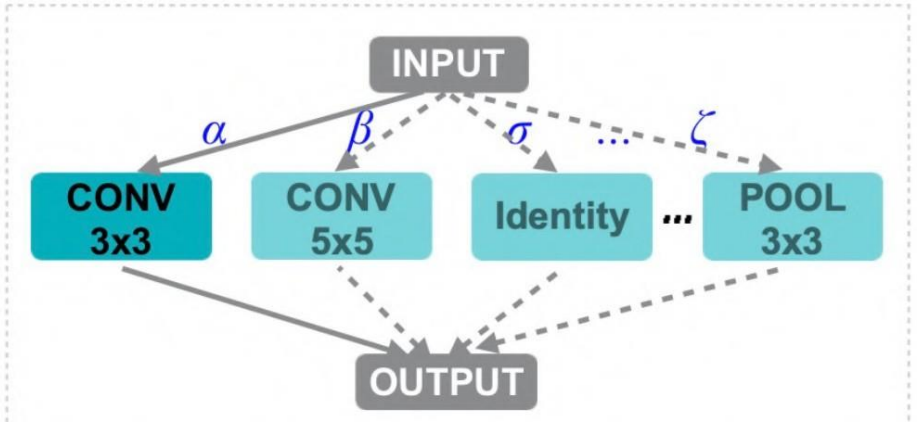
$$g = \text{binarize}(p_1, \cdots, p_N) = \begin{cases} [1, 0, \cdots, 0] & \text{with probability } p_1, \\ \cdots & \\ [0, 0, \cdots, 1] & \text{with probability } p_N. \end{cases} \tag{2}$$

Based on the binary gates g, the output of the mixed operation is given as:

$$m_{\mathcal{O}}^{\text{Binary}}(x) = \sum_{i=1}^{N} g_i o_i(x) = \begin{cases} o_1(x) & \text{with probability } p_1 \\ \cdots & \\ o_N(x) & \text{with probability } p_N. \end{cases} . \tag{3}$$

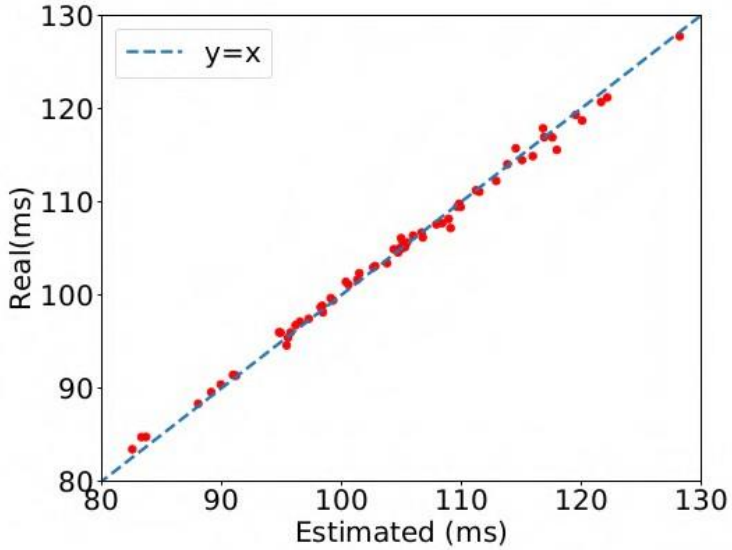
Latency Modeling on Target Hardware

Make Latency Differentiable



$$\mathbb{E}[\text{Latency}] = \alpha \times F(\text{conv\_3x3}) + \beta \times F(\text{conv\_5x5}) + \sigma \times F(\text{identity}) + \ldots\ldots + \zeta \times F(\text{pool\_3x3})$$

$$\mathbb{E}[\text{latency}] = \sum_i \mathbb{E}[\text{latency}_i]$$



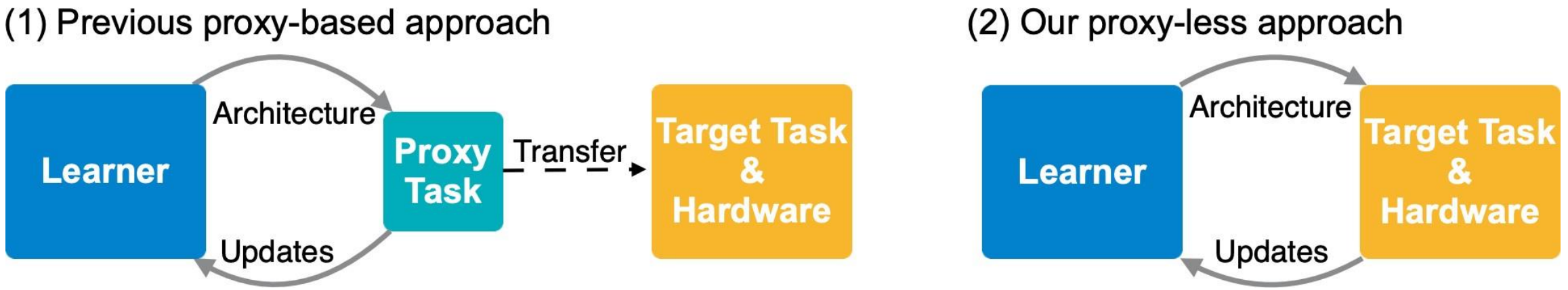
- Mobile farm infrastructure is expensive
- Measuring latency has high variance (thermal throttling)
- Use the latency estimation model as an economical alternative
- Make latency differentiable

Results: ProxylessNAS on ImageNet, Mobile Platform

	Model	Top-1	Latency	Hardware Aware	No Proxy	No Repeat	Search Cost
Manually Designed	MobilenetV1	70.6	113ms	-	-	x	-
	MobilenetV2	72.0	75ms	-	-	x	-
NAS	NASNet-A	74.0	183ms	x	x	x	48000
	AmoebaNet-A	74.4	190ms	x	x	x	75600
	MNasNet	74.0	76ms	yes	x	x	40000
ProxylessNAS	ProxylessNAS	74.6-75.1	78ms	yes	yes	yes	200

200-300x less GPU hours

Efficiently Search an Efficient Model without Proxy



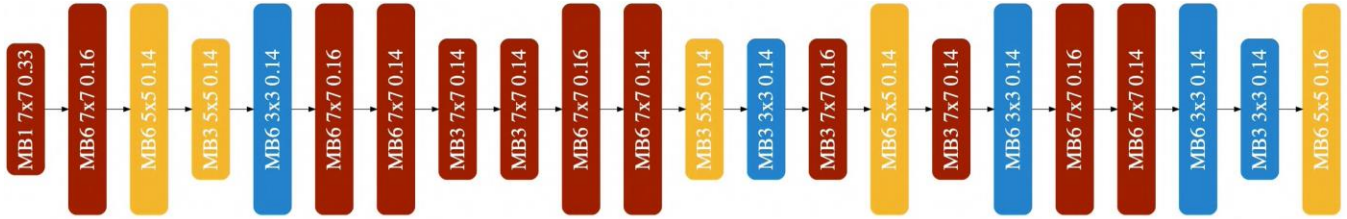
previous work have to utilize **proxy tasks:**

- CIFAR-10 -> ImageNet
- Small architecture space -> repeat the building blocks
- Fewer epochs training -> full training

Demo: the Search History on Different HW

Learning both the weight and architecture

Mobile



(1) The history of finding efficient Mobile model

CPU

Epoch-00

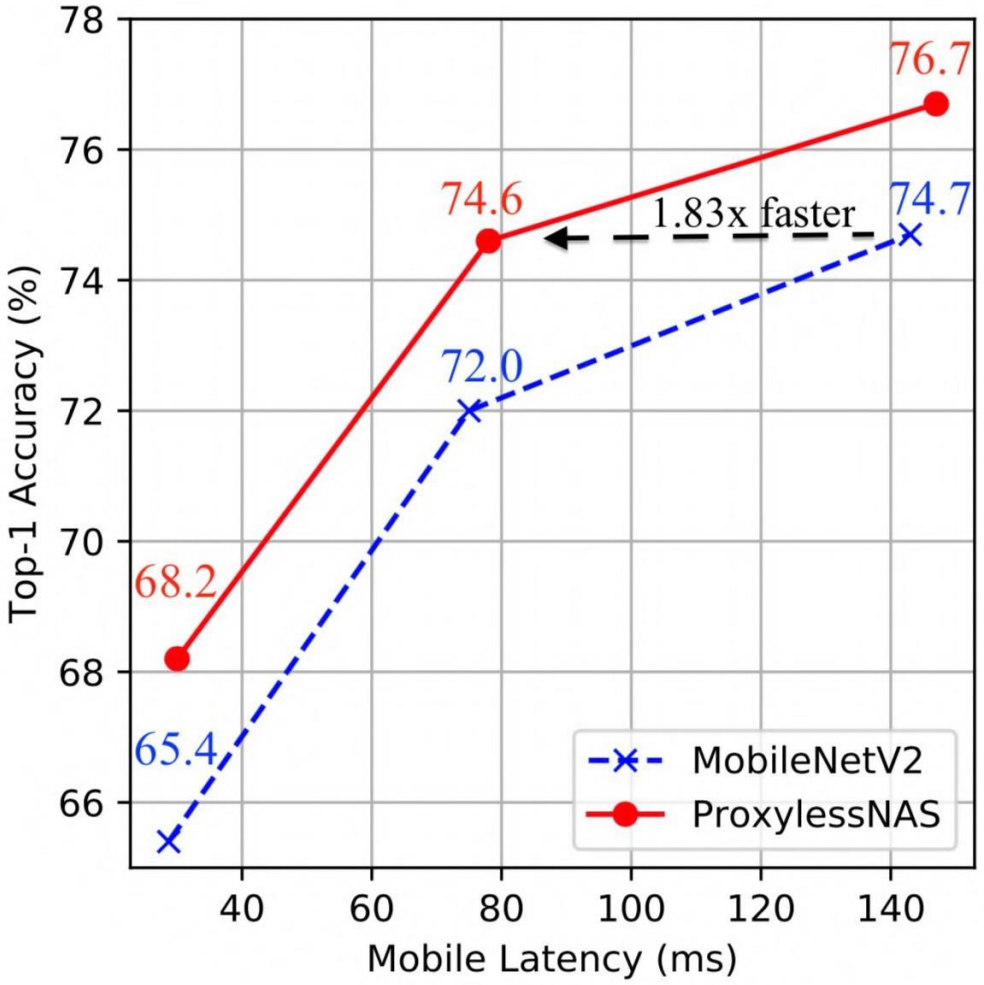
(2) The history of finding efficient CPU model

GPU

(3) The history of finding efficient GPU model

https://drive.google.com/file/d/1nut1owvACc9yz1ZPqcbqoJLS2XrVPp1Q/view

Results: Proxyless-NAS on ImageNet, Mobile Platform



- With >74.5% top-1 accuracy, ProxylessNAS is **1.8x faster** than MobileNet-v2

Results: Proxyless-NAS on ImageNet, GPU Platform

Model	Top-1	Top-5	GPU latency
MobileNetV2 (Sandler et al., 2018)	72.0	91.0	6.1ms
ShuffleNetV2 (1.5) (Ma et al., 2018)	72.6	-	7.3ms
ResNet-34 (He et al., 2016)	73.3	91.4	8.0ms
NASNet-A (Zoph et al., 2018)	74.0	91.3	38.3ms
DARTS (Liu et al., 2018c)	73.1	91.0	-
MnasNet (Tan et al., 2018)	74.0	91.8	6.1ms
Proxyless (GPU)	**75.1**	**92.5**	**5.1ms**

When targeting GPU platform, the accuracy is further improved to 75.1%.
3.1% higher than MobilenetV2.

ProxylessNAS is Quantization Friendly

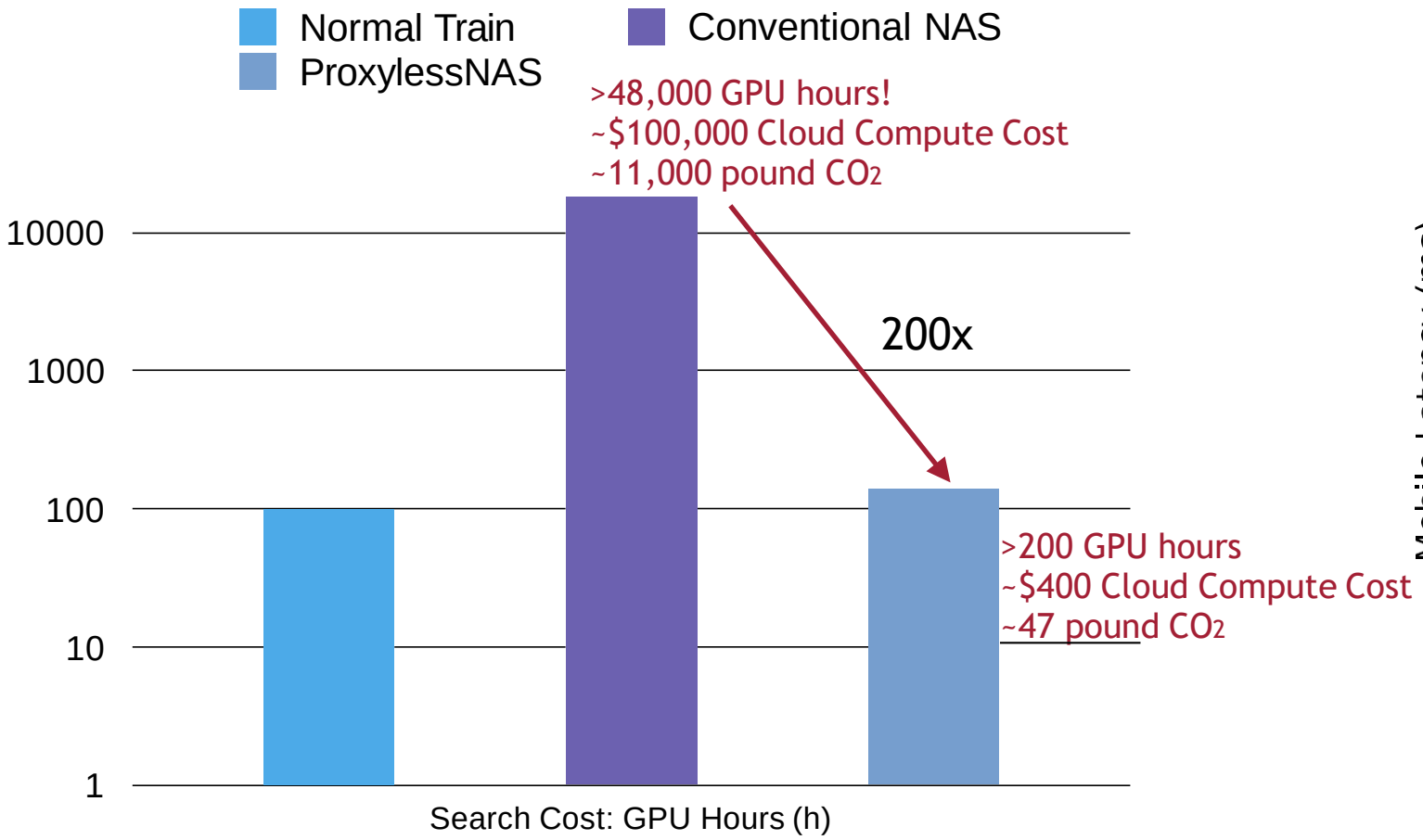
Model	Latency	Top-1 Acc
quant_mobilenetv2_128_100	20ms	62.9%
quant_mobilenetv2_192_50	20ms	61.6%
quant_mobilenetv2_224_35	21ms	58.1%
quant_mobilenetv2_160_75	26ms	64.6%
quant_mobilenetv2_224_50	28ms	63.7%
quant_mobilenetv2_160_100	31ms	67.4%
float_mnasnet_224_50	**36ms**	**67.9%**
quant_mobilenetv2_192_75	36ms	67.4%
quant_mobilenetv2_192_100	44ms	69.5%

ProxylessNAS:

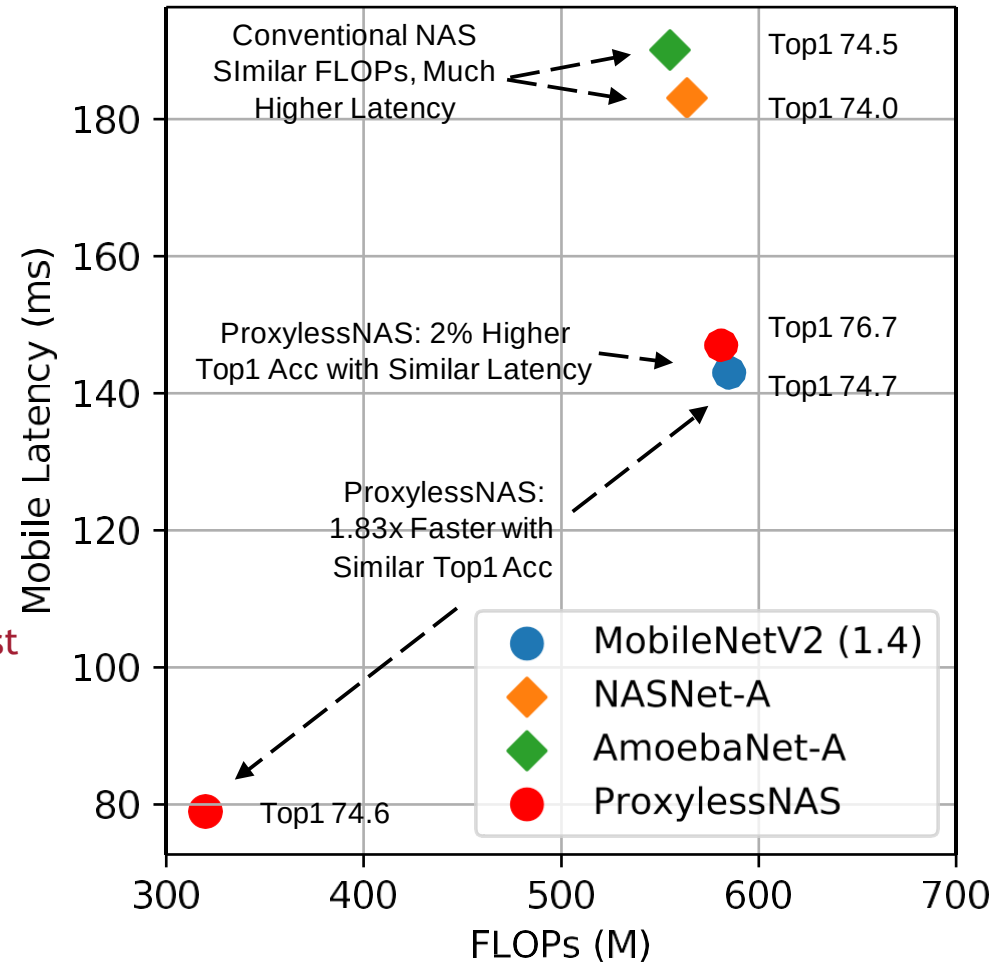
35ms 69.2%

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/java/ovic#sample-benchmarks

Efficiently Search a Model

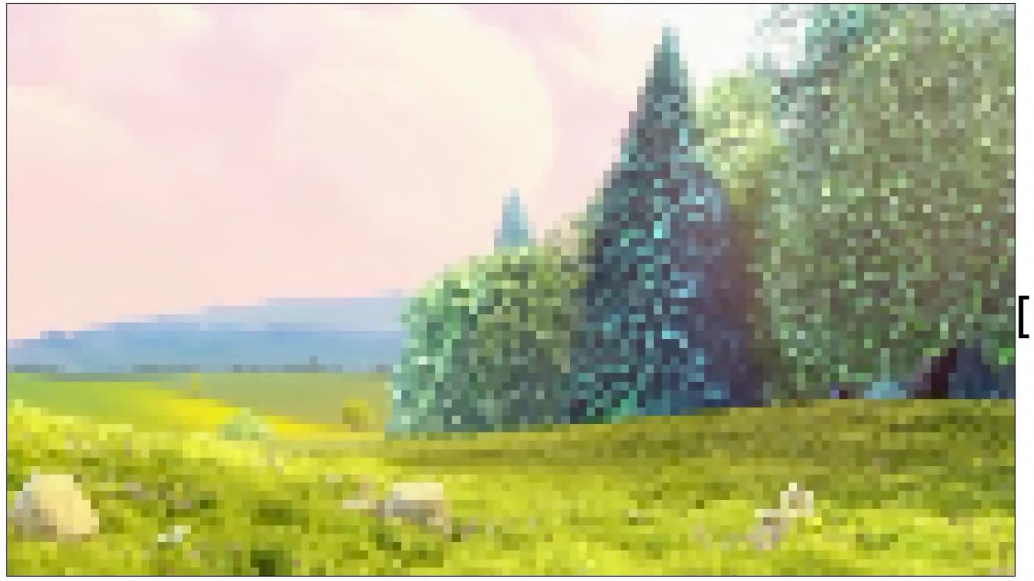


Search an Efficient Model



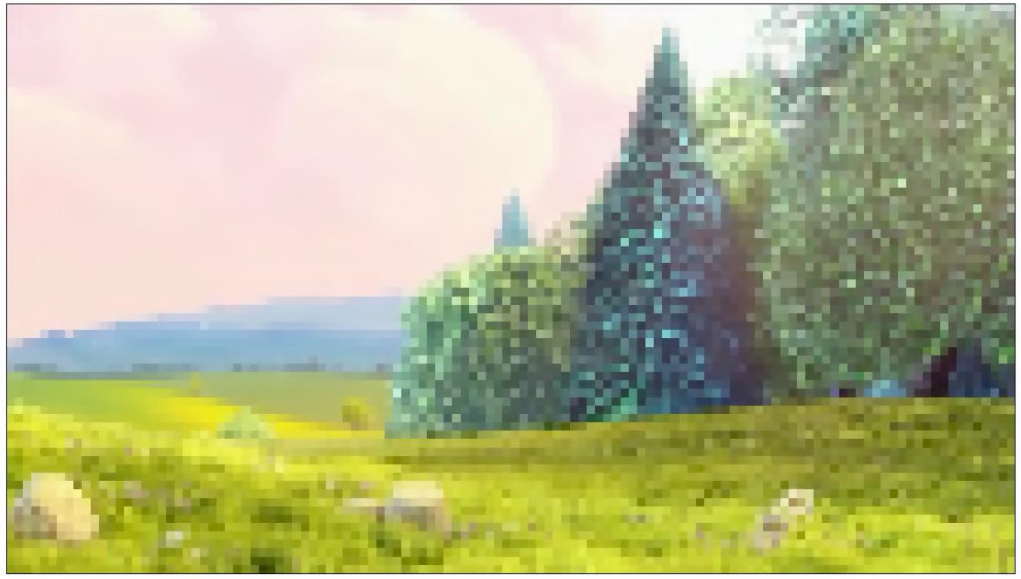
ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware, ICLR'19
https://arxiv.org/pdf/1812.00332.pdf

Accelerate Super Resolution with ProxylessNAS



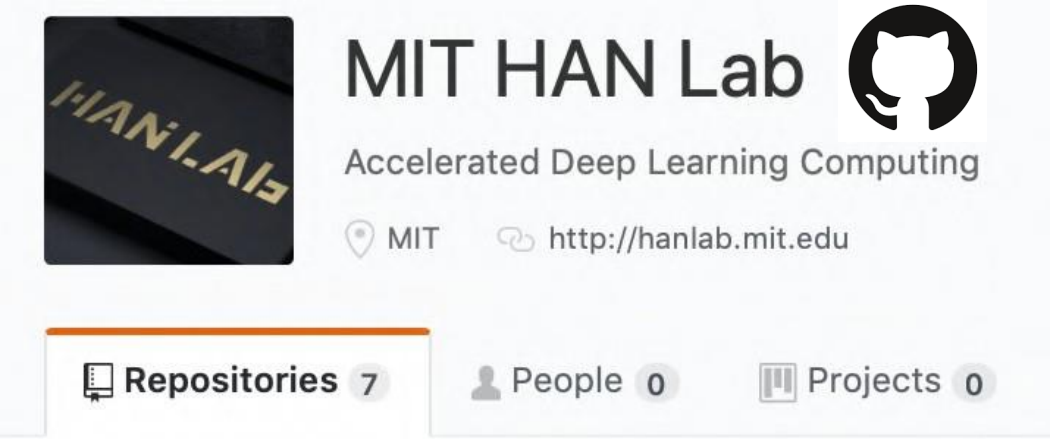
CARN
[ECCV'18]
==>

CARN 16FPS PNSR:21.09



Proxyless
NAS
==>

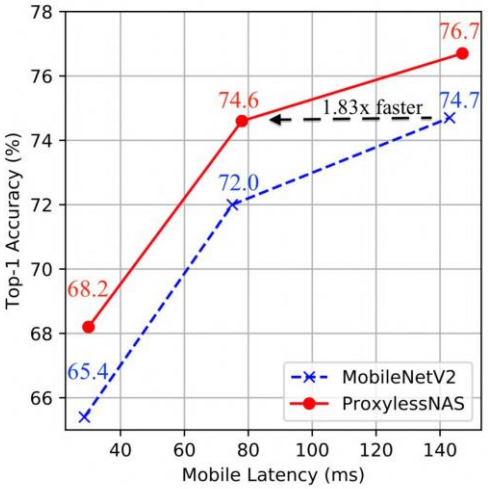
Ours 41FPS PNSR:21.26

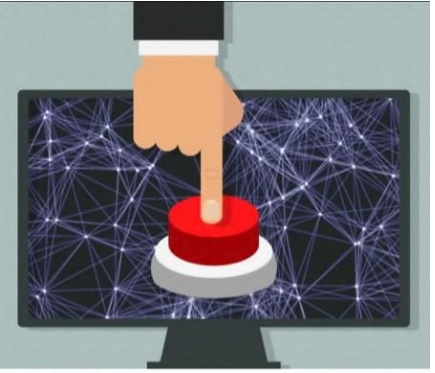


ProxylessNAS is Available on Github

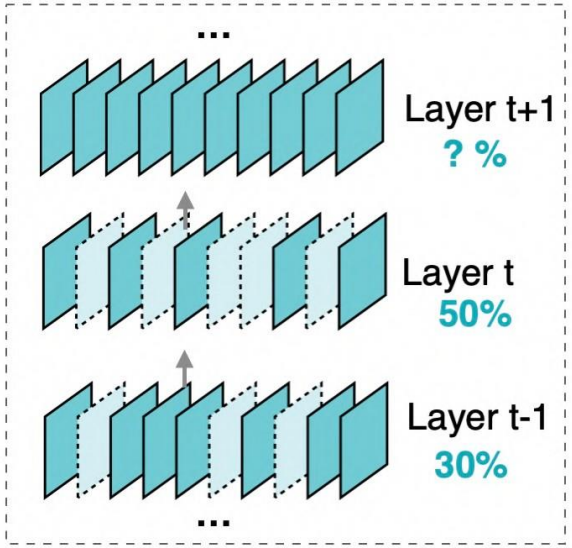
```
from proxyless_nas import *
net = proxyless_cpu(pretrained=True)
```

github.com/MIT-HAN-LAB/ProxylessNAS



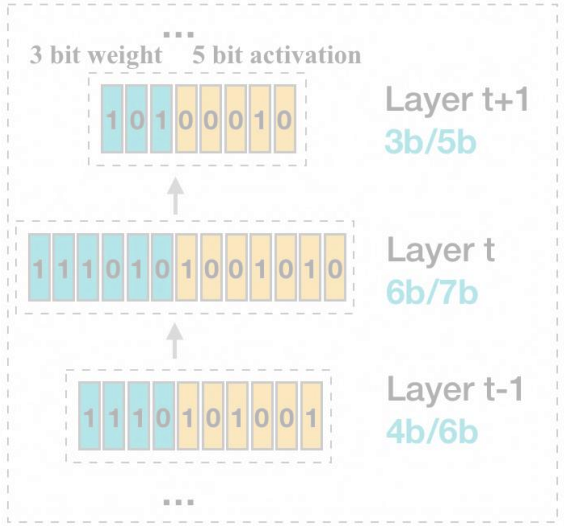


Design Automation for Efficient Deep Learning Computing



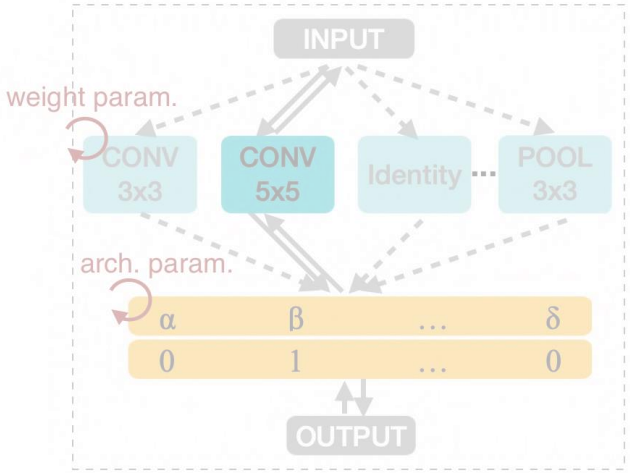
AMC: AutoML for Model Compression

[ECCV 2018]



HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



Proxyless Neural Architecture Search

[ICLR 2019]

HAN LAB

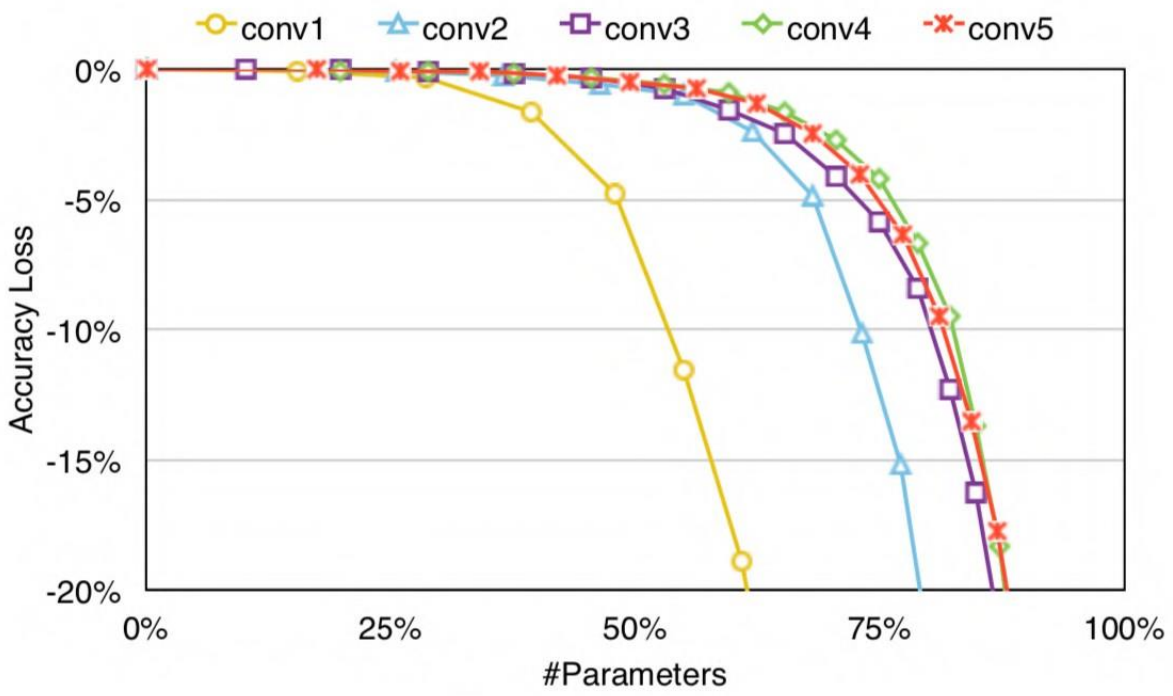
AMC: Automatic Model Compression and Acceleration for Mobile Devices

Yihui He[2] *, Ji Lin[1] *, Zhijian Liu[1], Hanrui Wang[1], Li-Jia Li[3], Song Han[1]

[1]Massachusetts Institute of Technology,
[2]Xi'an Jiaotong University,
[3]Google

ECCV'18

Sensitivity Analysis (Manual Design)



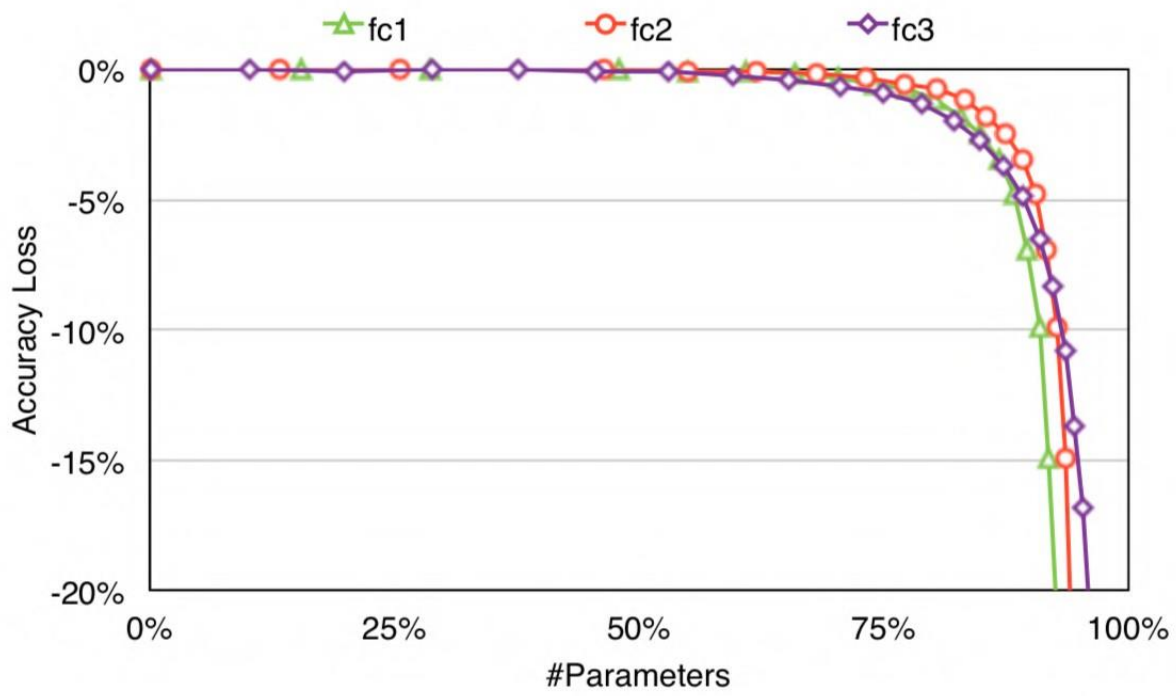
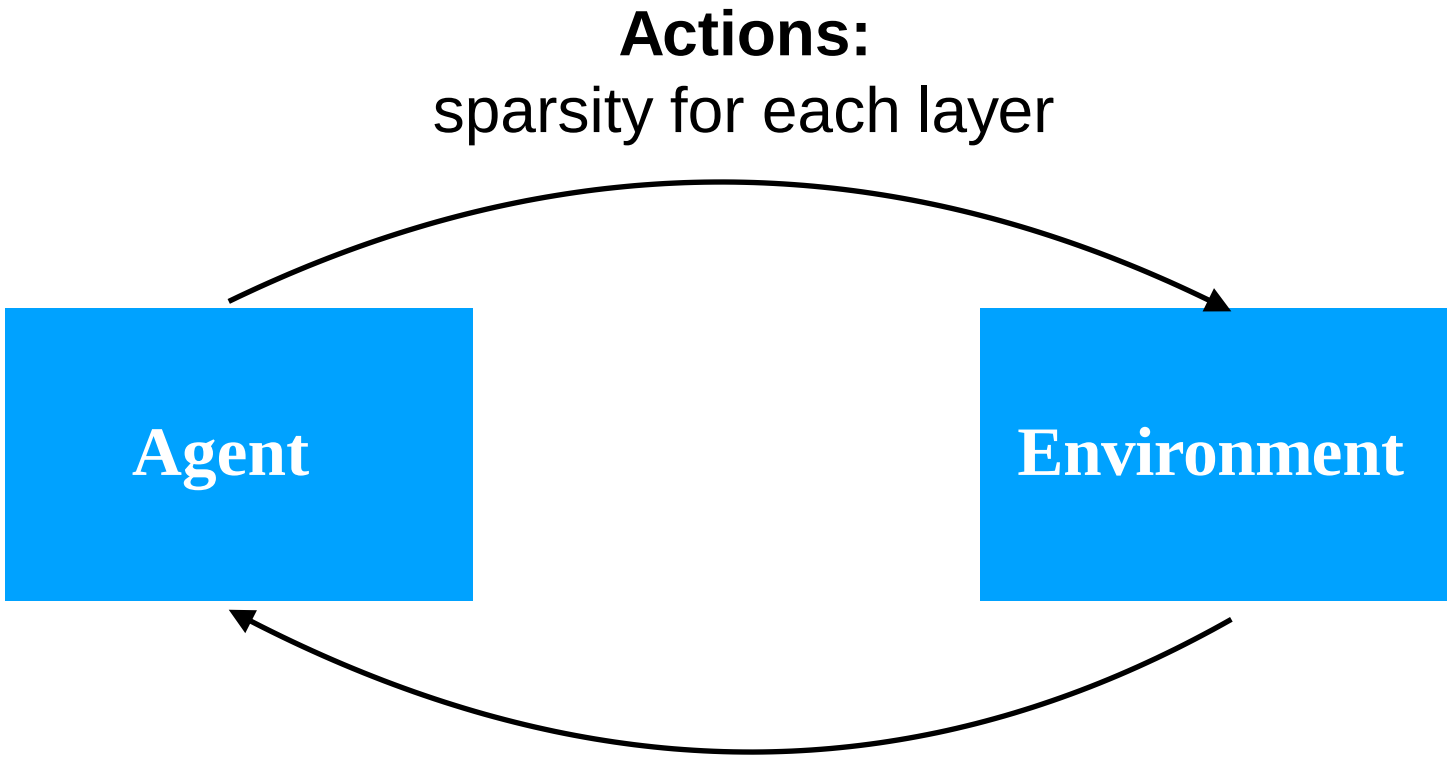


Figure 6: Pruning sensitivity for CONV layer (left) and FC layer (right) of AlexNet.

AMC: Automatic Model Compression



Actor Critic

DDPG

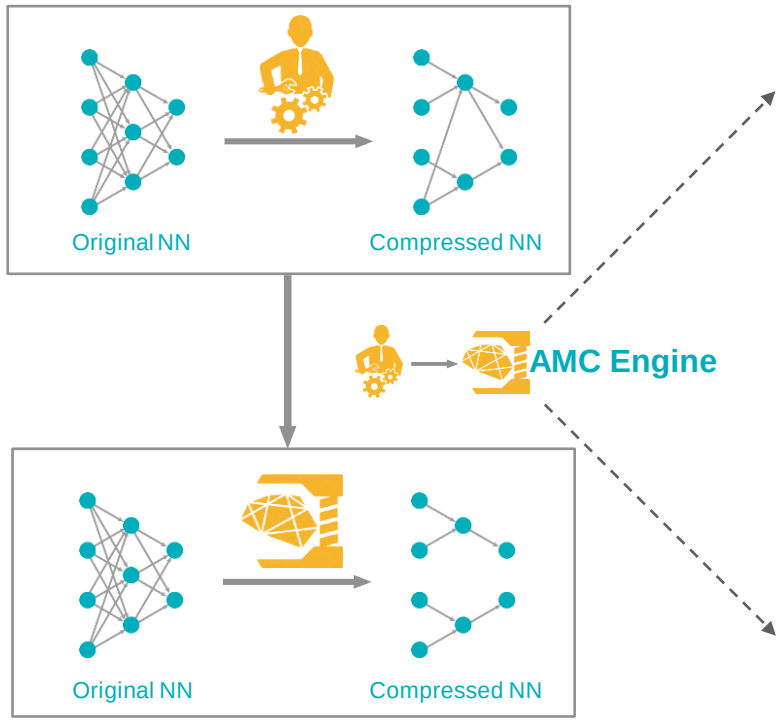
Channel

Pruning

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

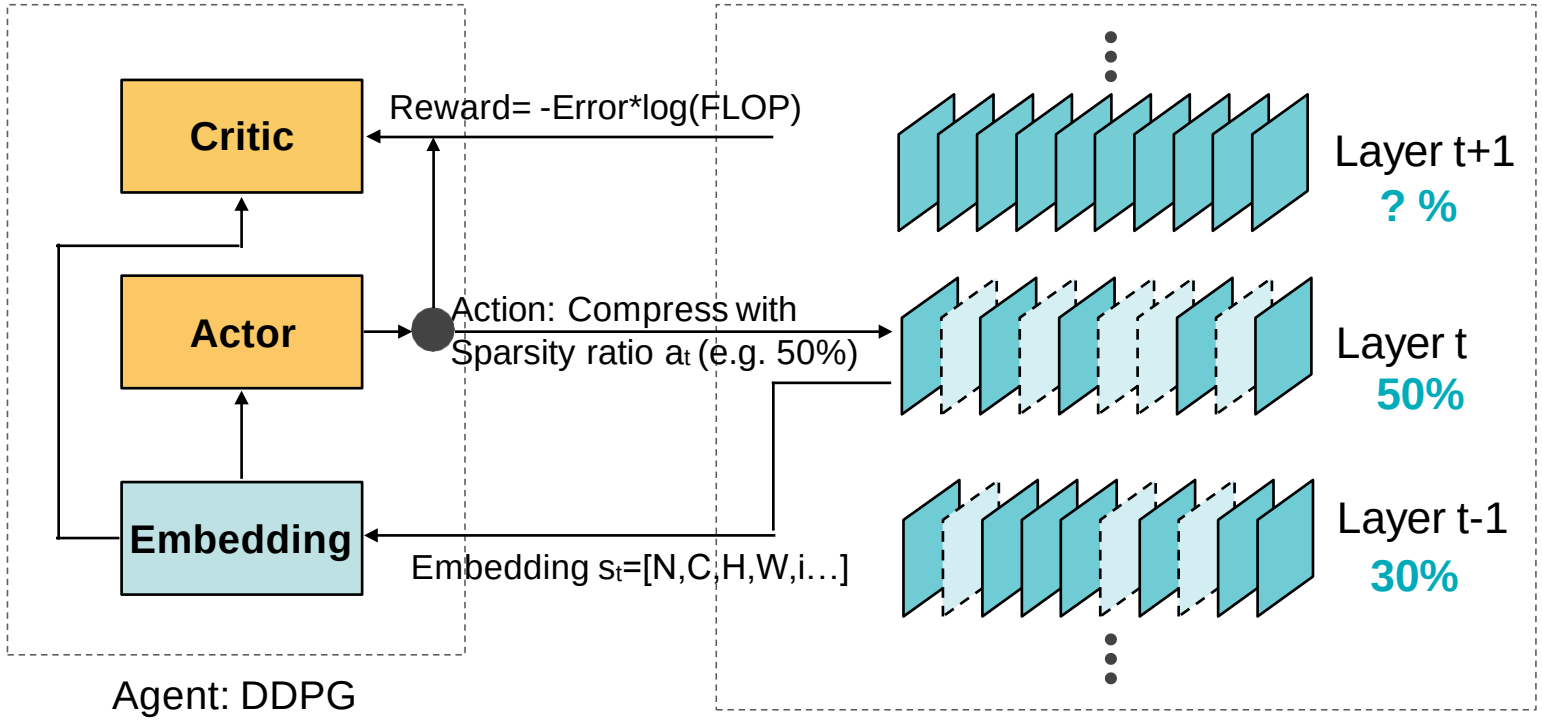
AMC: Automatic Model Compression

Model Compression by Human:
Labor Consuming, Sub-optimal



Model Compression by AI:
Automated, Higher Compression Rate, Faster

To get accuracy, retraining takes a long time;
We solve it by LMS, not retraining, which quickly gives the reward



Agent: DDPG

Environment: Channel Pruning

Previous actuation impacts the future states.
If you pruned a lot in layer i, then layer i+1 has less pressure to be pruned.

AMC: Automatic Model Compression

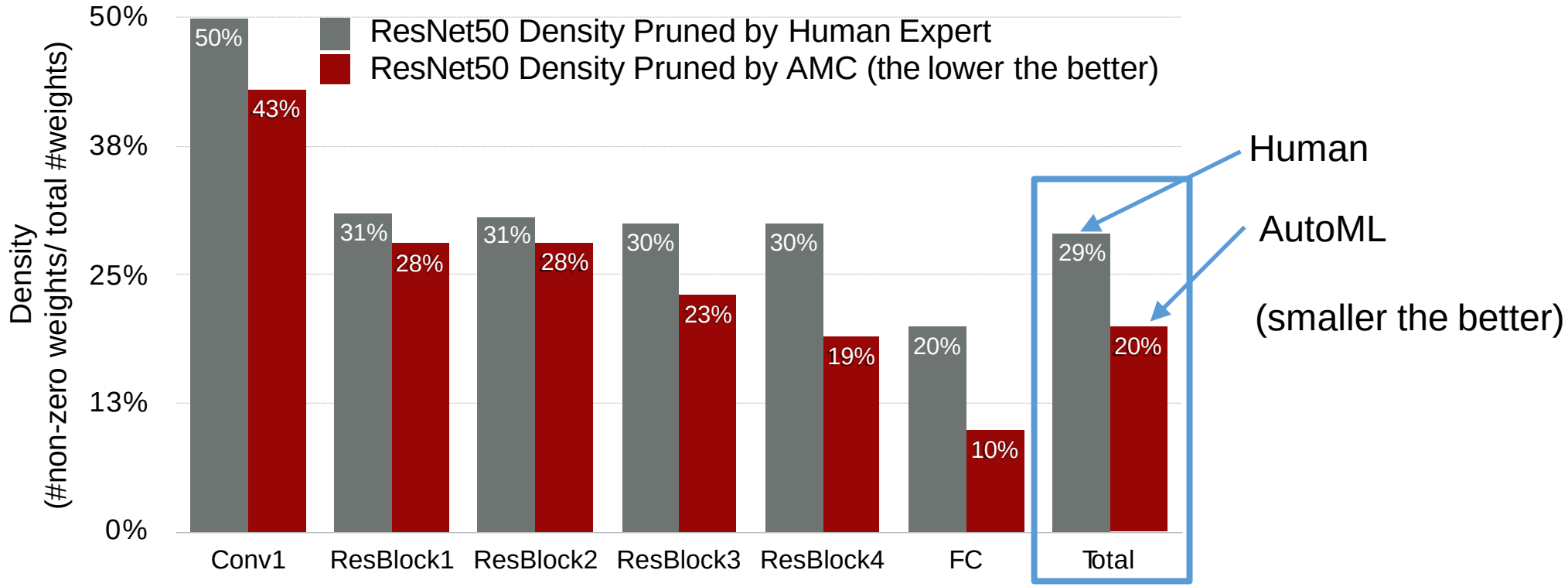


Figure 15: Our reinforcement learning agent (AMC) can prune the model to a lower density than achieved by human experts without loss of accuracy. (Human expert: 3.4× compression on ResNet50. AMC : 5× compression on ResNet50.)

AMC: Automatic Model Compression

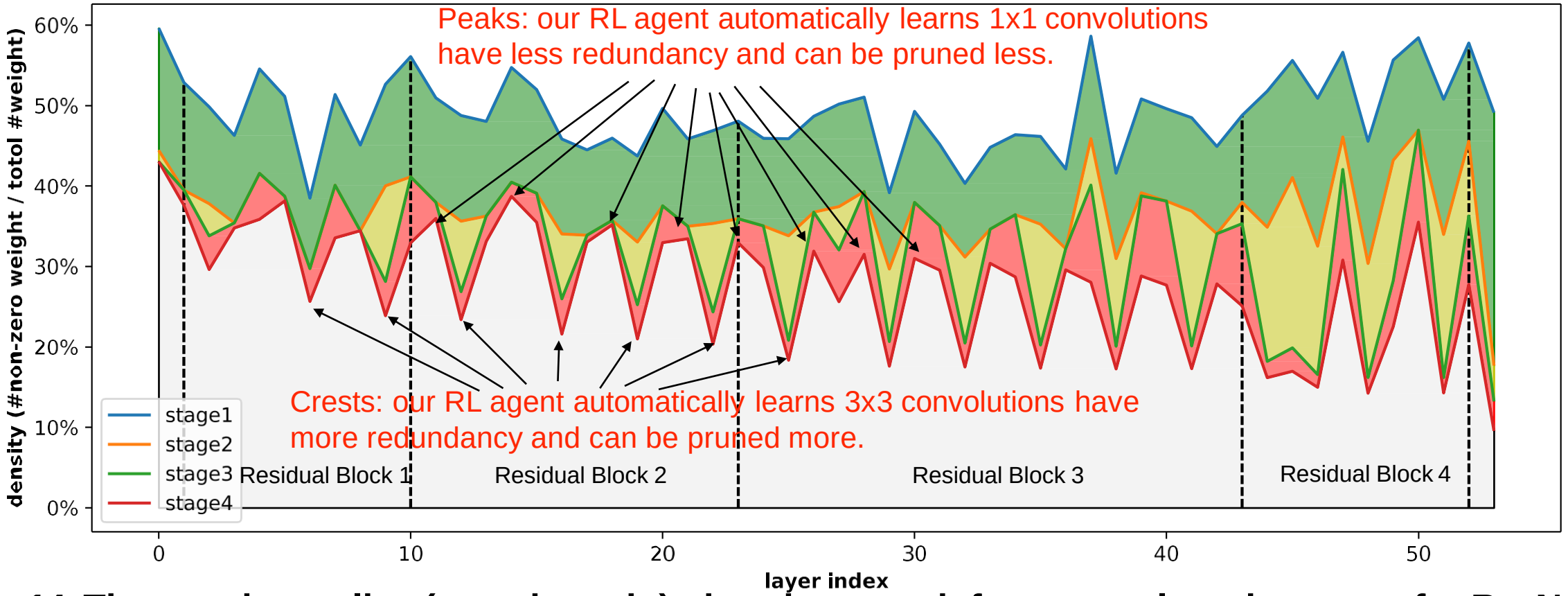
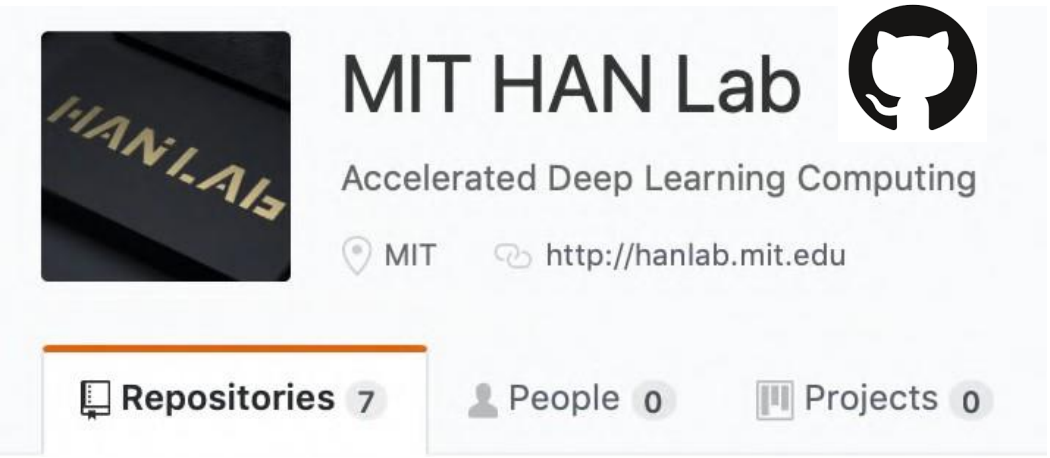


Figure 14: The pruning policy (sparsity ratio) given by our reinforcement learning agent for ResNet-50.

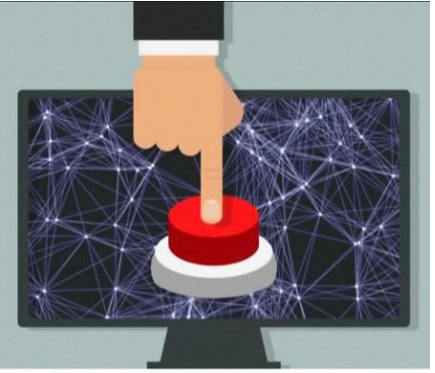
AMC : Accelerating MobileNet

Model	MAC	Top-1	Top-5	Latency	Speed	Memory
1.0 MobileNet	569M	70.6%	89.5%	**119.0ms**	8.4 fps	20.1MB
AMC (50% MAC)	285M	70.5%	89.3%	**64.4ms**	15.5 fps (1.8x)	14.3MB
AMC (50% Time)	272M	70.2%	89.2%	**59.7ms**	16.8 fps (2.0x)	13.2MB
0.75 MobileNet	325M	68.4%	88.2%	**69.5ms**	14.4 fps (1.7x)	14.8MB

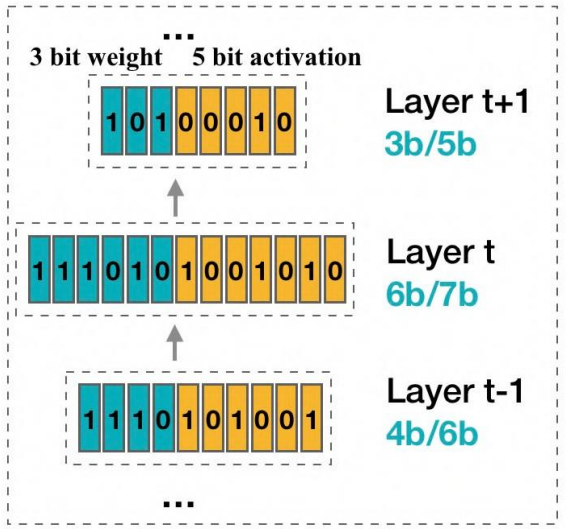


AMC is Available on Github

github.com/mit-han-lab/AMC

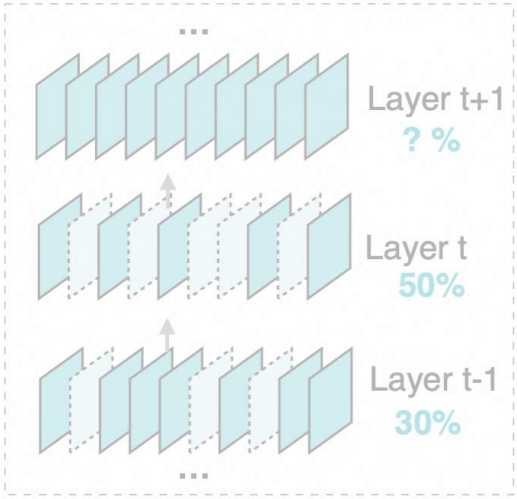


Design Automation for Efficient Deep Learning Computing



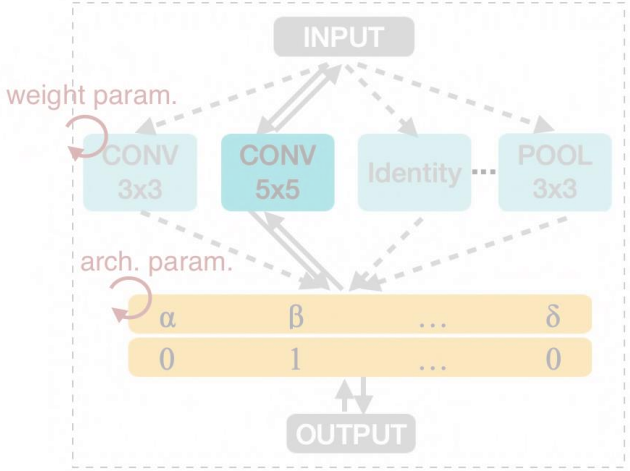
HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



AMC: AutoML for Model Compression

[ECCV 2018]



Proxyless Neural Architecture Search

[ICLR 2019]

HAN LAB

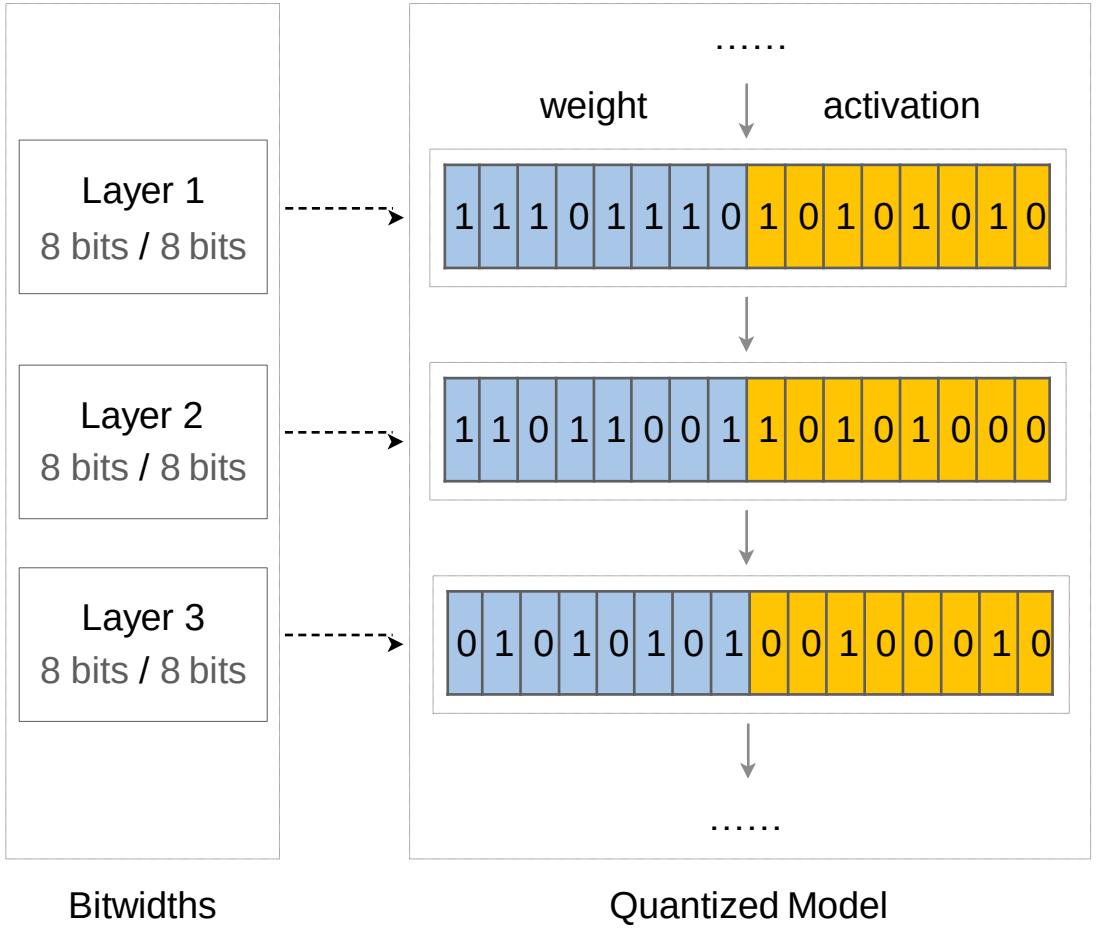
Hardware-aware Automated Quantization with Mixed Precision

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, Song Han

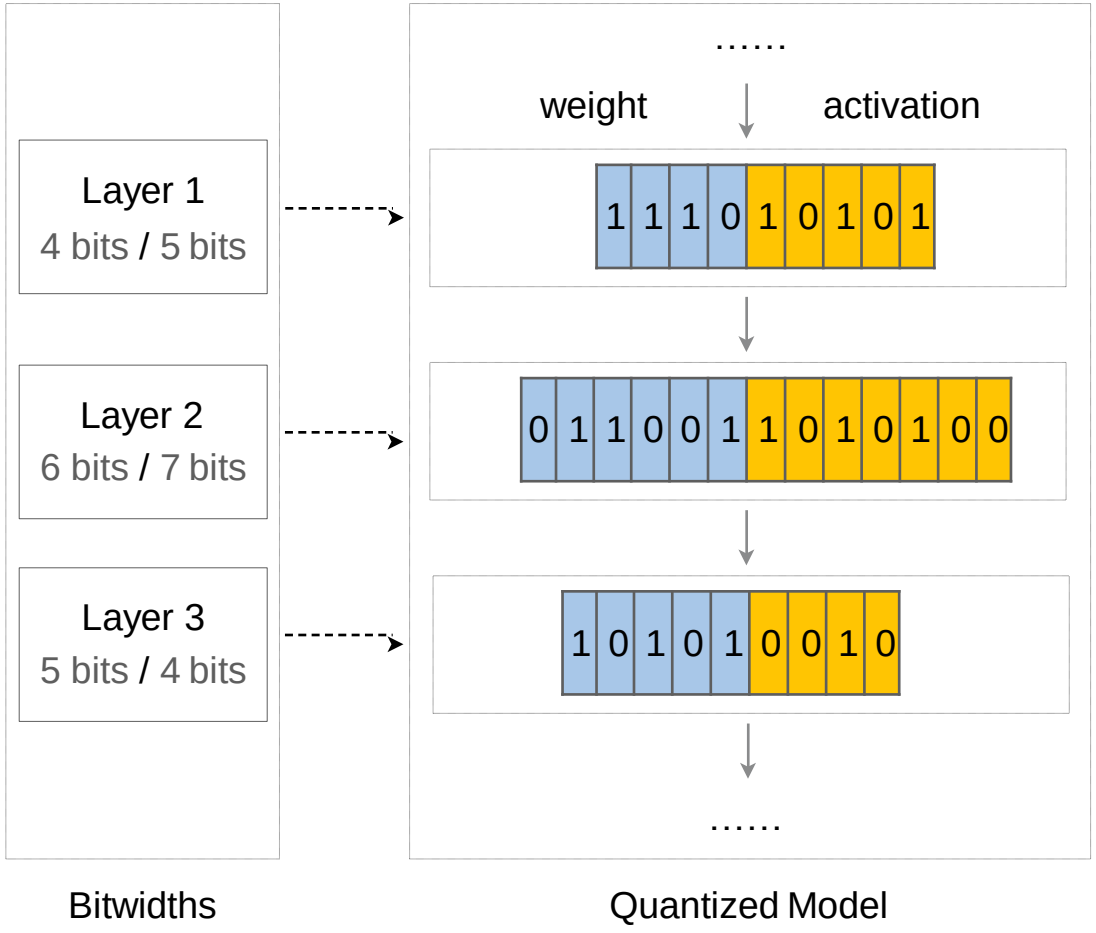
Massachusetts Institute of Technology

CVPR'19, oral

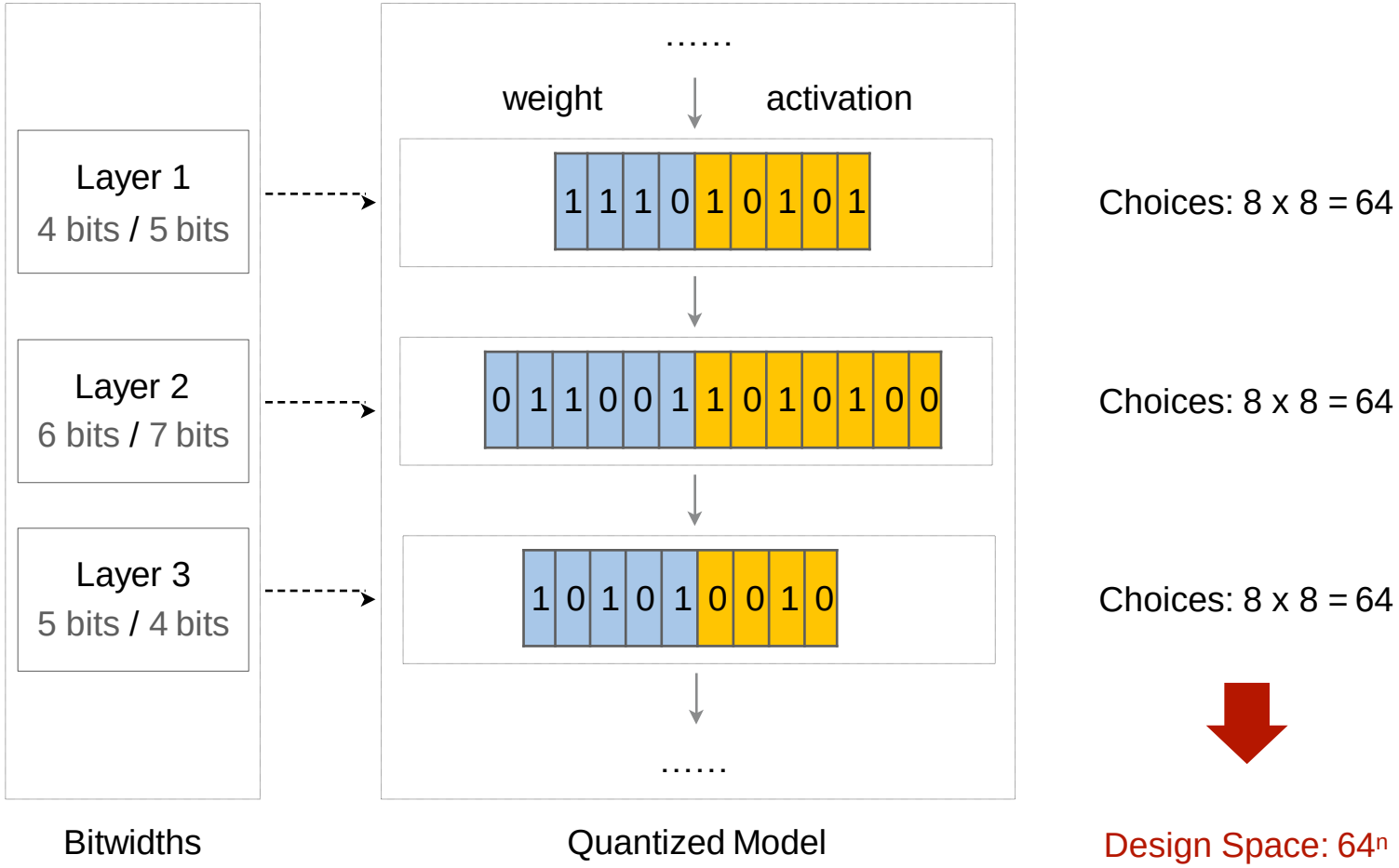
Fixed-Precision Quantization



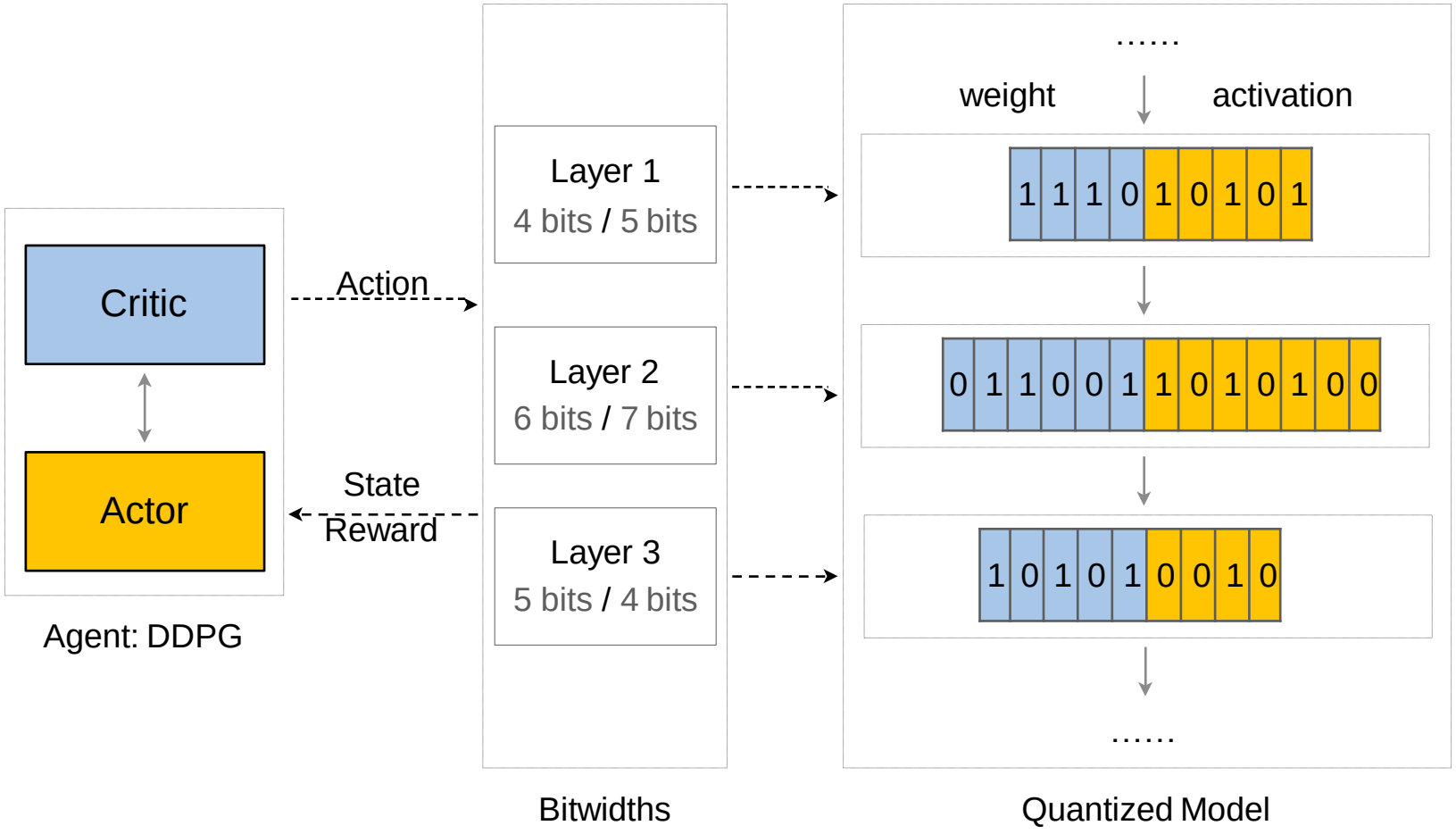
Contribution I: Mixed-Precision Quantization



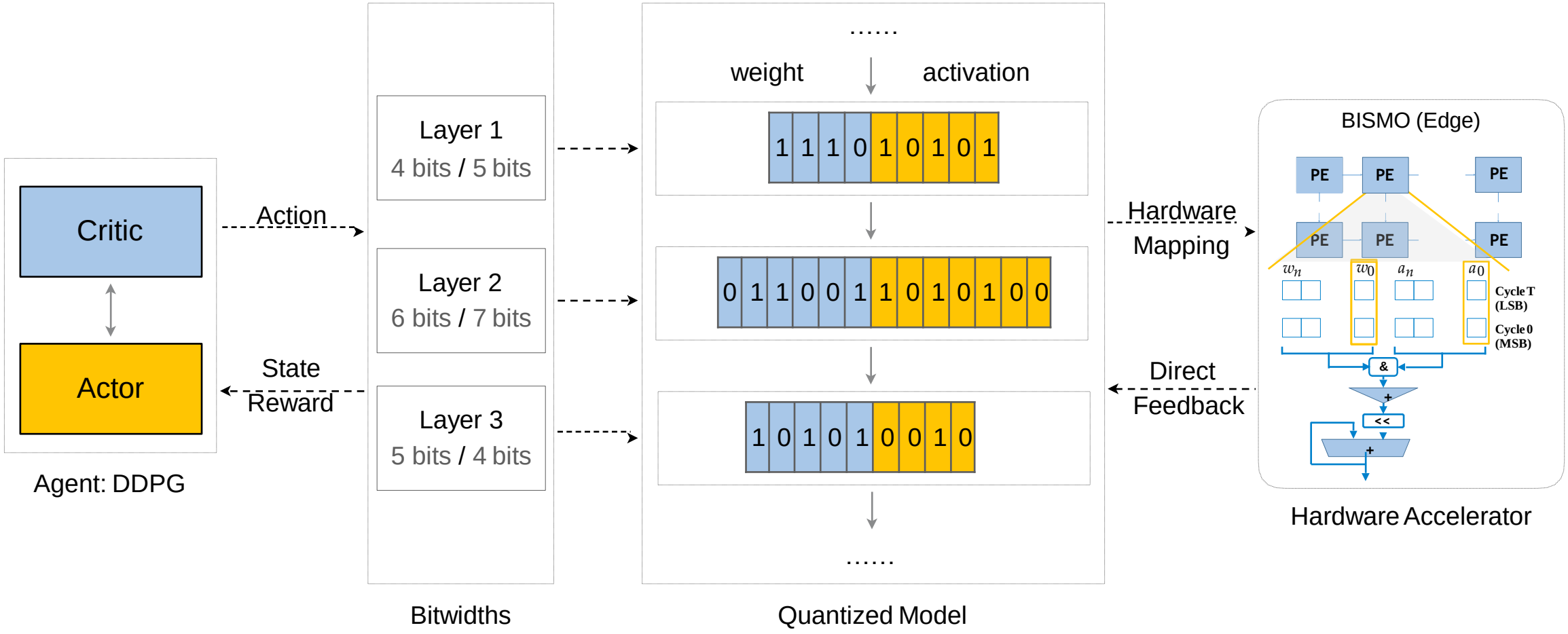
Contribution II: Design Automation



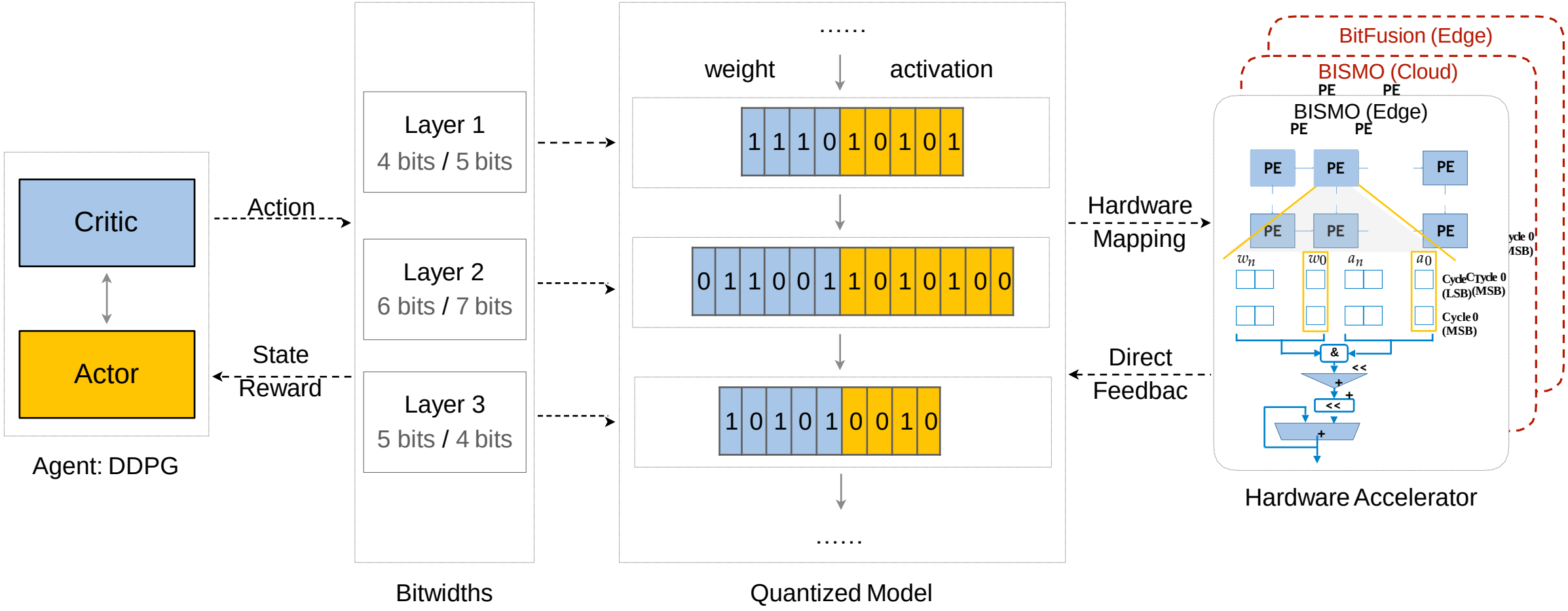
Contribution II: Design Automation



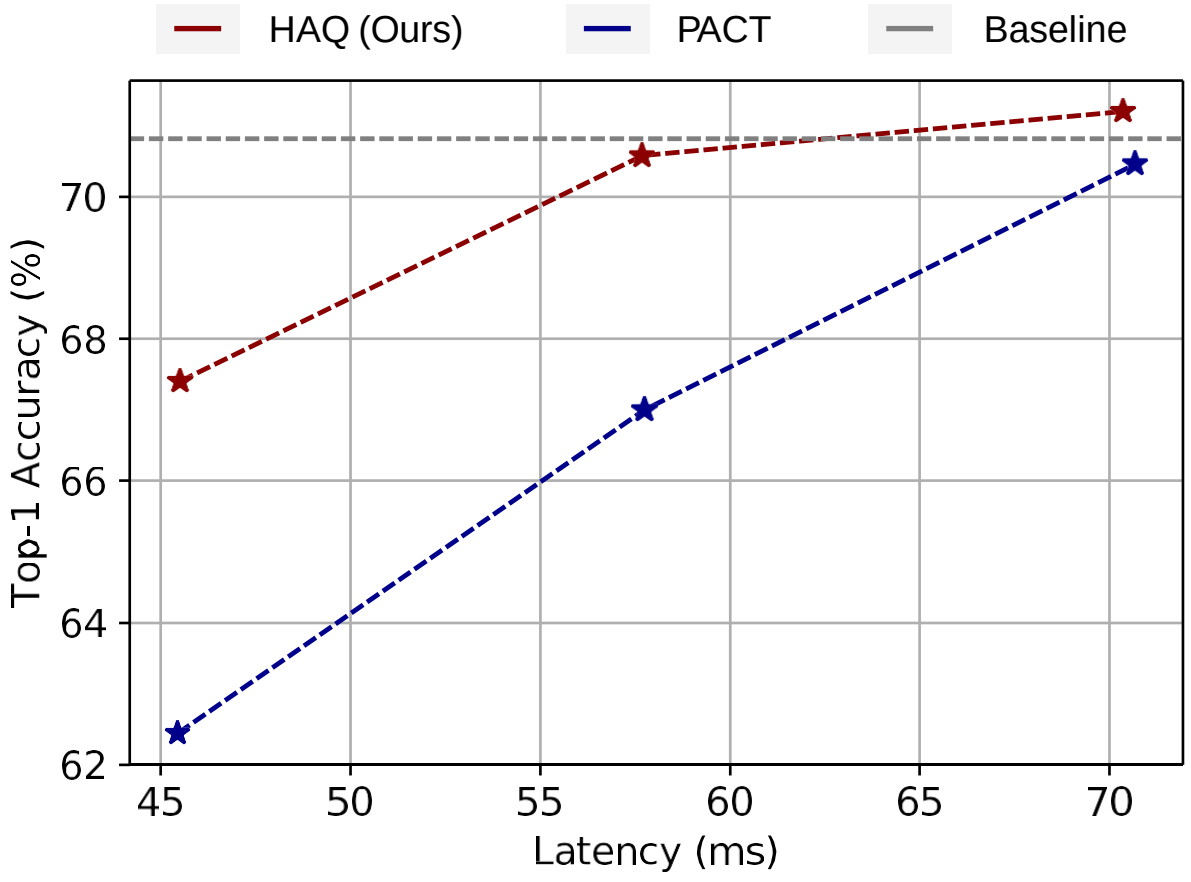
Contribution III: Hardware-Aware Specialization



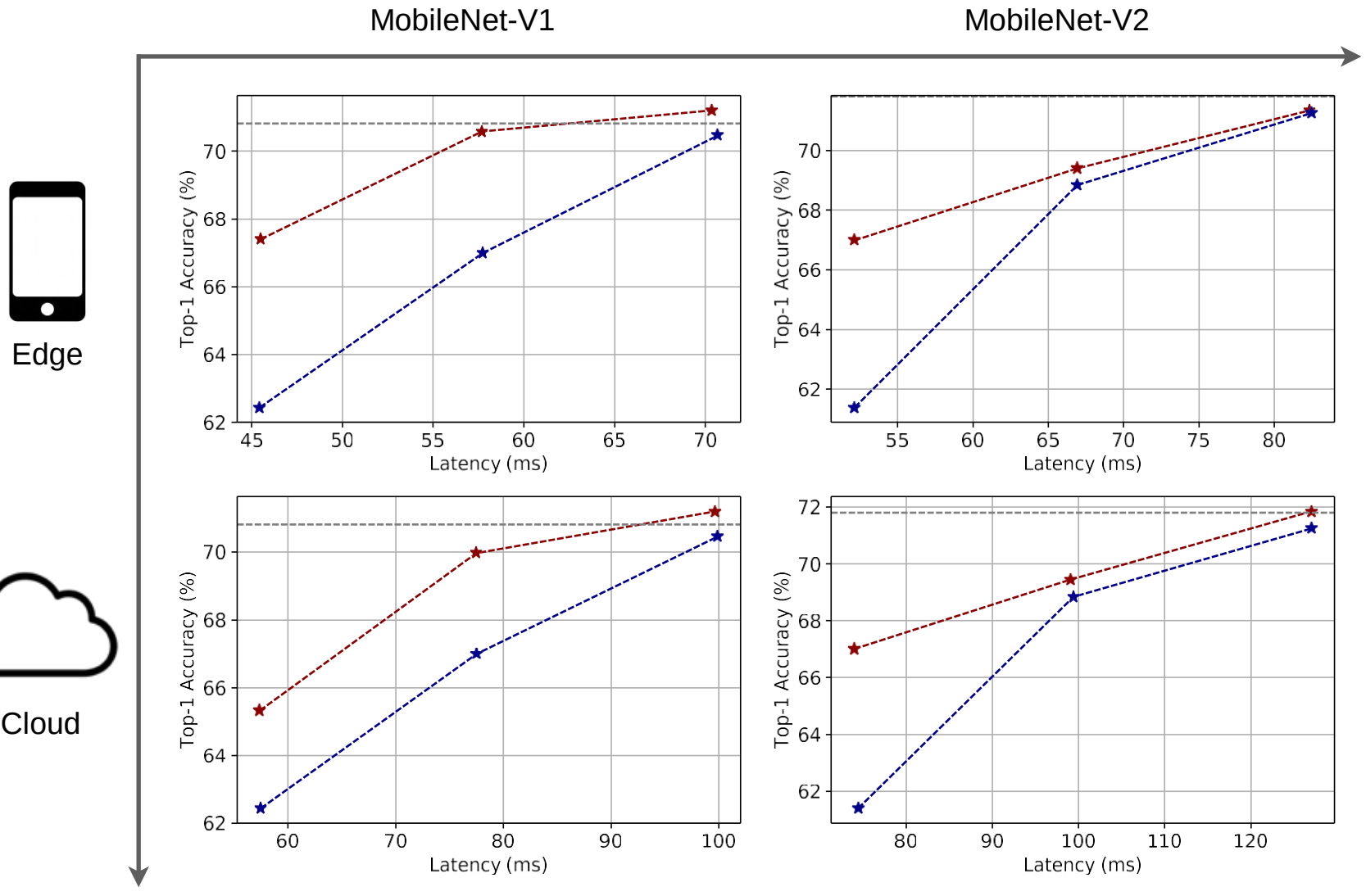
Contribution III: Hardware-Aware Specialization



Results: HAQ Outperforms Uniform Quantization



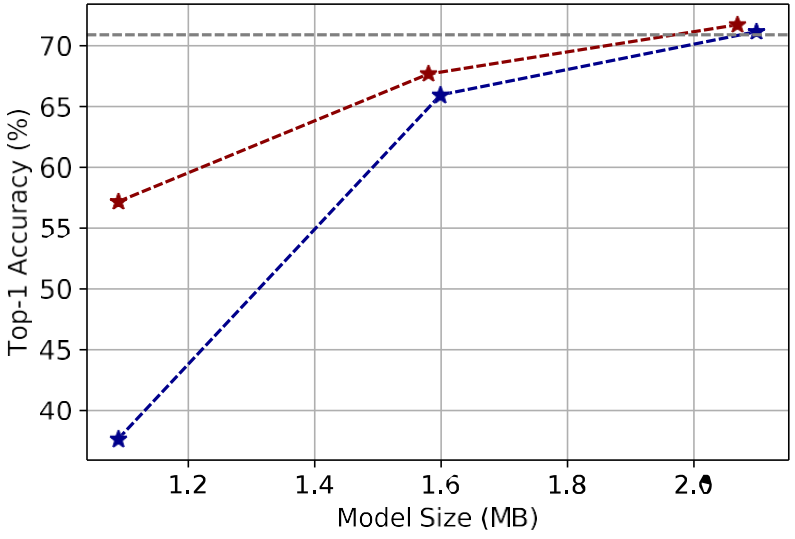
Results: Model and Hardware Specialization



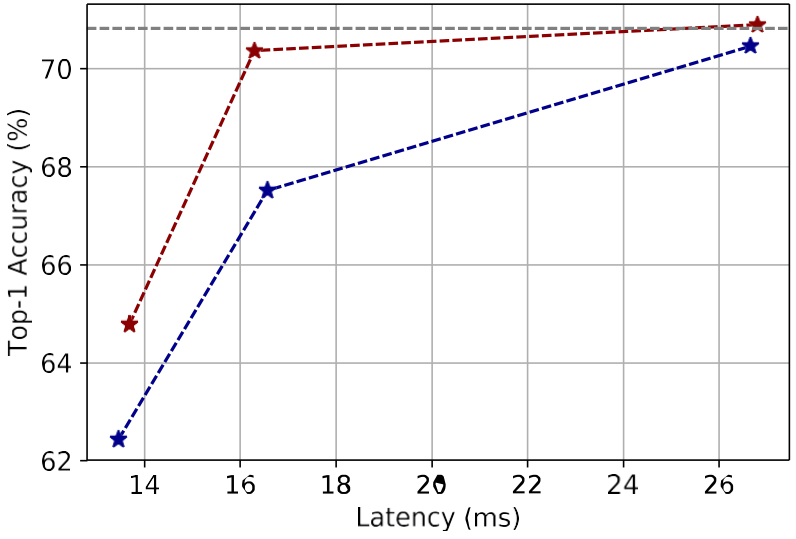
- Baseline (Full Precision)
- **HAQ (Ours)**
- **PACT (Uniform Quantization)**

Results: HAQ Supports Multiple Objectives

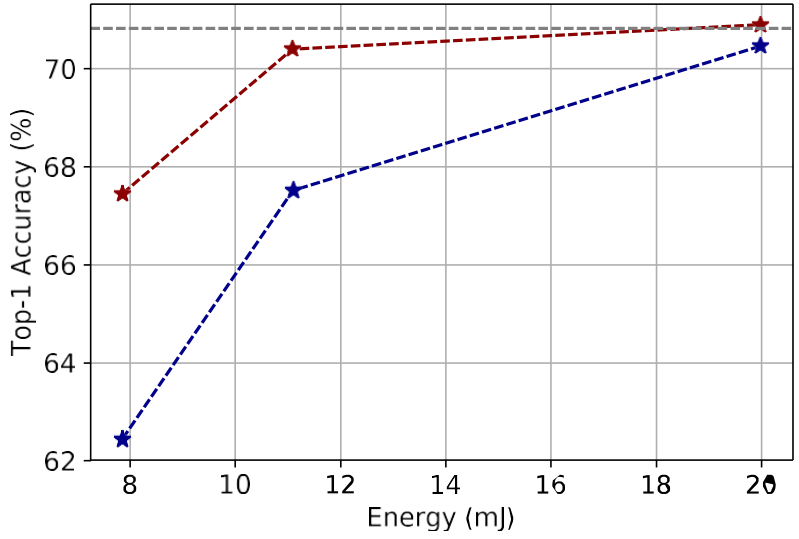
Model Size Constrained



Latency Constrained



Energy Constrained



MobileNet

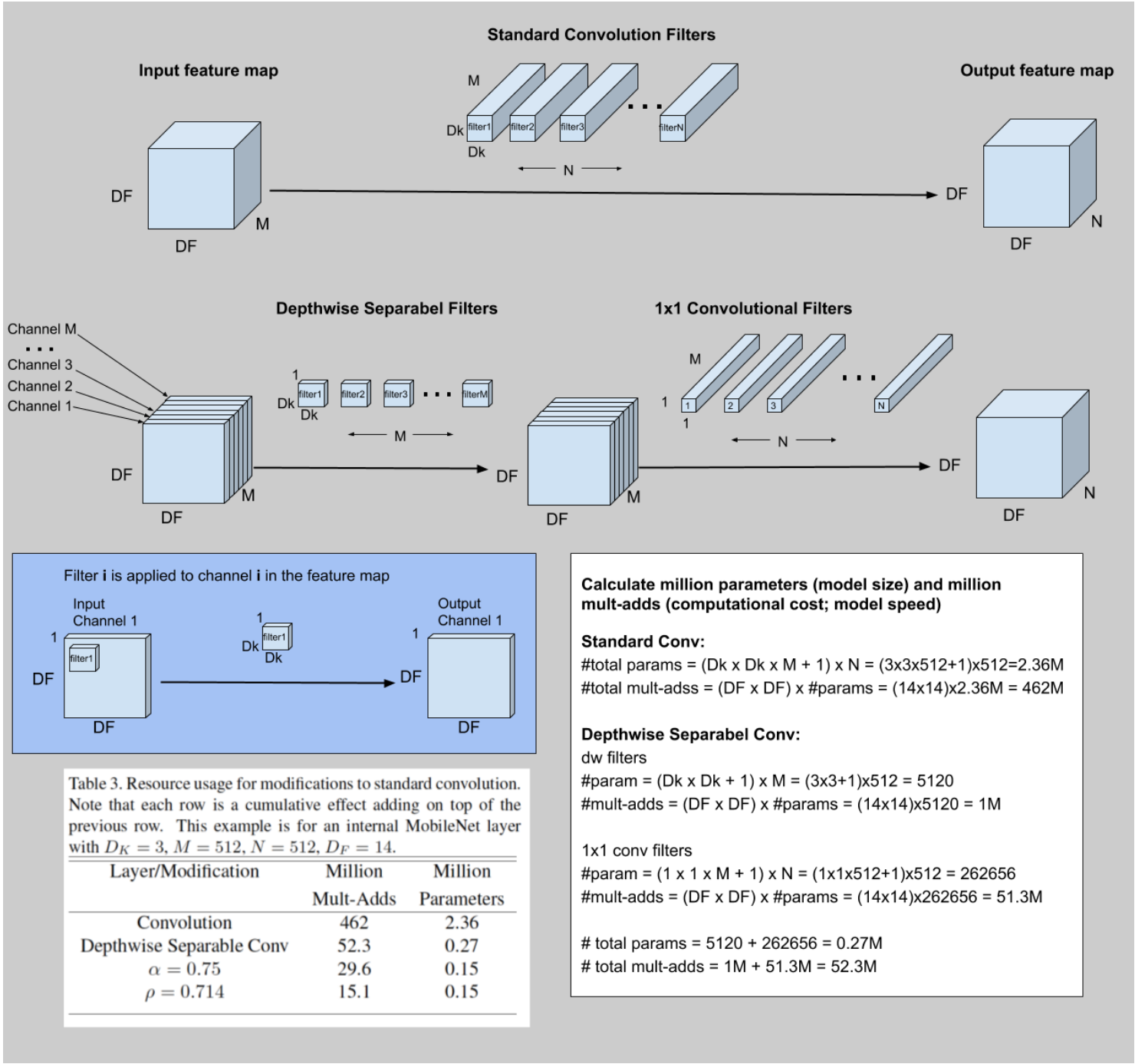


Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the previous row. This example is for an internal MobileNet layer with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

Calculate million parameters (model size) and million mult-adds (computational cost; model speed)

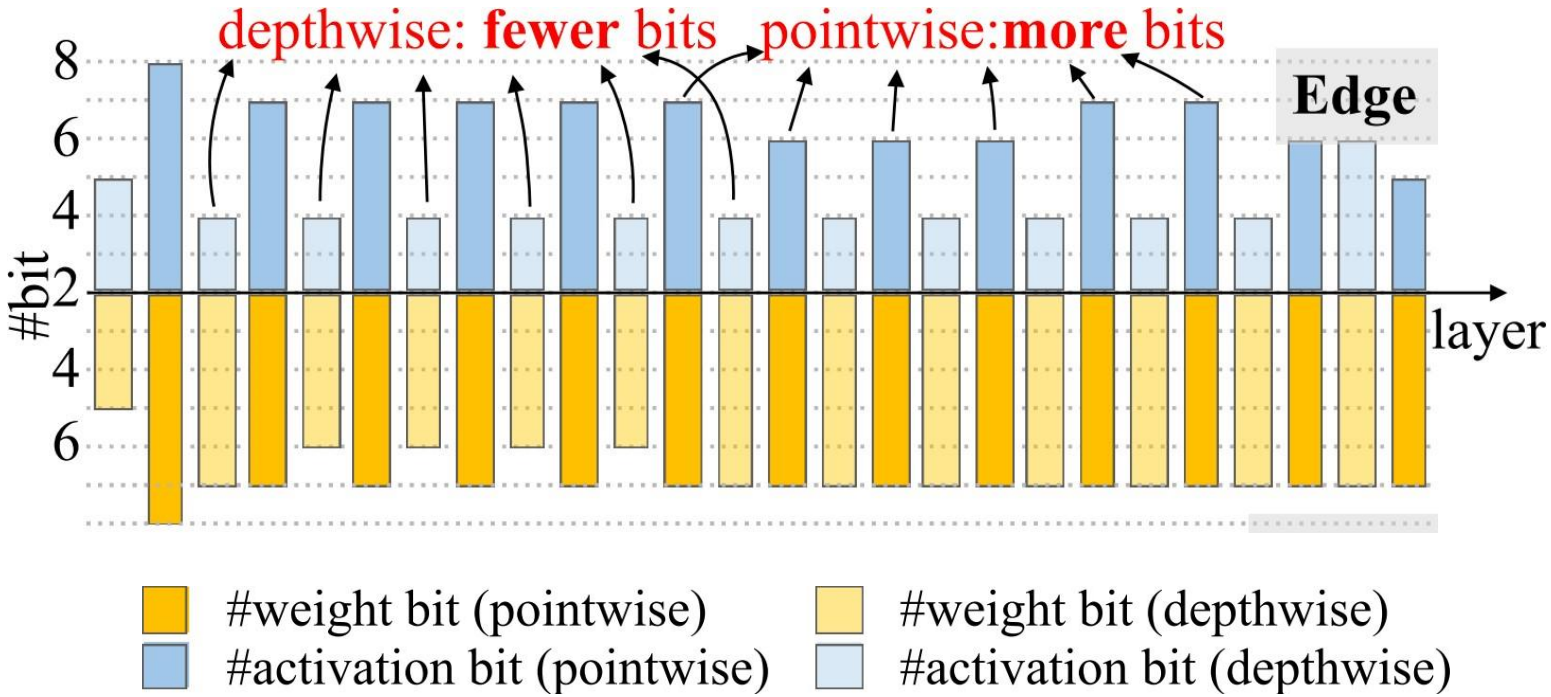
Standard Conv:
#total params = (Dk x Dk x M + 1) x N = (3x3x512+1)x512=2.36M
#total mult-adss = (DF x DF) x #params = (14x14)x2.36M = 462M

Depthwise Separabel Conv:
dw filters
#param = (Dk x Dk + 1) x M = (3x3+1)x512 = 5120
#mult-adds = (DF x DF) x #params = (14x14)x5120 = 1M

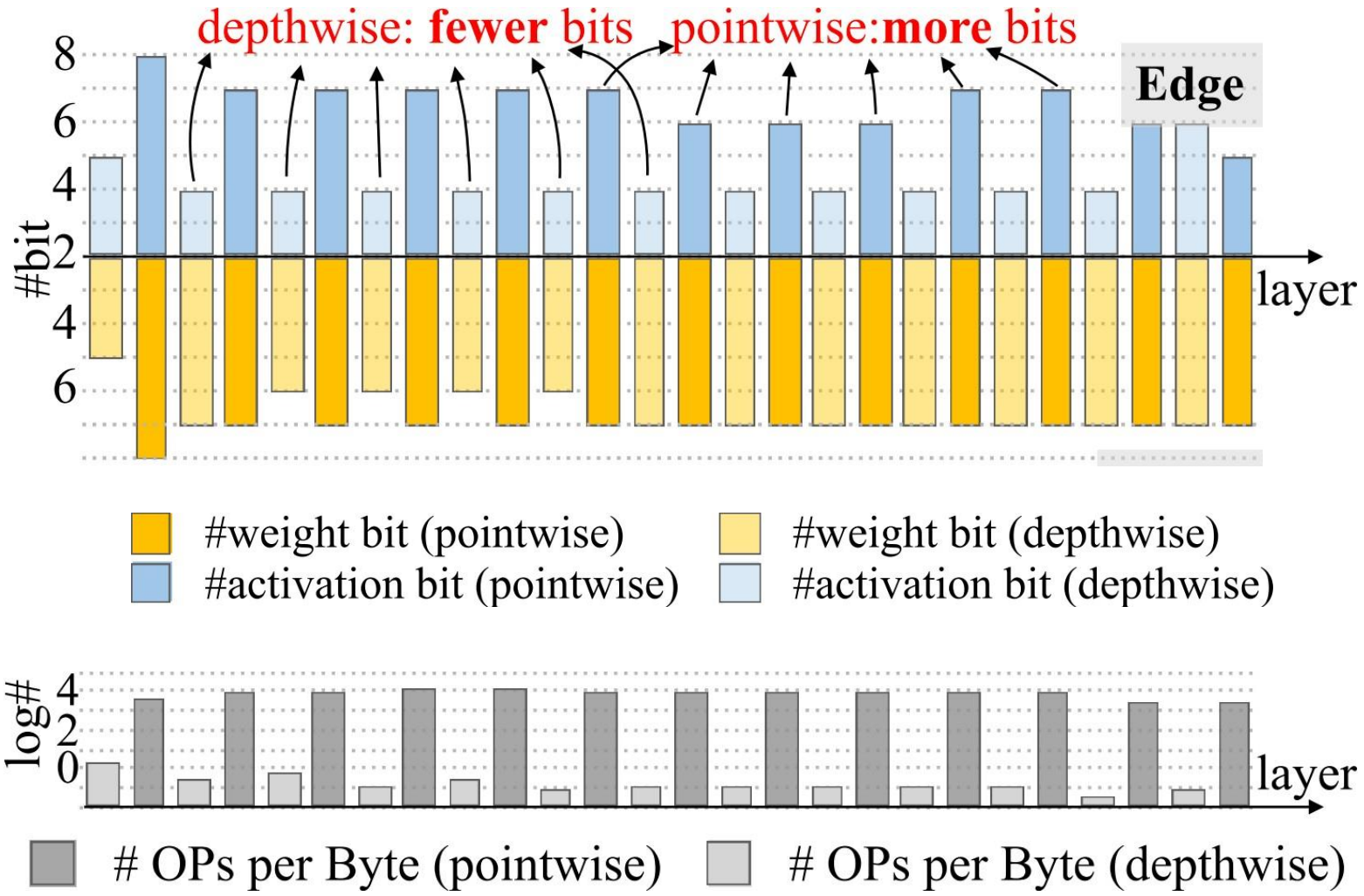
1x1 conv filters
#param = (1 x 1 x M + 1) x N = (1x1x512+1)x512 = 262656
#mult-adds = (DF x DF) x #params = (14x14)x262656 = 51.3M

total params = 5120 + 262656 = 0.27M
total mult-adds = 1M + 51.3M = 52.3M

Interpreting the Quantize Policy on the Edge



Interpreting the Quantize Policy on the Edge (MobileNet)



Low Search Cost

	Search Cost	Top-1	Top-5	Latency
ES	**17 hours**	65.7%	86.8%	45.5 ms
BO	74 hours	66.3%	87.2%	45.5 ms
Ours	**17 hours**	**67.4%**	**87.9%**	45.5 ms

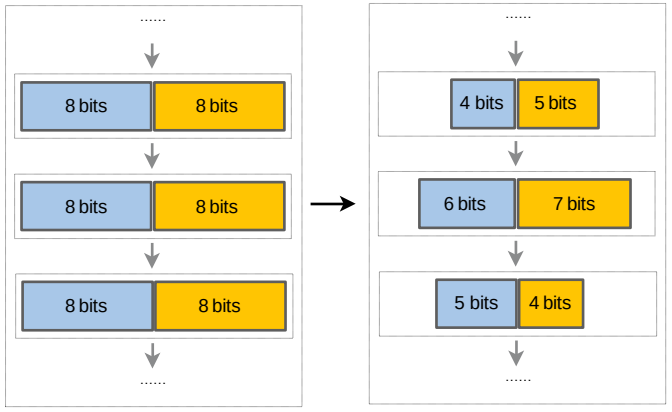
Good Transfer Ability

	Top-1	Top-5	Latency
PACT	61.4%	83.7%	52.2 ms
Ours (search for V2)	66.9%	87.3%	52.1 ms
Ours (transfer from V1)	**65.8%**	**86.6%**	52.1 ms

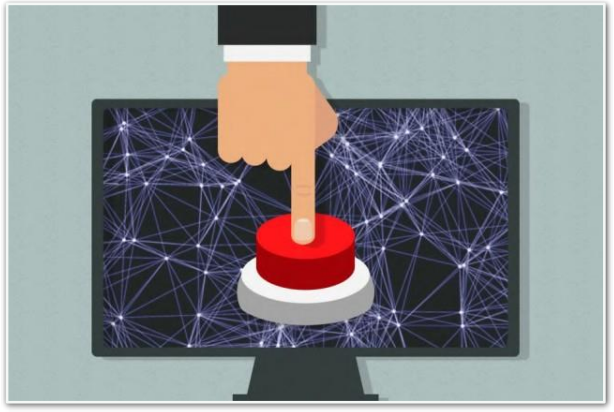
(Transfer the RL agent from MobileNet-v1 to MobileNet-v2)

Contributions

Mixed Precision



Design Automation



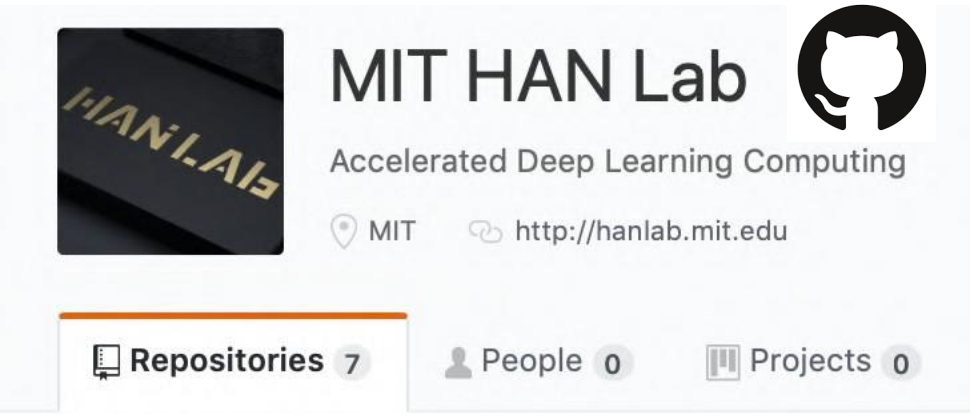
Hardware-Aware Specialization

Related Papers

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware, ICLR'19

AMC: AutoML for Model Compression and Acceleration on Mobile Devices, ECCV'18

HAQ: Hardware-Aware Automated Quantization with Mixed Precision, CVPR'19

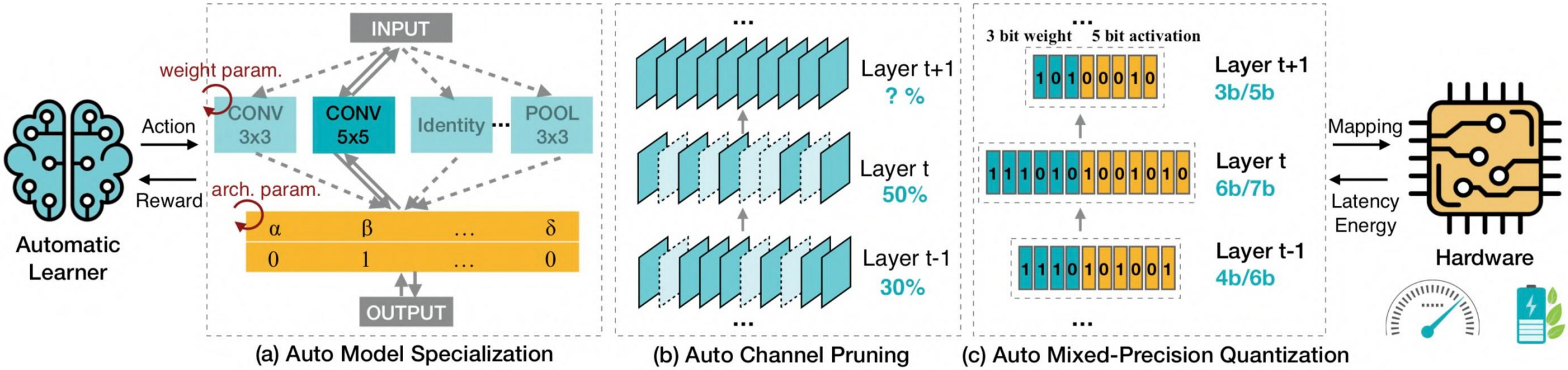


Thursday 9:24am, #199 @CVPR'19

HAQ is Available on Github

github.com/mit-han-lab/HAQ

Summary



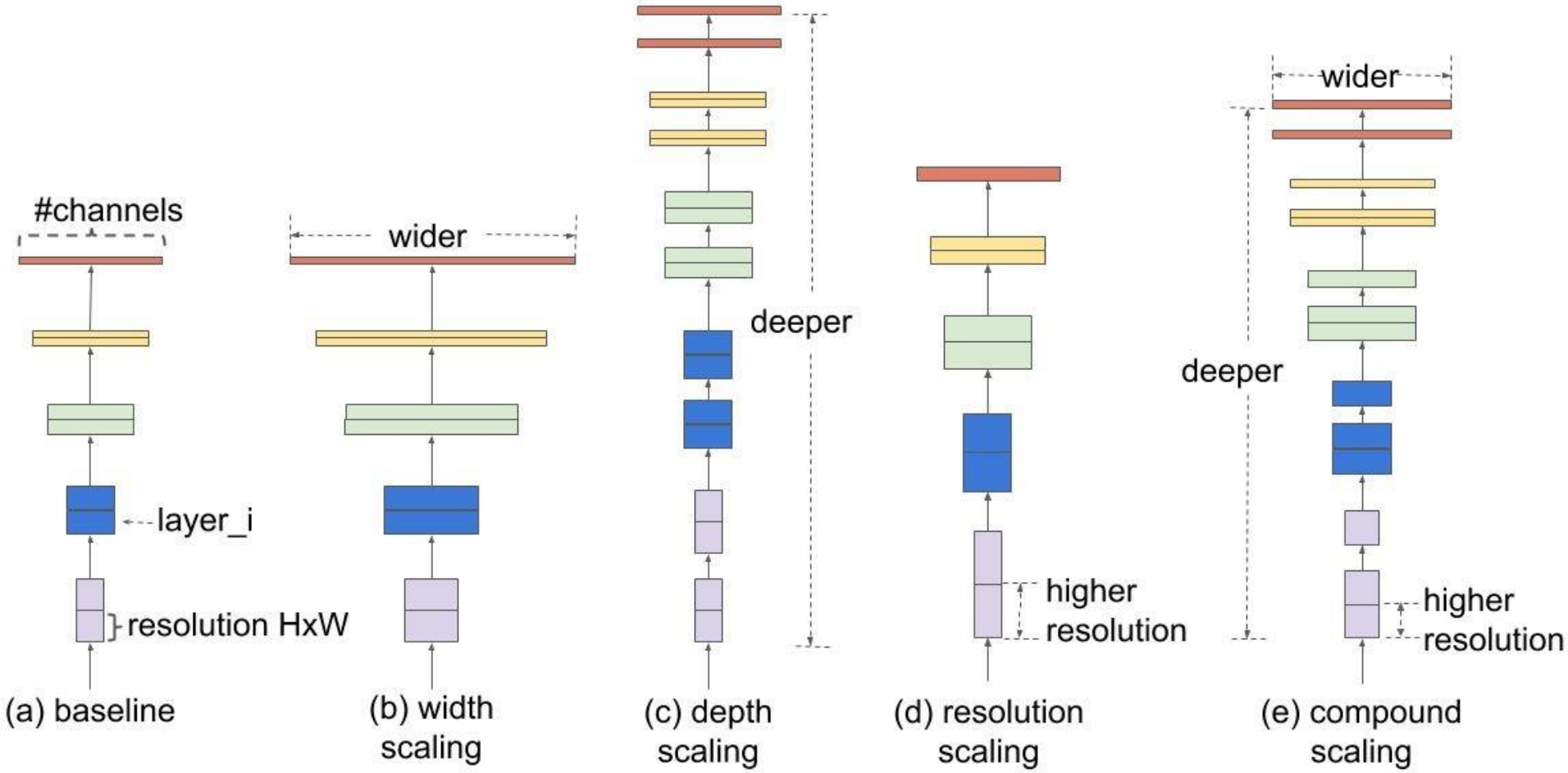
1. ProxylessNAS: automatically architect efficient neural networks
2. AMC: automatic model compression
3. HAQ: automatic quantization with mixed precision

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan, Quoc V. Le

Proprietary + Confidential

How to Scale Up A ConvNet?



Proprietary + Confidential

Compound Scaling

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

Step1:

- First fix φ = 2, and find α, β, γ with local search.

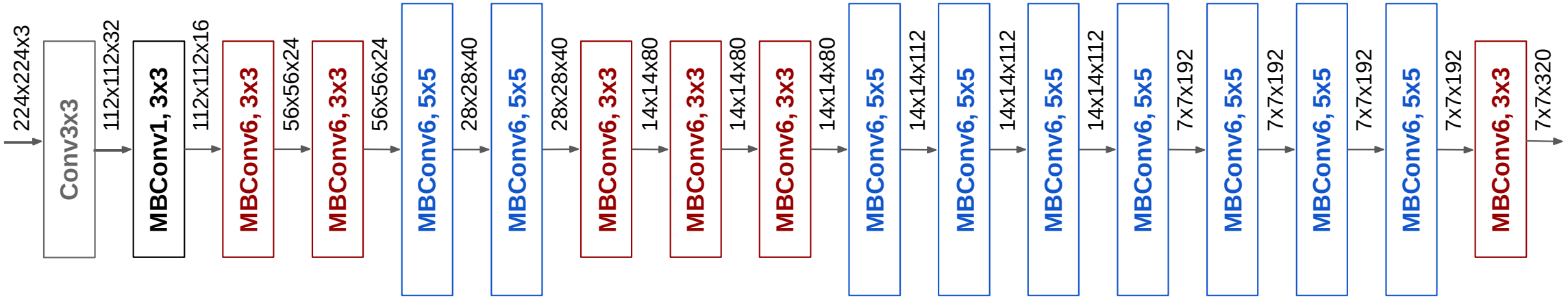
Step2:

- Then fix α, β, γ, and scale the network with different φ.

Compound scaling improves MobileNetV1, MobileNetV2, and ResNet-50.

Proprietary + Confidential

EfficientNet-B0: A New Baseline Network Found by AutoML

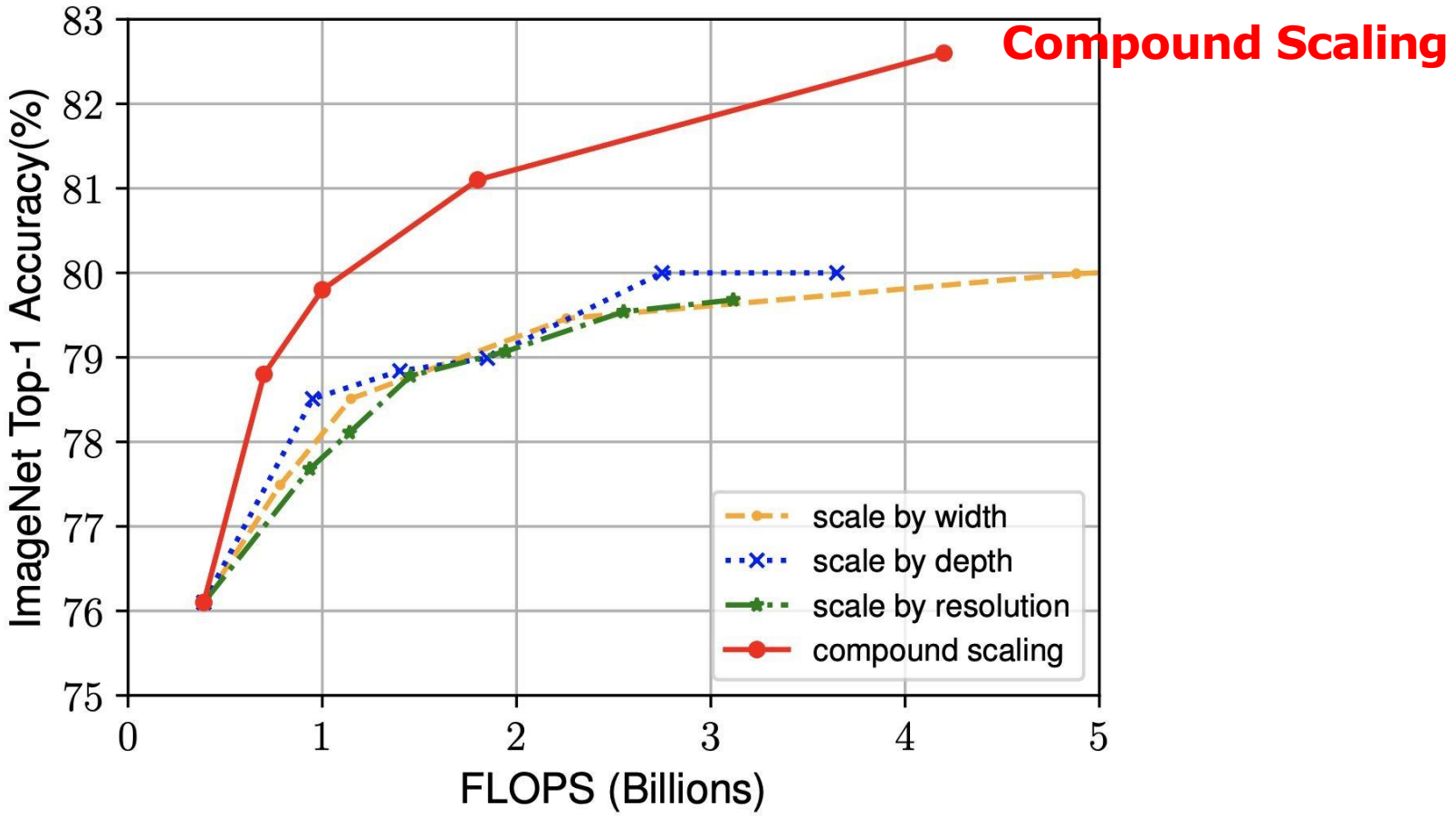


Simple, clean, no branches
irregularity in layer types

MBConv represents "mobile inverted bottleneck" [See MnasNet for more details]

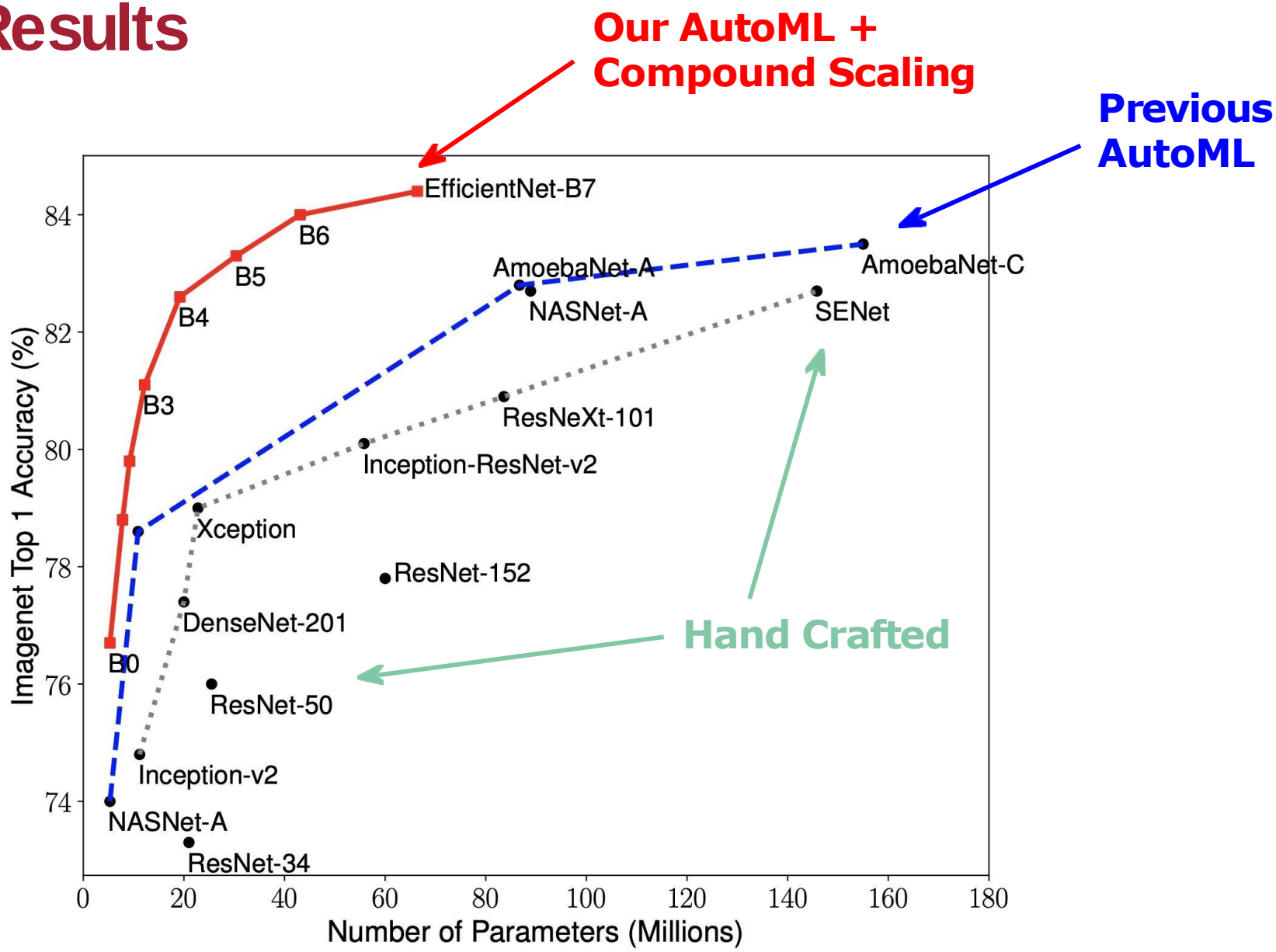
Proprietary + Confidential

Scaling the Same Baseline EfficientNet-B0



Proprietary + Confidential

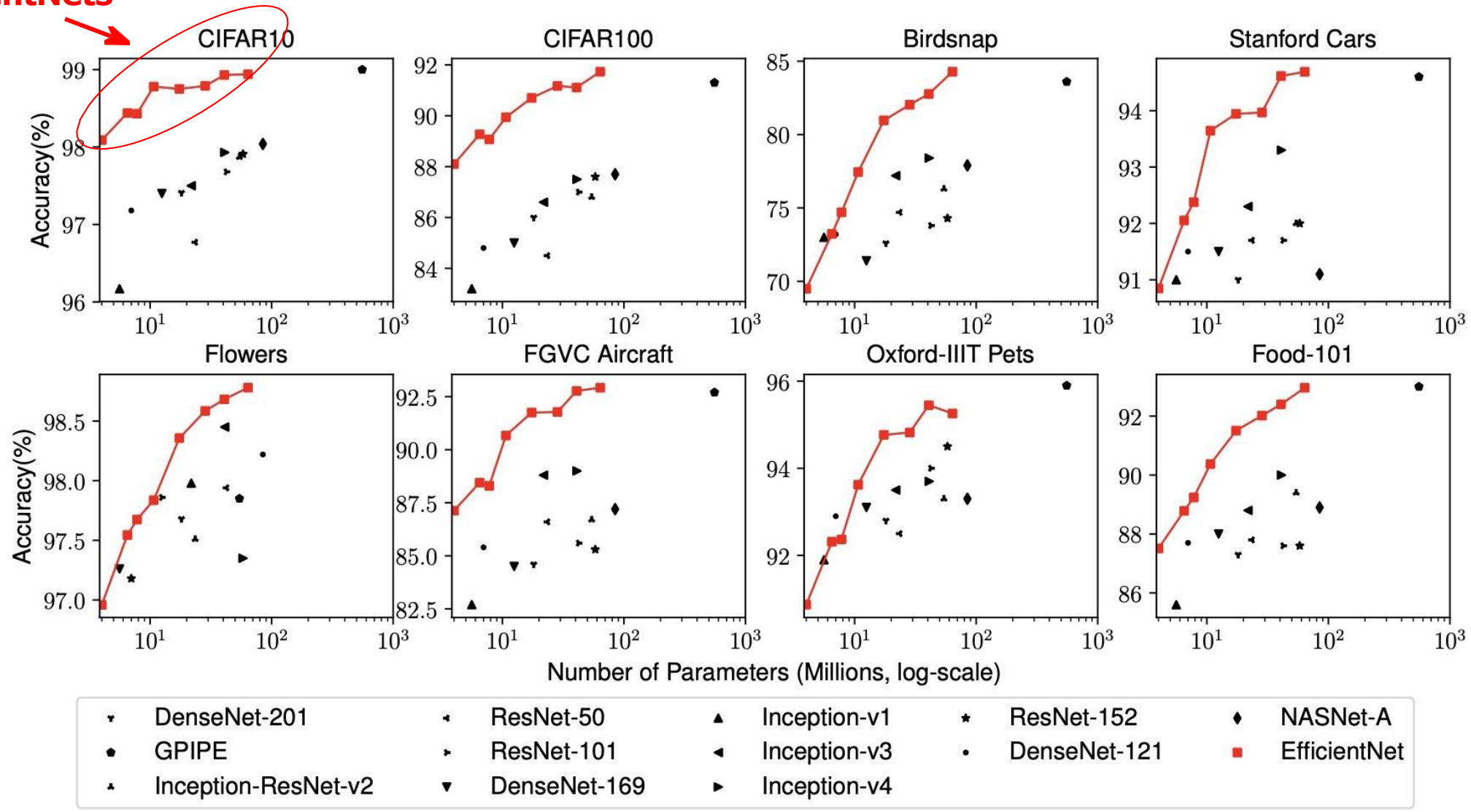
ImageNet Results



Proprietary + Confidential

Transfer Learning Results

EfficientNets



The End