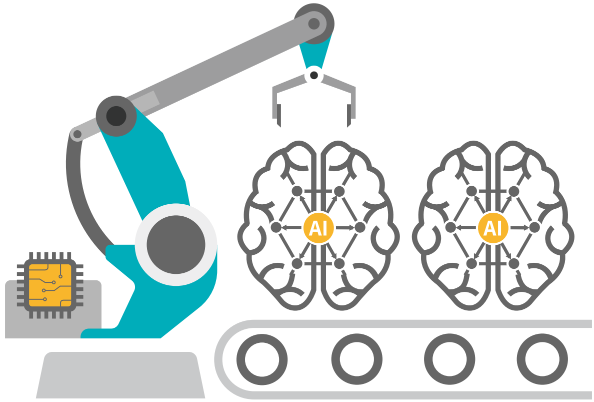
Hardware Efficiency Aware Neural Architecture Search and Compression



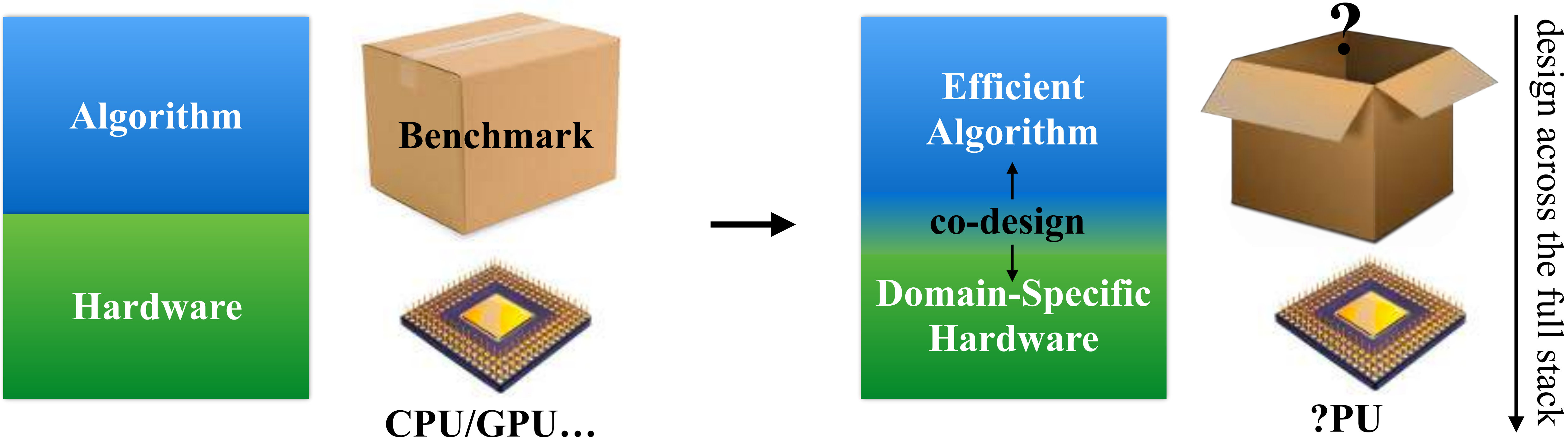
Song Han
Assistant Professor
Massachusetts Institute of Technology

https://songhan.mit.edu

A Challenge for Deep Learning Computing

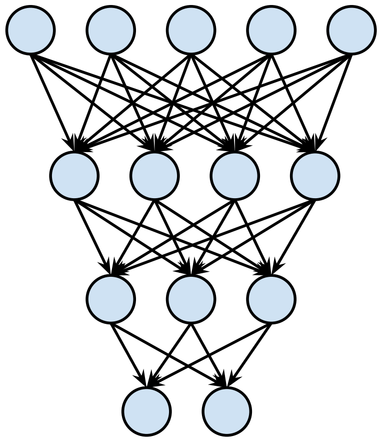
- We are solving more complicated AI problems with larger datasets, which **requires more computation**.
- However, Moore's Law is slowing down; the amount of computation per unit cost is **no longer increasing** at its historic rate.

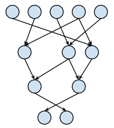
We Need Algorithm and Hardware Co-Design



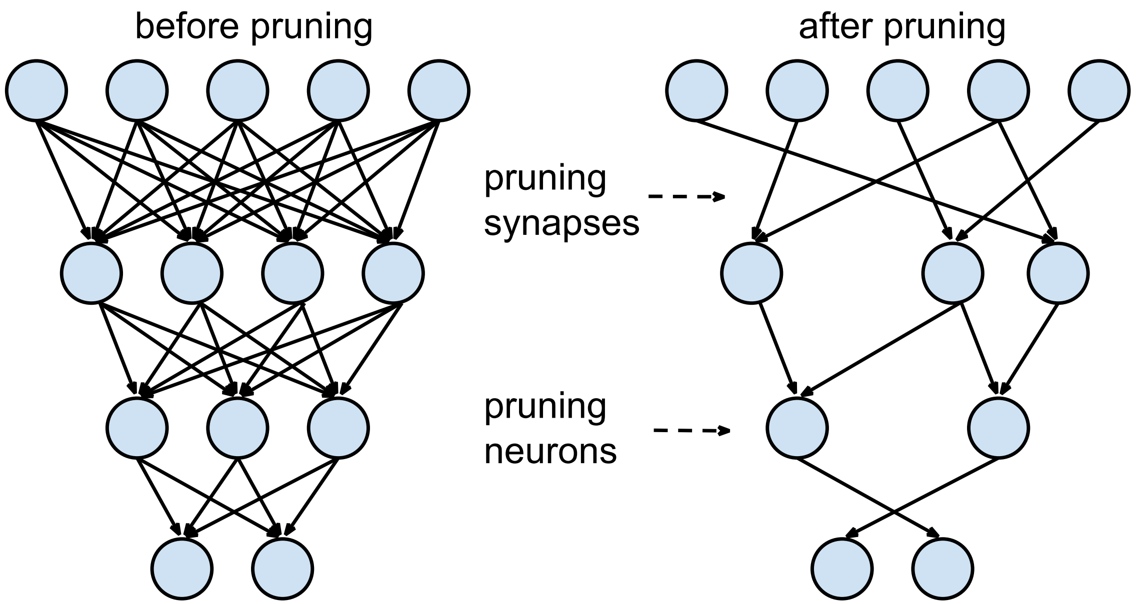
"**There is plenty of room at the top** by optimizing the algorithm. We found that DNN models can be significantly compressed and simplified"

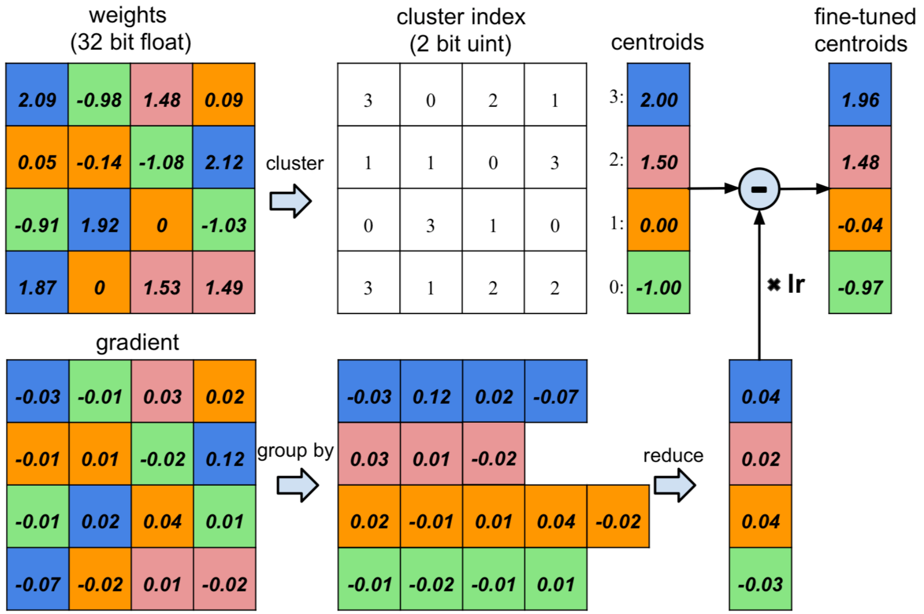
Model Compression





Deep Compression





Pruning

Han et al [NIPS'15]

Quantization

Han et al [ICLR'16]
Best Paper Award

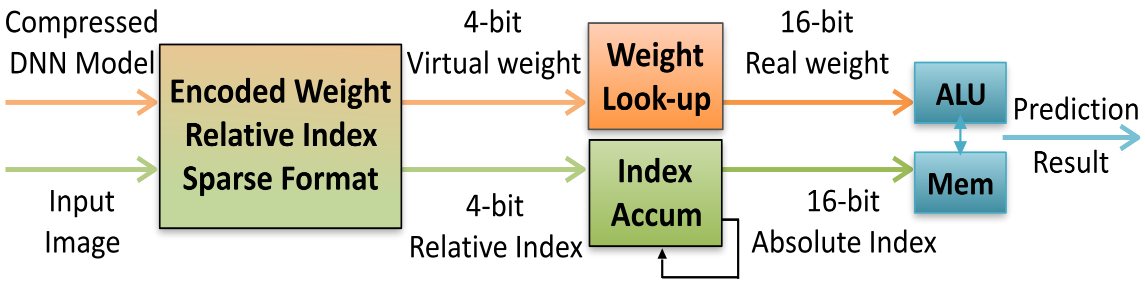
[Learning both Weights and Connections for Efficient Neural Networks]

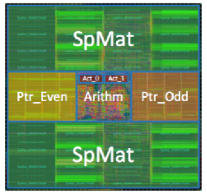
[Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding]

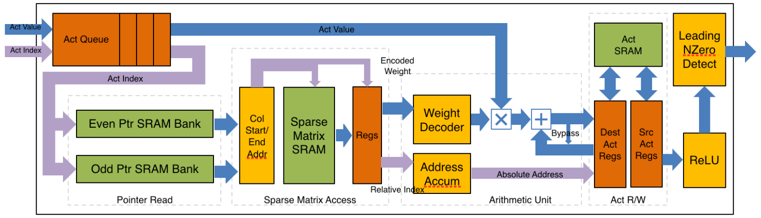
Results: Compression Ratio

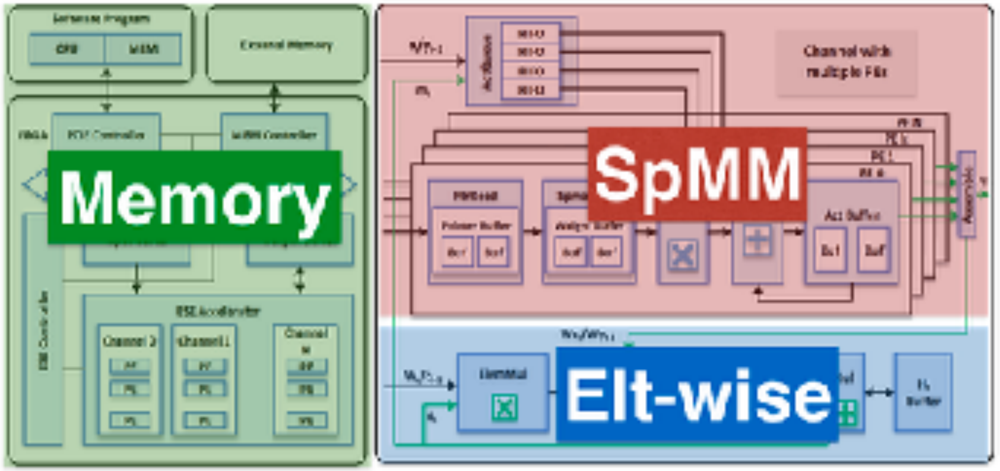
Network	Original Size		Compressed Size	Compression Ratio	Original Accuracy		Compressed Accuracy
LeNet-300	1070KB	→	27KB	**40x**	98.36%	→	98.42%
LeNet-5	1720KB	→	44KB	**39x**	99.20%	→	99.26%
AlexNet	240MB	→	6.9MB	**35x**	80.27%	→	80.30%
VGGNet	550MB	→	11.3MB	**49x**	88.68%	→	89.09%
Inception-V3	91MB	→	4.2MB	**22x**	93.56%	→	93.67%
ResNet-50	97MB	→	5.8MB	**17x**	92.87%	→	93.04%

Hardware Acceleration









EIE Accelerator

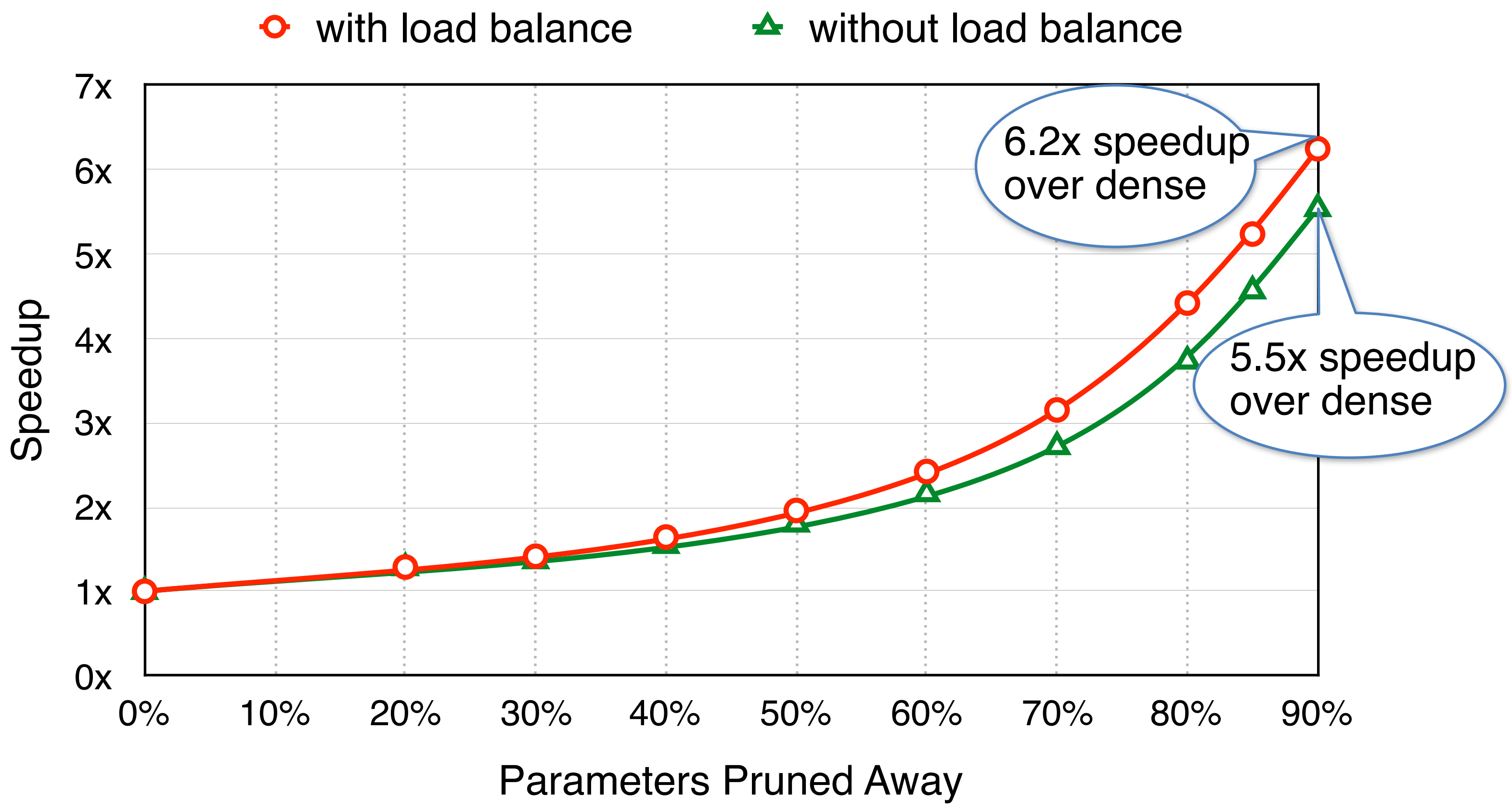
Han et al [ISCA'16]

ESE Accelerator

Han et al [FPGA'17]
Best Paper Award
Available on AWS Marketplace

[EIE: Efficient Inference Engine on Compressed Deep Neural Network]()
[ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA]()

Speeding Up Sparse Neural Network



https://arxiv.org/abs/1612.00694
https://aws.amazon.com/marketplace/pp/B079N2J42R

Deep Compression is Available at:

DNNDK User Guide

UG1327 (v1.5) June 7, 2019

XILINX

Xilinx AI SDK User Guide

UG1354 (v1.0) April 29, 2019

XILINX

DECENT Overview

The Deep Compression Tool (DECENT) includes two capabilities: Coarse-Grained Pruning and trained quantization. These reduce the number of required operations and quantize the weights. The entire working flow of DECENT is shown in the following figure. In this release, only the quantization tool is included. Contact the Xilinx support team if pruning tool is necessary for your project evaluation.

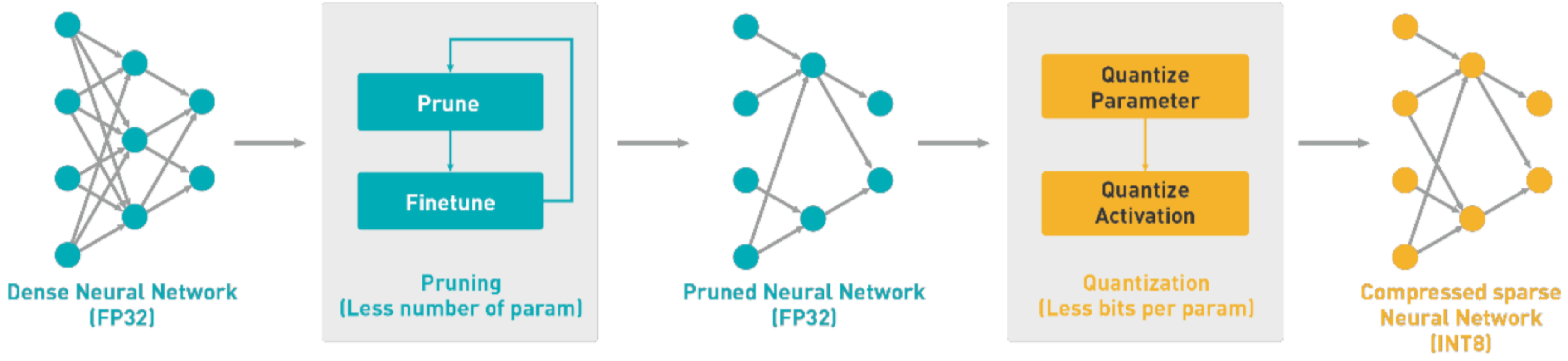


Figure 27: DECENT Pruning and Quantization Flow

DECENT (Caffe Version) Working Flow

Prepare the Neural Network Model

Before running DECENT, prepare the Caffe model in floating-point format and calibration data set, including:

- Caffe floating-point network model prototxt file.
- Pre-trained Caffe floating-point network model caffemodel file.
- Calibration data set. The calibration set is usually a subset of the training set or actual application images (at least 100 images). Make sure to set the source and root_folder in image_data_param to the actual calibration image list and image folder path, as shown in the following figure.

```
# ResNet-50
name: "ResNet-50"
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: false
    mean_value: 104
    mean_value: 107
    mean_value: 123
  }
  image_data_param {
    source: "./data/imagenet_256/calibration.txt"
    root_folder: "./data/imagenet_256/calibration_images/"
    batch_size: 10
    shuffle: false
    new_height: 224
    new_width: 224
  }
}
```

Figure 29: Sample Caffe Layer for Quantization

Note: Only the 3-mean-value format is supported by DECENT. Convert to the 3-mean-value format as required.

DECENT (TensorFlow Version) Usage

The options supported by DECENT_Q are shown in tables 8 and 9.

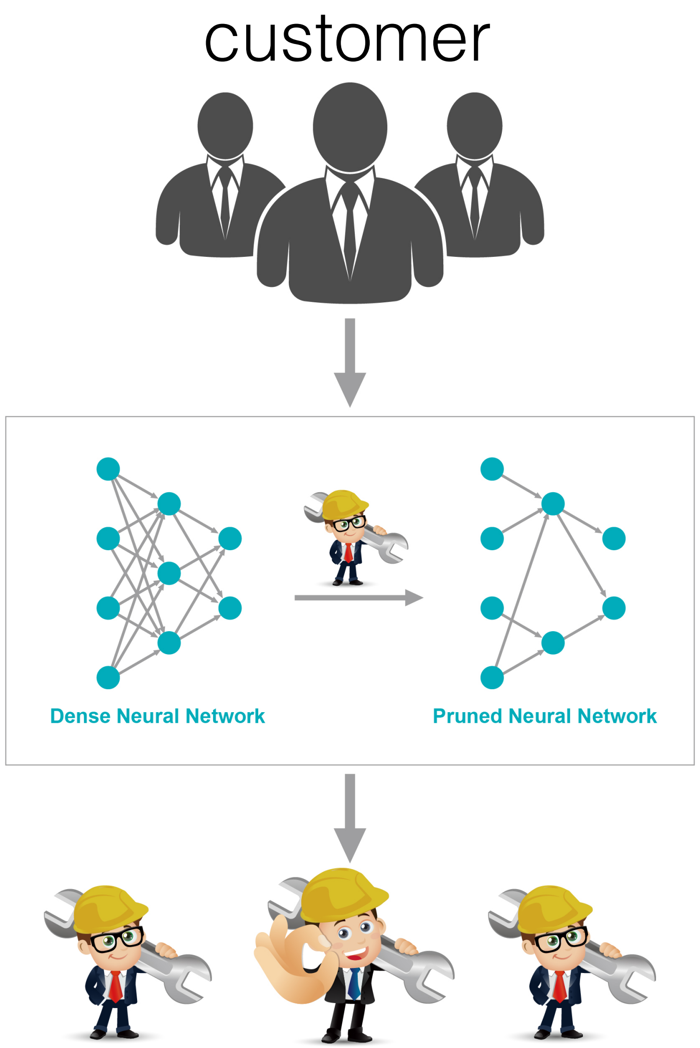
Table 8: DECENT Required Options List

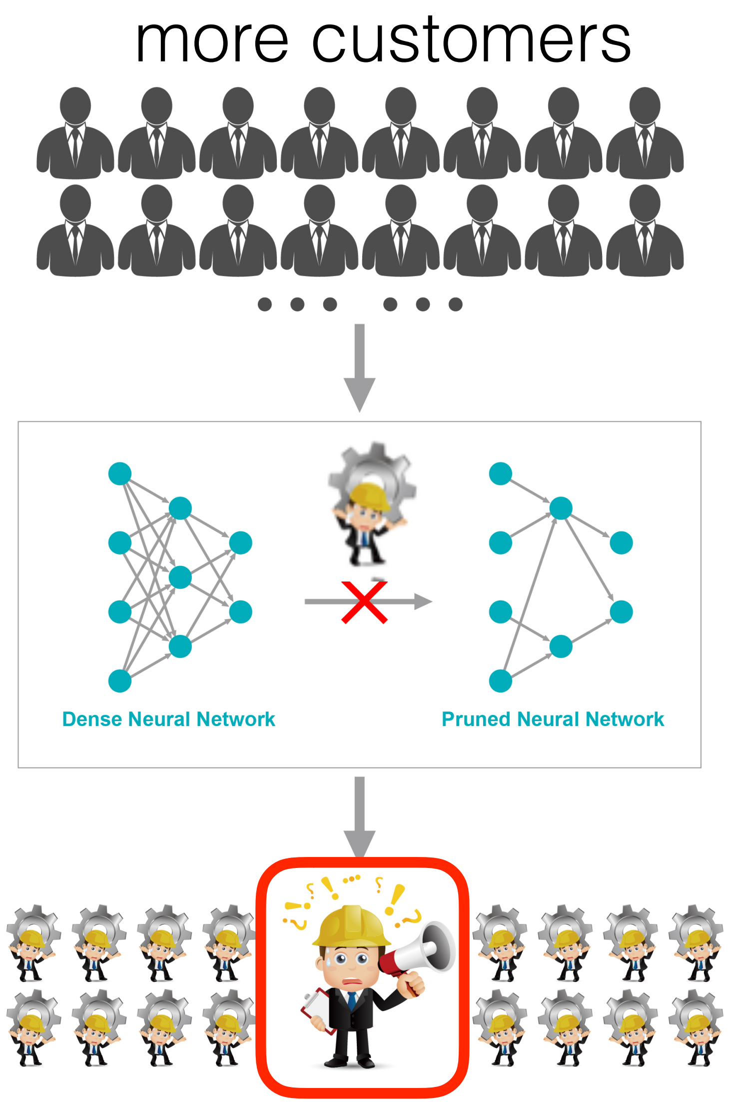
Option Name	Type	Description
--input_frozen_graph	String	TensorFlow frozen `GraphDef` file of the floating-point model.
--input_nodes	String	The name list of input nodes, comma separated.
--output_nodes	String	The name list of output nodes, comma separated.
--input_shapes	String	The shape list of input_nodes. Must be a 4-dimension shape for each node, comma separated, such as 1,224,224,3; support unknown size for batchsize, such as ?,224,224,3. In case of multiple input_node options, assign shape list of each node, separated by':'. For example, ?,224,224,3:?,300,300,1.
--input_fn	String	The function that provides input data for the input_nodes option, used with calibration dataset. The function format is `module_name.input_fn_name`, such as `my_input_fn.input_fn`. The `input_fn` command should take an `int` object as input, which indicates the calibration step number, and should return a `dict' (input_node_name, numpy.Array)'` object for each call, which will be fed into the input nodes of the model. The shape of `numpy.Array` should be consistent with `input_shapes`. Meanwhile, two preset input functions are provided: `default`: a simple image load function to load raw image files and do preprocessings. Mean subtraction, central crop, resize and normalization are supported. Should be used with the `[DefaultInputFnConfig]` command, described below. `random`: a function to produce random numbers for all inputs.

Table 9: DNNC Optional Option List

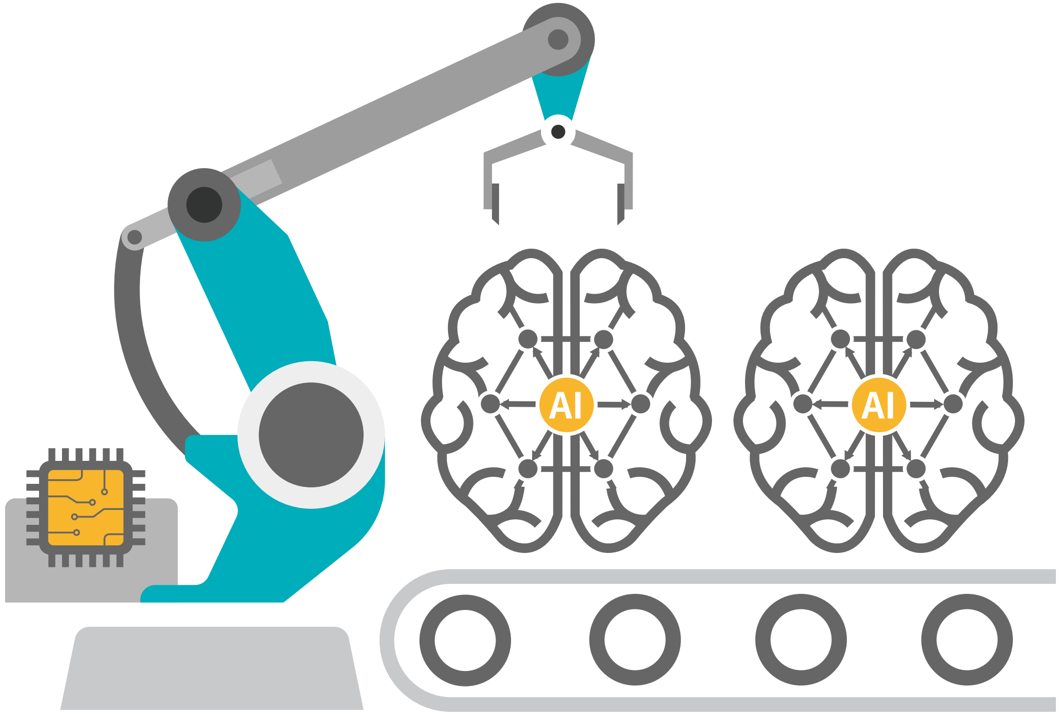
Has all the bottlenecks been solved?

There's a Shortage of Deep Learning Engineers





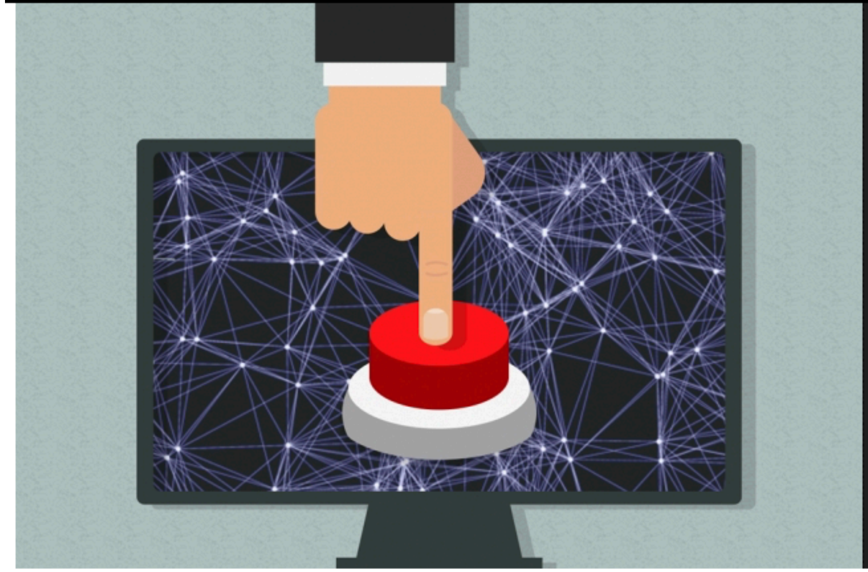
Design Automation for NN



MIT News

ON CAMPUS AND AROUND THE WORLD

Browse or Search



FULL SCREEN

MIT researchers have developed an efficient algorithm that could provide a "push-button" solution for automatically designing fast-running neural networks on specific hardware.

Image: Chelsea Turner, MIT

Kicking neural network design automation into high gear

Algorithm designs optimized machine-learning models up to 200 times faster than traditional methods.

Rob Matheson | MIT News Office
March 21, 2019

Press Inquiries

RELATED

Paper: "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware"

Song Han

Microsystems Technology Laboratories

A new area in artificial intelligence involves using algorithms to automatically design machine-learning systems known as neural networks, which are more accurate and efficient than those developed by human engineers. But this so-called neural architecture search (NAS) technique is computationally expensive.

A state-of-the-art NAS algorithm recently developed by Google to run on a squad of graphical processing units (GPUs) took 48,000 GPU hours to produce a single convolutional neural network, which is used for image classification and detection tasks. Google has the wherewithal to run hundreds of GPUs and other specialized hardware in parallel, but that's out of reach for many others.

Link

IEEE SPECTRUM

Topics Reports Blogs Multimedia

Using AI to Make Better AI

New approach brings faster, AI-optimized AI within reach for image recognition and other applications

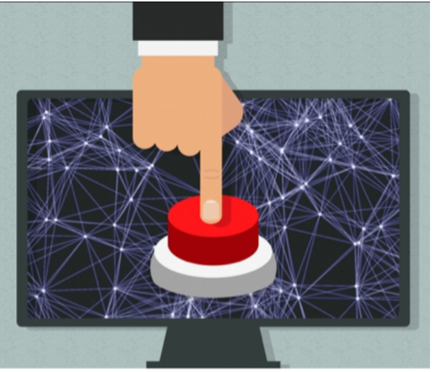
By **Mark Anderson**

Illustration: iStockphoto

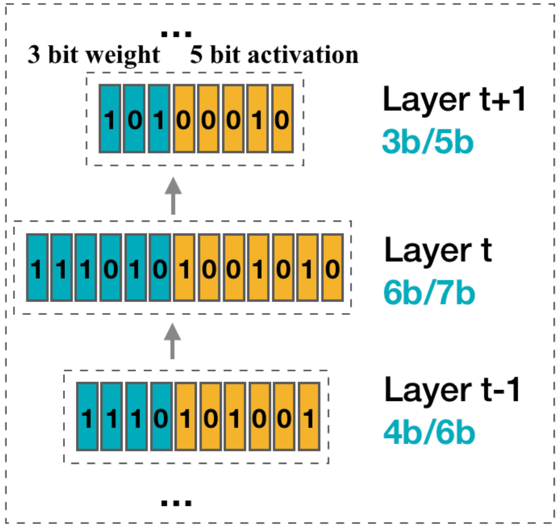
Since 2017, AI researchers have been using AI neural networks to help design better and faster AI neural networks. Applying AI in pursuit of better AI has, to date, been a largely academic pursuit—mainly because this approach requires tens of thousands of GPU hours. If that's what it takes, it's likely quicker and simpler to design real-world AI applications with the fallible guidance of educated guesswork.

Next month, however, a team of MIT researchers will be presenting a so-called "Proxyless neural architecture search" algorithm that can speed up the AI-optimized AI design process by 240 times or more. That would put faster and more accurate AI within practical reach for a broad class of image recognition algorithms and other related applications.

Link

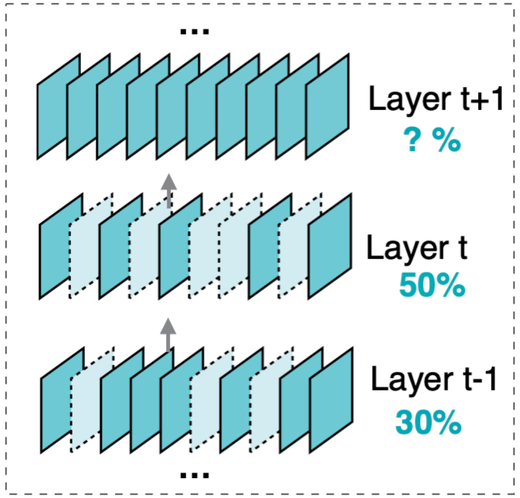


Design Automation for Efficient Deep Learning Computing



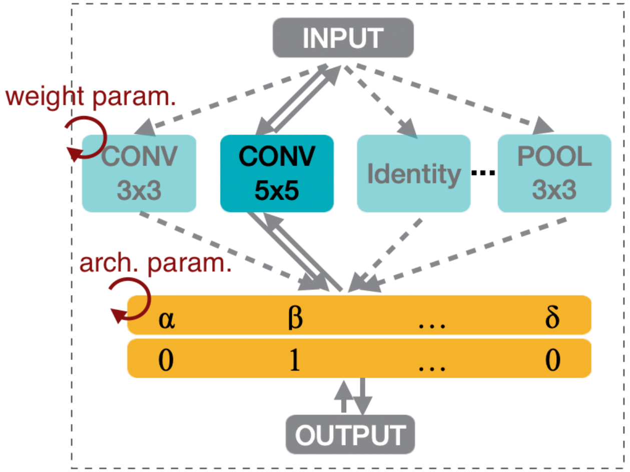
HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



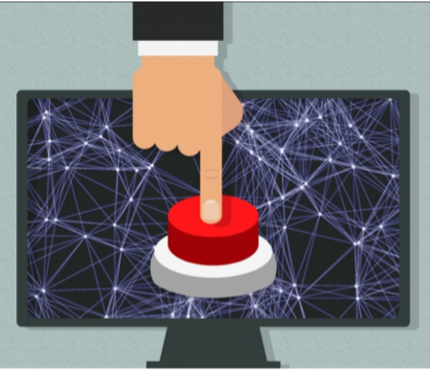
AMC: AutoML for Model Compression

[ECCV 2018]

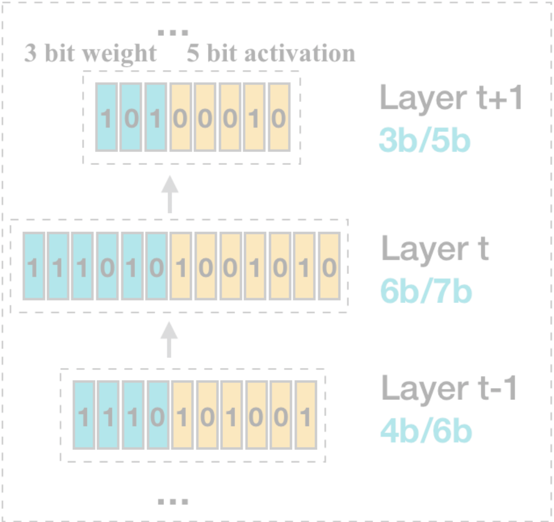


Proxyless Neural Architecture Search

[ICLR 2019]

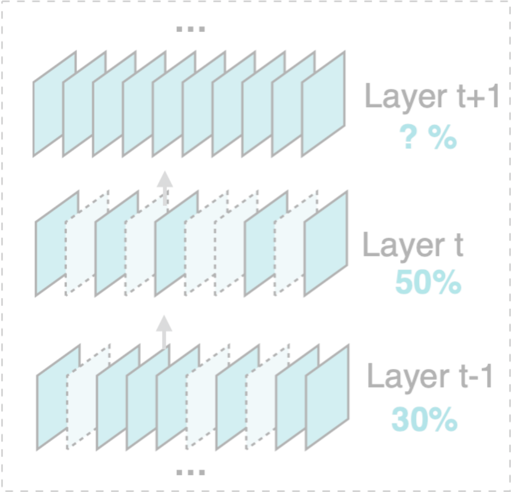


Design Automation for Efficient Deep Learning Computing



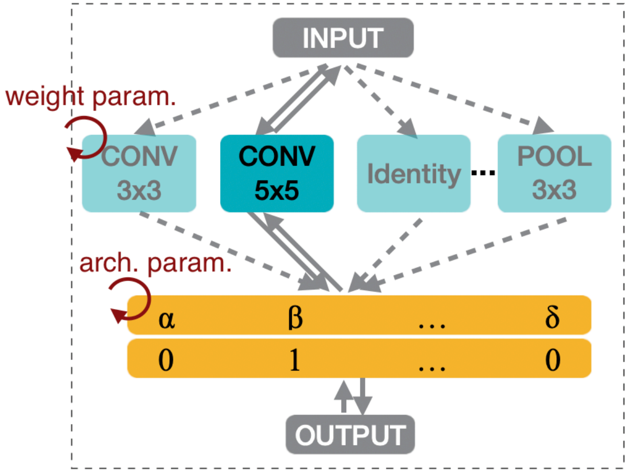
HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



AMC: AutoML for Model Compression

[ECCV 2018]



Proxyless Neural Architecture Search

[ICLR 2019]

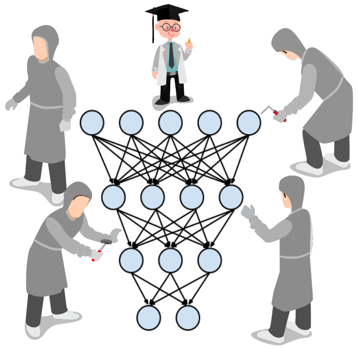
ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware

Han Cai, Ligeng Zhu, Song Han

Massachusetts Institute of Technology

ICLR'19

From Manual Design to Automatic Design

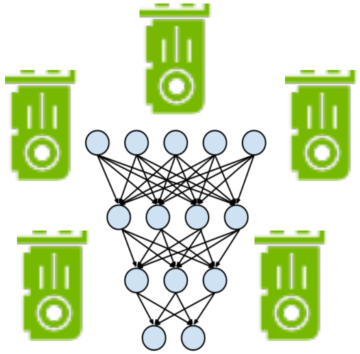


Use Human Expertise

Manual Architecture Design

VGGNets
Inception Models
ResNets
DenseNets
....

Computational Resources

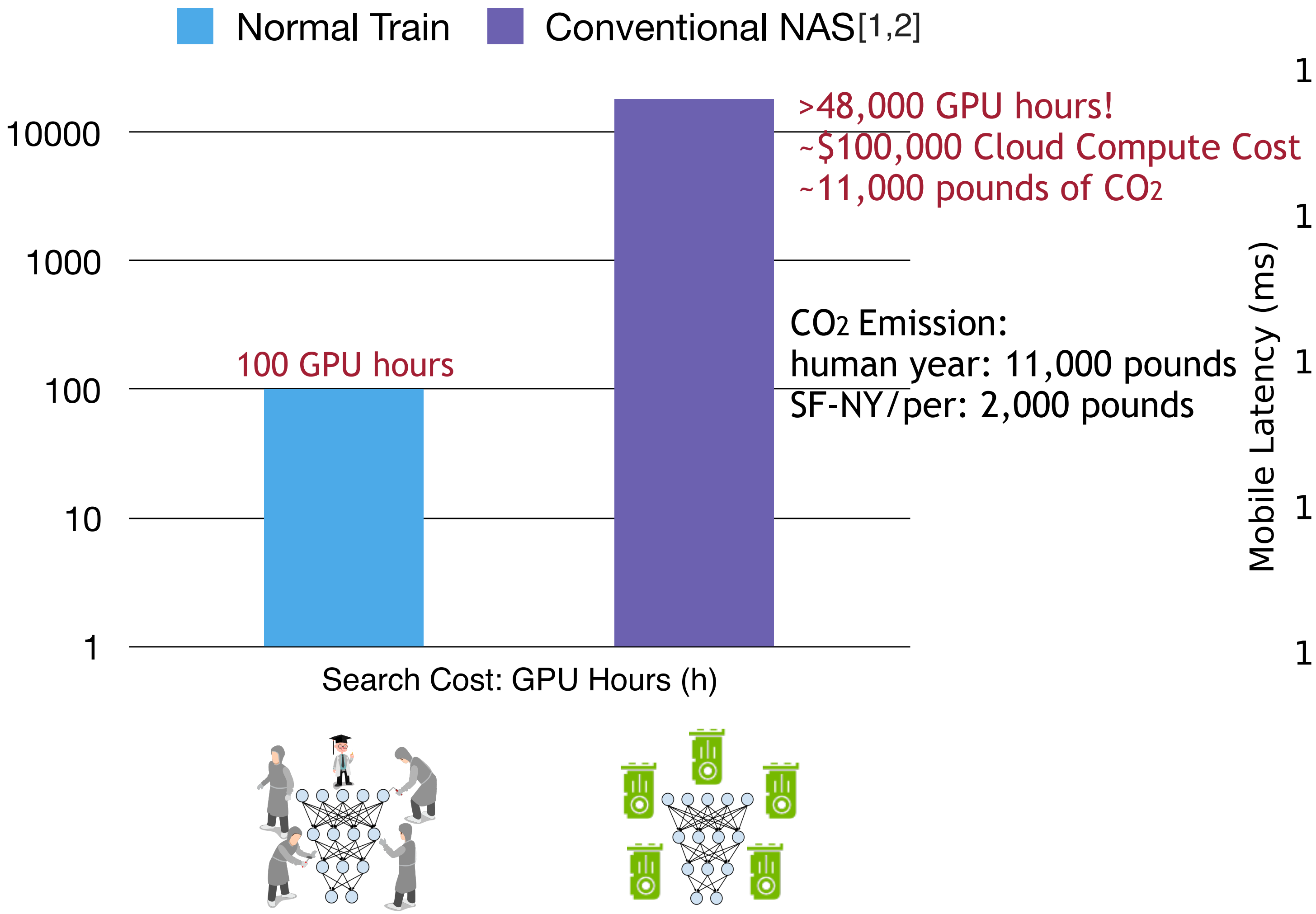


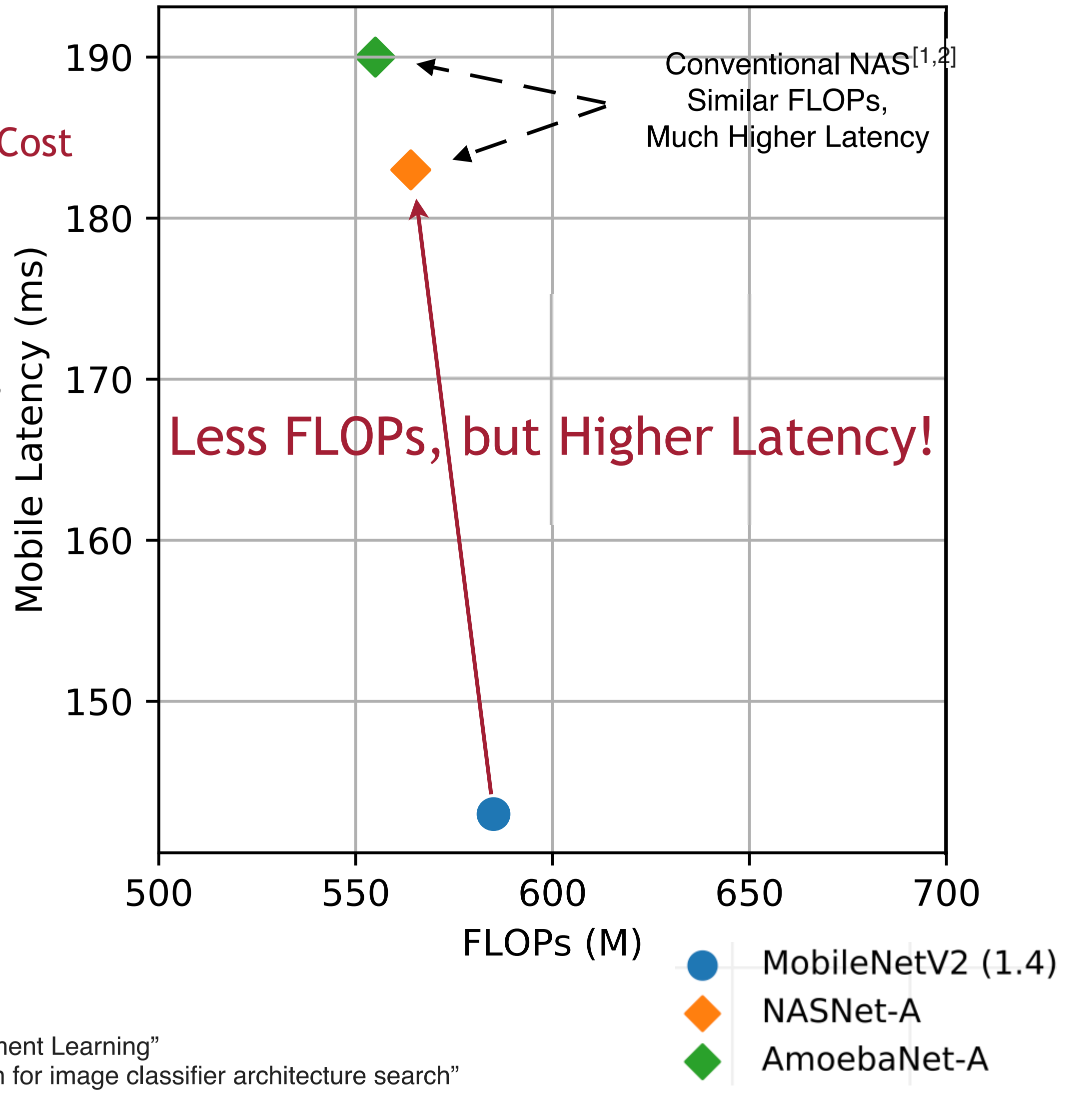
Use Machine Learning

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

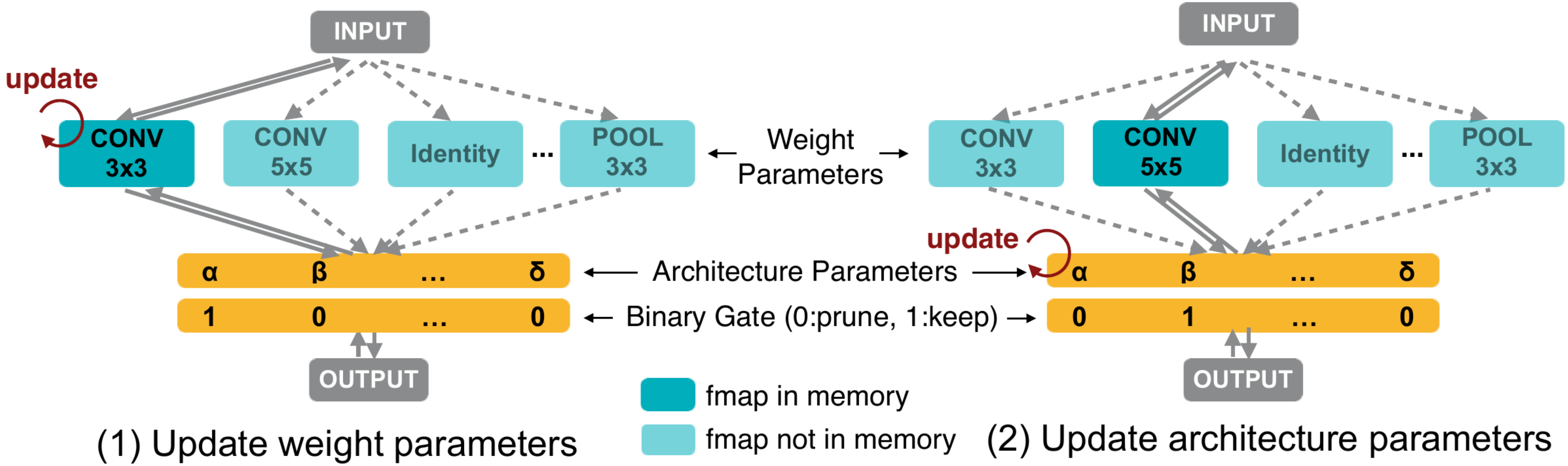
Binarization

Save GPU hours

Save GPU Memory

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

ProxylessNAS: Implementation

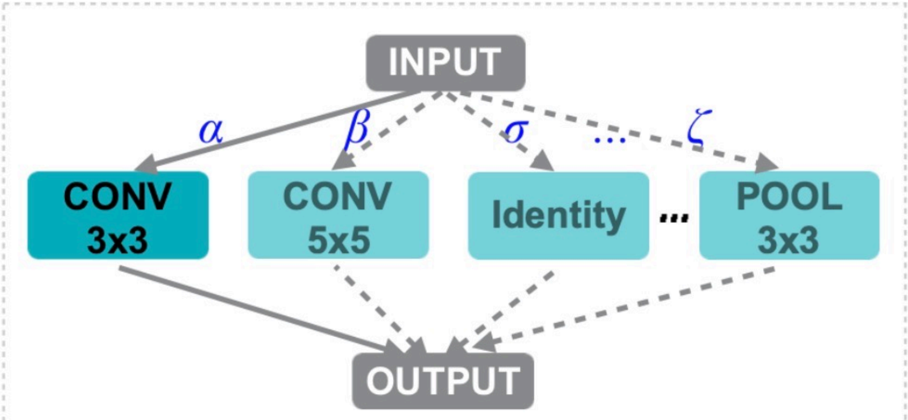


Only one path in GPU memory. Scalable to a large design space.

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

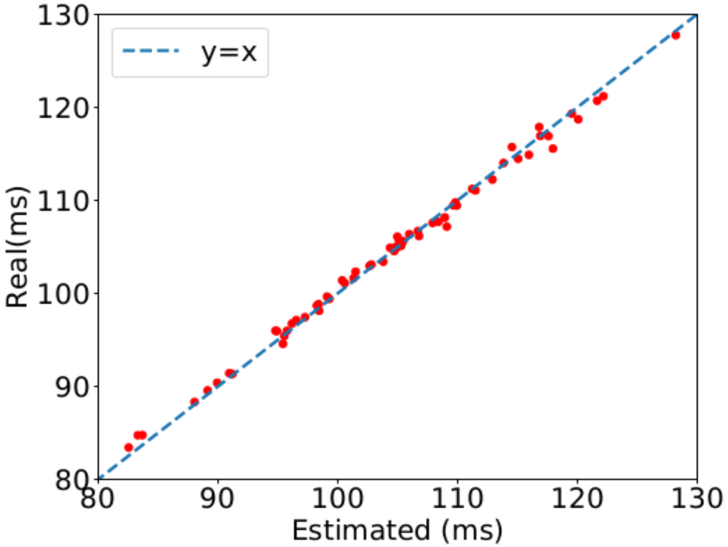
Latency Modeling on Target Hardware

Make Latency Differentiable



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

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



- 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

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

Efficiently Search an Efficient Model without Proxy

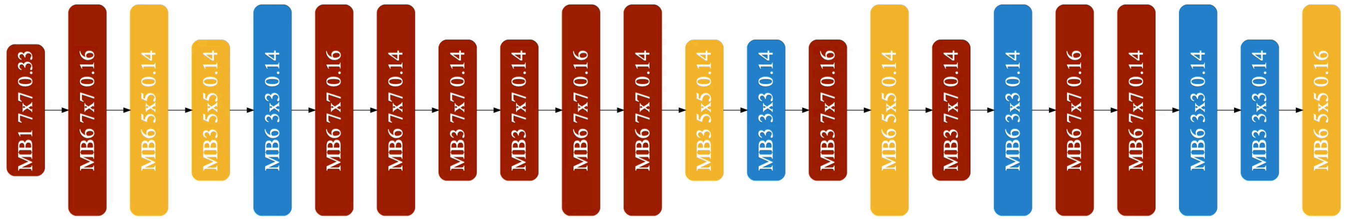
previous work have to utilize **proxy tasks:**

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

Demo: the Search History on Different HW

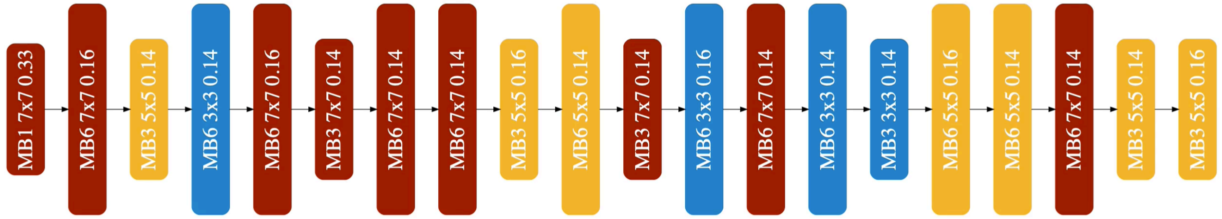
Learning both the weight and architecture

Mobile



(1) The history of finding efficient Mobile model

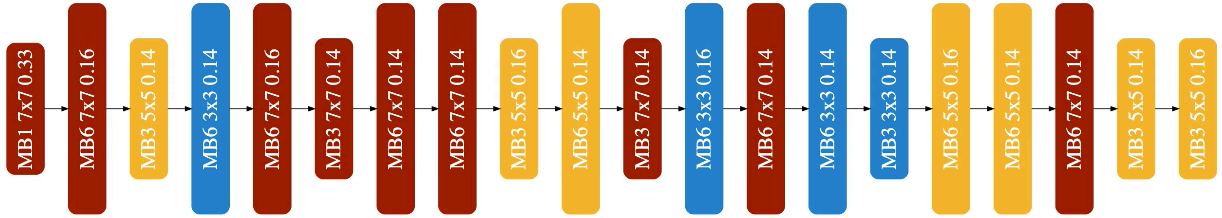
CPU



Epoch-00

(2) The history of finding efficient CPU model

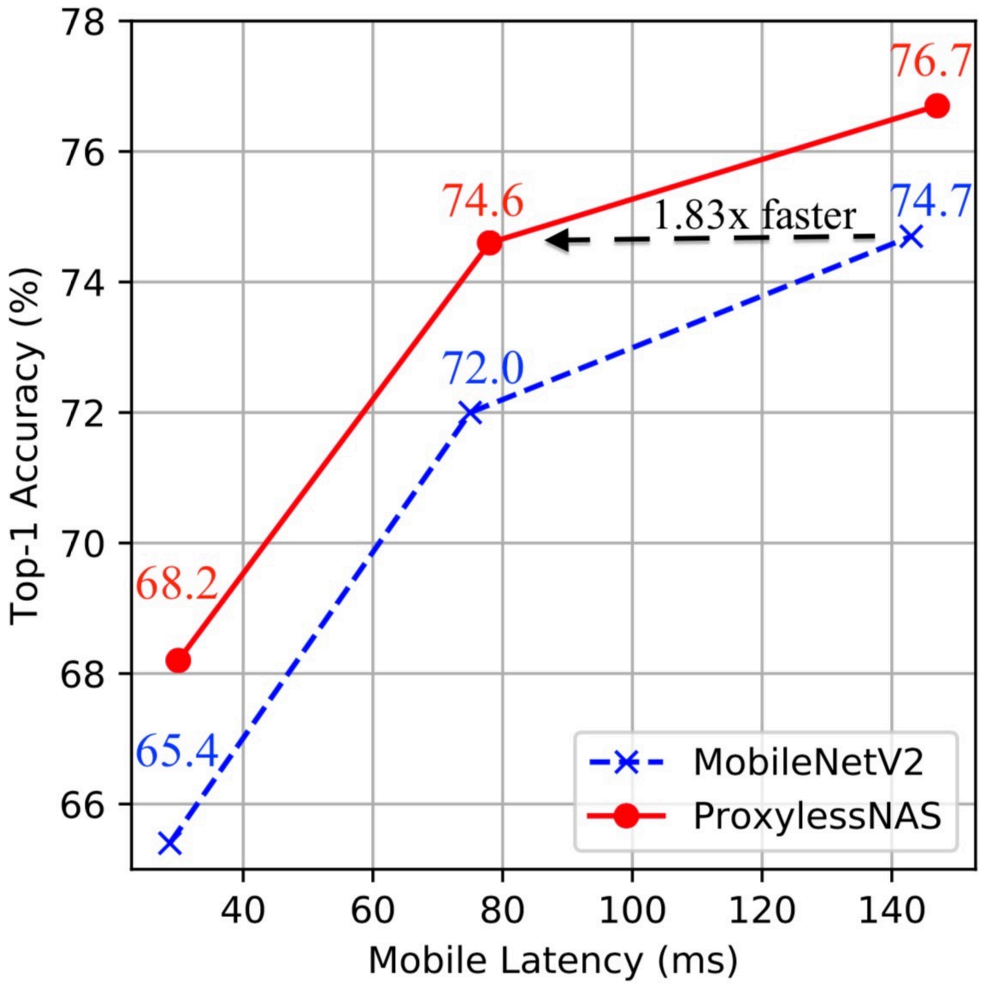
GPU



(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

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

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

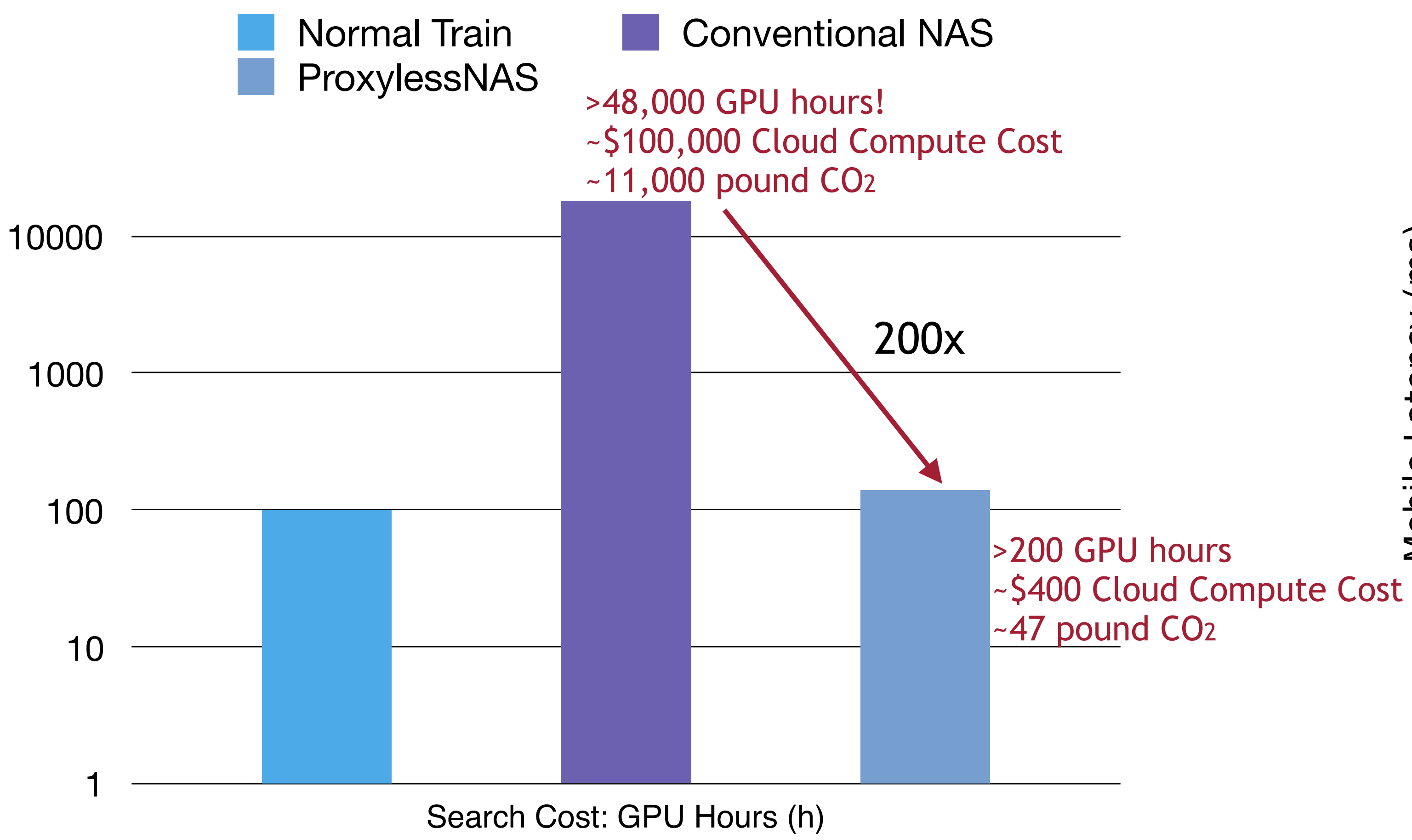
Model	Latency	Top-1 Acc
quant_mobilenetv2_128_100	20ms	62.9%
quant_mobilenetv2_192_50	20ms	61.6%
quant_mobilenetv2_224_35	21ms	58.1%
quant_mobilenetv2_160_75	26ms	64.6%
quant_mobilenetv2_224_50	28ms	63.7%
quant_mobilenetv2_160_100	31ms	67.4%
float_mnasnet_224_50	**36ms**	**67.9%**
quant_mobilenetv2_192_75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

ProxylessNAS:

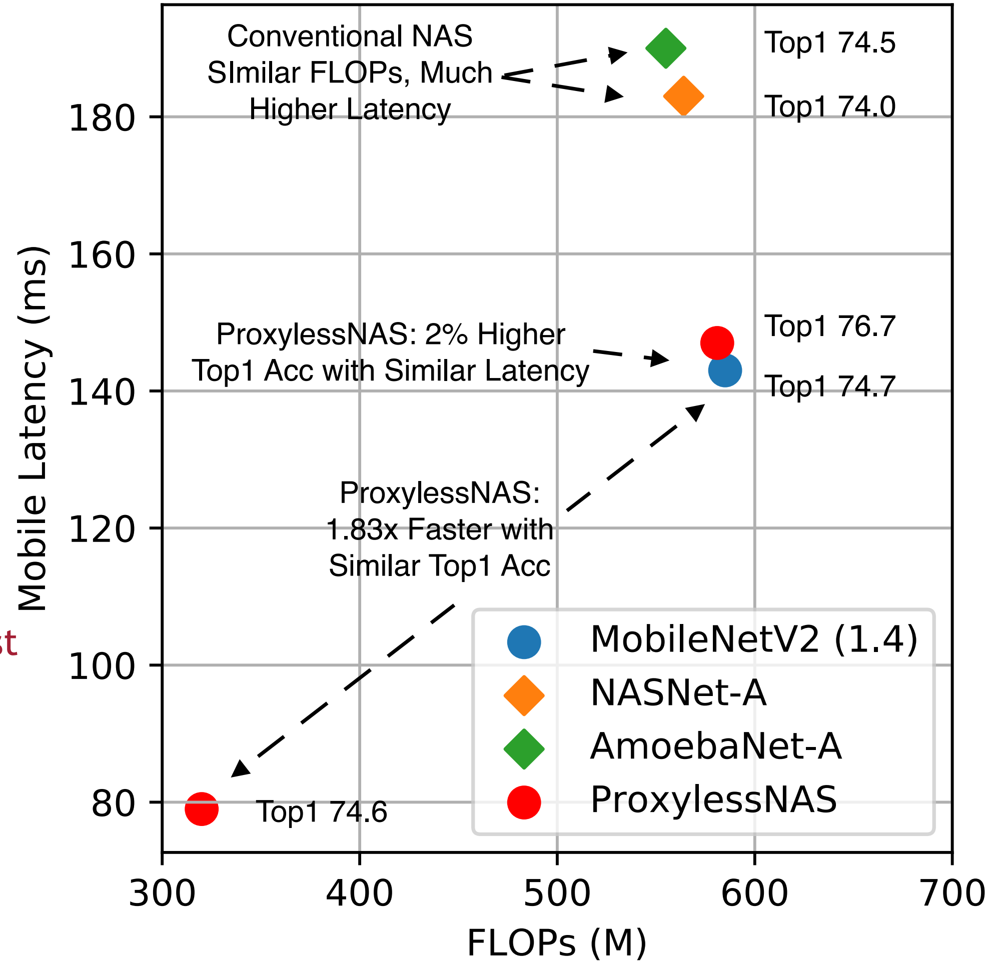
35ms 69.2%

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

Efficiently Search a Model



Search an Efficient Model

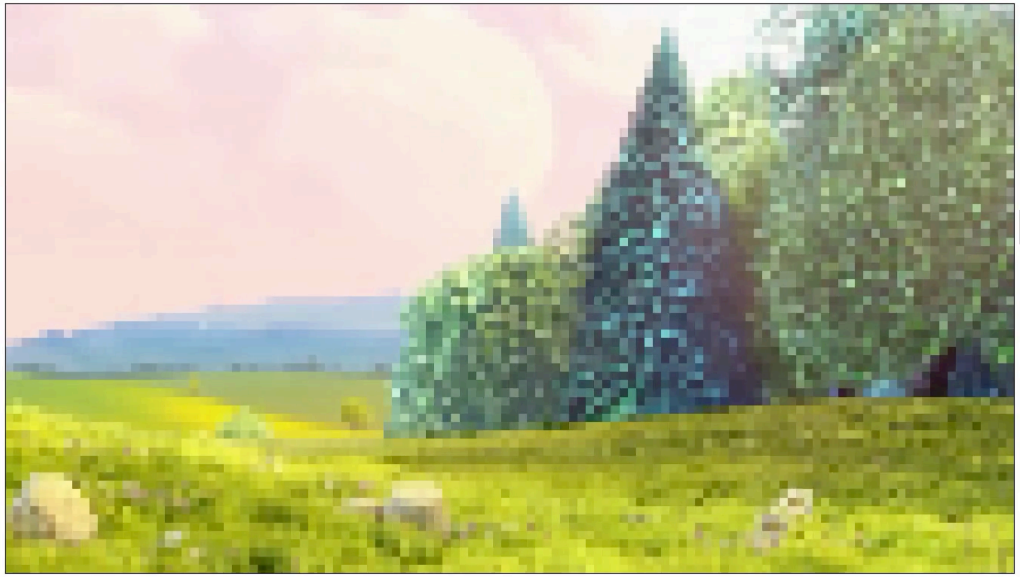


ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware, ICLR'19
https://arxiv.org/pdf/1812.00332.pdf

Accelerate Super Resolution with ProxylessNAS

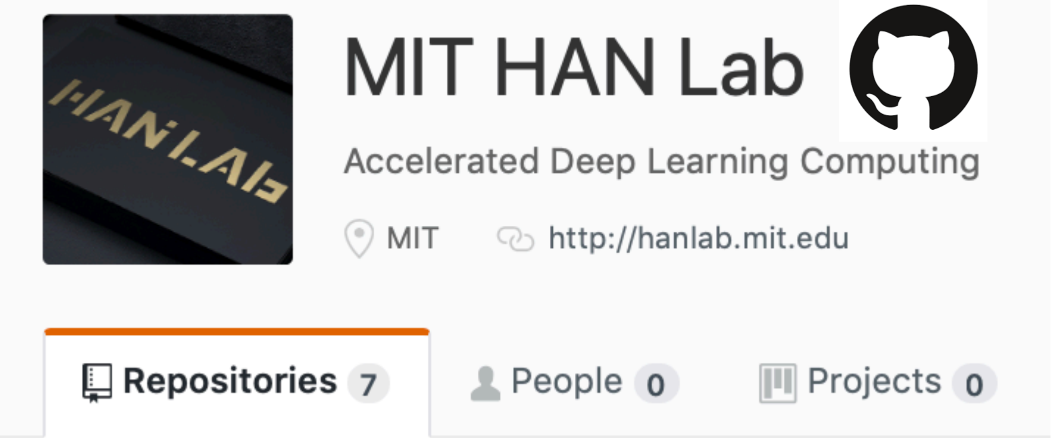
CARN
[ECCV'18]
==>

CARN 16FPS PNSR:21.09



Proxyless
NAS
==>

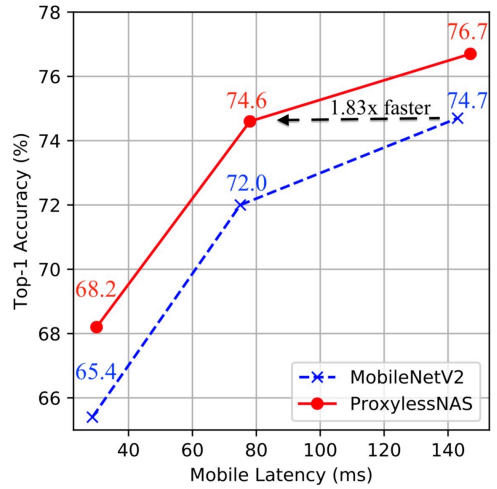
Ours 41FPS PNSR:21.26

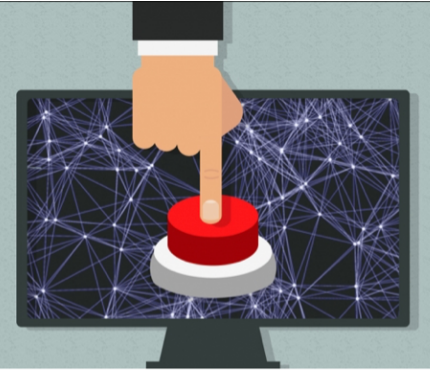


ProxylessNAS is Available on Github

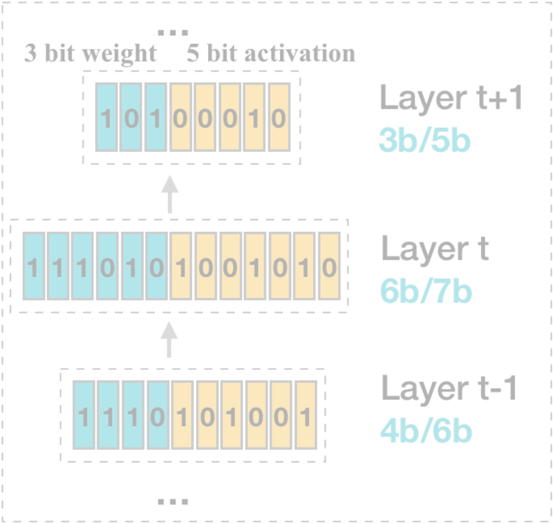
```python
from proxyless_nas import *
net = proxyless_cpu(pretrained=True)
```

github.com/MIT-HAN-LAB/ProxylessNAS



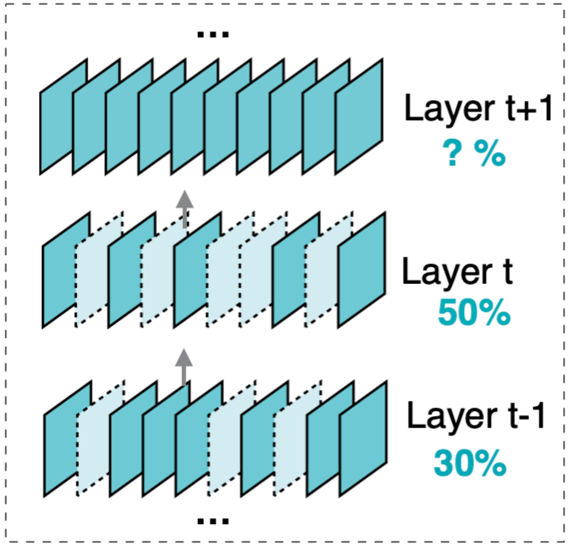


Design Automation for Efficient Deep Learning Computing



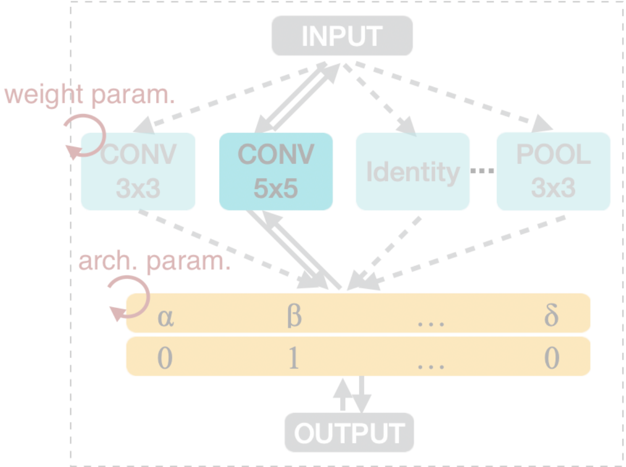
HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



AMC: AutoML for Model Compression

[ECCV 2018]



Proxyless Neural Architecture Search

[ICLR 2019]

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)

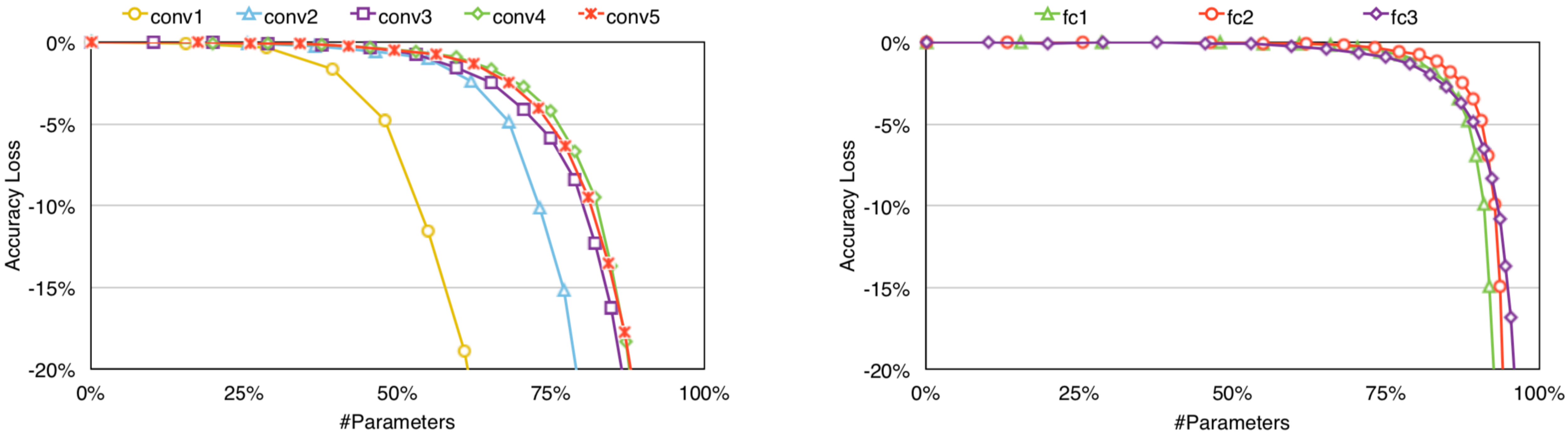


Figure 6: Pruning sensitivity for CONV layer (left) and FC layer (right) of AlexNet.

AMC: Automatic Model Compression

Actions:
sparsity for each layer

Actor Critic
DDPG

Agent

Environment

Channel
Pruning

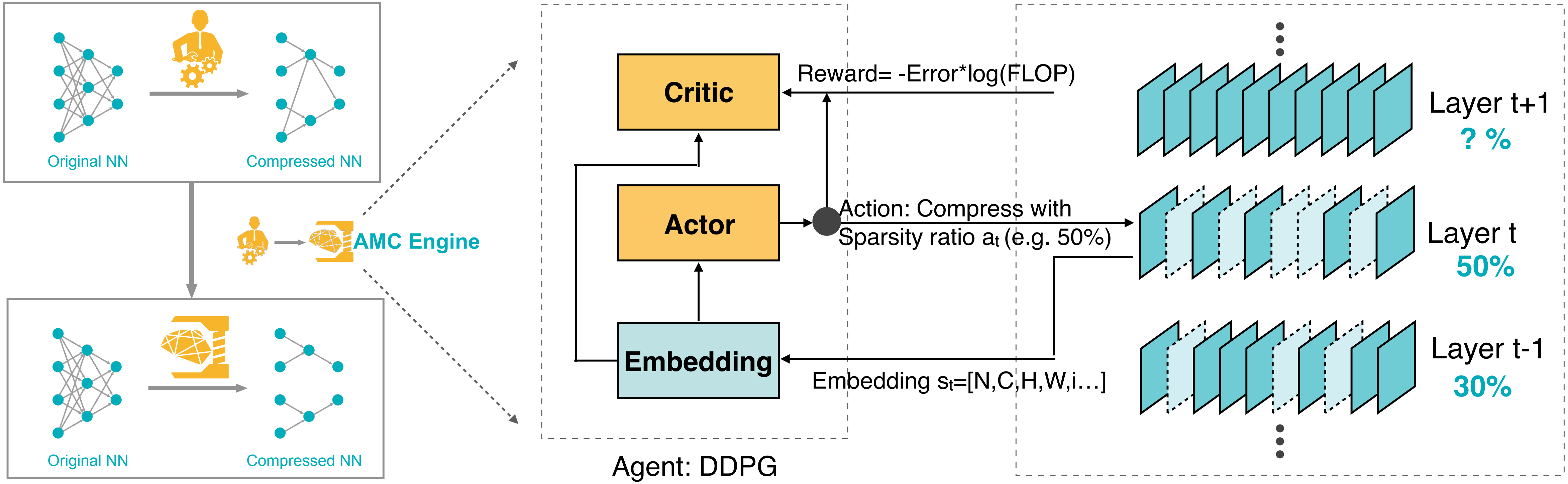
State / Reward:
N,C,H,W, accuracy, latency…

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

AMC: Automatic Model Compression

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

Model Compression by Human:
Labor Consuming, Sub-optimal



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

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

AMC: Automatic Model Compression

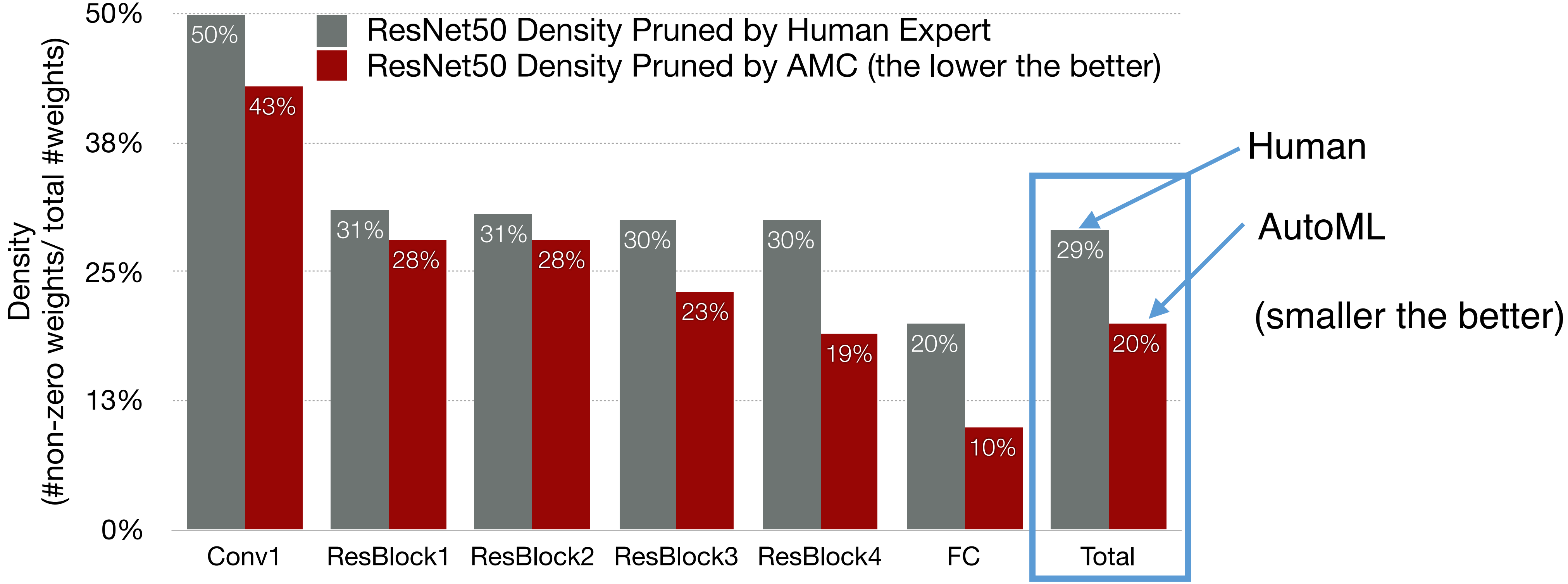


Figure 15: Our reinforcement learning agent (AMC) can prune the model to a lower density than achieved by human experts without loss of accuracy. (Human expert: 3.4× compression on ResNet50. AMC : 5× compression on ResNet50.)

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

AMC: Automatic Model Compression

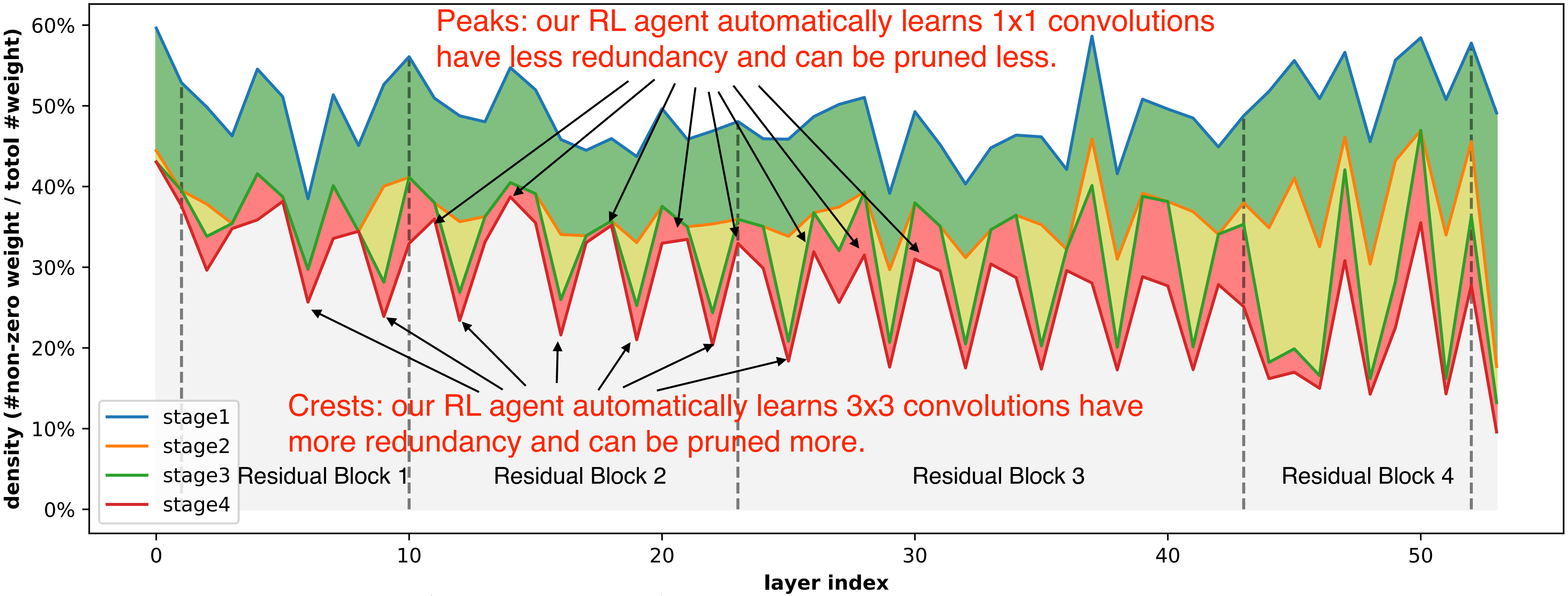
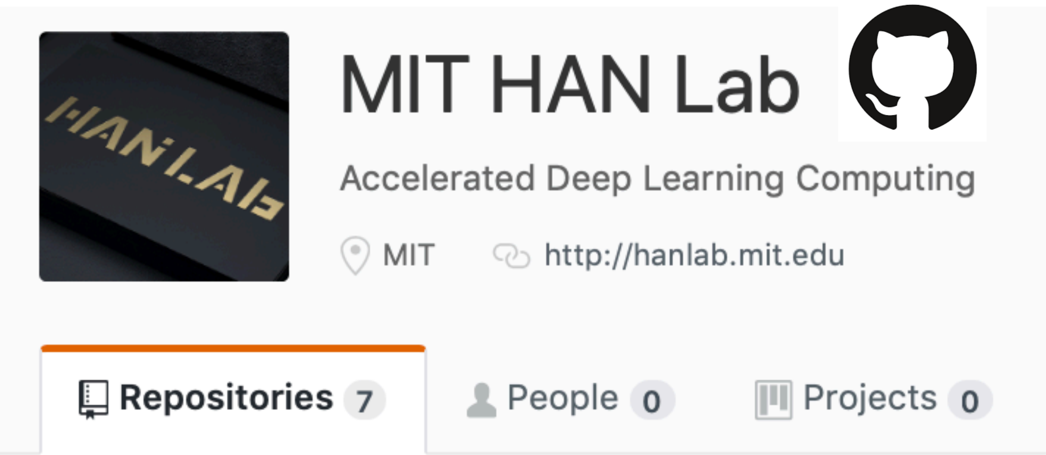


Figure 14: The pruning policy (sparsity ratio) given by our reinforcement learning agent for ResNet-50.

AMC : Accelerating MobileNet

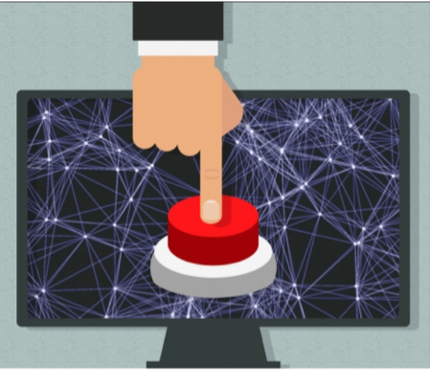
Model	MAC	Top-1	Top-5	Latency	Speed	Memory
1.0 MobileNet	569M	70.6%	89.5%	**119.0ms**	8.4 fps	20.1MB
AMC (50% MAC)	285M	70.5%	89.3%	**64.4ms**	15.5 fps (1.8x)	14.3MB
AMC (50% Time)	272M	70.2%	89.2%	**59.7ms**	16.8 fps (2.0x)	13.2MB
0.75 MobileNet	325M	68.4%	88.2%	**69.5ms**	14.4 fps (1.7x)	14.8MB

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

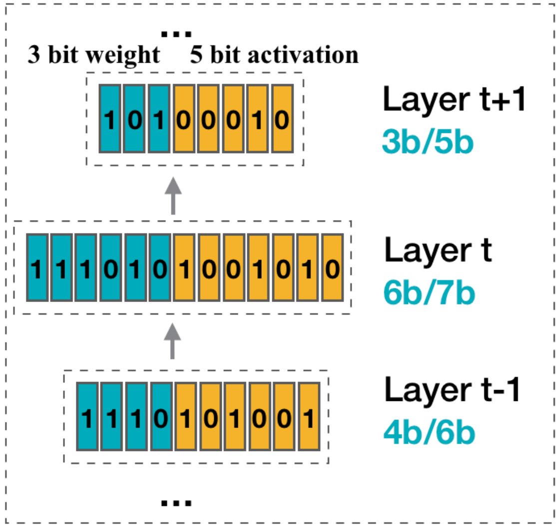


AMC is Available on Github

github.com/mit-han-lab/AMC

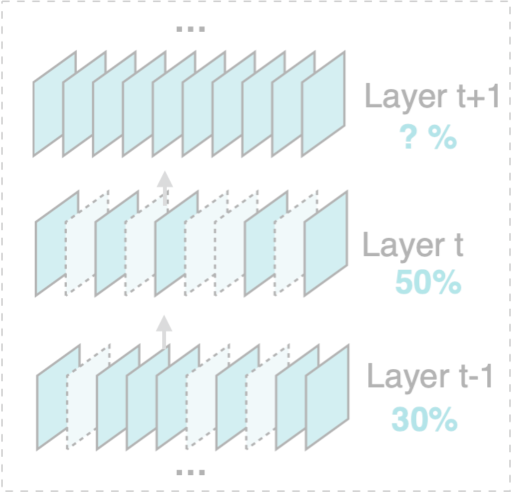


Design Automation for Efficient Deep Learning Computing



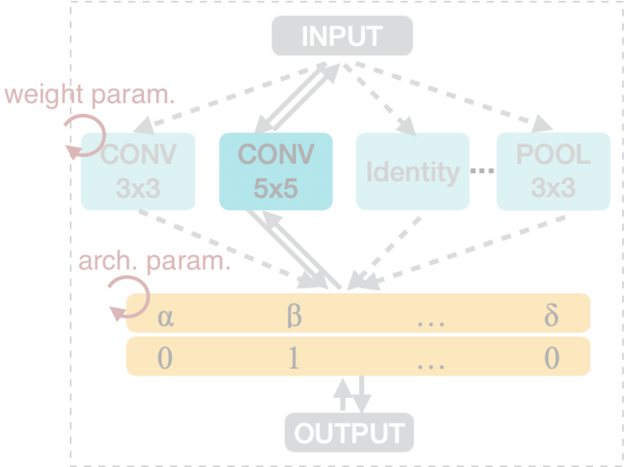
HAQ: Hardware-aware Automated Quantization

[CVPR 2019], oral



AMC: AutoML for Model Compression

[ECCV 2018]



Proxyless Neural Architecture Search

[ICLR 2019]

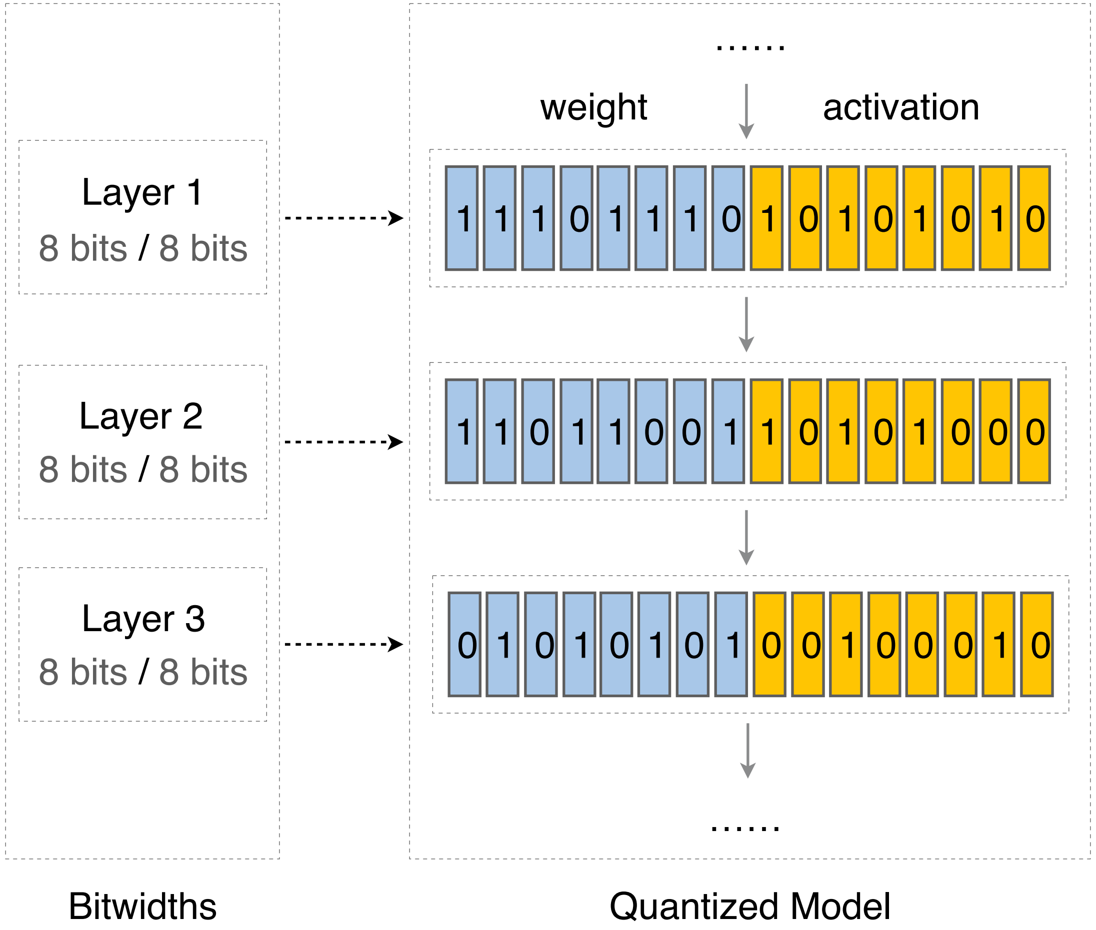
Hardware-aware Automated Quantization with Mixed Precision

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

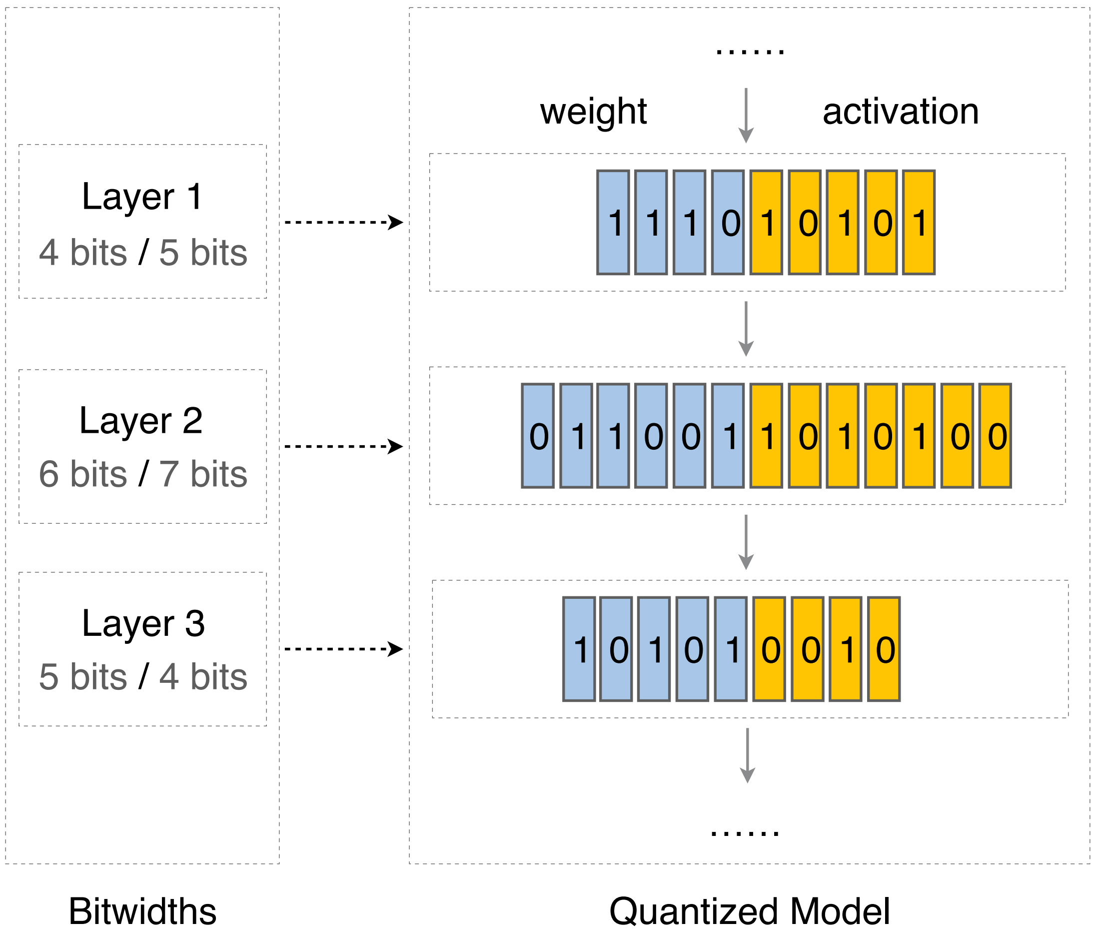
Massachusetts Institute of Technology

CVPR'19, oral

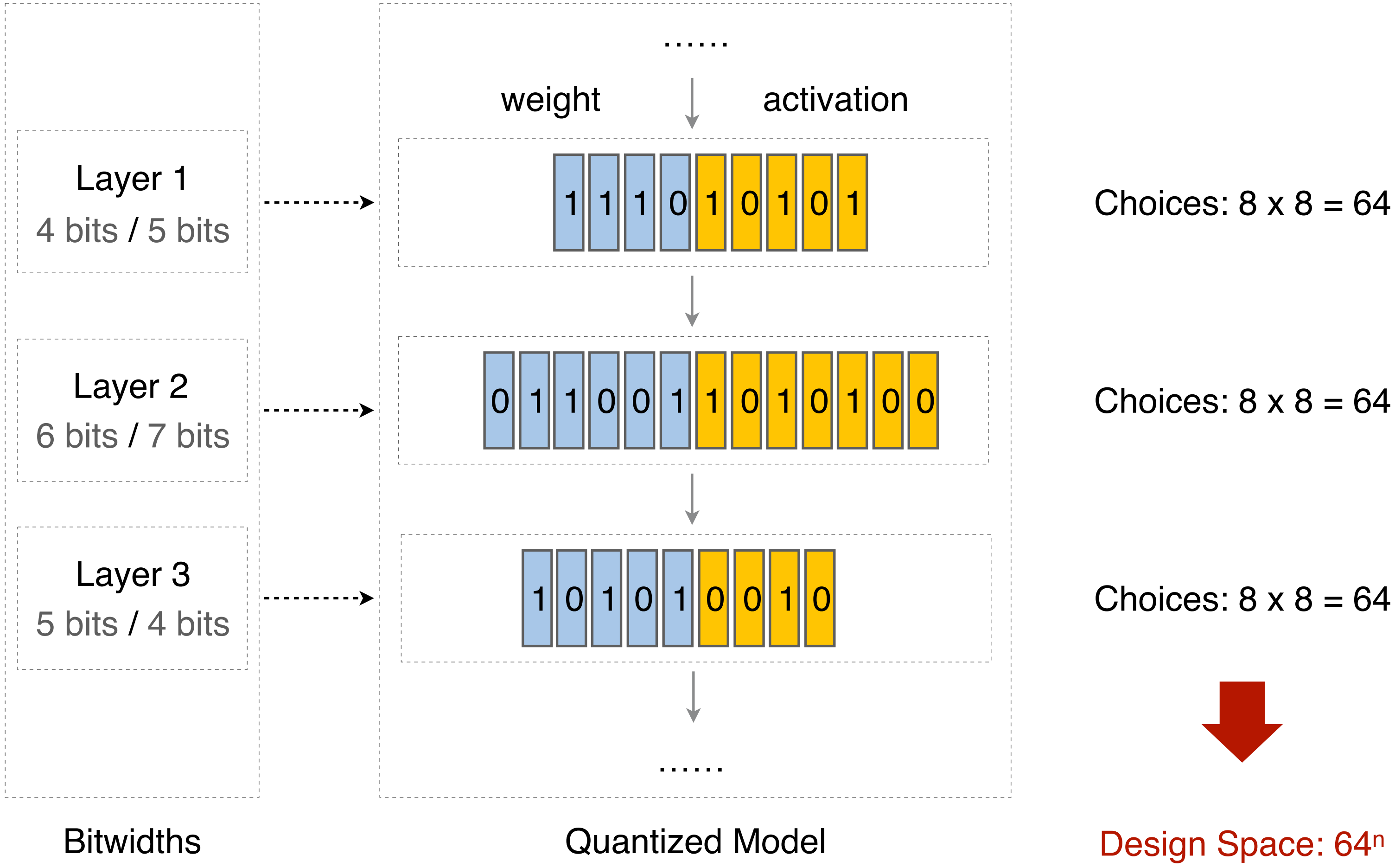
Fixed-Precision Quantization



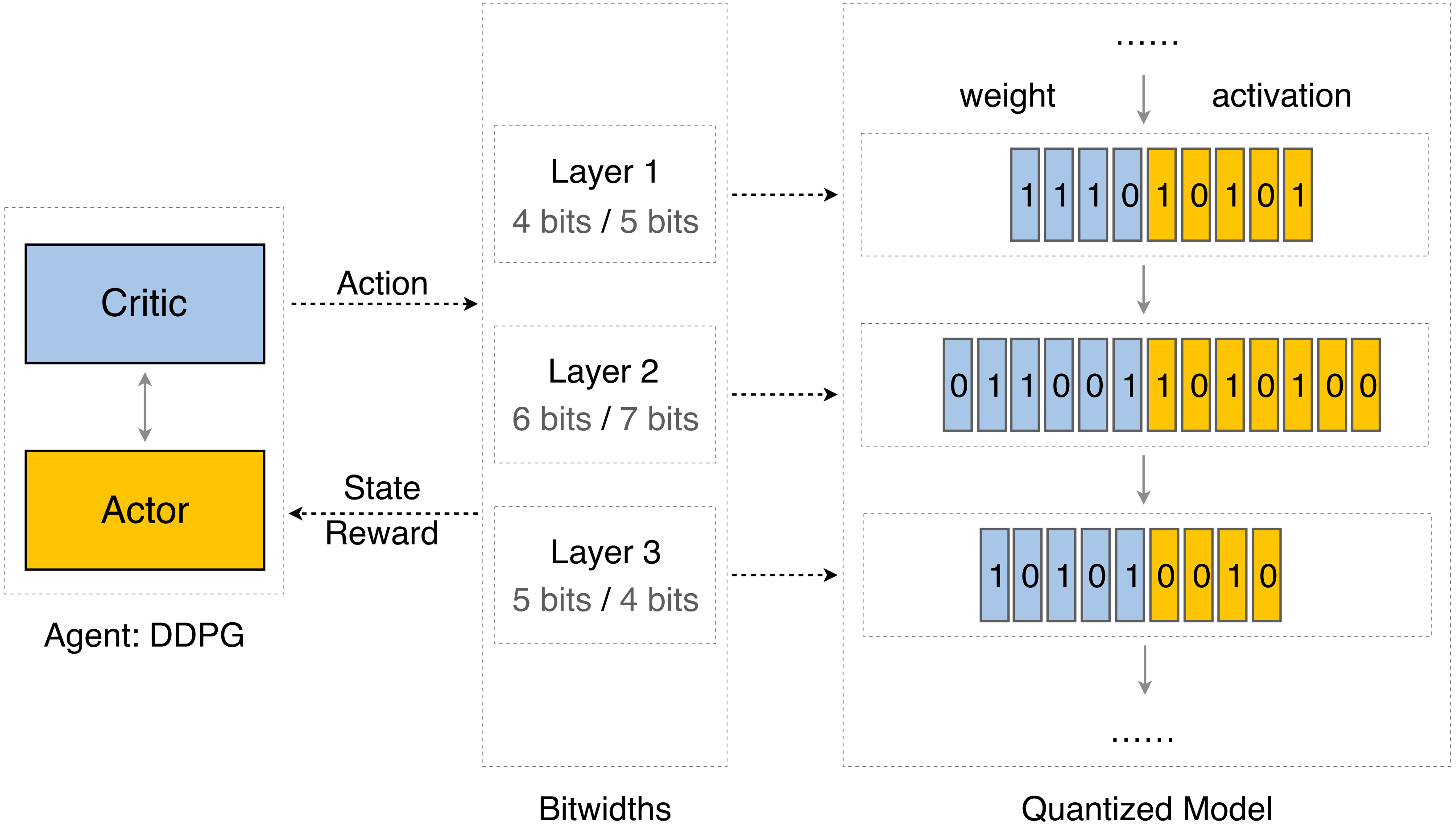
Contribution I: Mixed-Precision Quantization



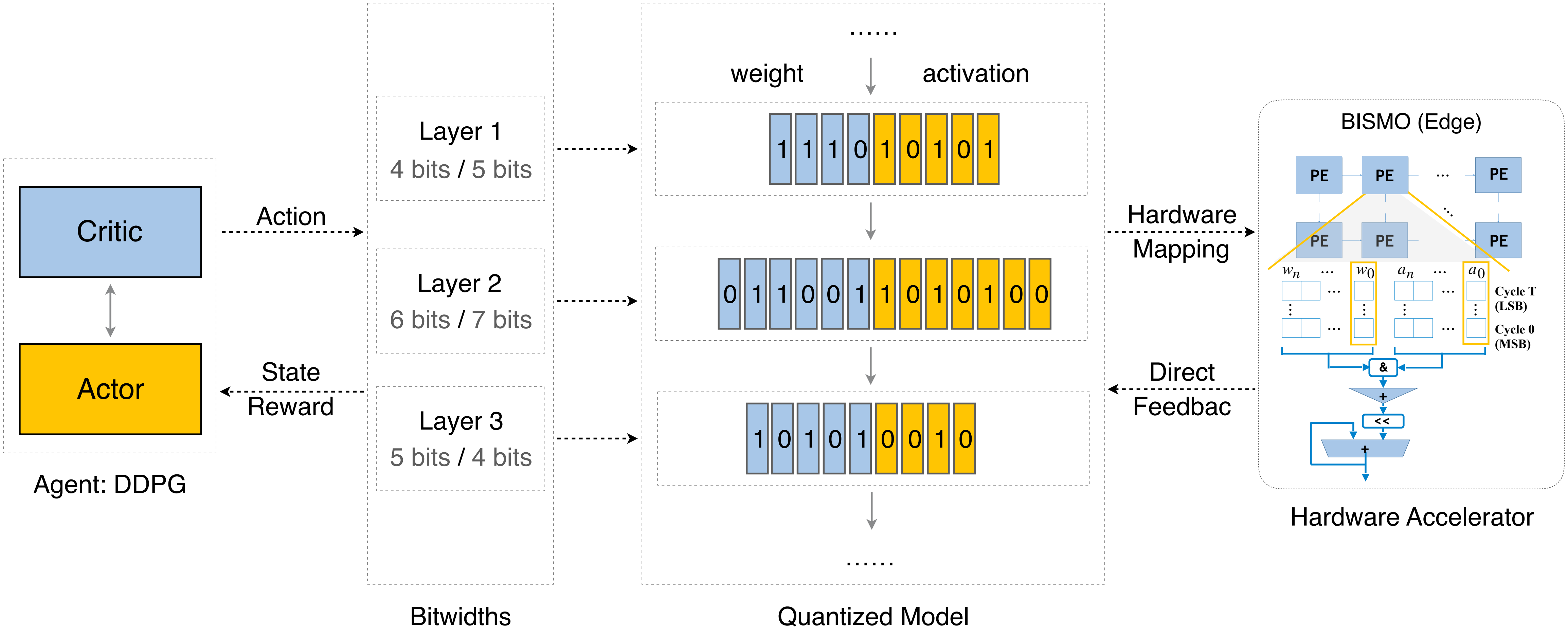
Contribution II: Design Automation



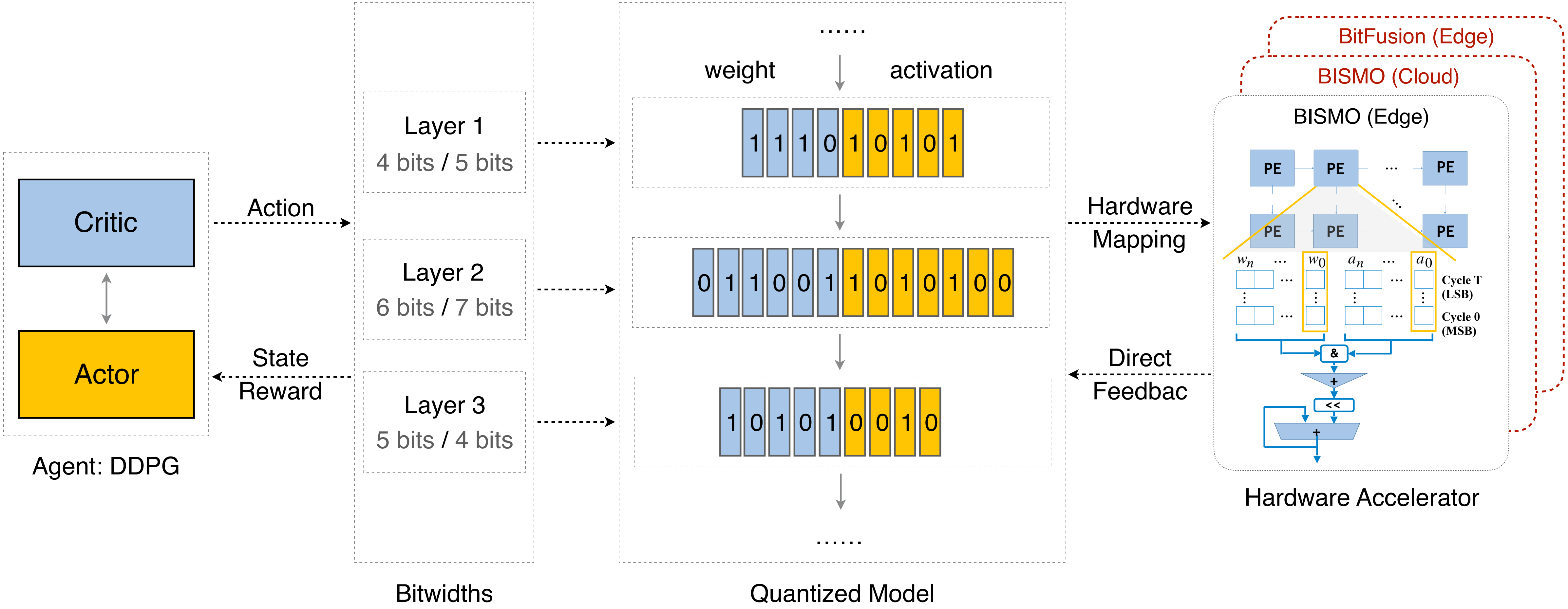
Contribution II: Design Automation



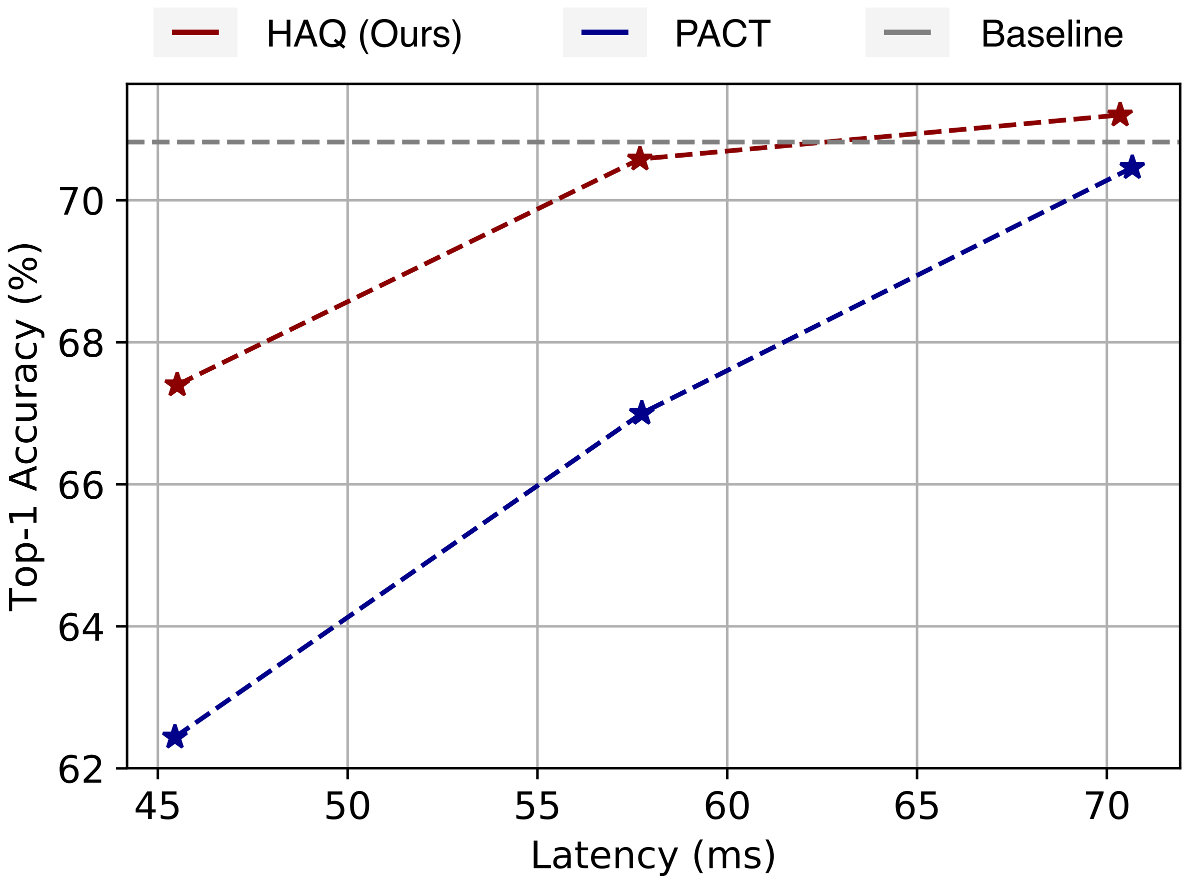
Contribution III: Hardware-Aware Specialization



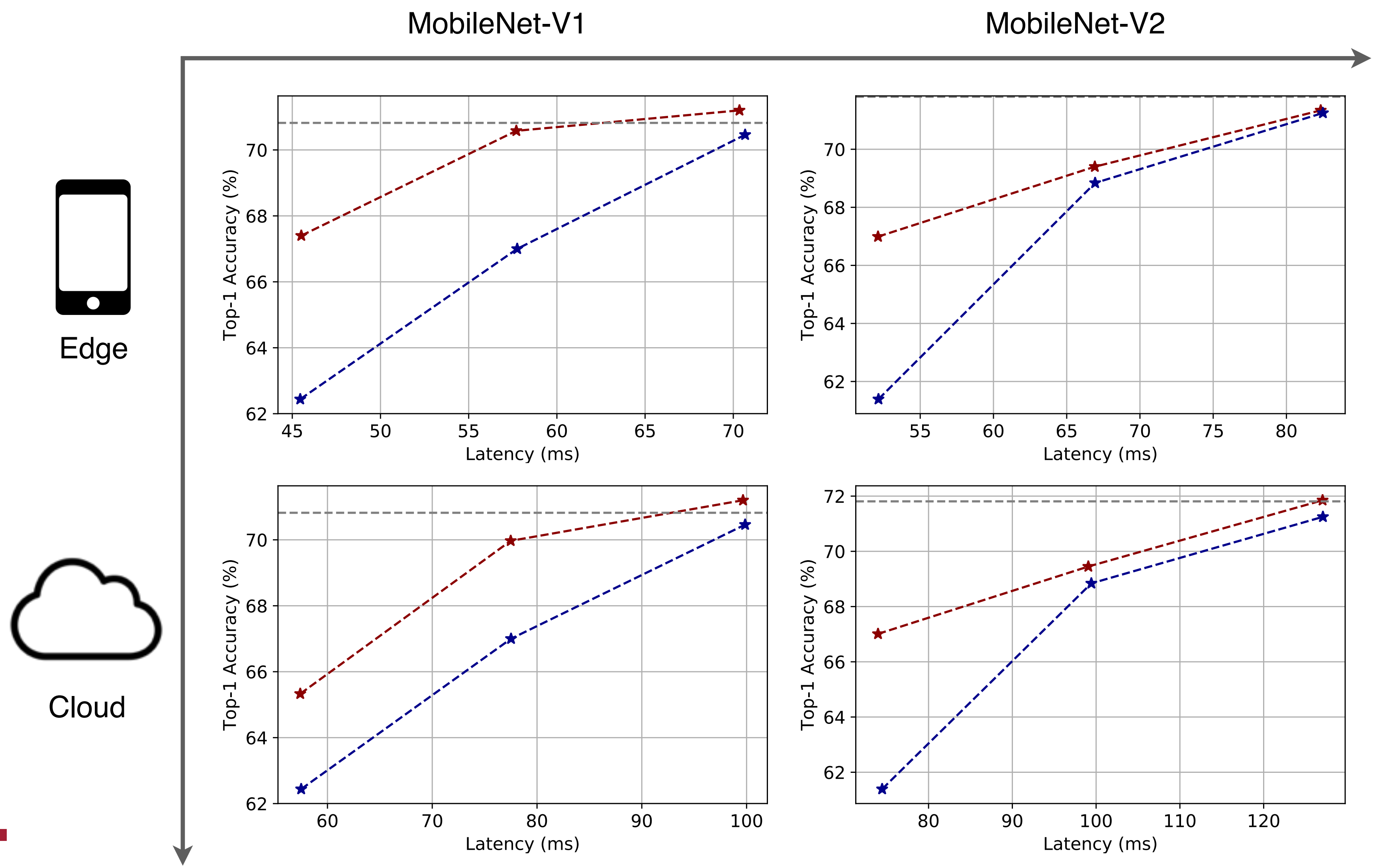
Contribution III: Hardware-Aware Specialization



Results: HAQ Outperforms Uniform Quantization



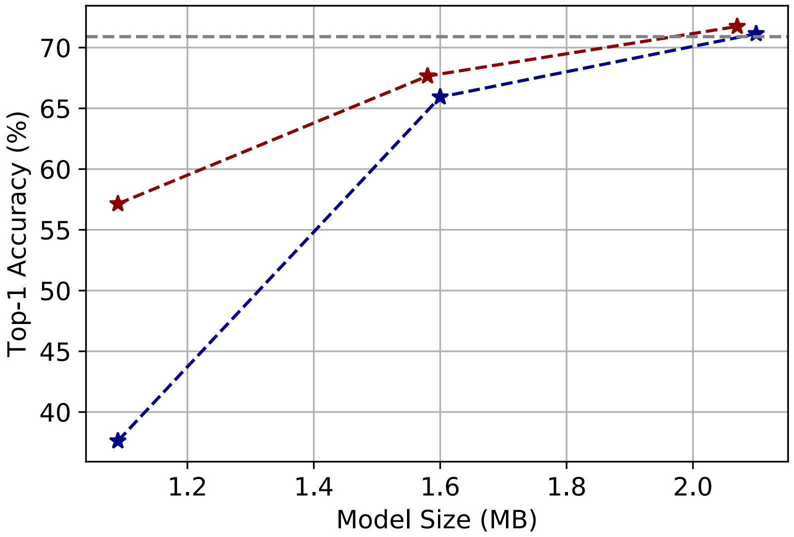
Results: Model and Hardware Specialization



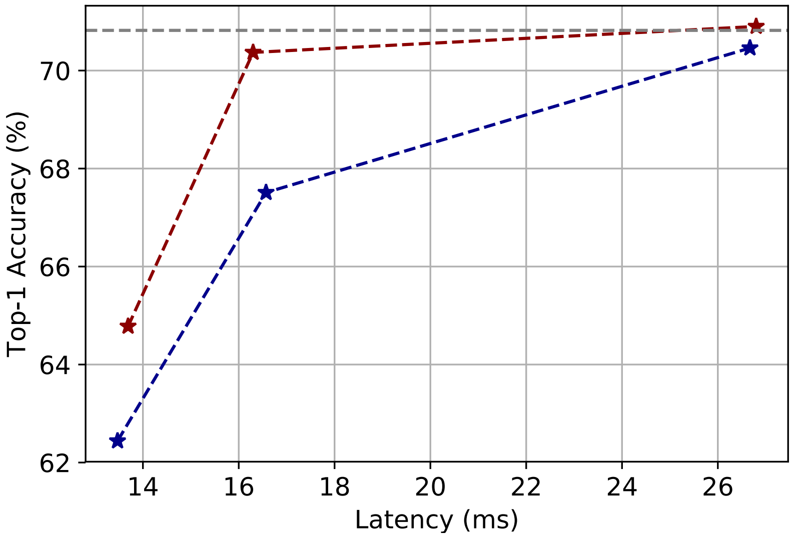
- Baseline (Full Precision)
- HAQ (Ours)
- PACT (Uniform Quantization)

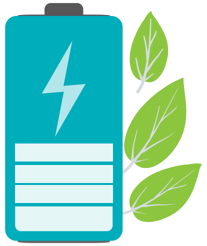
Results: HAQ Supports Multiple Objectives

Model Size Constrained

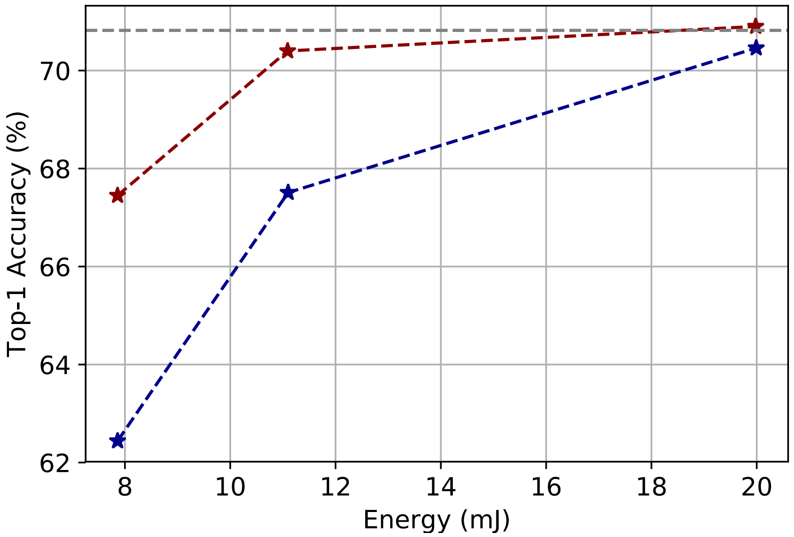


Latency Constrained

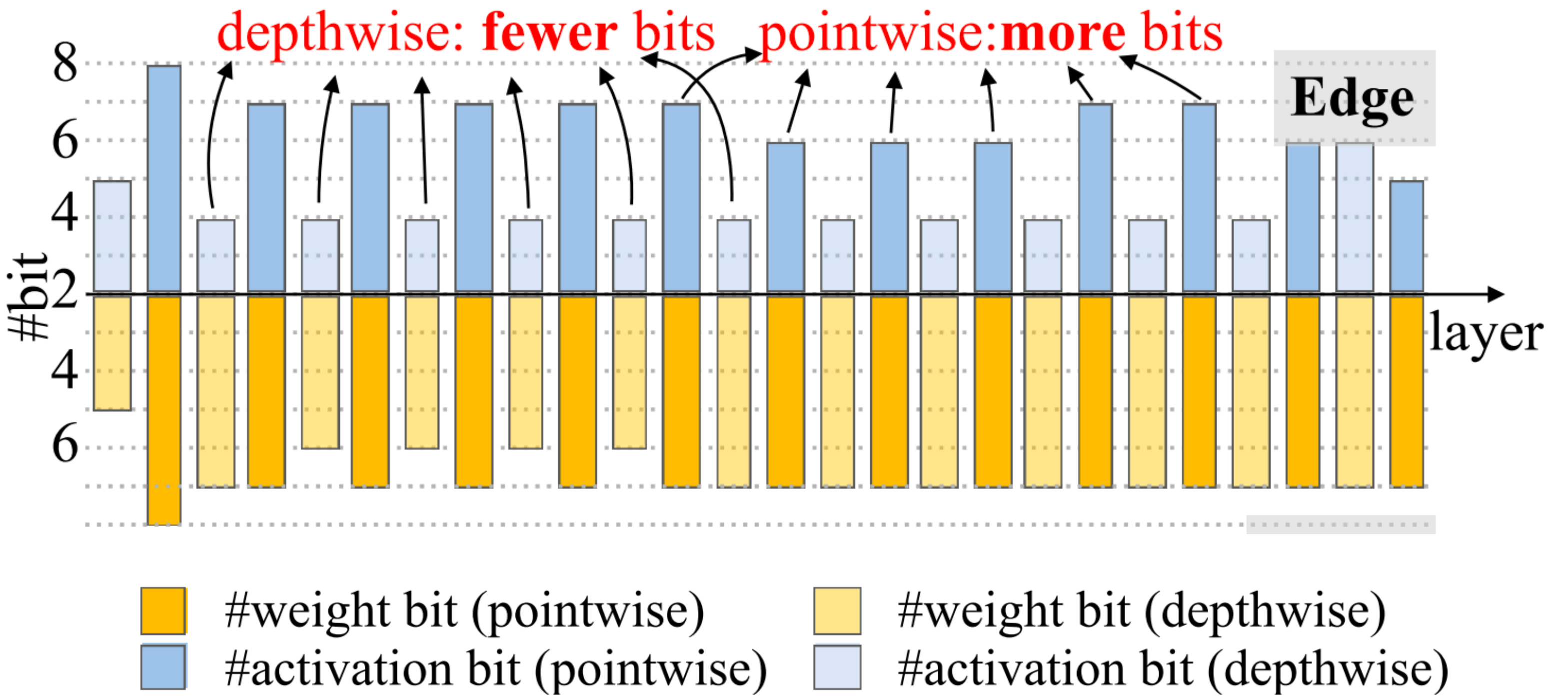




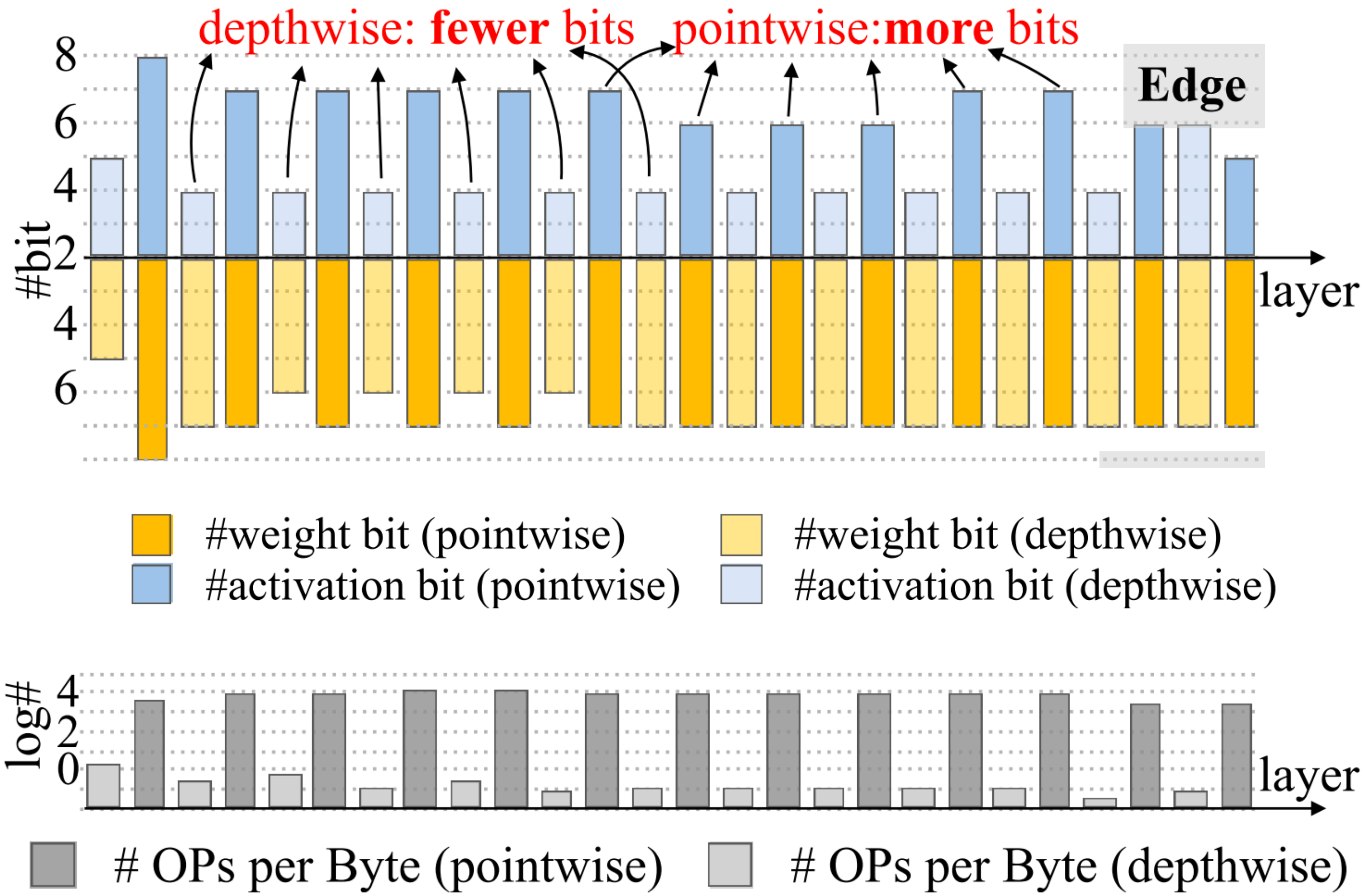
Energy Constrained



Interpreting the Quantize Policy on the Edge



Interpreting the Quantize Policy on the Edge



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

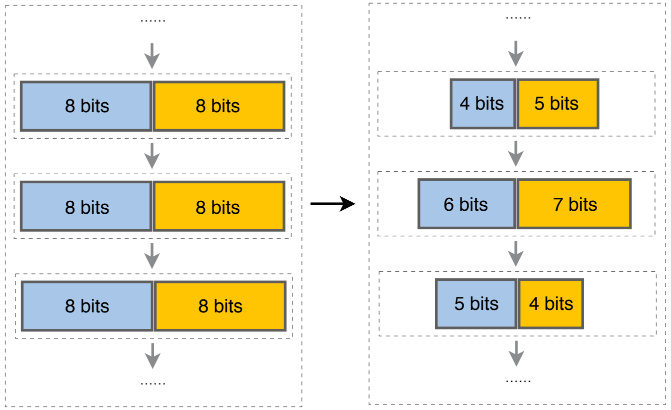
Good Transfer Ability

	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

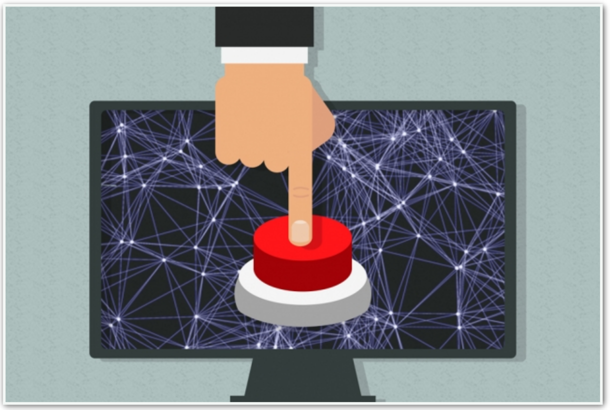
(Transfer the RL agent from MobileNet-v1 to MobileNet-v2)

Contributions

Mixed Precision



Design Automation



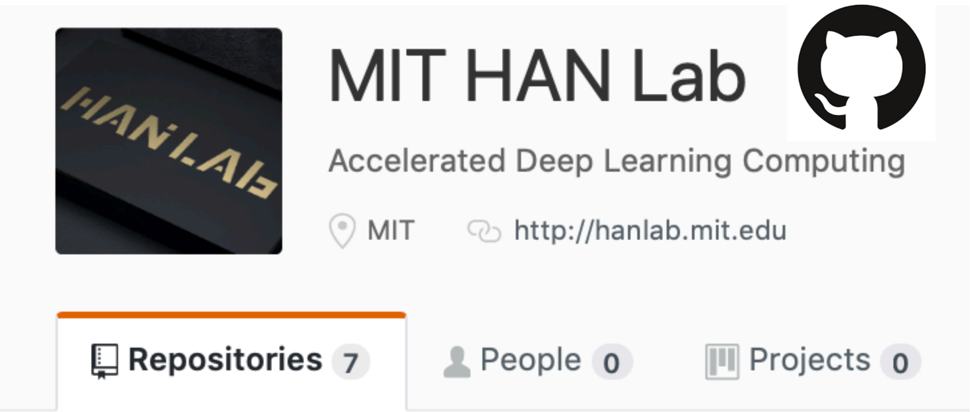
Hardware-Aware Specialization

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

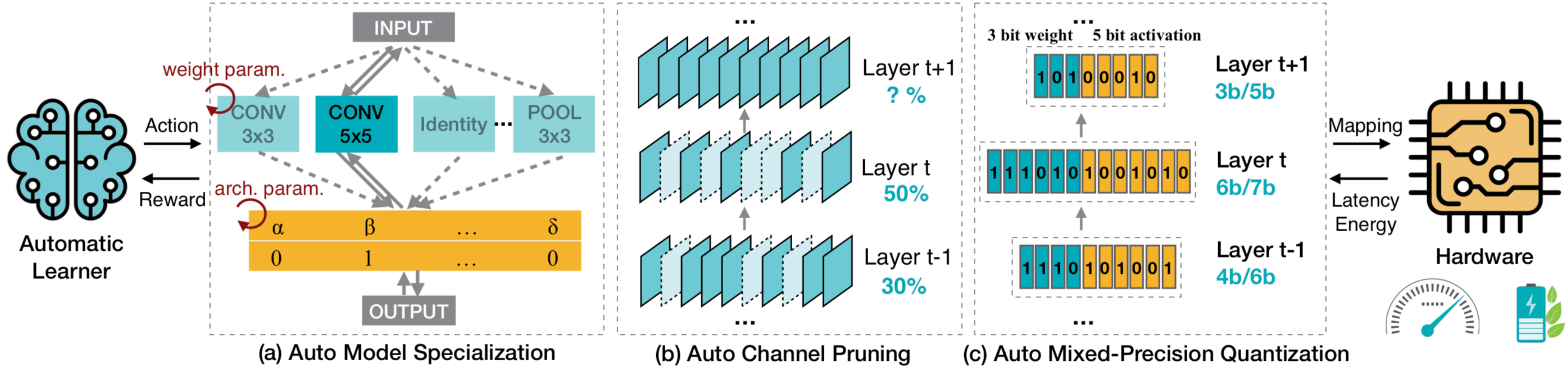


Thursday 9:24am, #199 @CVPR’19

HAQ is Available on Github

github.com/mit-han-lab/HAQ

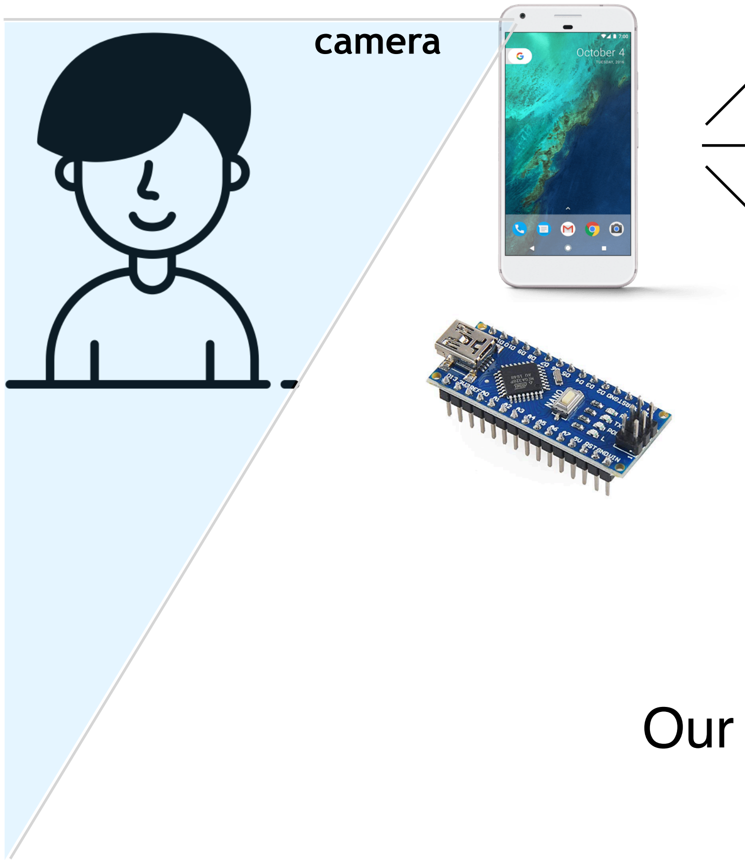
Summary



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

Put them together: ProxylessNAS => AMC => HAQ

1st place in CVPR'19 Visual Wake Words Challenge



Peak Memory Usage < 250KB

Model Size < 250KB

MAC < 60M

Efficient mobile CNN model

People? No People?
(Wake or Sleep)

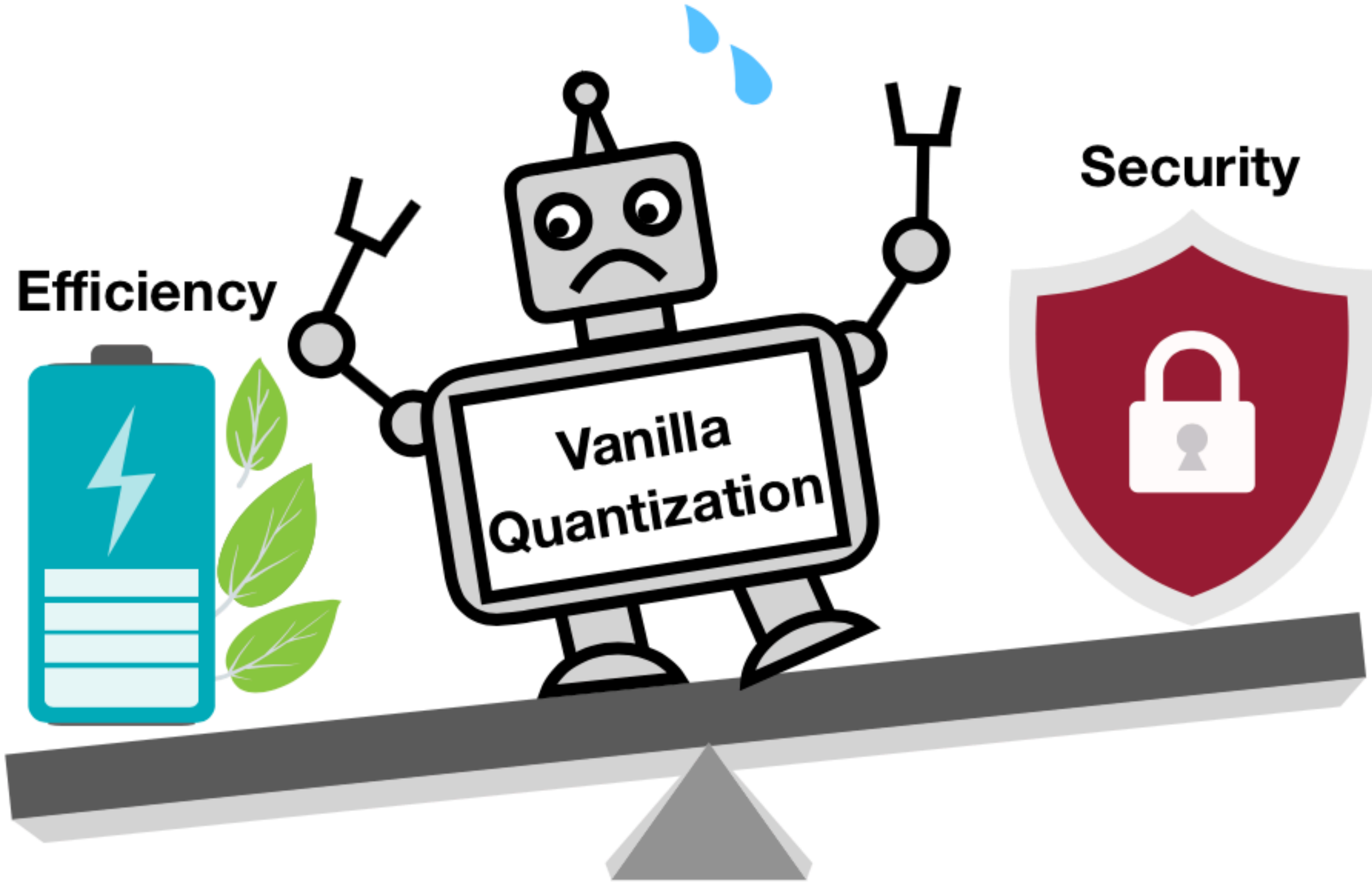
Our result:

Peak Memory: 245KB

Modle Size: 242KB

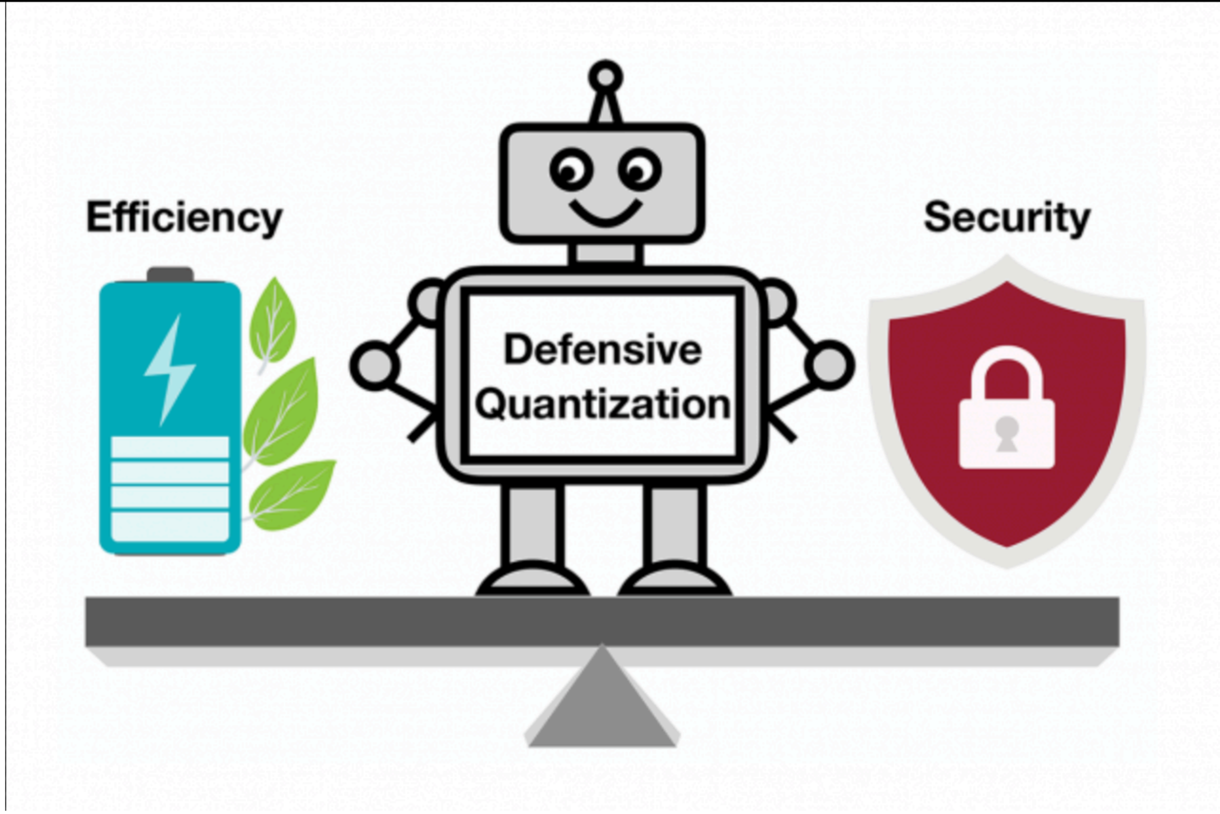
MAC: 50M

Top1: 94.575%


Efficiency
Vanilla Quantization
Security

MIT News

ON CAMPUS AND AROUND THE WORLD



FULL SCREEN

Image: Ji Lin

Improving security as artificial intelligence moves to smartphones

Researchers unveil a tool for making compressed deep learning models less vulnerable to attack.

Defensive Quantization
When Efficiency Meets Robustness

Ji Lin[1], Chuang Gan[2], Song Han[1]

[1] Massachusetts Institute of Technology
[2] MIT-IBM Watson AI Lab

ICLR'19

Problem Overview

- **Efficiency:** Deep neural nets deployment is hard due to limited resource. **Quantization** can reduce the computation needed for deep neural networks.

- **Robustness:** Deep neural nets are vulnerable to **adversarial attack**, leading to potential security issue.

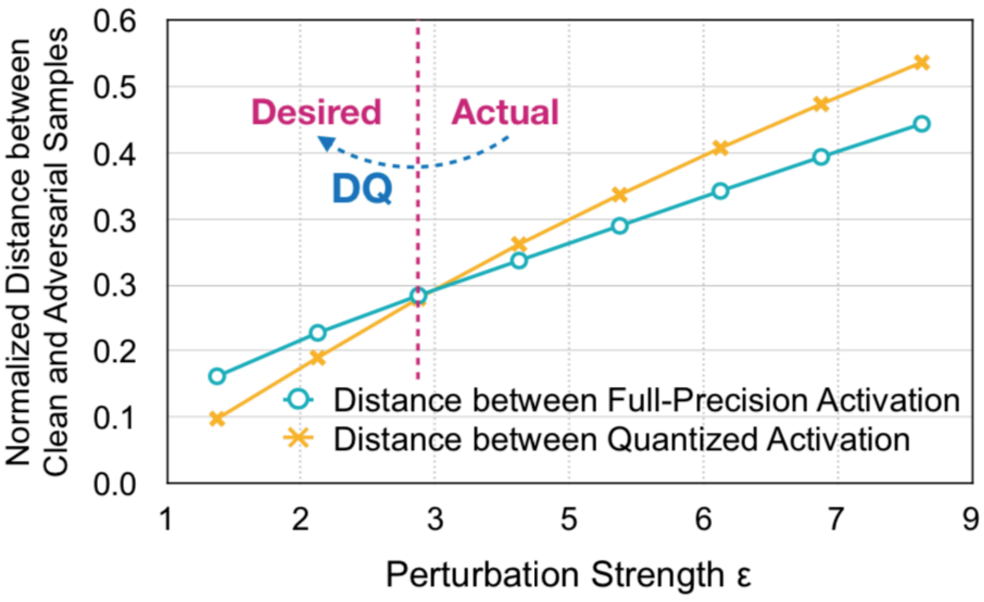
* Han *et al.* Deep Compression
* Eykholt *et al.* Robust Physical-World Attacks on Deep Learning Visual Classification

Quantized Model is not Robust

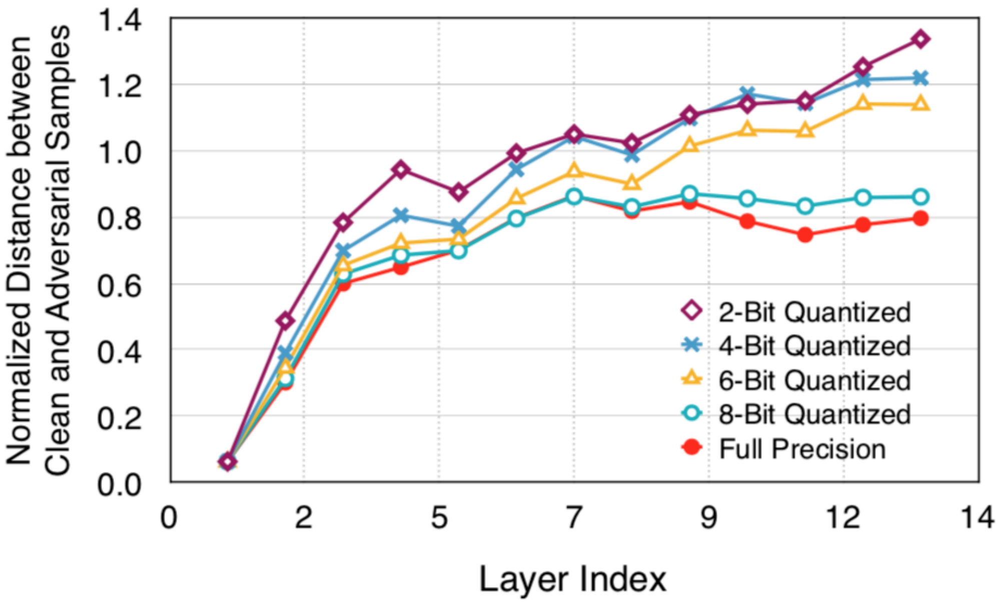
Compressed 4-bit Model: Deer
50% certainty

Why?

- **Error Amplification Effect!**
- Quantization helps robustness when noise is small; it hurts robustness when noise is amplified



(a) Noise increases with perturbation strength. Quantization makes the slope deeper.



(b) With conventional quantization, noise increases with layer index (the amplification effect).

Defensive Quantization (DQ)

- **Main Idea**: suppress noise amplification so that quantization helps robustness

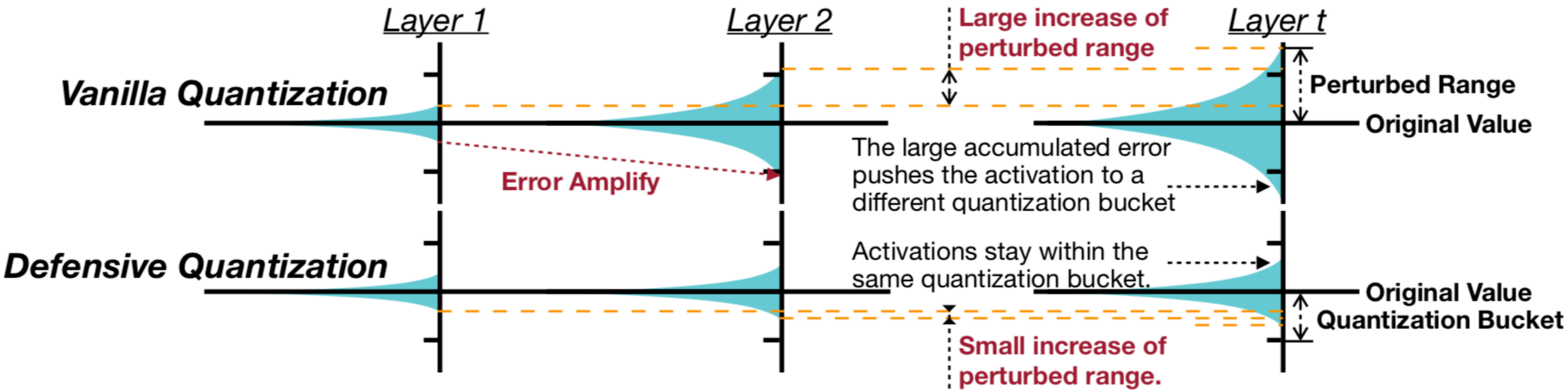
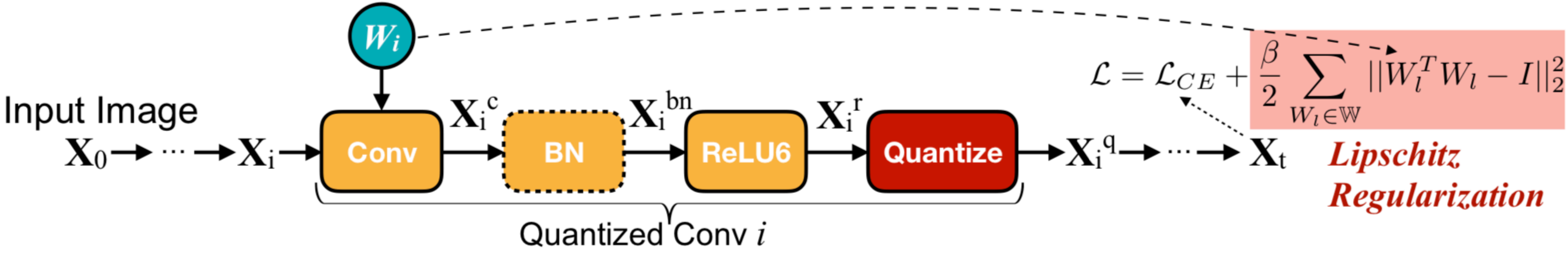


Figure 4. The error amplification effect prevents activation quantization from defending adversarial attacks.

Defensive Quantization: When Efficiency Meets Robustness, ICLR'19
https://arxiv.org/abs/1904.08444

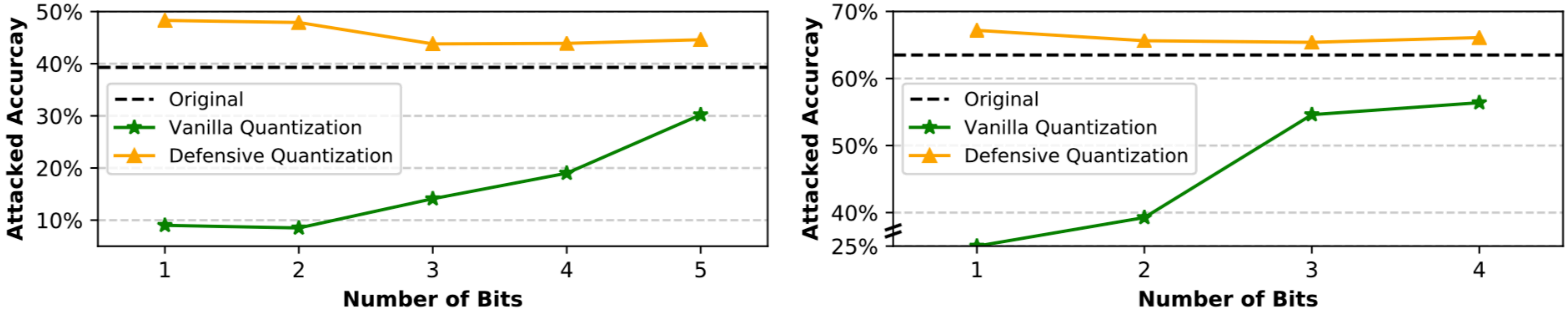
- Lipschitz constant describes: when input changes, how much does the output change correspondingly. $\frac{|f(y)-f(x)|}{|y-x|} \leq k$
- to keep the Lipschitz constant of the whole network small, we need to keep the Lipschitz constant of each layer Lip(ϕi) ≤ 1
- The Lipschitz constant is by definition the maximum singular value of W
- However, computing the singular values of each weight matrix is not computationally feasible during training
- Luckily, if we can keep the weight matrix row orthogonal, the singular values are by nature equal to 1
- Therefore we transform the problem of keeping ρ(W) ≤ 1 into keeping $W^TW \approx I$

Solution: Add Lipschitz Regularization



- Controlling ***Lipschitz constant*** during quantization, to suppress the noise amplification
- We introduce a regularization term $||W^T W - I||$
- For convolutional layers with weight $W \in R^{cout \times cin \times k \times k}$, we can view it as a two-dimension matrix of shape $W \in R^{cout \times (cin \times k \times k)}$ and apply the same regularization

DQ Fixes Robustness Decrease (And Further Improves It)



(a) White-Box Robustness ($\epsilon = 8$) **(b)** Black-Box Robustness ($\epsilon = 8$)

Defensive Quantization: When Efficiency Meets Robustness, ICLR'19
https://arxiv.org/abs/1904.08444

Defensive Quantization is both Efficient and Robust

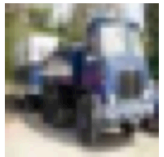
Compressed 4-bit Model: Deer
50% certainty

With Defensive Quantization: Truck
100% certainty

Defensive Quantization: When Efficiency Meets Robustness, ICLR'19
https://arxiv.org/abs/1904.08444

Defensive Quantization is both Efficient and Robust

2. **GT:** truck

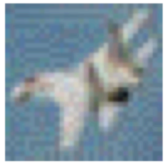


FP: truck (1.00) ✓
VQ: truck (1.00) ✓
DQ: truck (1.00) ✓

FP: truck (1.00) ✓
VQ: deer (0.50) ✗
DQ: truck (1.00) ✓

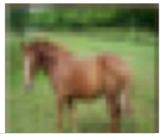
3. **GT:** plane

FP: plane (1.00) ✓
VQ: plane (1.00) ✓
DQ: plane (1.00) ✓



FP: deer (1.00) ✗
VQ: cat (0.99) ✗
DQ: deer (0.51) ✗

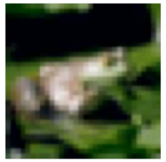
4. **GT:** horse



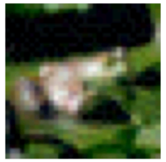
FP: horse (1.00) ✓
VQ: horse (1.00) ✓
DQ: horse (1.00) ✓

FP: horse (0.97) ✓
VQ: deer (0.97) ✗
DQ: horse (0.99) ✓

5. **GT:** frog



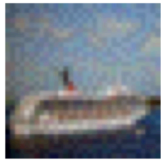
FP: frog (1.00) ✓
VQ: frog (1.00) ✓
DQ: frog (1.00) ✓



FP: car (0.95) ✗
VQ: deer (0.45) ✗
DQ: frog (0.73) ✓

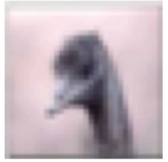
6. **GT:** ship

FP: ship (1.00) ✓
VQ: ship (1.00) ✓
DQ: ship (1.00) ✓



FP: ship (0.96) ✓
VQ: frog (0.57) ✗
DQ: ship (1.00) ✓

7. **GT:** bird



FP: bird (1.00) ✓
VQ: bird (1.00) ✓
DQ: bird (1.00) ✓

FP: horse (0.32) ✗
VQ: cat (0.87) ✗
DQ: bird (0.99) ✓

Take Home

- Aim to raise people's awareness about the security of the quantized and deployed neural networks

- Pave a possible direction to bridge two important areas in deep learning: <u>Efficiency</u> and <u>Robustness</u>

- Design a novel **Defensive Quantization (DQ)** module to defend adversarial attacks while maintain the efficiency.

Thank you!

Hardware, AI and Neural-nets

References

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

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

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

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

Code available at: github.com/mit-han-lab