arm AI

AI Virtual Tech Talks Series

arm

Optimizing Power and Performance for Machine Learning at the Edge : Model Deployment Overview

Arm

Lingchuan Meng, Principal Engineer, Arm
Naveen Suda, Principal Engineer, Arm

October 20, 2020

AI Virtual Tech Talks Series

Date	Title	Host
October 20, 2020	Optimizing Power and Performance For Machine Learning at the Edge - Model Deployment Overview	Arm
November 3, 2020	Small is big: Making Deep Neural Nets faster, smaller and energy-efficient on low power hardware	DeepLite
November 17, 2020	The Smart City in Motion - AI in intelligent transportation	Clever Devices, NXP Semiconductor, Arcturus

Visit: developer.arm.com/solutions/machine-learning-on-arm/ai-virtual-tech-talks

Presenters

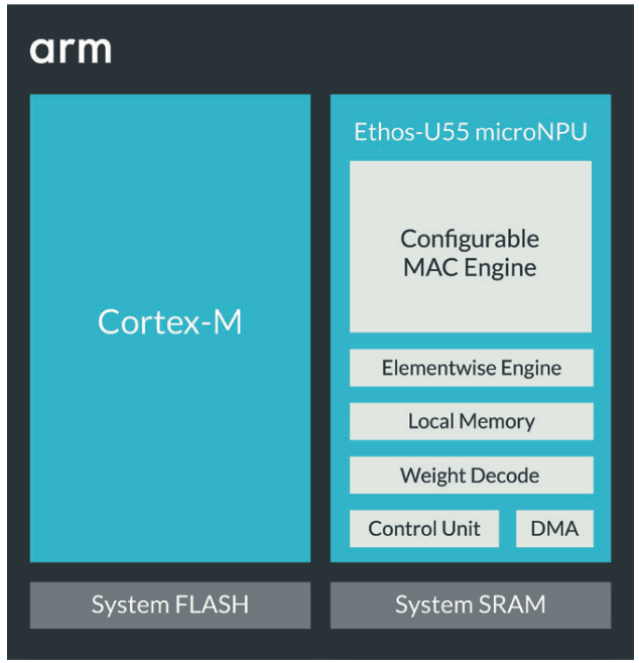
Lingchuan Meng, Principal Engineer, Arm

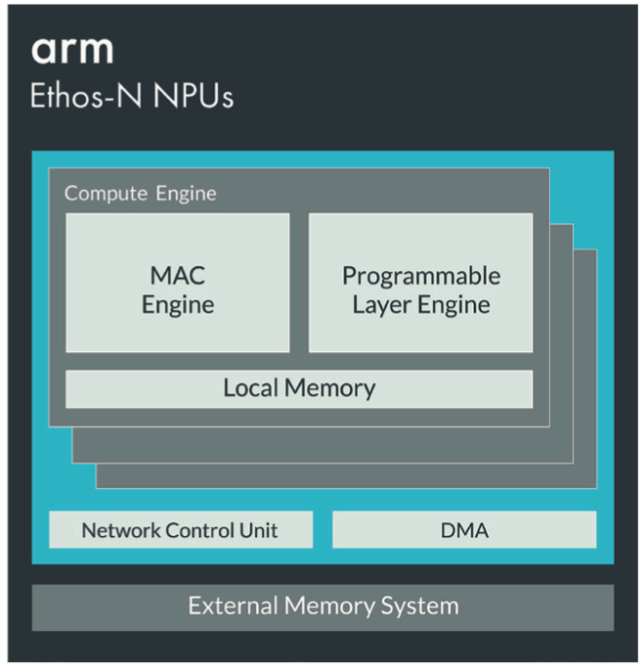
Naveen Suda, Principal Engineer, Arm

ML on the Edge - Challenges

- Edge device constraints for deploying ML algorithms
  - Limited memory
    - Flash (32 kB - few MB)
    - SRAM (16 kB - 512 kB)
  - Limited compute capability (100 MHz - 1 GHz)

- Hardware/software features
  - Compression HW: pruning, clustering, etc.
  - Mixed precision: 8-bit, 16-bit, etc.
  - Algorithmic: Winograd, etc.
  - Layer fusion: conv-add-pool-relu, etc.

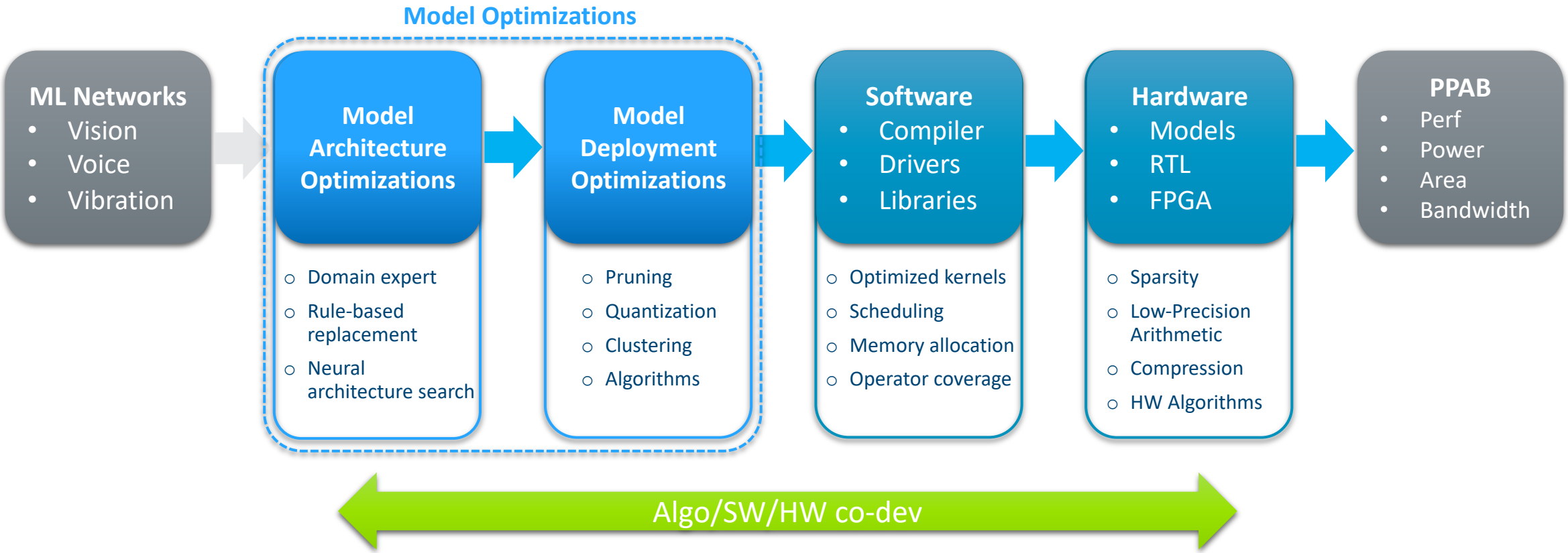




ML solutions = Model Optimization → Software → Hardware

arm

End-to-end Technology Exploration

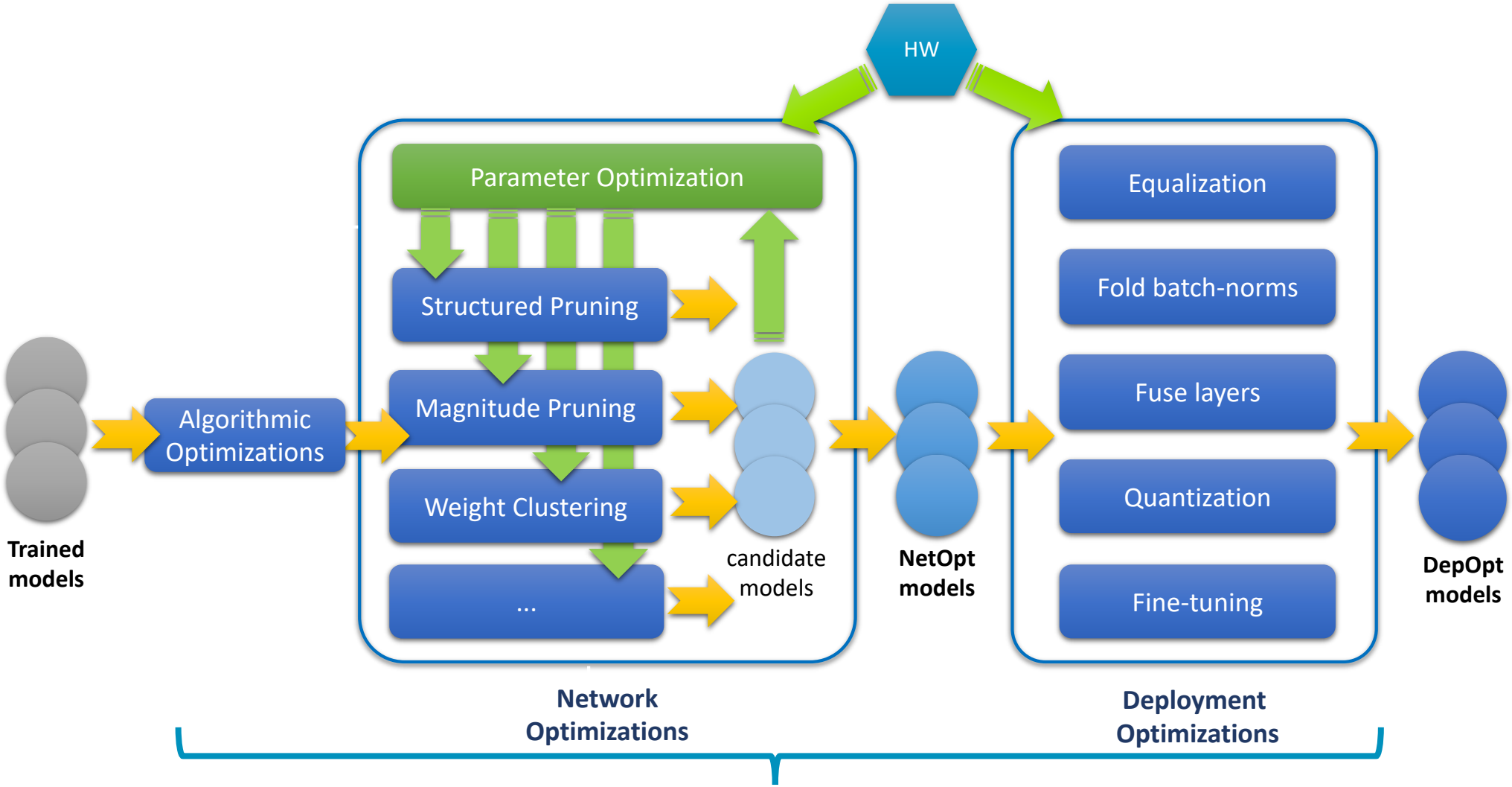


arm

arm

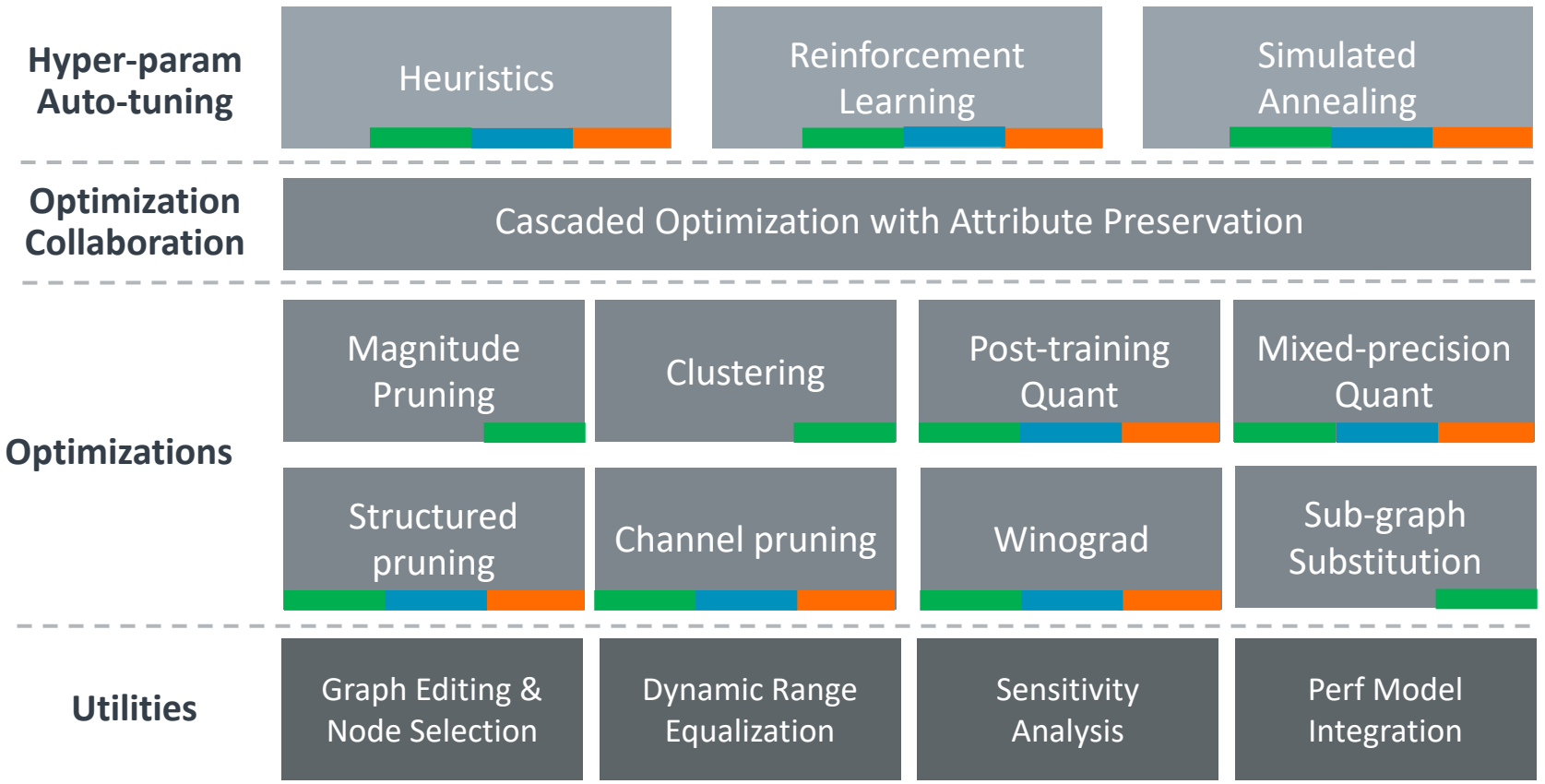
Model Deployment Optimizations

Deployment Optimization Flow



arm

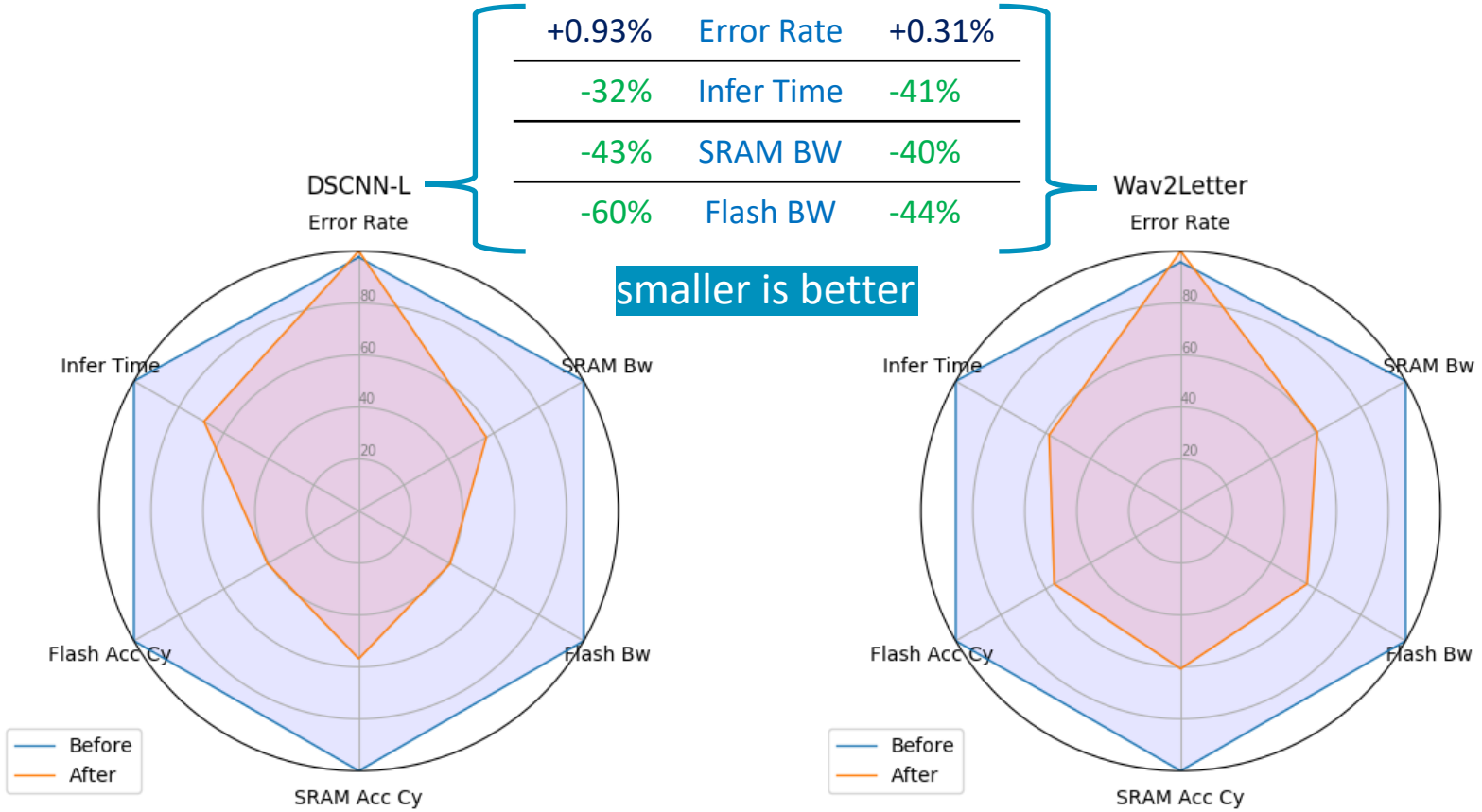
Overview of Technologies



Optimization Results

Model	Sparsity	Accuracy Δ
Inception V3	50%	+0.1%
ResNet 50	50%	+0.7%
VGG 16	50%	+1.6%
MobileNet V1	50%	-0.9%
Wav2Letter	50%	-1.34%
DS-CNN Large	80%	-0.5%

unstructured pruning



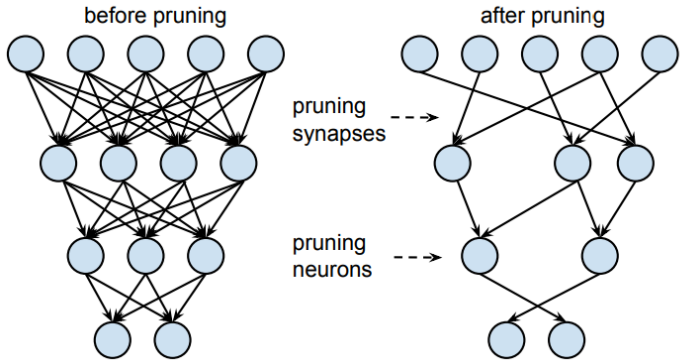
Keyword Spotting

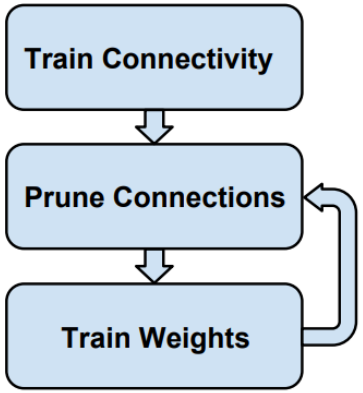
Speech Recognition

Lingchuan Meng, et al. "*Neural Network Optimizations for On-Device AI*" Embedded World Conference (2020).

Overview of Pruning Techniques

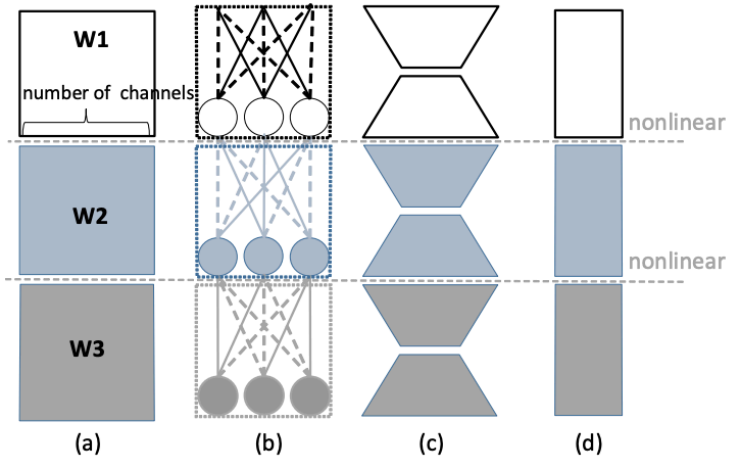
Magnitude Pruning

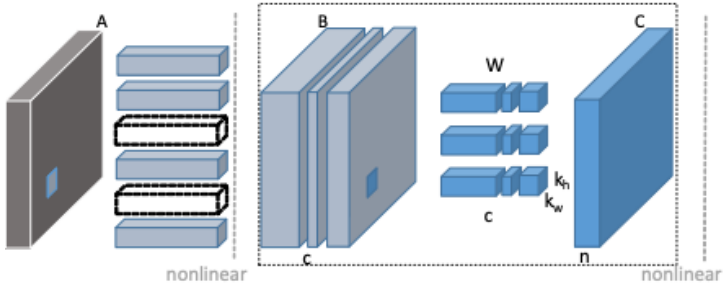




Song Han, et al. "*Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*" arXiv: 1510.00149 (2015).

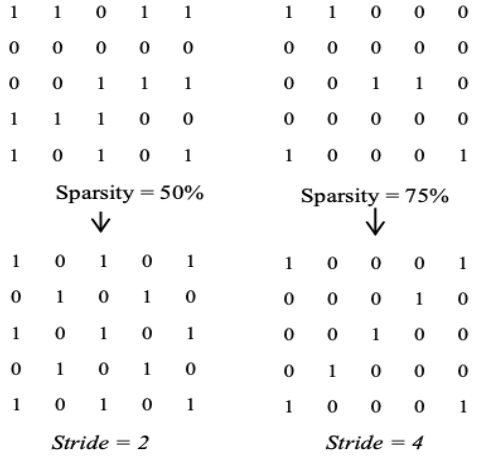
Channel Pruning

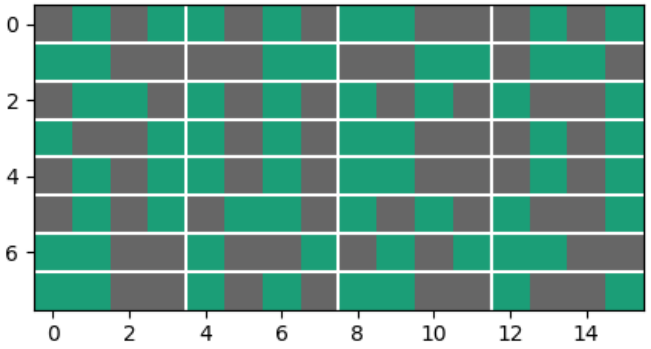




Yihui He, et al. "*Channel Pruning for Accelerating Very Deep Neural Networks*" arXiv: 1707.06168 (2017).

Structured Pruning

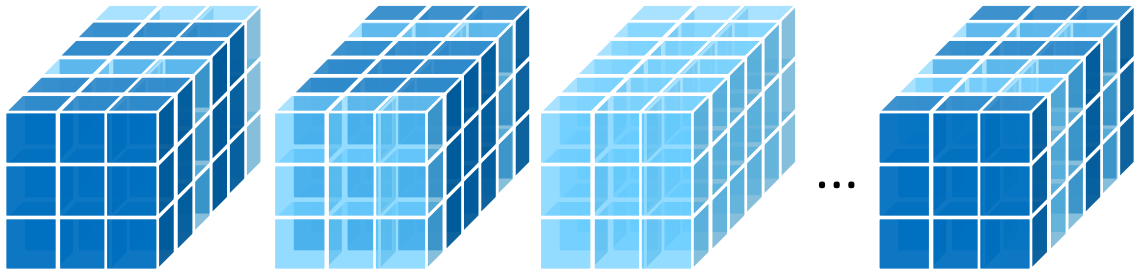




Sajid Anwar, et al. "*Structured Pruning of Deep Convolutional Neural Networks*" arXiv: 1512.08571 (2015).
Jeff Pool, *"Accelerating Sparsity in the Nvidia Ampere Architecture"* GTC 2020

Pruning

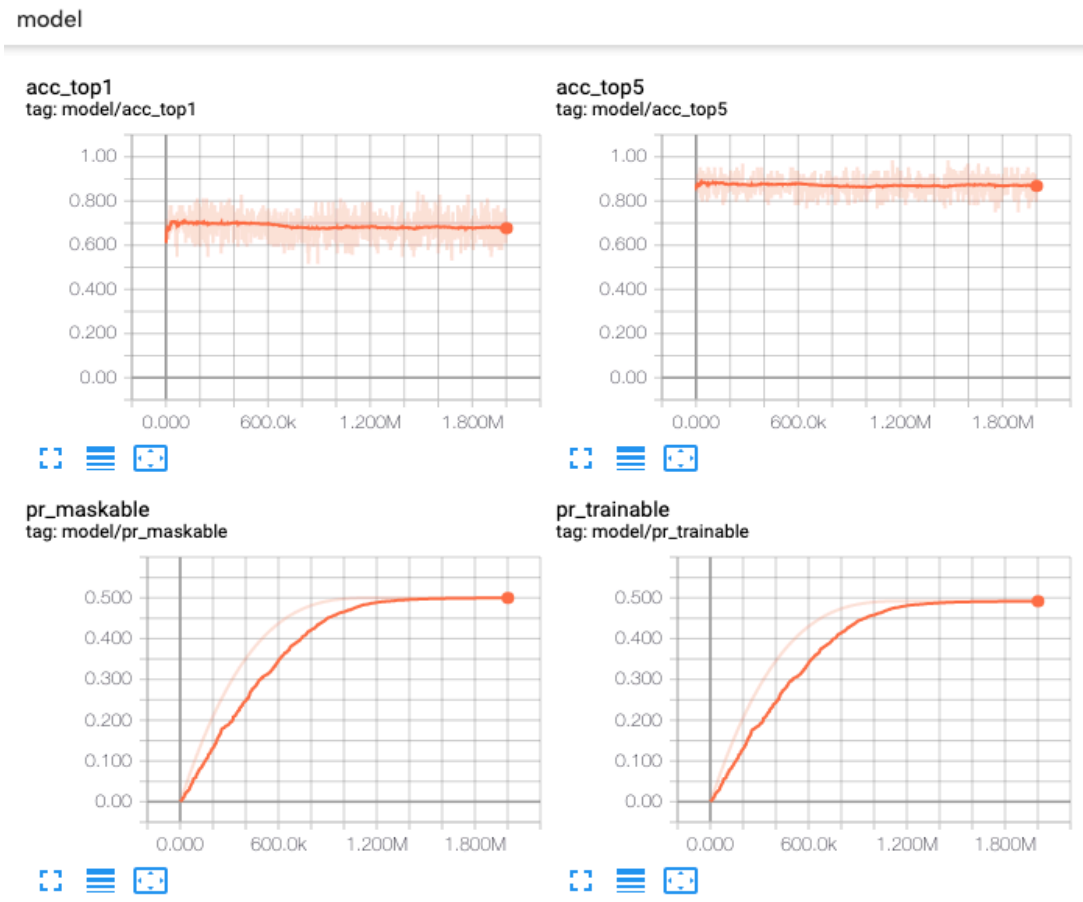
- Inducing sparsity to overly-parametrized models
  - Improve model compression and computation efficiency
- Structure
  - Unstructured: irregular locations of zeros
  - Structured: pre-defined patterns of zeros
- Spatial granularity
  - Layer / filter / kernel / weight
  - Compressibility vs. acceleration
- Techniques
  - Magnitude / Variational dropout / Regularization
- Challenges
  - Accuracy degradation
  - High sparsity → better performance?



arm

Pruning – Key Concepts

- Pruning schedule
  - Pruning induces damages to model
  - Increase sparsity gradually
  - Strategies: inverse power/linear/cosine
- Distribution of sparsity
  - Not all layers are equal
  - Uniform: same sparsity for all layers
  - Heuristic: sparsity $\propto$ # parameters
  - Reinforcement-learning (RL)
- Hardware-aware hyper-parameter tuning
  - Tuning for a single optimization
  - Joint tuning for multiple optimizations



arm

Hyper-Parameter Tuning

Deterministic

- Uniform
  - Same sparsity for all prunable layers
- Heuristic
  - Per-layer target sparsity: $\alpha \cdot \log|var_i|$

$$\alpha = \frac{pr \cdot \sum|var_i|}{\sum(|var_i| \cdot \log|var_i|)}$$

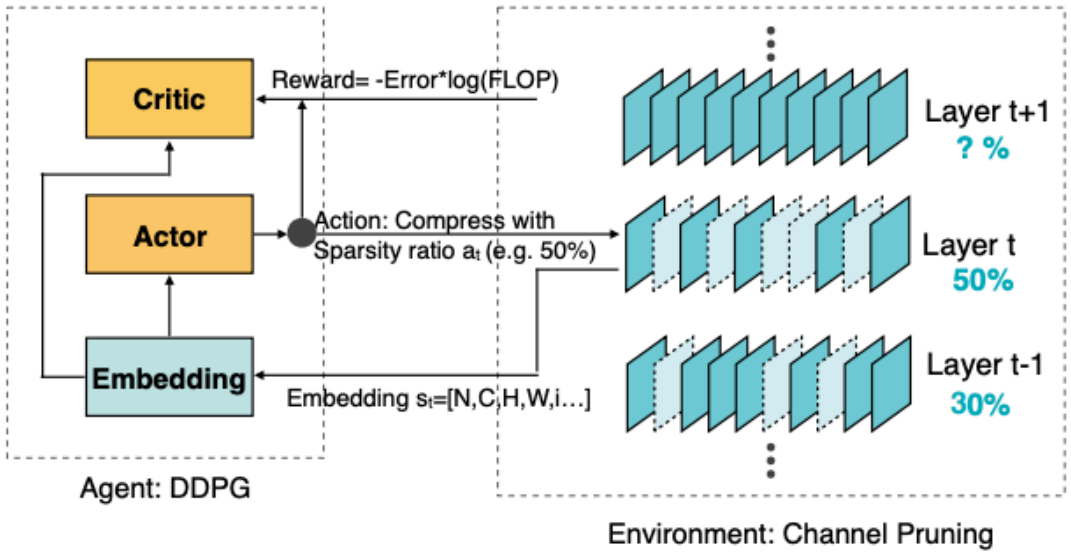
  - Dynamically increase pruning ratio during training

$$\overline{pr} = pr \cdot (1 - \left(1 - \frac{t - t_0}{n\Delta t}\right)^3)$$

Michael Zhu, et al. "*To prune, or not to prune: exploring the efficacy of pruning for model compression*" arXiv: 1710.01878 (2017).

Jiaxiang Wu, et al. "*PocketFlow: An Automated Framework for Compressing and Accelerating Deep Neural Networks*" https://openreview.net/pdf?id=H1fWoYhdim (2018)

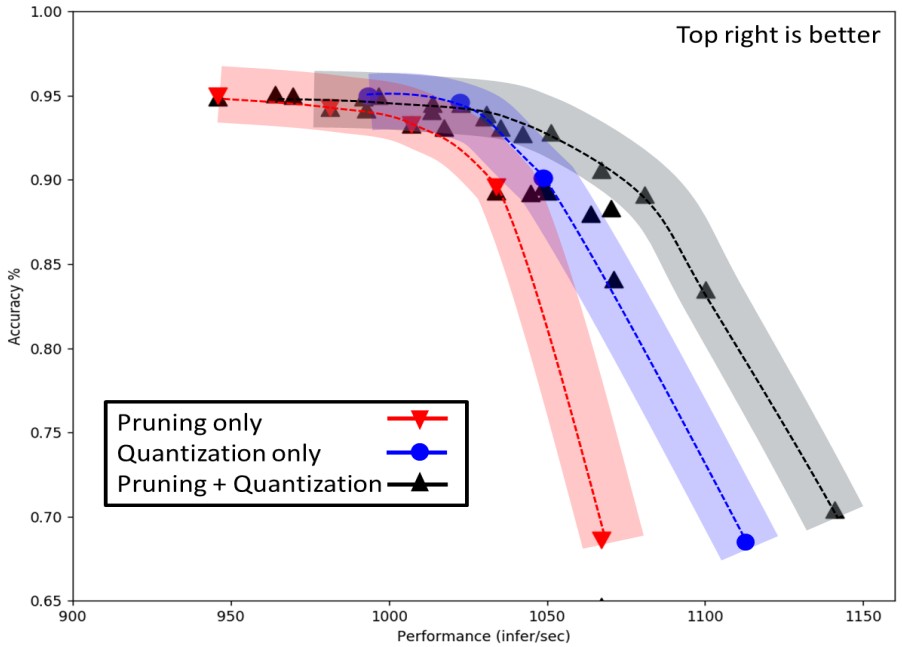
Reinforcement Learning

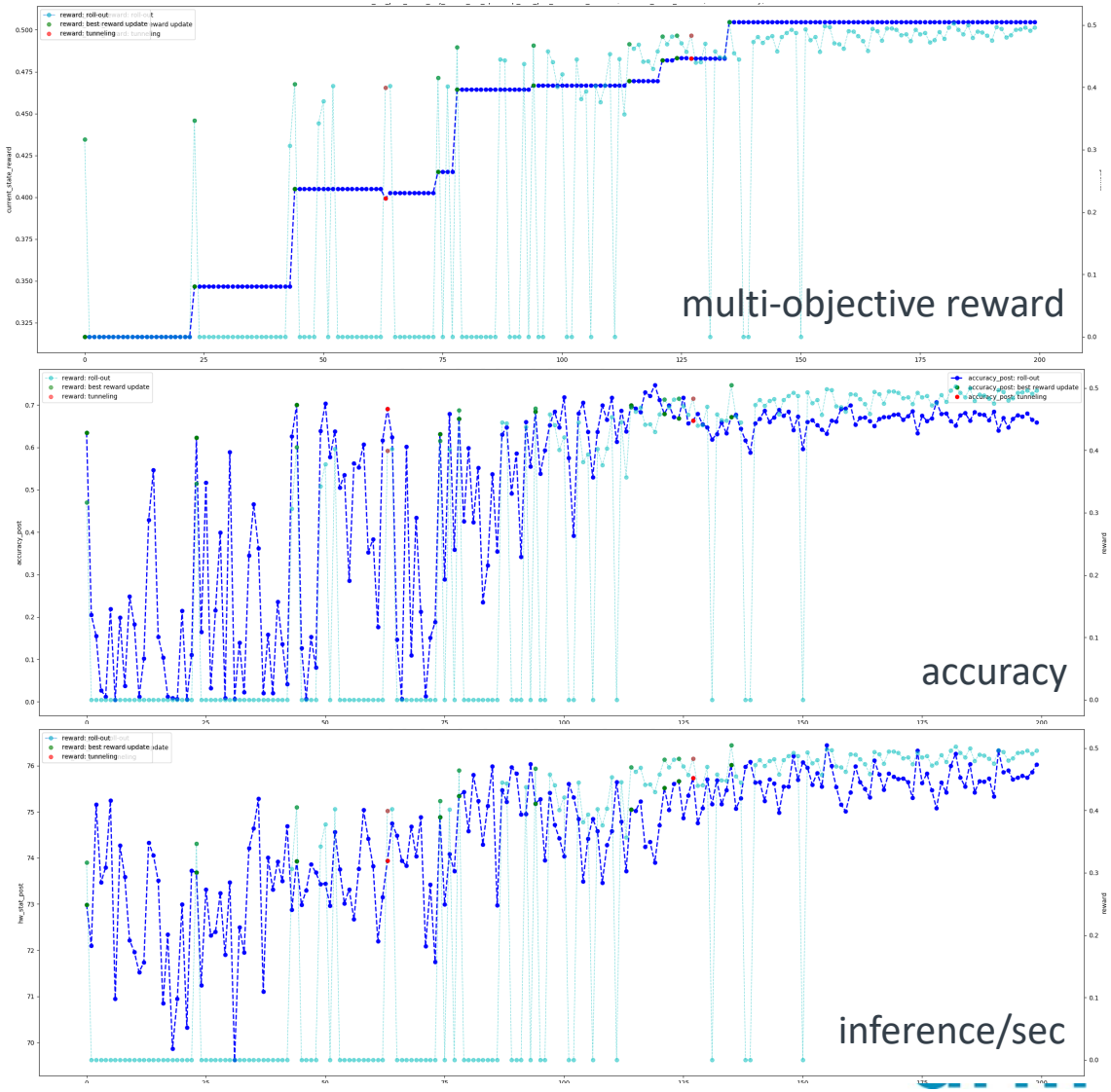


Yihui He, et al. "*AMC: AutoML for Model Compression and Acceleration on Mobile Devices*" arXiv: 1802.03494 (2018).

Hardware-Aware Hyper-Parameter Tuning

- Optimizing for accuracy + hardware metrics
  - HW metrics: latency, compression, bandwidth …
  - Multi-objective reward/fitness functions
  - Joint tuning for multiple optimizations
    - Larger search space and better results

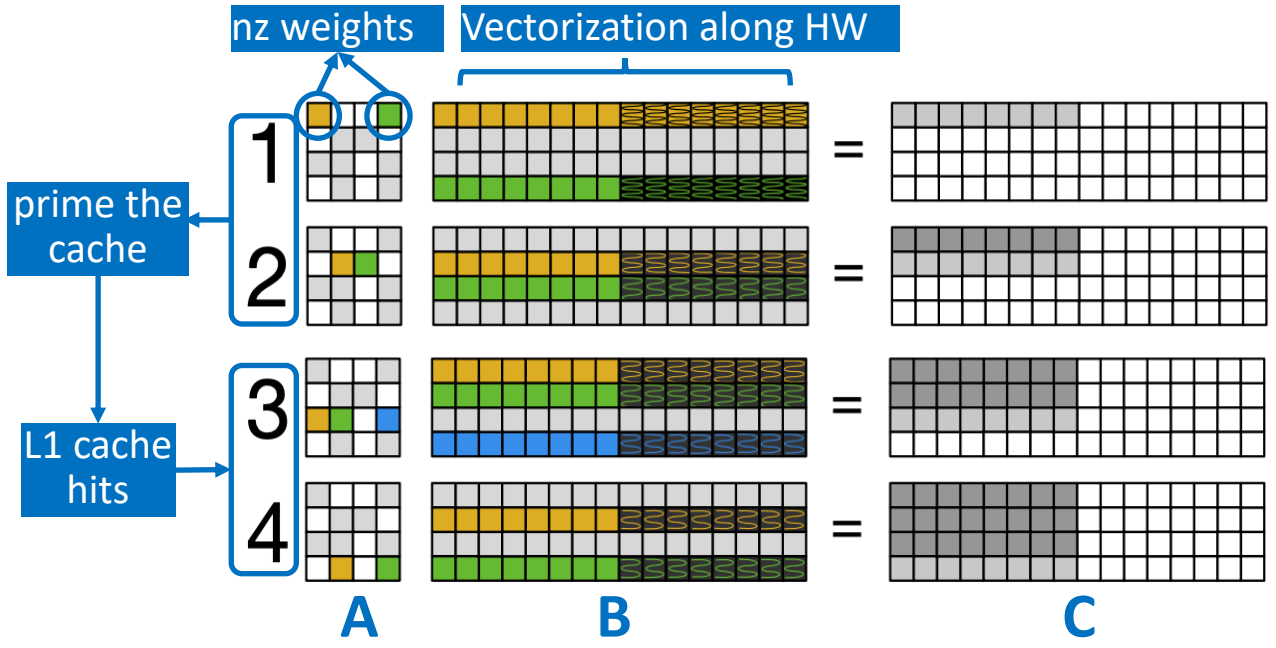




Convolution with Sparse Tensors

- Accelerating sparse matrix multiply
  - Sparse weights + dense activations
  - Dense math primitives → sparse primitives
  - Vector loads of activations
  - Randomly-accessed values cached in L1
  - Prefetching activations to reduce cache misses
  - Block-structured sparsity for additional speedup

	Model	Width	Top-1	Mega FLOPs	Mega Params	Time SD835	Time SD670	Time Wasm
MBv1	Dense	1.0	70.9	1120	4.30	125	106	271
	Sparse	1.4	**72.0**	**268**	**2.28**	**58**	**64**	**97**
MBv1	Dense	.75	68.4	636	2.59	73	64	170
	Sparse	1.0	68.4	**146**	**1.48**	**31**	**34**	**56**

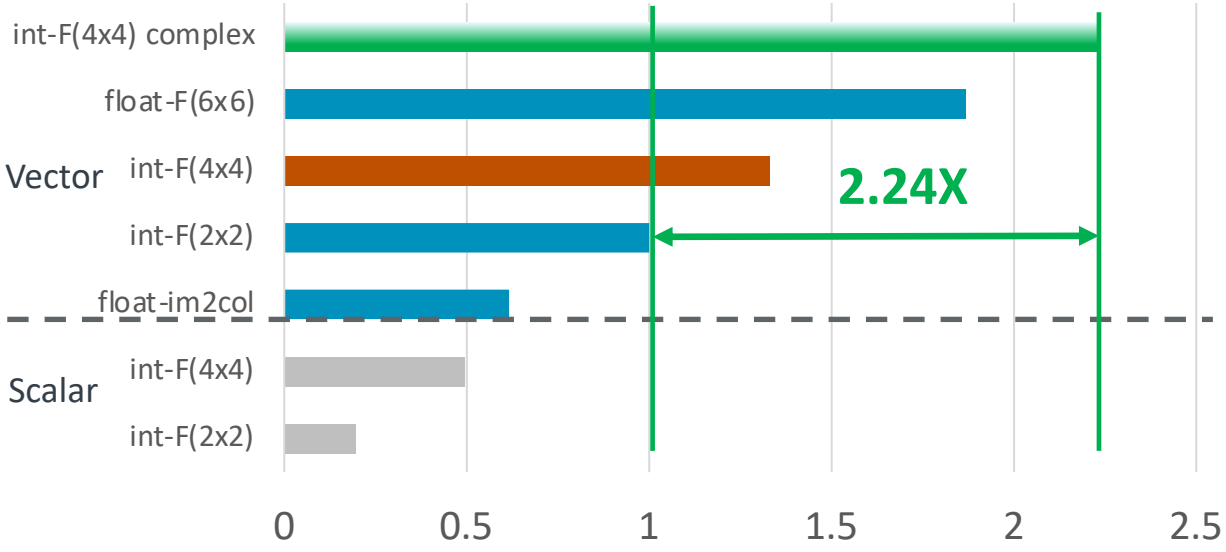


Erich Elsen, et al. "*Fast Sparse ConvNets*" arXiv: 1911.09723 (2019).

arm

Algorithmic Optimizations

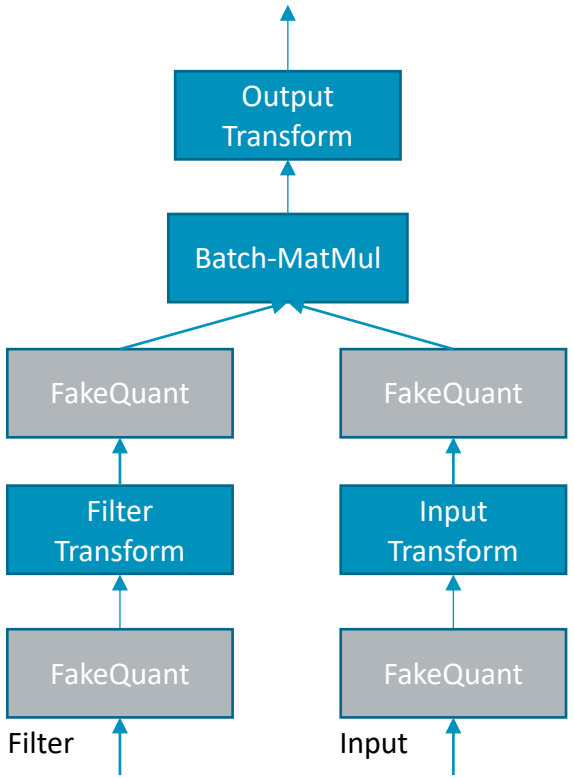
Complex-domain Winograd



	VGG 16	ResNet 18	GoogleNet	SqueezeNet
Speedup vs NCNN 2x2	94.55%	21.13%	12.82%	8.86%

- Lingchuan Meng, et al. "*Efficient Winograd Convolution via Integer Arithmetic*" arXiv: 1901.01965
- NCNN: https://github.com/Tencent/ncnn

8-bit Winograd



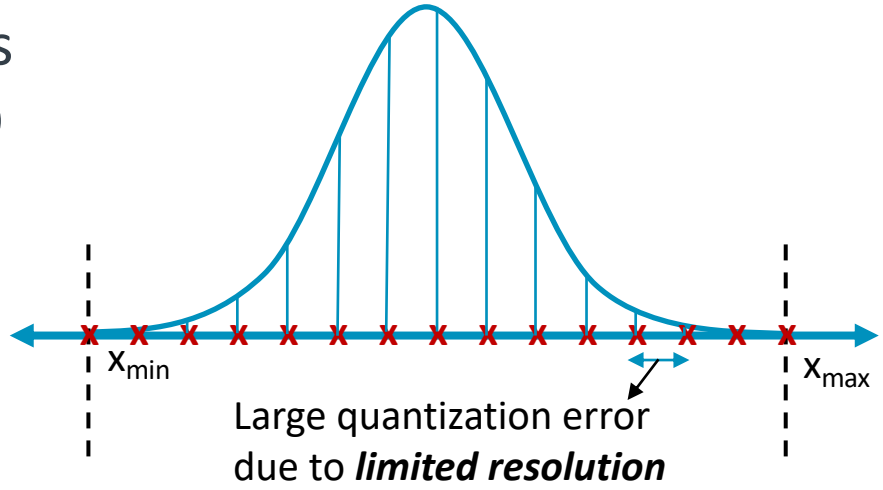
Winograd Conv2D

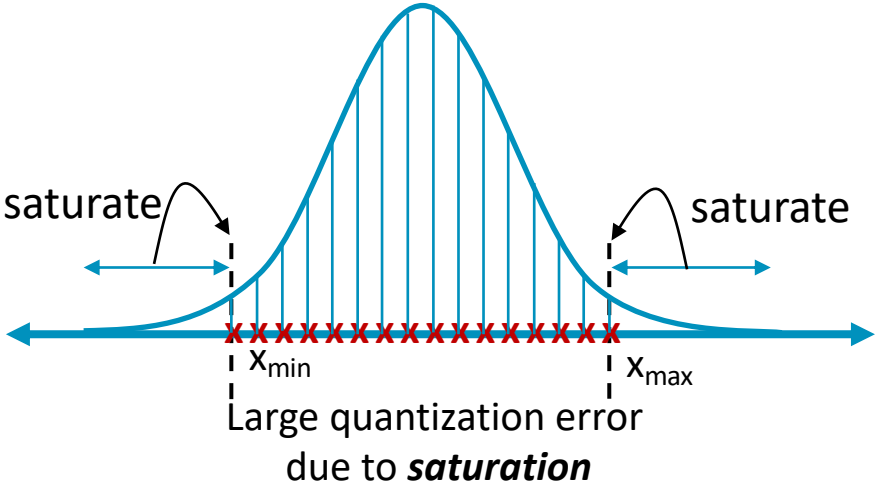
	Accuracy		
	Training	**Top-1**	**Top-5**
FP32	X	76.94%	93.40%
Int8 (FQ)			
Im2Col	X	76.38%	93.13%
F(4x4) Real	X	0.52%	1.79%
	√	60.49%	82.86%
F(4x4) Complex	X	74.86%	92.43%
	√	76.27%	93.10%

Inception V3

Quantization

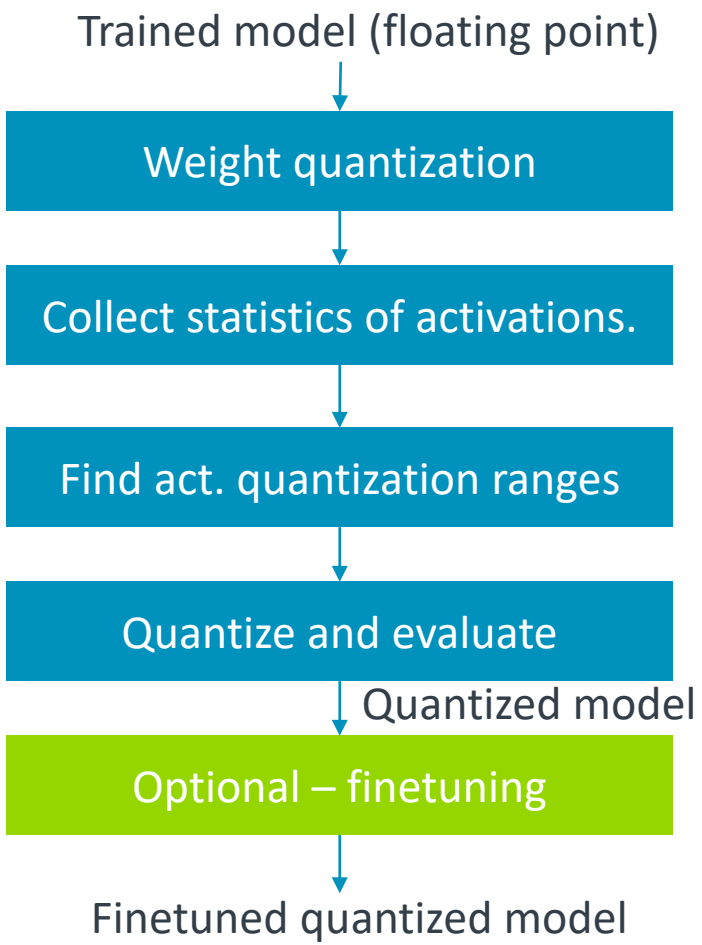
- Storing and computing with tensors at lower bitwidths
  - Typically FP32 -> INT8: $x_{FP32} = scale \cdot (x_{INT8} - zero\_point)$
  - 4x savings in model size and memory bandwidth
  - Inference speedup: 2-4x
  - More aggressive quantization in active research
- Granularity: per-layer vs. per-channel
- Symmetric vs. asymmetric
  - Weights: symmetric with *zero_point*=0
  - Activations: asymmetric
- Finding optimal quantization ranges
  - Balancing range vs. resolution
  - Techniques: minimize Quantization error, KL-divergence, etc.





arm

Quantization Workflow



Find optimal quantization ranges for weights and evaluate model with quantized weights

Insert histogram Ops and collect the distribution of activations.

Find optimal quantization thresholds for activations

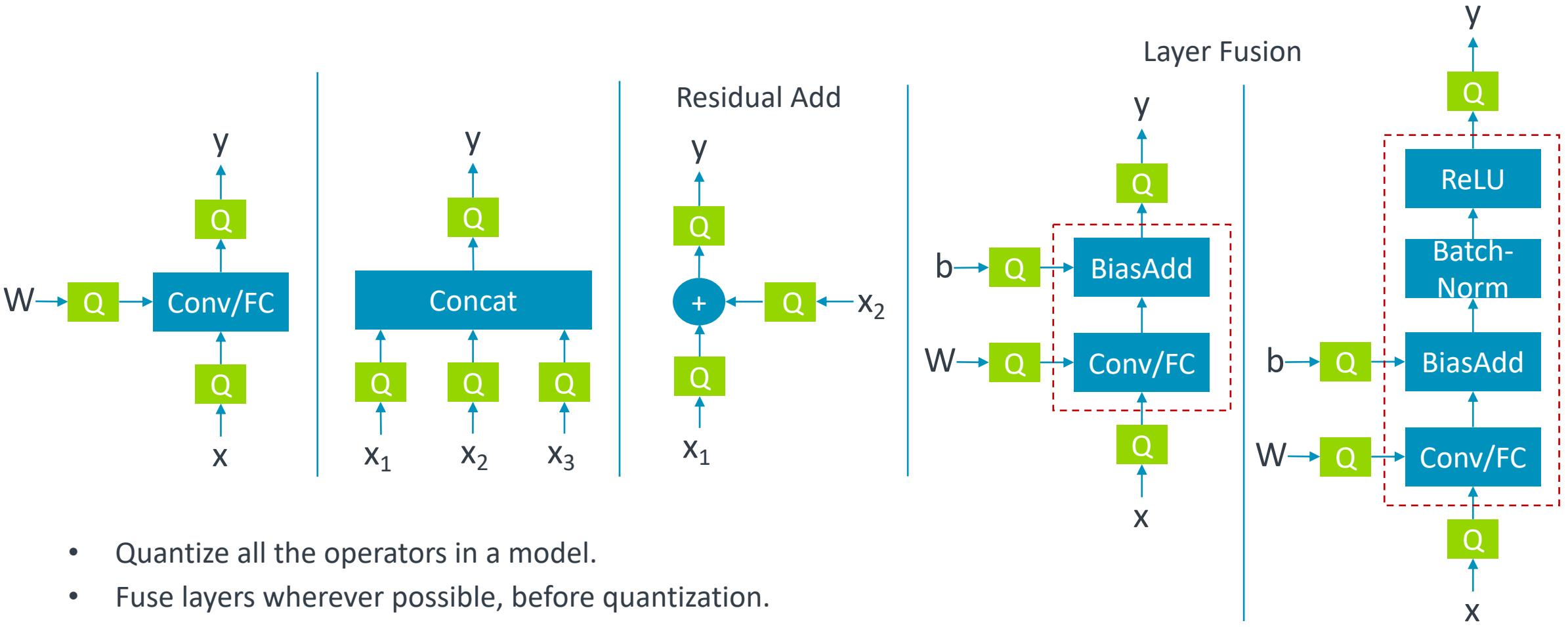
Evaluate weight/activation quantized model

Optionally finetune if training data is available
=> Quantization aware training (QAT)

Quantizing Activation Nodes

Q = tf.fake_quant

- Simulate quantization in forward pass.
- Straight-through-estimator (STE) in backward pass during QAT.

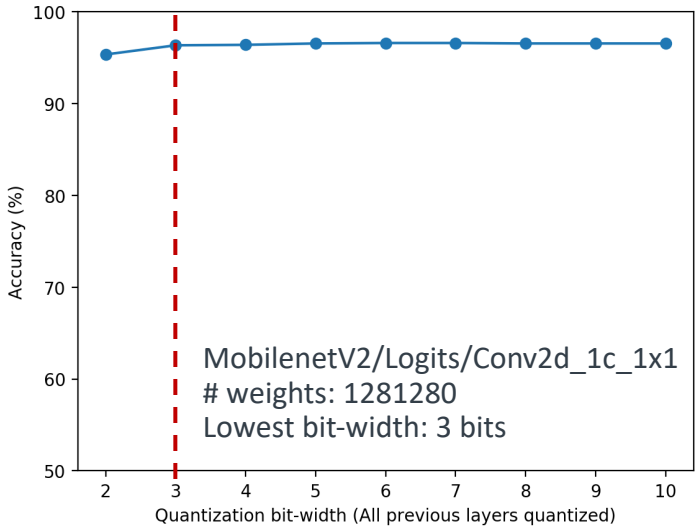


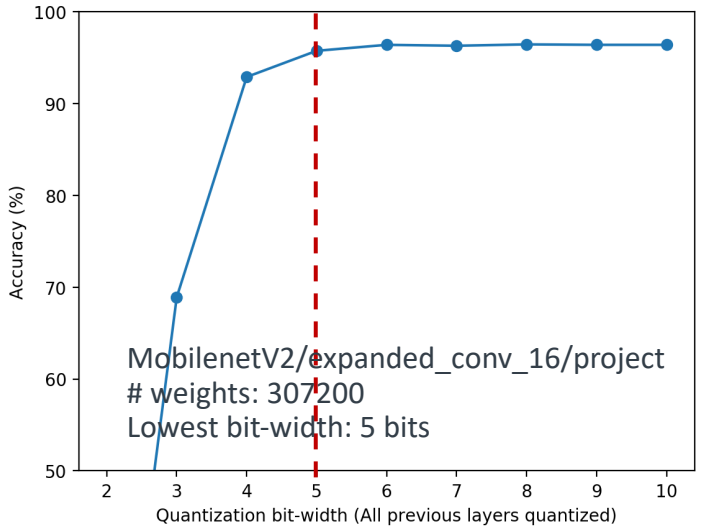
- Quantize all the operators in a model.
- Fuse layers wherever possible, before quantization.

arm

Mixed-Precision Quantization

- Some layers are less sensitive to aggressive quantization.
- How to find the optimal bit-width per layer ?
- A solution: Sensitivity-based mixed precision quantization
  - Find lowest bit-width without significant accuracy drop.
  - Consider the cascaded effect of quantization error from layer-to-layer.
  - Start from the largest layer, so it is compressed the most.



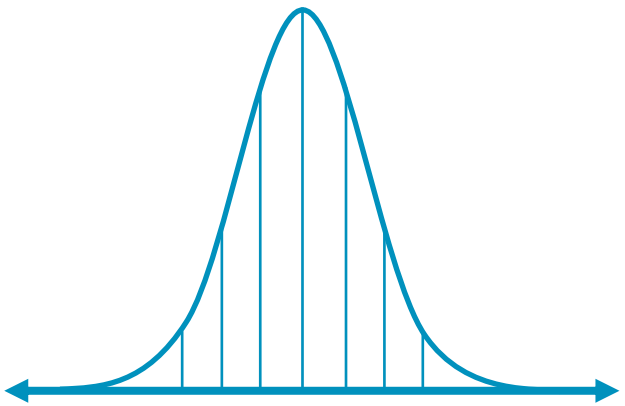


Mobilenet V2

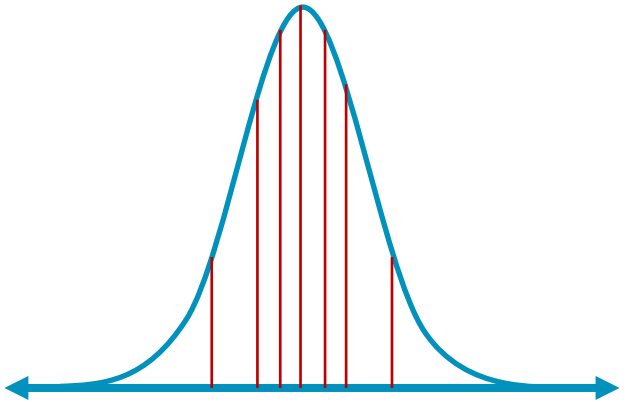
- Average bitwidth: 4.5 bits
- 2% accuracy drop (without retraining)
- Fine-tuning recovered 1.5% accuracy

Clustering: Non-Uniform Quantization

- Non-uniform quantization yields smaller quantization error than uniform quantization.
- Better weight compression – especially for large layers.
- K-means clustering algorithm to find initial cluster centroids.
- Fine-tune the cluster centroids with clustering constraints in the graph.
- Preserve sparsity during fine-tuning using sparsity masks.



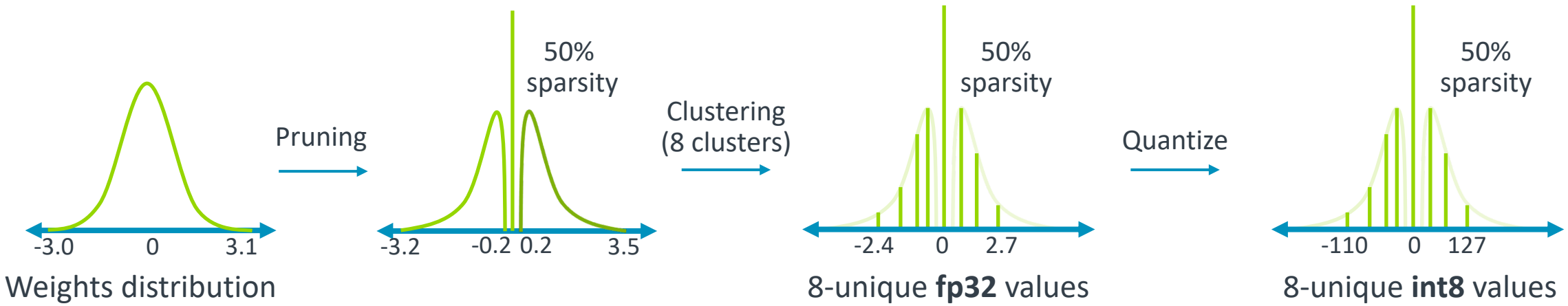
Uniform Quantization

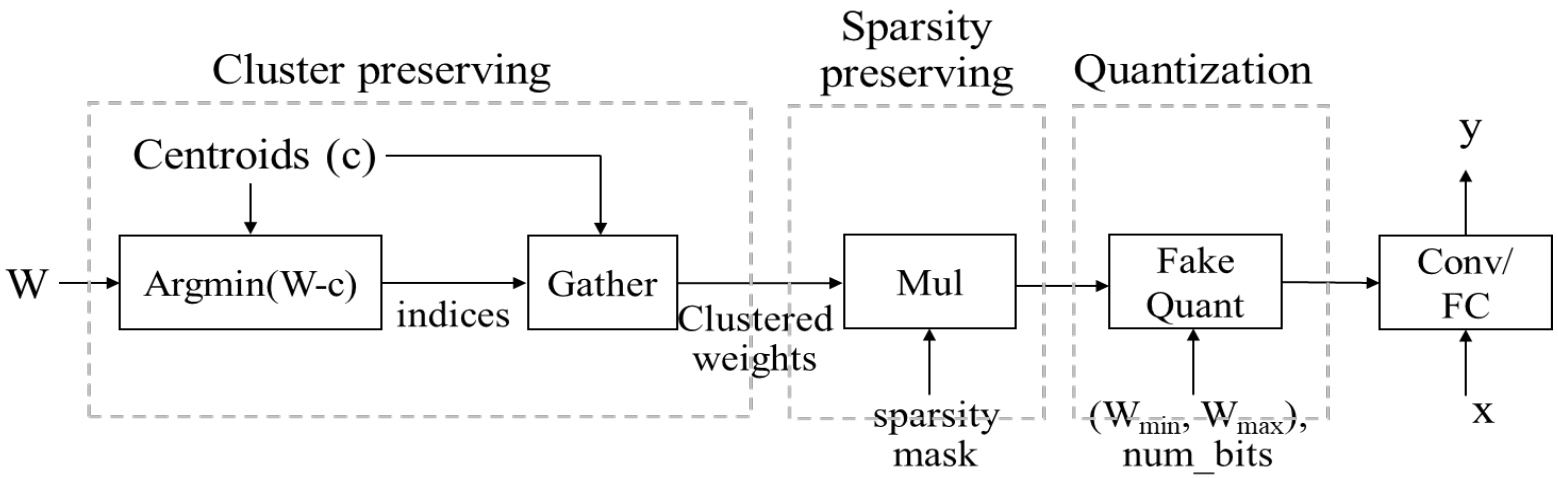


Non-uniform Quantization
(smaller quantization error)

arm

Collaborative Optimizations





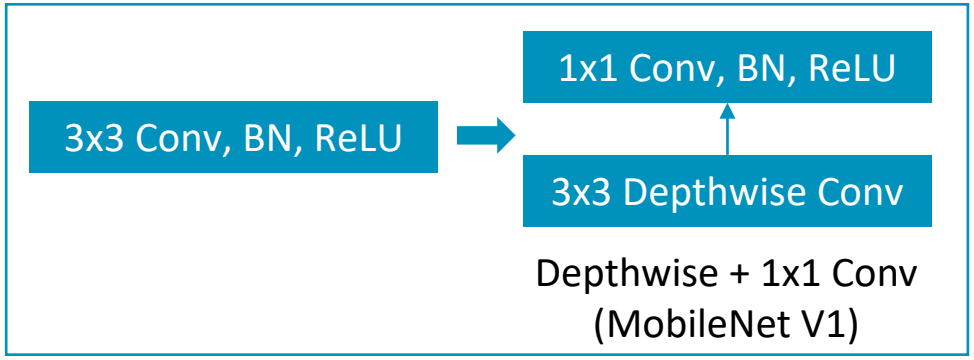
arm

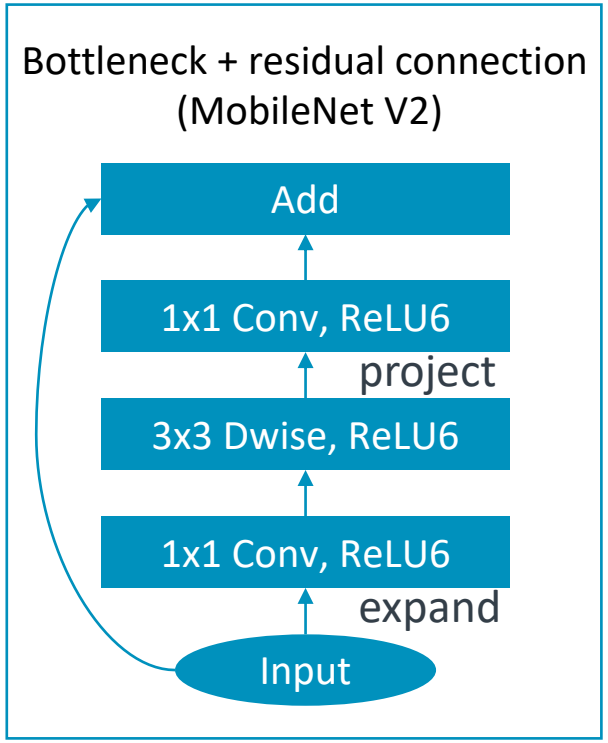
arm

Model Architecture Optimizations

Efficient Network Building Blocks

- Arithmetic reduction by operator decomposition/approximation
  - Depthwise convolution – Mobilenet-V1
  - Inverted bottleneck with residual – Mobilenet-V2
- Operator sparsification
  - sparsity → performance?
  - Replace dense ops with sparse ops
- Asymptotically-faster operators
  - Winograd convolution
  - FFT

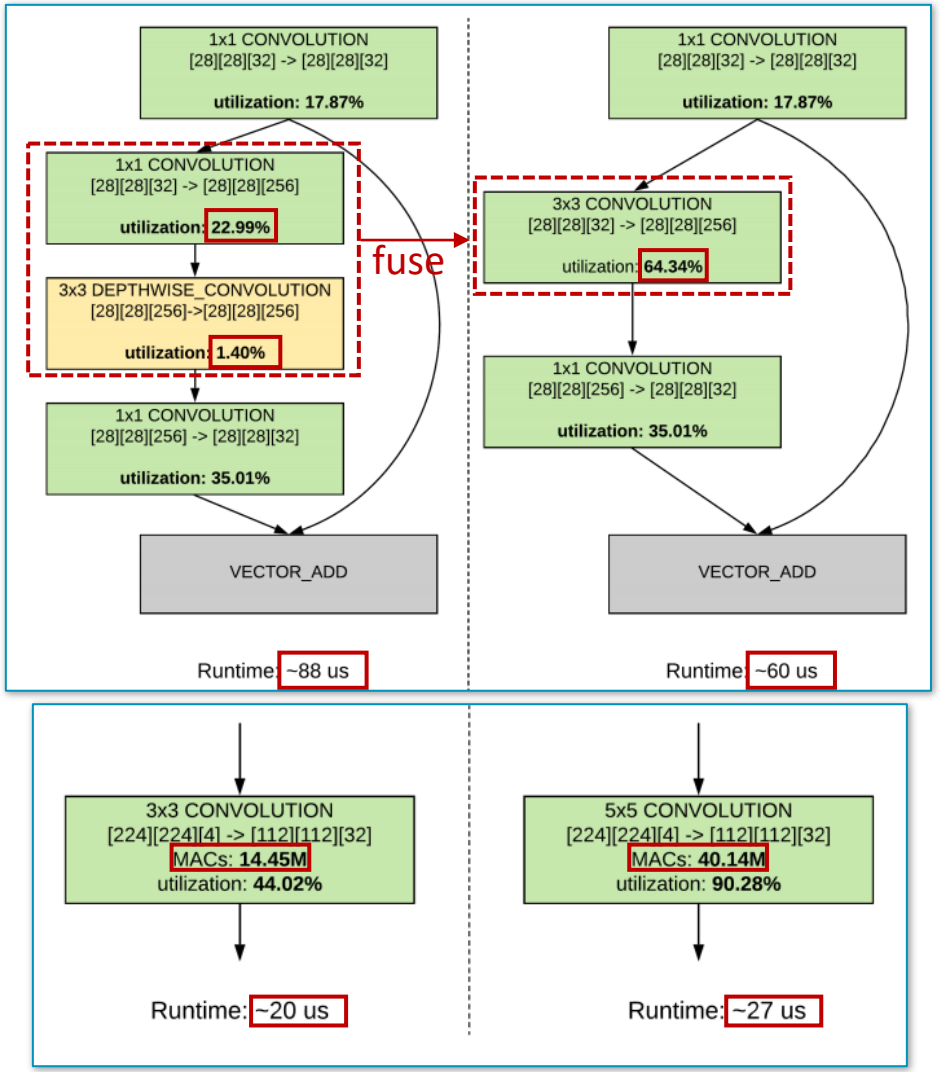




arm

Efficiency is Target-Specific

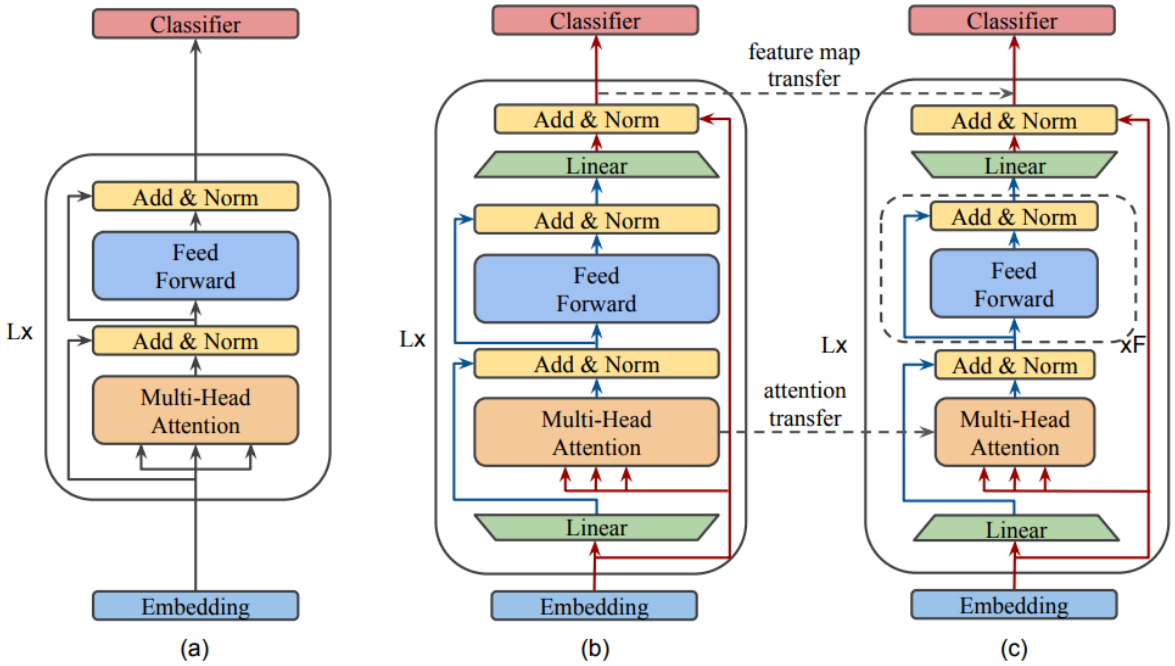
- Hardware utilization matters!
  - Inverted bottleneck conv block
  - Fused inverted bottleneck conv block
    - 1x1 conv + 3x3 depthwise -> 3x3 conv
    - More trainable parameters
    - Better HW utilization (hence a good latency-accuracy trade-off)
  - 3x3 vs. 5x5 convolution
    - 5x5 convolution leads to 2.78x increase of MACs and parameters
    - Only a 35% runtime increase
    - Good trade-off for more trainable parameters at a marginal latency cost



Gupta, Suyog, and Berkin Akin. "Accelerator-aware Neural Network Design using AutoML." *arXiv preprint arXiv:2003.02838* (2020).

Efficiency is More than Conv/FC

- Layer normalization and Gaussian Error LU (GELU) impact latency
- Layer normalization replaced by element-wise linear transform
  - $NoNorm(h) = \gamma \circ h + \beta$
- GELU replaced by ReLU



	#Params	#FLOPS	Latency	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 5.7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	GLUE
MobileBERT$_{\text{TINY}}$	15.1M	3.1B	40 ms	46.7	91.7	87.9	80.1	68.9	81.5/81.6	89.5	65.1	75.8
MobileBERT	25.3M	5.7B	62 ms	50.5	92.8	88.8	84.4	70.2	83.3/82.6	90.6	66.2	77.7
MobileBERT w/o OPT	25.3M	5.7B	192 ms	51.1	92.6	88.8	84.8	70.5	84.3/**83.4**	**91.6**	**70.4**	**78.5**

Lower is better

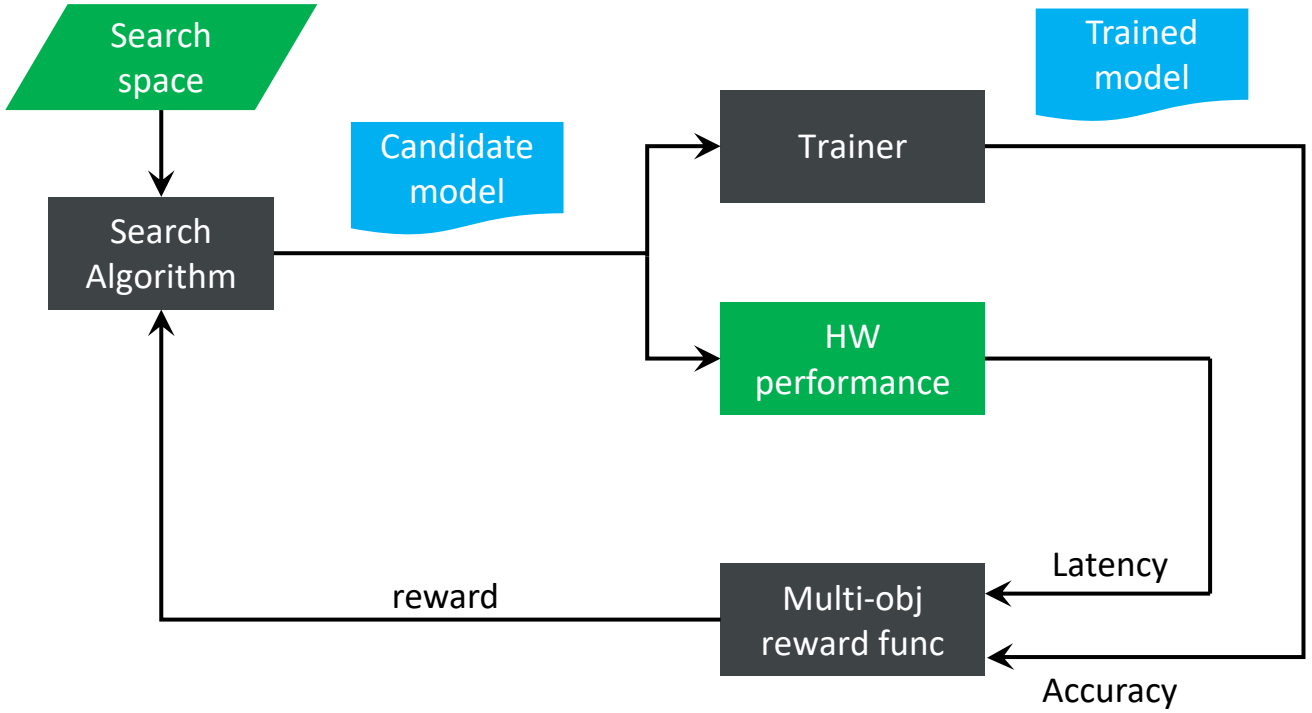
Higher is better

Sun, Zhiqing, et al. "MobileBert: a compact task-agnostic BERT for resource-limited devices." *arXiv preprint arXiv:2004.02984* (2020).

Hardware-aware Neural Architecture Search (NAS)

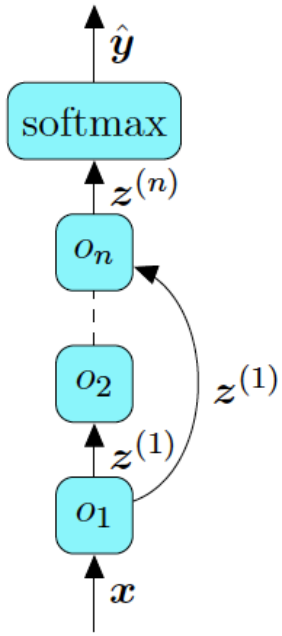
- Diversity in ML Hardware – CPUs, GPUs and NPUs
  - Different programming models, compute capabilities, memory organization
  - Same neural network architecture cannot map efficiently across multiple HW
- Awareness of the target HW architecture by NN architectures
  - Hand tuning model architectures
  - Automated neural architecture search (NAS)
- Search objectives
  - Latency proxies: # MACs, # parameters (NASNet)
  - Hardware performance model (MNASNet, ProxylessNAS, FBNet)
- State-of-the-art models searched by NAS
  - MobileNet v3, FBNet, EfficientNet, MobileBERT

NAS – Key Concepts

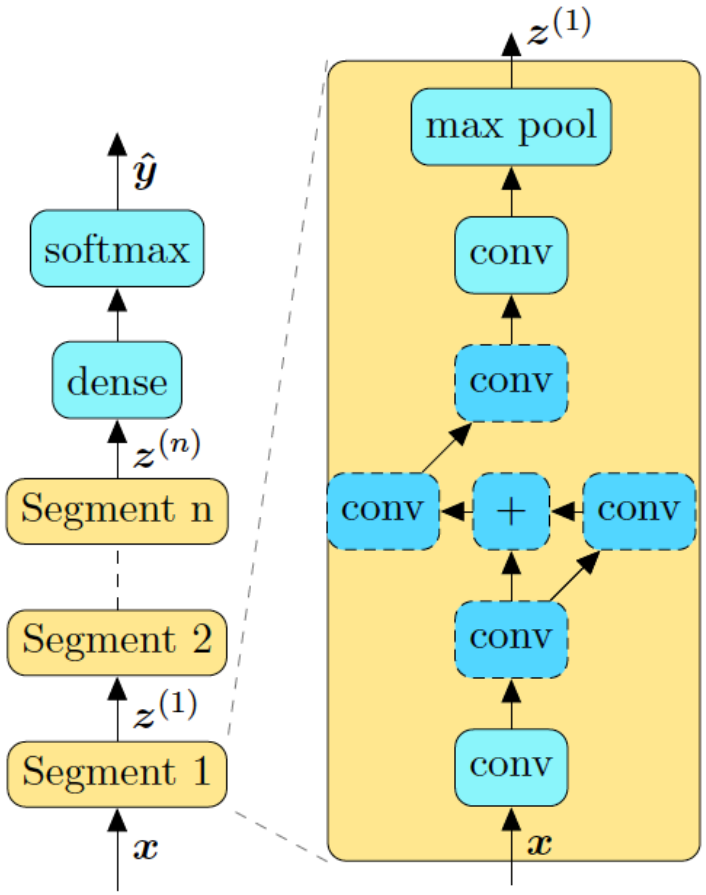


- Search space
  - Chain-structured
  - Architecture template
  - Cell-based
- Search strategy
  - Reinforcement learning
  - Gradient-based methods
  - Evolutionary algorithms
  - Random search with rejection sampling
- Performance estimation
  - Accuracy estimation
  - Latency estimation

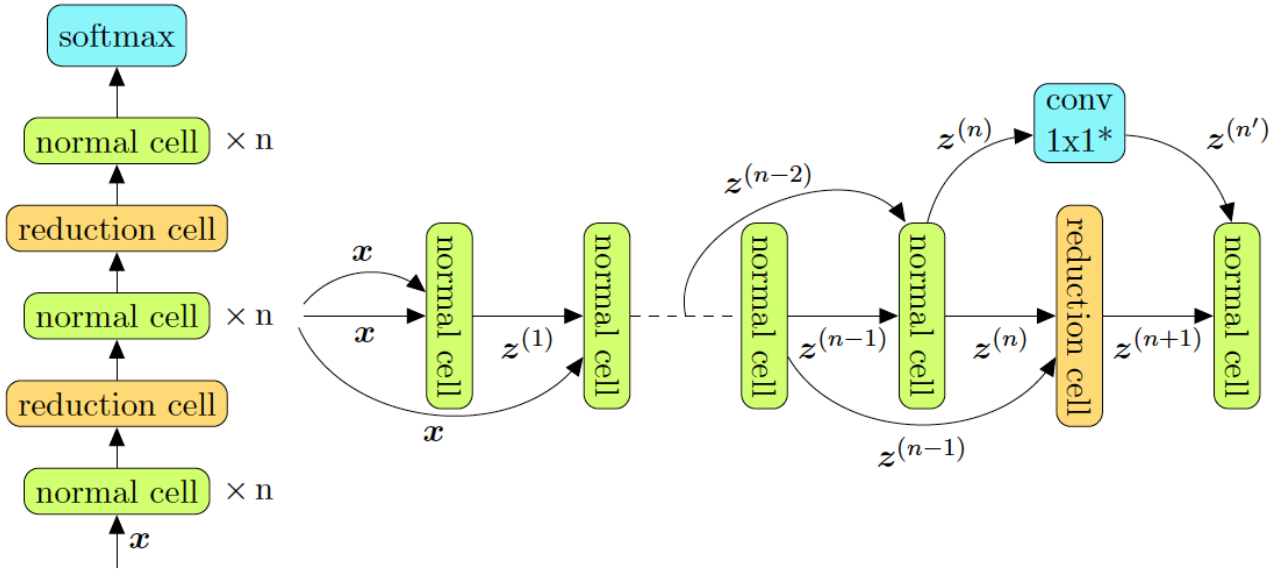
NAS Search Space



Chain-structured



Architecture template



Cell-based

Wistuba, Martin, et. al. "A survey on neural architecture search." *arXiv preprint arXiv:1905.01392* (2019).

Summary

- Resource constraints and diverse SW + HW features require co-development
- Model optimizations bridge the gap between models + SW + HW
- Efficient building blocks and architectures for higher accuracy and performance
- NAS efficiently navigates in design spaces to automate model design
- New opportunities in joint optimization of NAS, pruning, and quantization

arm

Questions?

arm

Thank you!

Tweet us: @ArmSoftwareDev

Check out our Arm YouTube channel and our Arm Software Developers YouTube channel

Signup now for our next AI Virtual Tech Talk here

Don’t forget to fill out our survey to be in with a chance of winning an Arduino Nano 33 BLE board

arm AI

AI Virtual Tech Talks Series

arm

Thank You

Danke

Merci

谢谢

ありがとう

Gracias

Kiitos

감사합니다

धन्यवाद

شكرًا

תודה