Tsinghua University

sensetime

Chapter 2 - Section 11

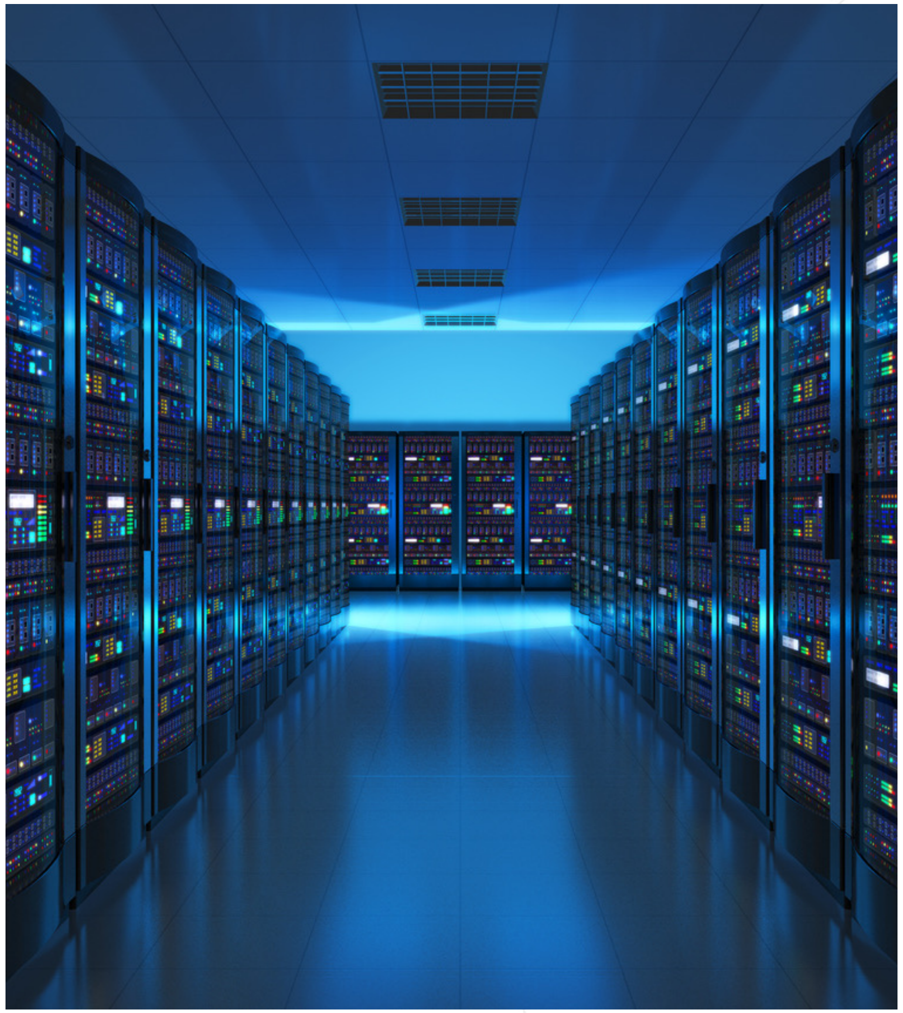
Model Compression

Ruihao Gong

Friday, May 6, 2022

Highlights

Background


High Performance Server

VS

Low-power Edge Device

Background

Less Parameters

Faster Inference

Less Memory Occupation

Quantization

Pruning

KD

NAS

商汤 sensetime

清華大學 Tsinghua University

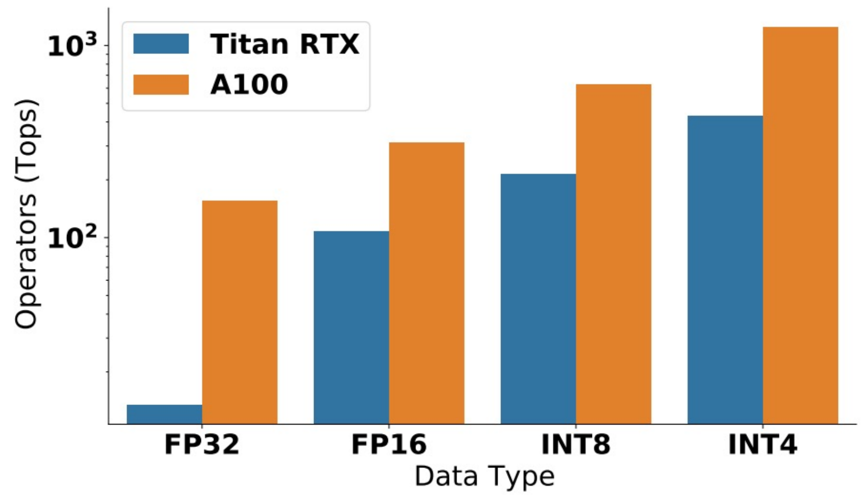
Highlights

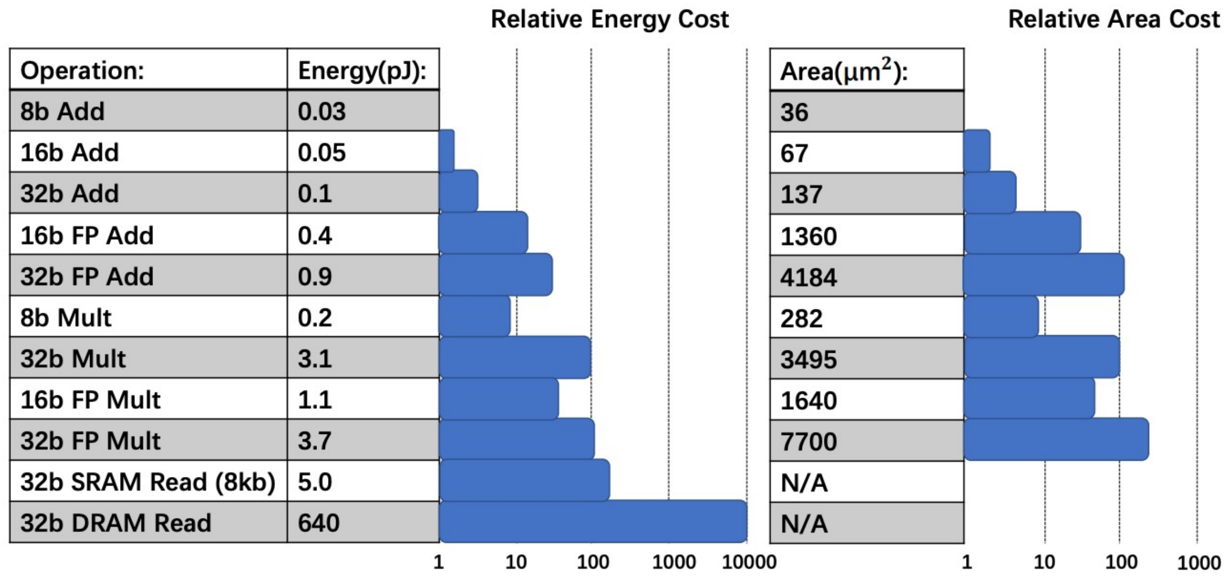
Quantizer Design

- What is Model Quantization?
  - Quantization maps the 32-bit floating-point numbers into low-bit fixed-point numbers, or a mapping from **continues** real numbers to **discrete** integers.
  - Applying quantization to model parameters (e.g. weights & bias) can save memory footprint. For example, 8-bit quantization can save **4x** memory space.
  - Applying quantization to both parameters and activations can accelerate the inference by replacing the **floating-point** multi-adds operations to low-power **fixed-point** ones.

Quantizer Design

- What can Quantization do?





Operation:	Energy(pJ):	Area(μm^2):
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8kb)	5.0	N/A
32b DRAM Read	640	N/A

Figure 7: *(Left) Comparison between peak throughput for different bit-precision logic on Titan RTX and A100 GPU. (Right) Comparison of the corresponding energy cost and relative area cost for different precision for 45nm technology [95]. As one can see, lower precision provides exponentially better energy efficiency and higher throughput.*

Quantizer Design

- Uniform Quantization
  - Can be represented by fixed-point integers.
  - Can compress and accelerate the inference.
- Non-Uniform Quantization
  - Levels are arbitrarily spaced.
  - Non-uniform quantization schemes are difficult to be deployed efficiently on general computation hardware.

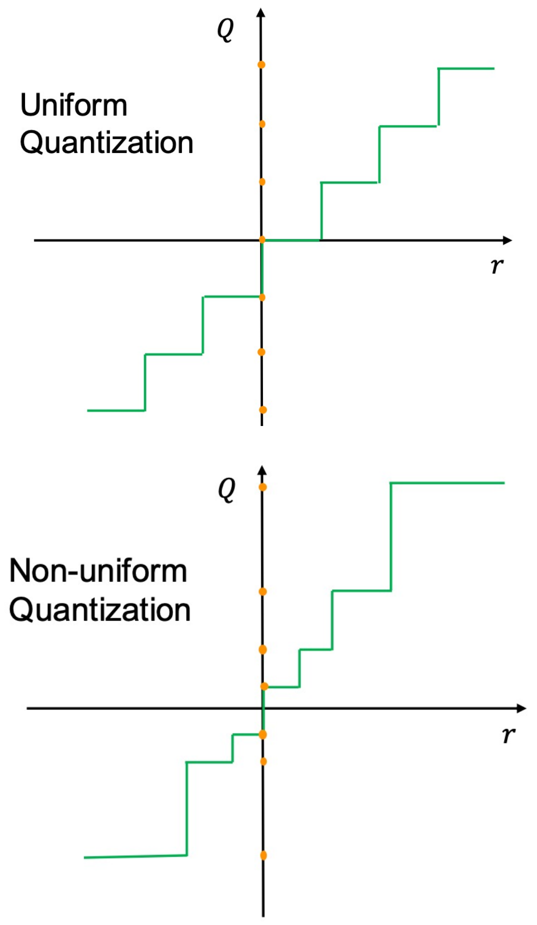


Figure taken from Gholami et al., 2021, A Survey of Quantization Methods for Efficient Neural Network Inference

Quantizer Design

- Symmetric Quantization

$$x_q = clip\left(\text{round}\left(\frac{x}{s}\right), n, p\right)$$

  - Symmetric quantization quantize parameters within $(-\alpha, \alpha)$.
  - 0 will be quantized to exactly integer 0.

- Asymmetric Quantization

$$x_q = clip\left(\text{round}\left(\frac{x}{s} - z\right), n, p\right)$$

  - Much more flexible $(-\alpha, \beta)$.
  - Must ensure 0 will be quantized to an integer Z exactly.

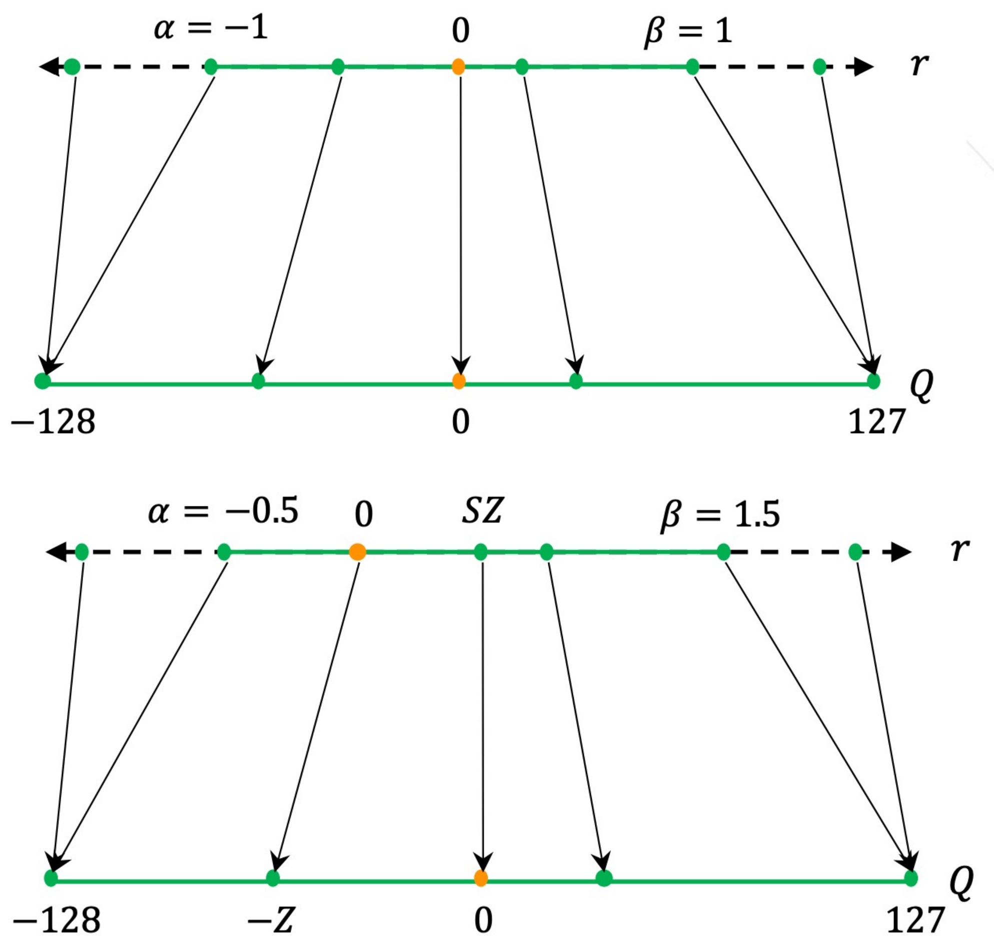
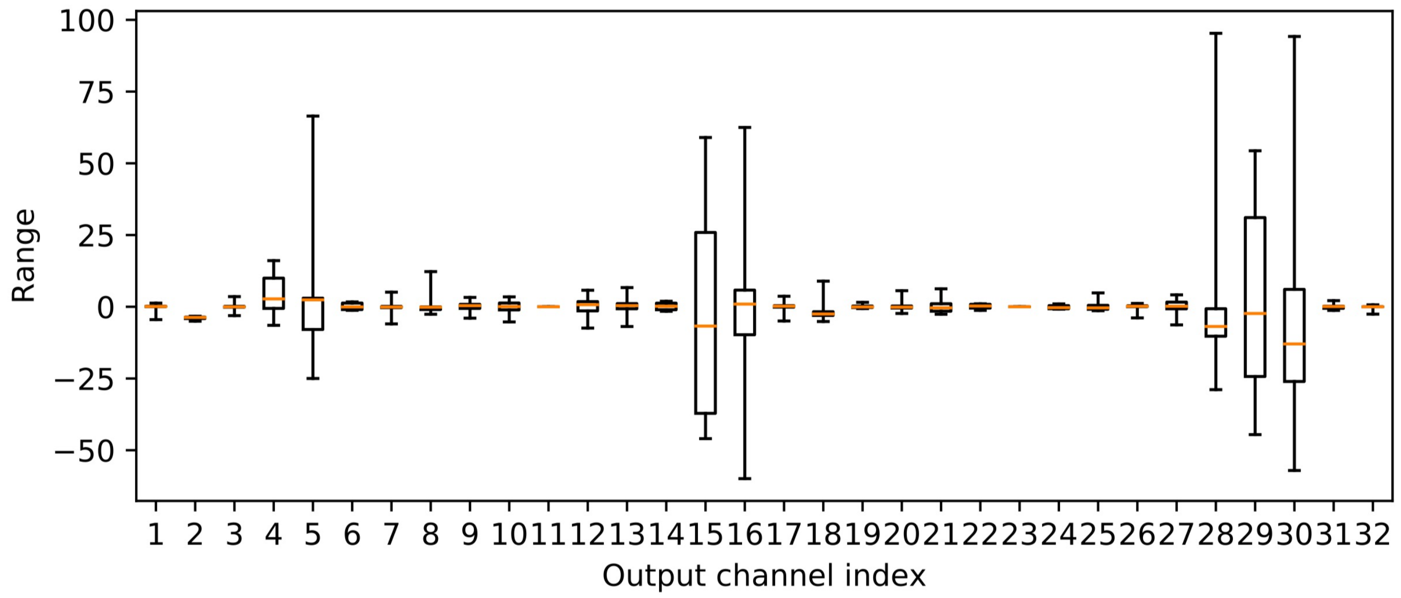


Figure taken from Gholami et al., 2021, A Survey of Quantization Methods for Efficient Neural Network Inference

Quantizer Design

- Layer-wise Quantization
  - The same clipping range is applied to all weights in a layer.
  - Could have bad results if channels differ a lot.
- Channel-wise Quantization
  - Assign each channel a unique clipping range.
  - The computation may become more complex than layer-wise.



Weight range of a DW-Conv layer in MobileNetV2

gure taken from Nagel et al., 2019, Data-Free
uantization Through Weight Equalization and Bias
orrection.

Two ways to produce quantized models

1. Post Training Quantization (PTQ)
2. Quantization Aware Training (QAT)

Post-Training Quantization

- Features of PTQ
  - Low-cost, only need a pretrained model and calibration data (10~1000 training images) to finish quantization.
  - Fast, PTQ can quantize model in several minutes.
  - Easy to use, only an API call.
  - Low performance: Quantizing a ResNet-18 to to 4-bit can only have 39% accuracy, as explained in [1].

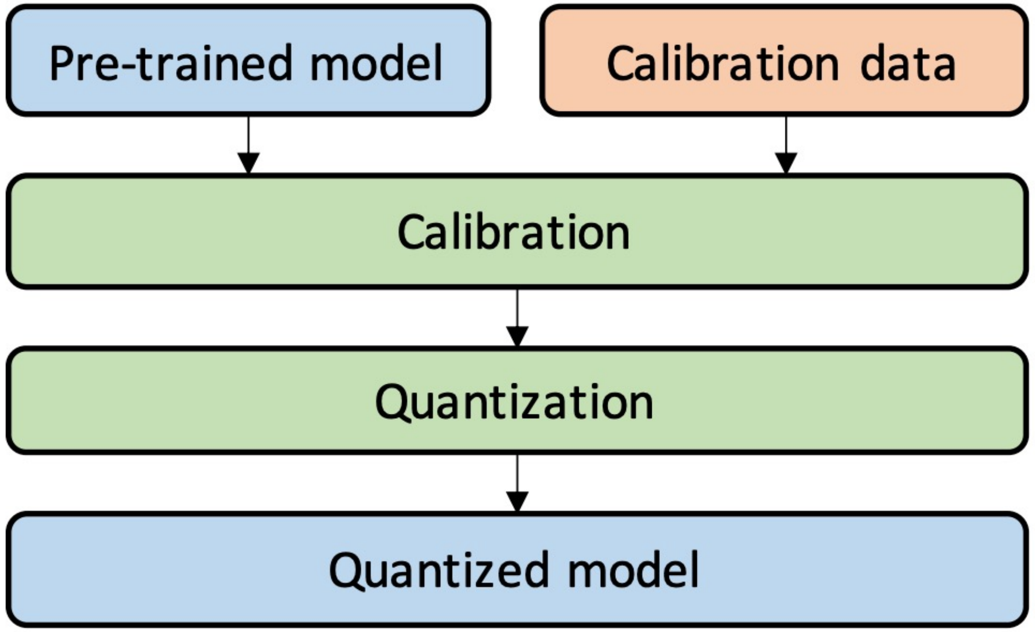
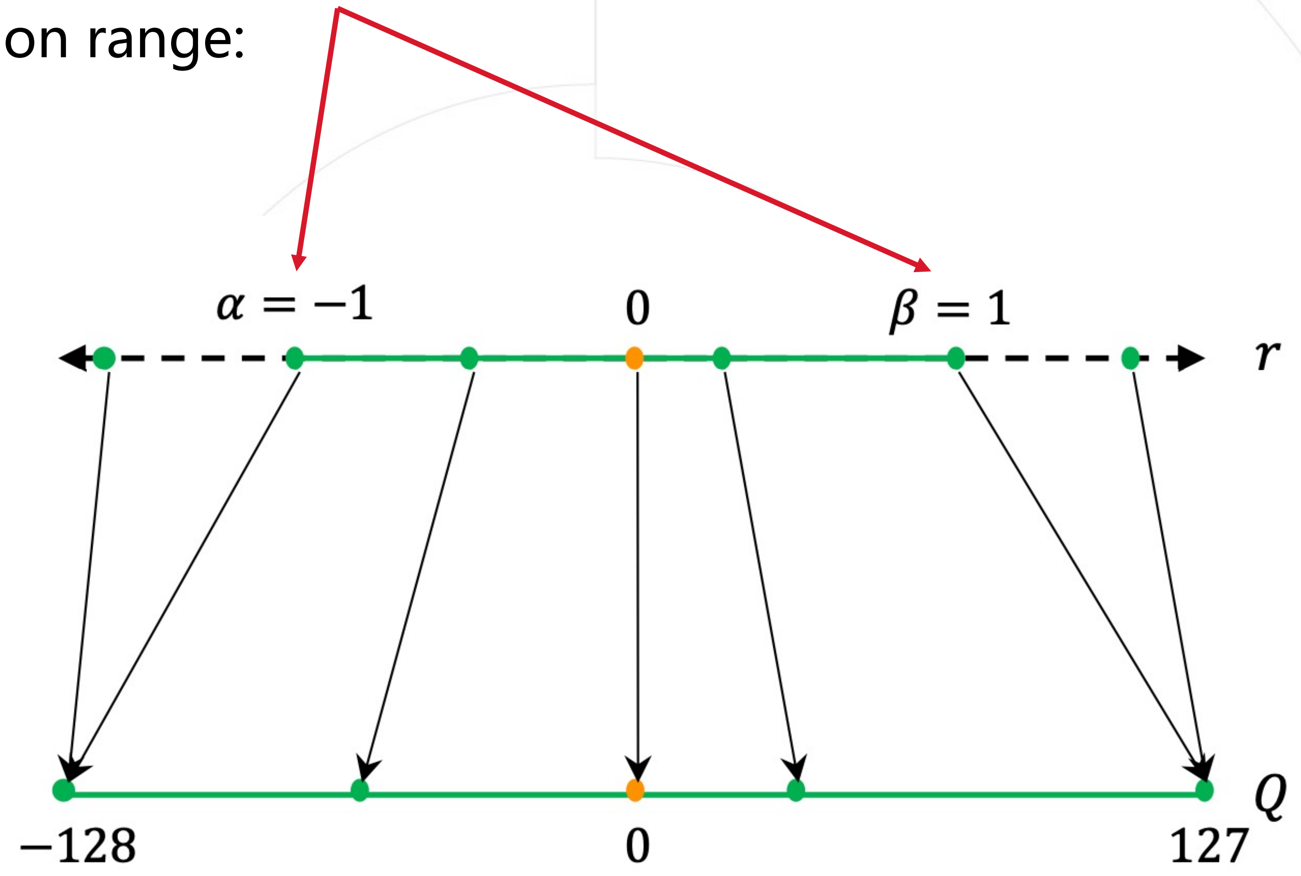


Figure taken from Gholami et al., 2021, A Survey of Quantization Methods for Efficient Neural Network Inference

Ref. [1] Nagel et al., 2019, Data-Free Quantization Through Weight Equalization and Bias Correction.

Post-Training Quantization

- How to calibrate quantized models?
  - In PTQ, we need to estimate the **quantization range** of both weights and activations.
  - Several ways to find the quantization range:
    - Use min-max range
    - Minimize Mean Squared Error
    - Minimize KL Divergence Loss



Quantization-Aware Training

- Features of QAT
  - End-to-end training. Requires all training images and huge computing resources.
  - Slow, need >100 GPU hours.
  - **Not** Easy to use, we have to modify the training codes.
  - High performance: Quantizing a ResNet-18 to 3-bit can retain original FP model performance [1].

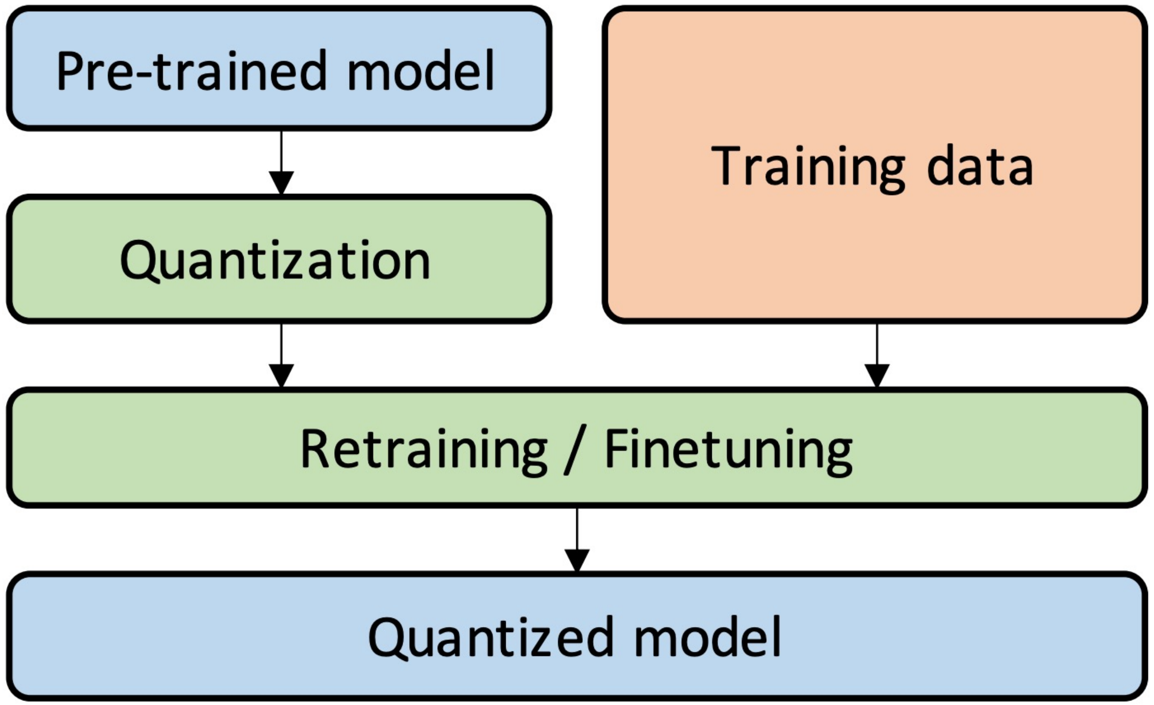
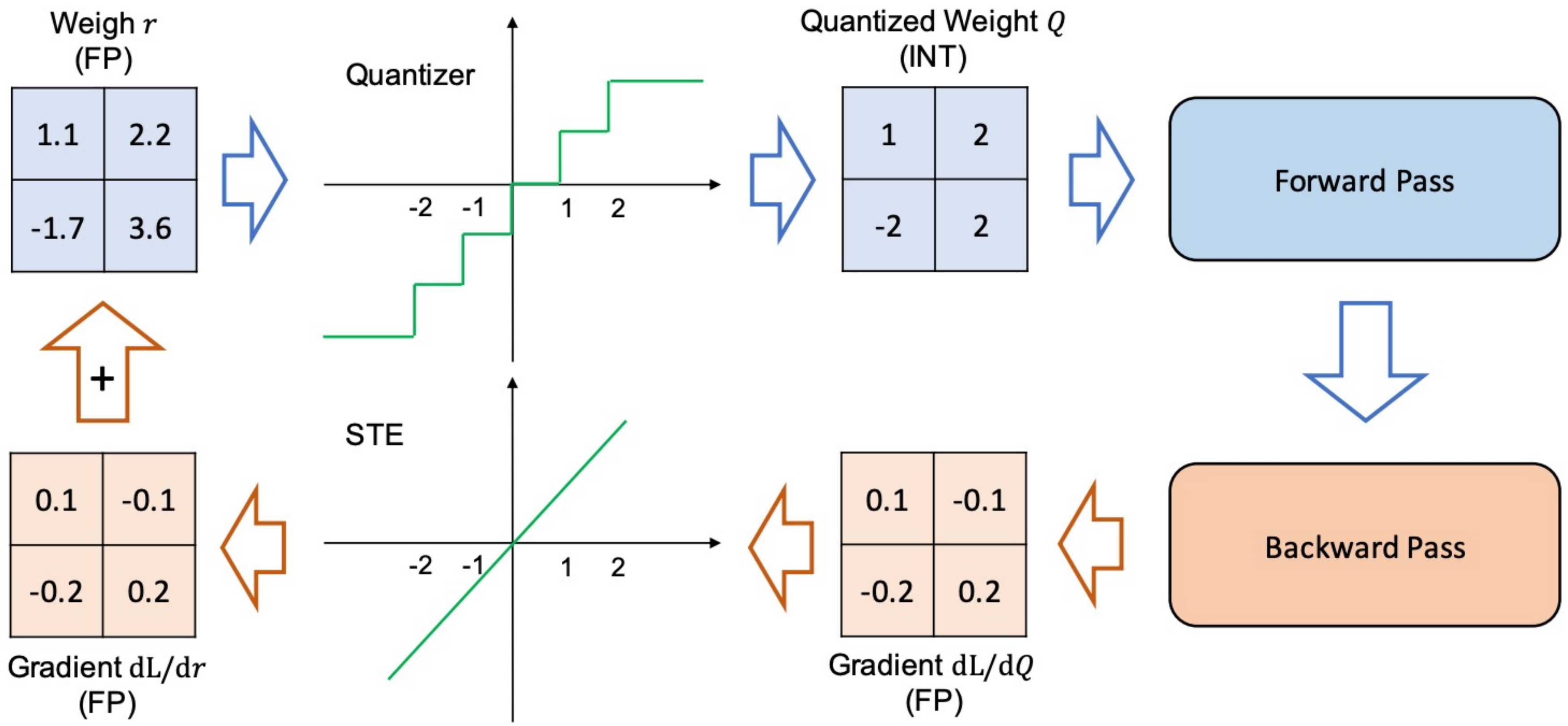


Figure taken from Gholami et al., 2021, A Survey of Quantization Methods for Efficient Neural Network Inference

Ref. [1] Esser et al., 2020, Learned step size quantization.

Quantization-Aware Training

- How to learn a quantized model?
  - The quantization function (round-to-integers) is not differentiable. To perform standard backpropagation, we need to estimate the gradients of step function:



Quantization-Aware Training

- Folding Batch Normalization Layers

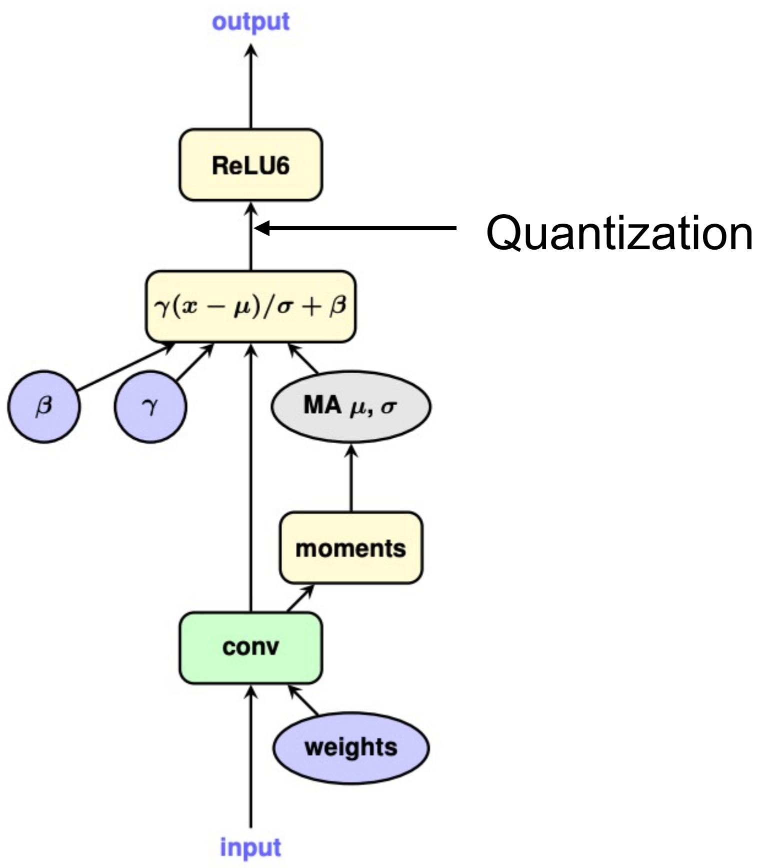


Figure C.5: Convolutional layer with batch normalization: training graph

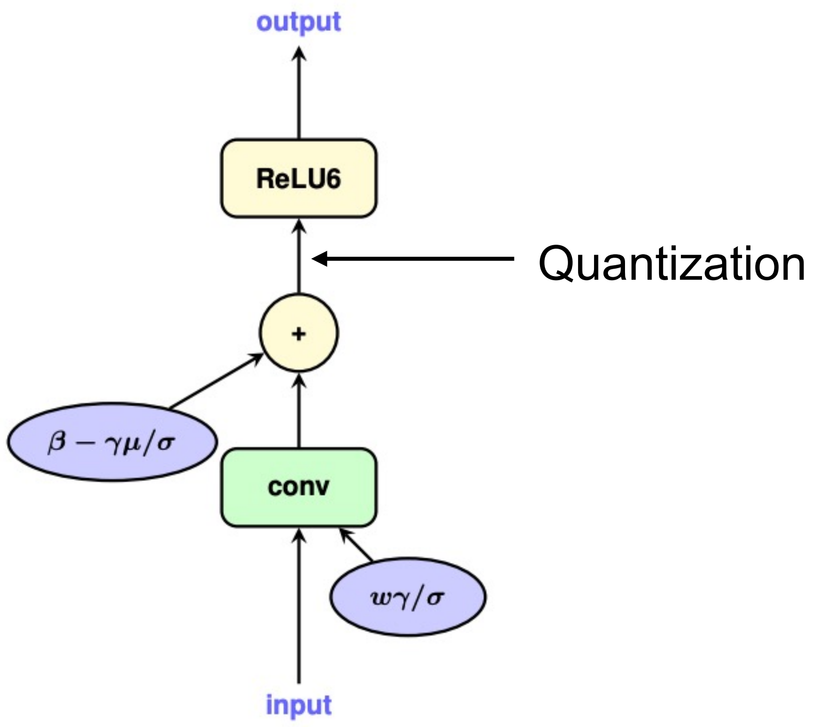
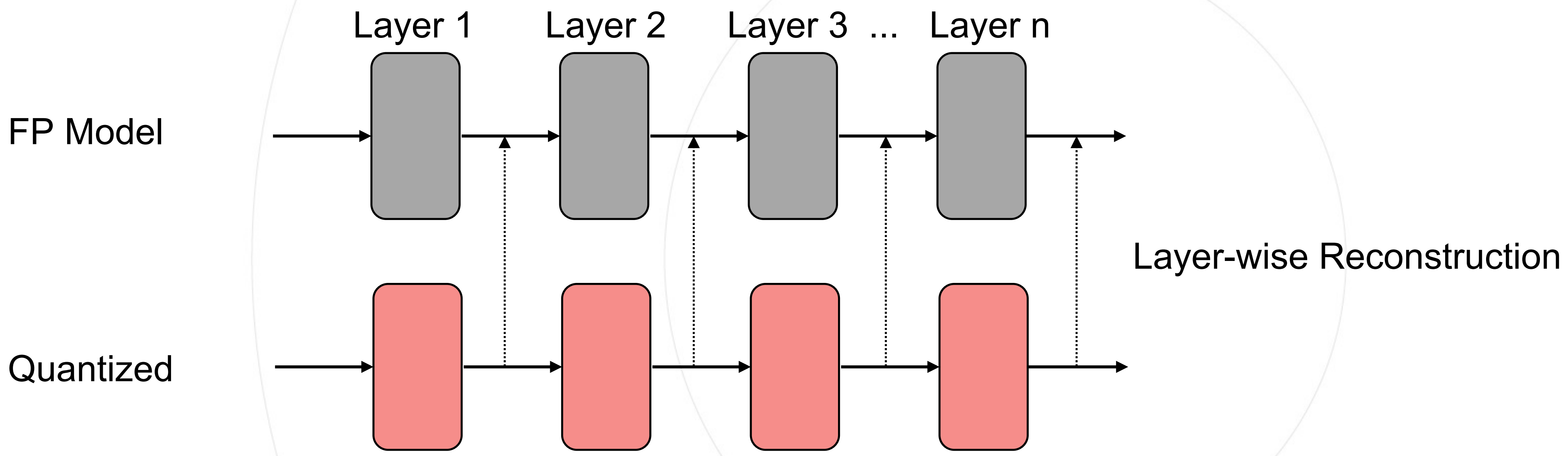


Figure C.6: Convolutional layer with batch normalization: inference graph

Post-Training Finetuning Quantization

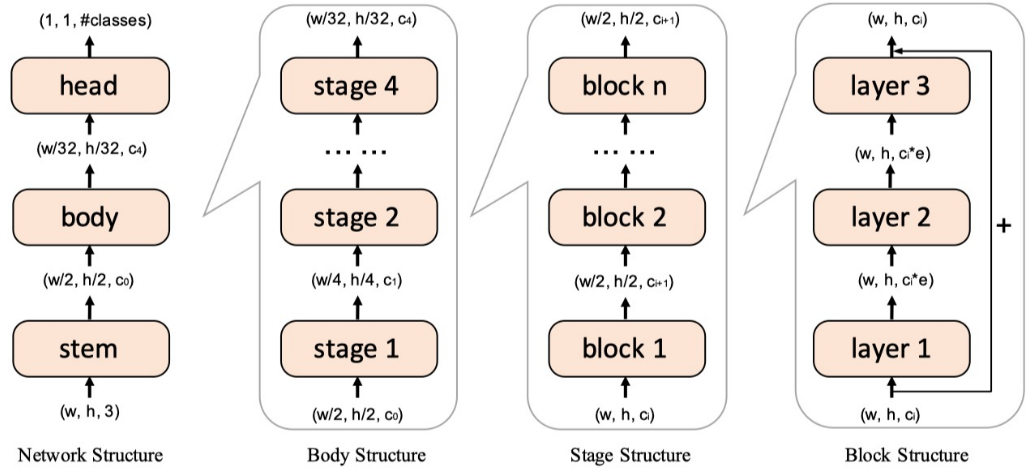
- Is there an intermediate space between PTQ and QAT?

  - Recently, Nagel et al. 2020 and Li et al. 2021 propose to **reconstruct** the internal output of the quantized model to optimize the quantized weights.

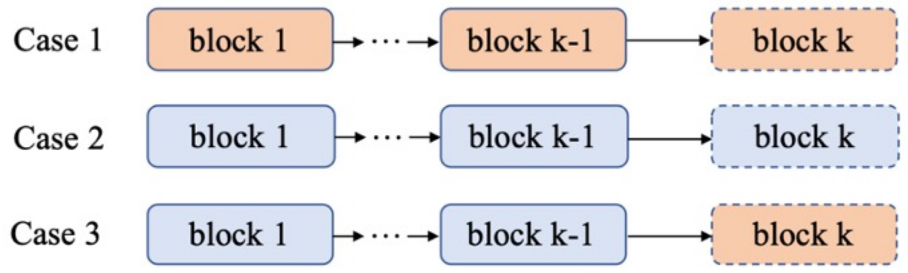


Post-Training Finetuning Quantization

Block-wise Reconstruction (BRECQ)



QDrop



$$\text{QDROP} : u = \begin{cases} 0 & \text{with probability } p \\ \frac{\hat{a}}{a} - 1 & \text{with probability } 1 - p \end{cases}.$$

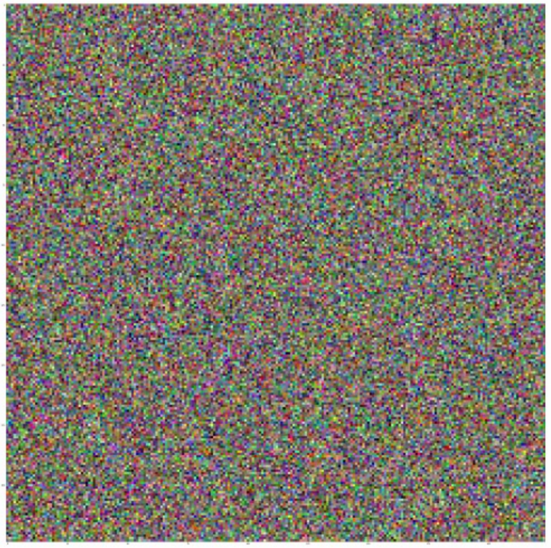
- Experimental Results

Methods	Bits (W/A)	ResNet-18	ResNet-50	MobileNetV2	RegNet-600MF	RegNet-3.2GF	MNasNet-2.0
Full Prec.	32/32	71.08	77.00	72.49	73.71	78.36	76.68
ACIQ-Mix (Banner et al., 2019)	4/4	67.0	73.8	-	-	-	-
ZeroQ (Cai et al., 2020)*	4/4	21.71	2.94	26.24	28.54	12.24	3.89
LAPQ (Nahshan et al., 2019)	4/4	60.3	70.0	49.7	57.71*	55.89*	65.32*
AdaQuant (Hubara et al., 2020)	4/4	67.5	73.7	34.95*	-	-	-
Bit-Split (Wang et al., 2020)	4/4	67.56	73.71	-	-	-	-
BRECQ (Ours)	4/4	**69.60**$\pm$0.04	**75.05**$\pm$0.09	**66.57**$\pm$0.67	**68.33**$\pm$0.28	**74.21**$\pm$0.19	**73.56**$\pm$0.24
ZeroQ (Cai et al., 2020)*	2/4	0.08	0.08	0.10	0.10	0.05	0.12
LAPQ (Nahshan et al., 2019)*	2/4	0.18	0.14	0.13	0.17	0.12	0.18
AdaQuant (Hubara et al., 2020)*	2/4	0.21	0.12	0.10	-	-	-
BRECQ (Ours)	2/4	**64.80**$\pm$0.08	**70.29**$\pm$0.23	**53.34**$\pm$0.15	**59.31**$\pm$0.49	**67.15**$\pm$0.11	**63.01**$\pm$0.35

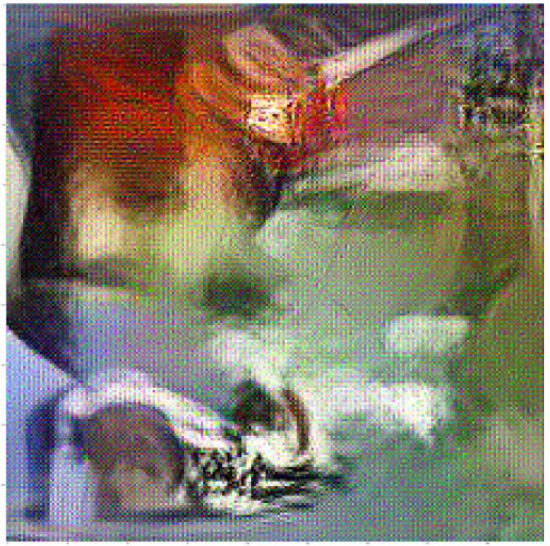
Zero-Shot Quantization

- Zero-Shot Quantization or Data-Free Quantization
  - ZSQ requires no real data for model quantization.
  - Need to **synthesize** artificial data.
  - In Cai et al. 2020, the data is **learned** by gradient descent by matching its statistics variable with BN running mean and variance.

Gaussian Random Data



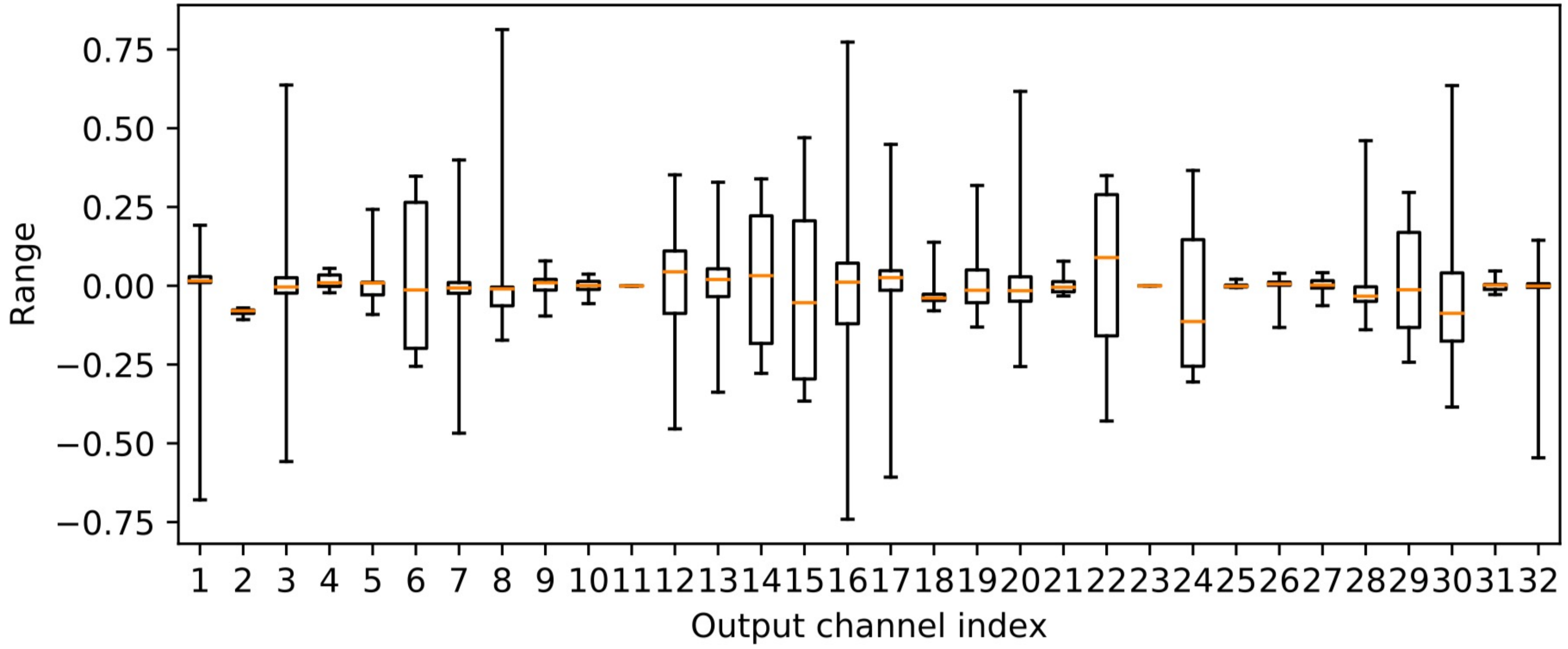
Synthesized Data



Zero-Shot Quantization

- Weight Equalization
  - Modify weights to suitable-for-quantization

$$W_i \leftarrow \frac{\alpha_{i-1}}{\alpha_i} W_i$$



Mixed-Precision

- Why mixed-precision?
  - Different layers have different sensitivities for quantization
  - Different layers have different hardware performances
  - We can assign less bits to non-sensitive layers and high hardware cost layers

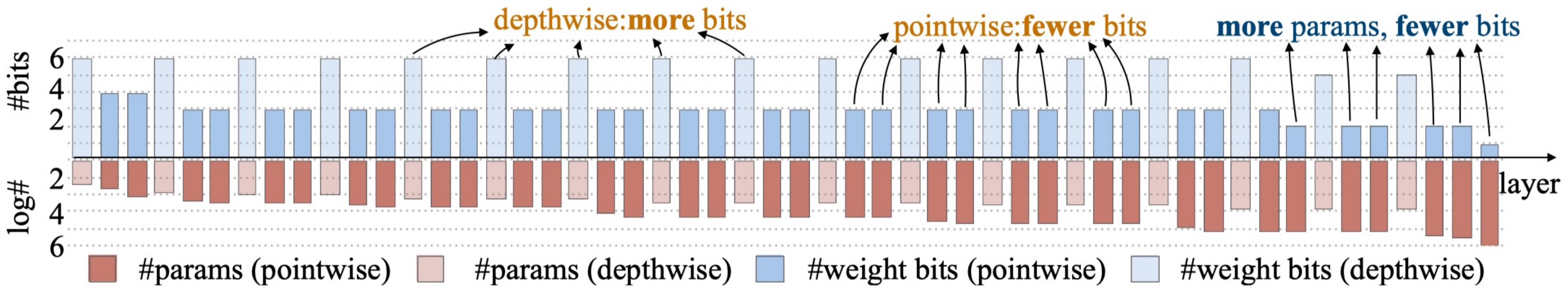


Figure 5: Quantization policy under model size constraints for MobileNet-V2. Our RL agent allocates *more* bits to the depthwise convolutions, since depthwise convolutions have *fewer* number of parameters.

Hardware Quantization Scheme

Hardware	Company	Inference Library	Bit–width	Quantization Scheme
GPU	NVIDIA	TensorRT	8	Uniform symmetric per channel
			FP16	IEEE 754
		NART-QUANT	4/8	Uniform symmetric per layer/channel
3559/3519/3516	Hisilicon	NNIE	8/16	Log
Ceva DSP	Ceva	-	8/16	Uniform asymmetric per layer/channel
Hexagon DSP	Qualcomm	SNPE	8	Uniform asymmetric per layer
Adreno 5/6 serial	Qualcomm	OCL	FP16	IEEE 754 without Subnormal
ARM	ARM	NART-QUANT	2-8	Uniform asymmetric per layer
WUQI	WUQI tech.	WUQI sdk	8/16	Ristretto
SigmaStar	SigmaStar Technology	SigmaStar sdk	8/16	Uniform symmetric weight(per channel)，symmetric activation
Ascend 310	HUAWEI	ACL	8	Uniform asymmetric per channel
			FP16	IEEE 754
Ambarella	Ambarella	CVFlow	8/16	Ristretto
			FP16	IEEE 754
FPGA	Xilinx	Vitis-AI	Int8	Ristretto

MQBench

✅ 支持近10种硬件平台后端量化，包括GPU、Vitis、DSP、NNIE、TVM、OpenVINO、Tengine等大类

✅ 同时支持CV和NLP任务

✅ 一键量化

✅ 支持了PTQ和QAT的SOTA算法

贡献 OpenVINO Tengine

使用 地平线 Horizon Robotics 京东

7种统计算法 | 8种量化算法 | Best Practice

https://github.com/ModelTC/MQBench

高级离线量化： https://mqbench.readthedocs.io/en/latest/user_guide/PTQ/advanced.html

支持的典型算法举例：

LSQ | DoReFa | PACT | AdaRound | BRECQ | QDrop

Highlights

What is Pruning

- Pruning
  - The process of **removing weight connections** in a network to increase inference speed and decrease model storage size.[1]
  - Removing unused parameters from the ***over-parameterized*** network.[1]

- Levels of Pruning
  - Channel/Filter; Layer; Block

[1] https://neuralmagic.com/

What is Pruning

- Pipeline

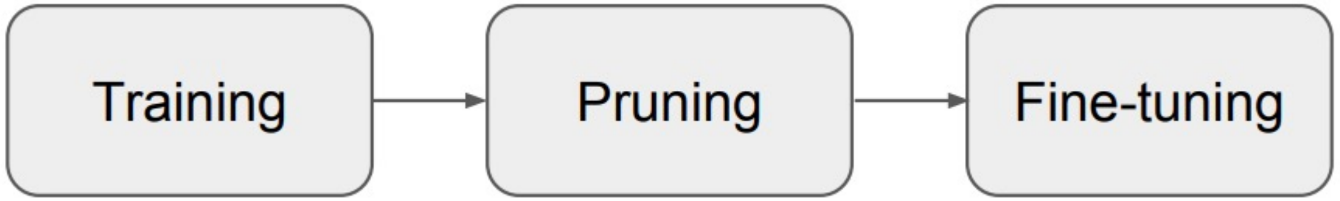
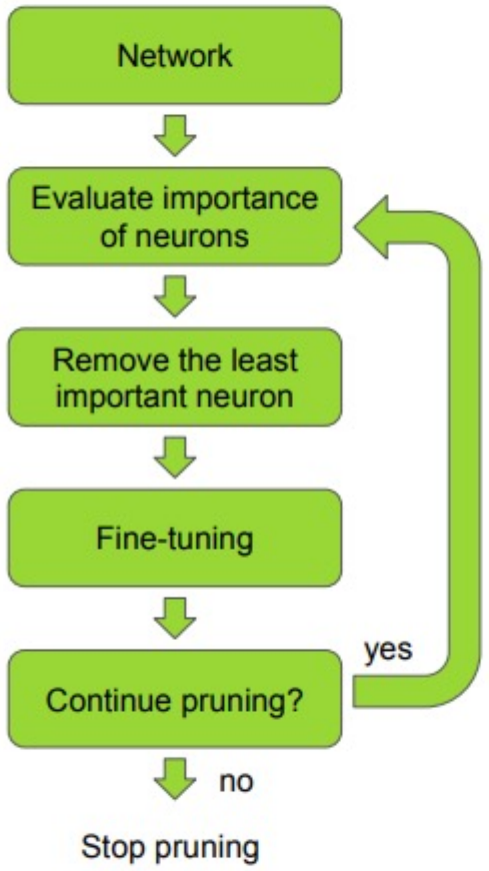


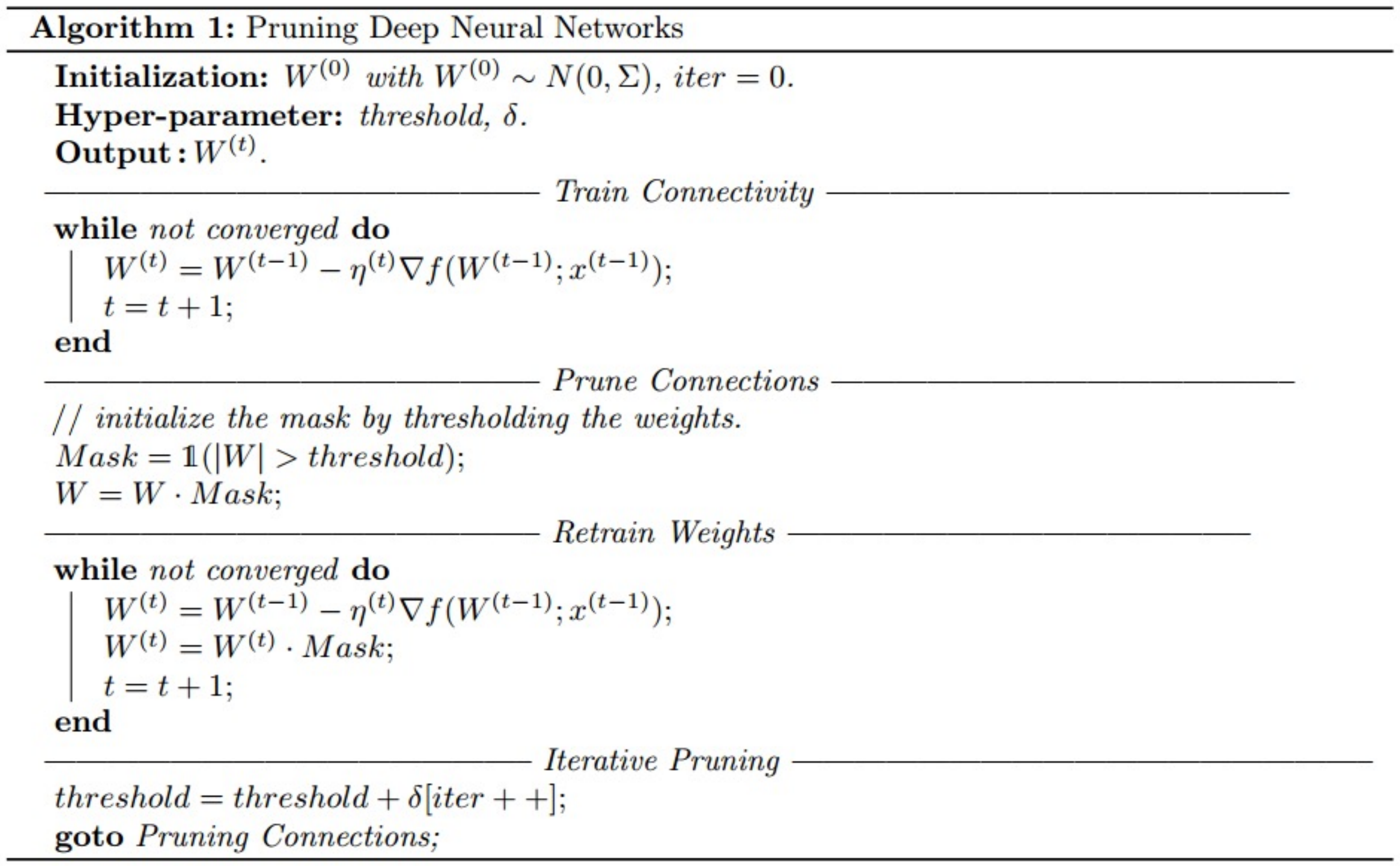
Figure 1: A typical three-stage network pruning pipeline.


Network
Evaluate importance of neurons
Remove the least important neuron
Fine-tuning
Continue pruning?
yes
no
Stop pruning

One example of iterative pruning

Ref. [1] Liu, et al. Rethinking the Value of Network Pruning. ICLR2019
[2] Molchanov, et al. Pruning Convolutional Neural Networks for Resource Efficient Inference. ICLR2017

What is Pruning

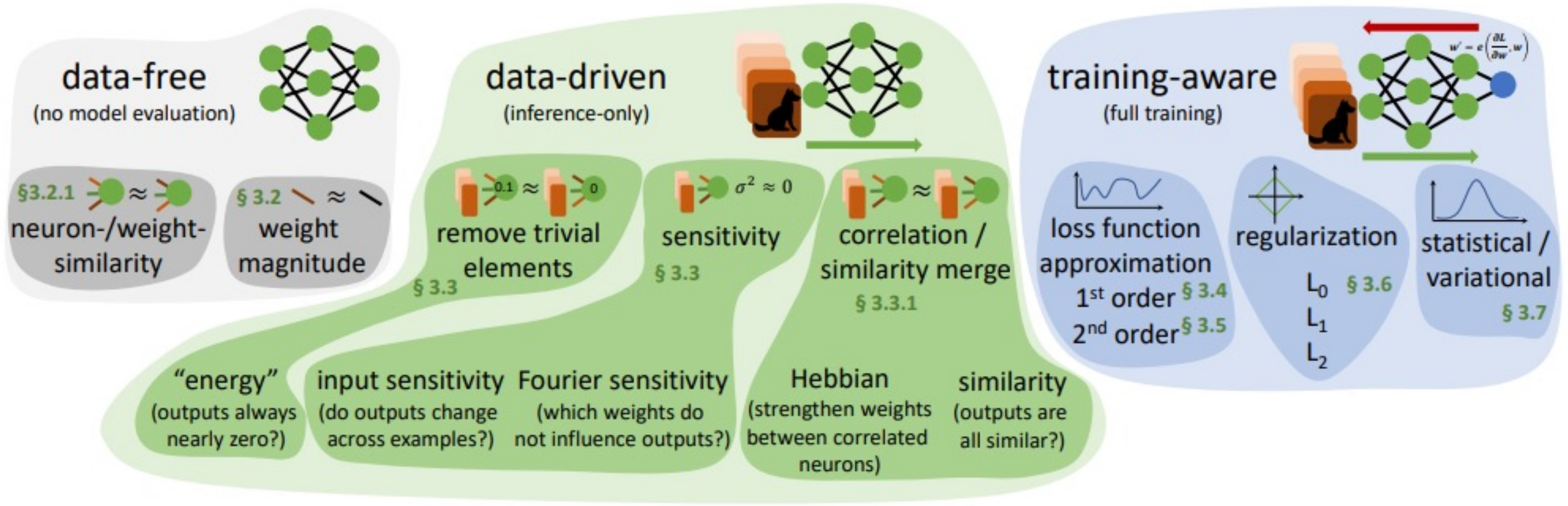


```
Algorithm 1: Pruning Deep Neural Networks
Initialization: $W^{(0)}$ with $W^{(0)} \sim N(0, \Sigma)$, $iter = 0$.
Hyper-parameter: $threshold$, $\delta$.
Output: $W^{(t)}$.
―――――――――― Train Connectivity ――――――――――
while not converged do
    $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
    $t = t + 1$;
end
―――――――――― Prune Connections ――――――――――
// initialize the mask by thresholding the weights.
$Mask = \mathbb{1}(|W| > threshold)$;
$W = W \cdot Mask$;
―――――――――― Retrain Weights ――――――――――
while not converged do
    $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
    $W^{(t)} = W^{(t)} \cdot Mask$;
    $t = t + 1$;
end
―――――――――― Iterative Pruning ――――――――――
$threshold = threshold + \delta[iter + +]$;
goto Pruning Connections;
```

Ref. [1] Han, et al. Learning both Weights and Connections for Efficient Neural Networks. NIPS2015

What is Pruning

- Sparsity Structure
  - Structured
  - Unstructured
- Schemes
  - Data-free
  - Data-driven
  - Training-aware



data-free
(no model evaluation)
§3.2.1
neuron-/weight-similarity
§ 3.2
weight magnitude
data-driven
(inference-only)
remove trivial elements
§ 3.3
sensitivity
§ 3.3
correlation / similarity merge
§ 3.3.1
"energy" (outputs always nearly zero?)
input sensitivity (do outputs change across examples?)
Fourier sensitivity (which weights do not influence outputs?)
Hebbian (strengthen weights between correlated neurons)
similarity (outputs are all similar?)
training-aware
(full training)
loss function approximation
1st order § 3.4
2nd order § 3.5
regularization
L0 § 3.6
L1
L2
statistical / variational
§ 3.7

Fig. 10. Overview of schemes to select candidate elements for removal during sparsification

Ref. [1] arxiv.org/abs/2102.00554

Data-free Pruning

- L1-norm based[1]
- Similarity based[2]

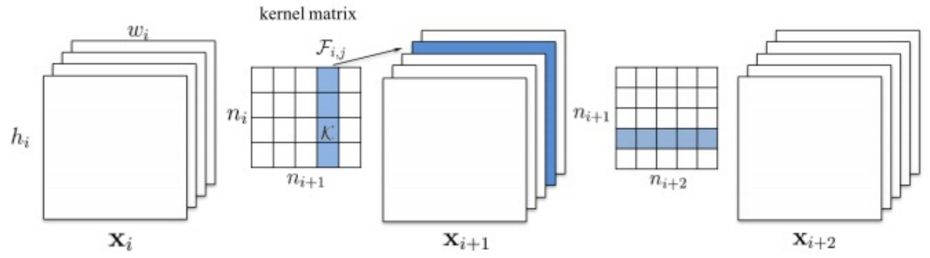


Figure 1: Pruning a filter results in removal of its corresponding feature map and related kernels in the next layer.

$$z_n = a_1 h(W_1^T X) + \ldots + a_i h(W_i^T X) + \ldots + a_j h(W_j^T X) + \ldots$$

$$z_{n-1} = a_1 h(W_1^T X) + \ldots + (a_i + a_j) h(W_i^T X) + \ldots$$

$$\Big\downarrow$$

$$min(E\langle (z_n - z_{n-1})^2 \rangle) \le min(\langle a_j^2 \rangle \, \|\varepsilon_{i,j}\|_2^2) \; E\|X\|_2^2$$

$$\Big\downarrow$$

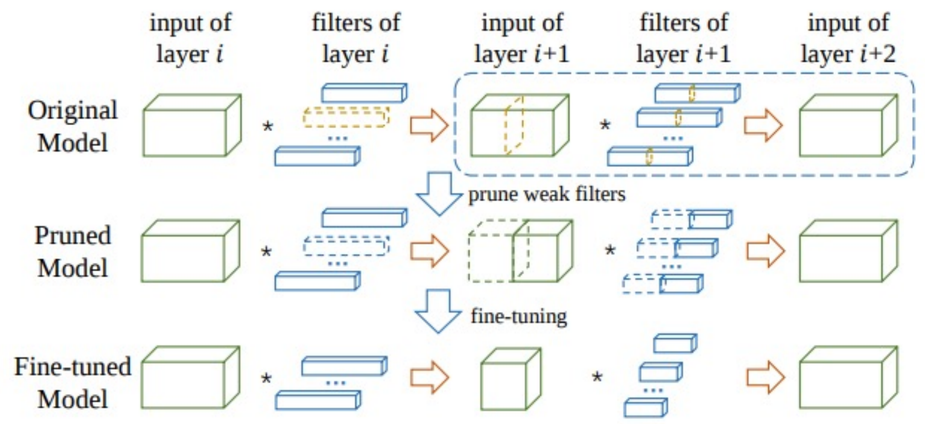
$$s_{i,j} = \langle a_j^2 \rangle \, \|\varepsilon_{i,j}\|_2^2.$$

1. Compute the saliency $s_{i,j}$ for all possible values of (i,j). It can be stored as a square matrix M, with dimension equal to the number of neurons in the layer being considered.
2. Pick the minimum entry in the matrix. Let it's indices be (i', j'). Delete the j'^{th} neuron, and update $a_{i'} \leftarrow a_{i'} + a_{j'}$.
3. Update M by removing the j'^{th} column and row, and updating the i'^{th} column (to account for the updated $a_{i'}$.)

Ref. [1] Li, et al. Pruning Filters for efficient convnets. ICLR2017
[2] Data-free parameter pruning for Deep Neural Networks. BMVC2015.

Data-driven Pruning

- ThiNet[1]
  - least effect on the next layer’ s output
- Regression based feature reconstruction[2]
  - LASSO



$$\begin{aligned}&\arg\min_{S}\sum_{i=1}^{m}\left(\hat{y}_i-\sum_{j\in S}\hat{\mathbf{x}}_{i,j}\right)^2 \qquad (5)\\ &\text{s.t.}\quad |S| = C\times r,\quad S\subset\{1,2,\ldots,C\}.\end{aligned}$$

Ref. [1] Luo, et al. A filter level of pruning method for deep neural network compression. ICCV2017.
[2] He, et al. Channel pruning for accelerating very deep neural networks. ICCV2017

Training-aware: Optimal Brain Damage[1]

Cost Function *E* = Cost(Train) + R(Network Complexity)

Approximate E by a Taylor series.

$$\delta E = \frac{1}{2}\sum_i h_{ii}\delta u_i^2 \qquad \frac{\partial^2 E}{\partial a_i^2} = f'(a_i)^2 \sum_l w_{li}^2 \frac{\partial^2 E}{\partial a_l^2} + f''(a_i)\frac{\partial E}{\partial x_i}$$

1. Choose a reasonable network architecture
2. Train the network until a reasonable solution is obtained
3. Compute the second derivatives h_{kk} for each parameter
4. Compute the saliencies for each parameter: $s_k = h_{kk}u_k^2/2$
5. Sort the parameters by saliency and delete some low-saliency parameters
6. Iterate to step 2

Ref. [1] LeCun, et al. http://yann.lecun.com/exdb/publis/pdf/lecun-90b.pdf

Training-aware: Optimal Brain Damage[1]

$$x_i = f(a_i) \qquad \text{and} \qquad a_i = \sum_j w_{ij} x_j$$

$$\delta E = \sum_i g_i \delta u_i + \boxed{\frac{1}{2} \sum_i h_{ii} \delta u_i^2} + \frac{1}{2} \sum_{i \neq j} h_{ij} \delta u_i \delta u_j + O(||\delta U||^3)$$

$$\frac{\partial^2 E}{\partial a_i^2} = f'(a_i)^2 \sum_l w_{li}^2 \frac{\partial^2 E}{\partial a_l^2} + f''(a_i) \frac{\partial E}{\partial x_i}$$

Ref. [1] LeCun, et al. http://yann.lecun.com/exdb/publis/pdf/lecun-90b.pdf

Training-aware Pruning

- Network Slimming[1]
  - pruning by channel scaling factors in the following BN layer

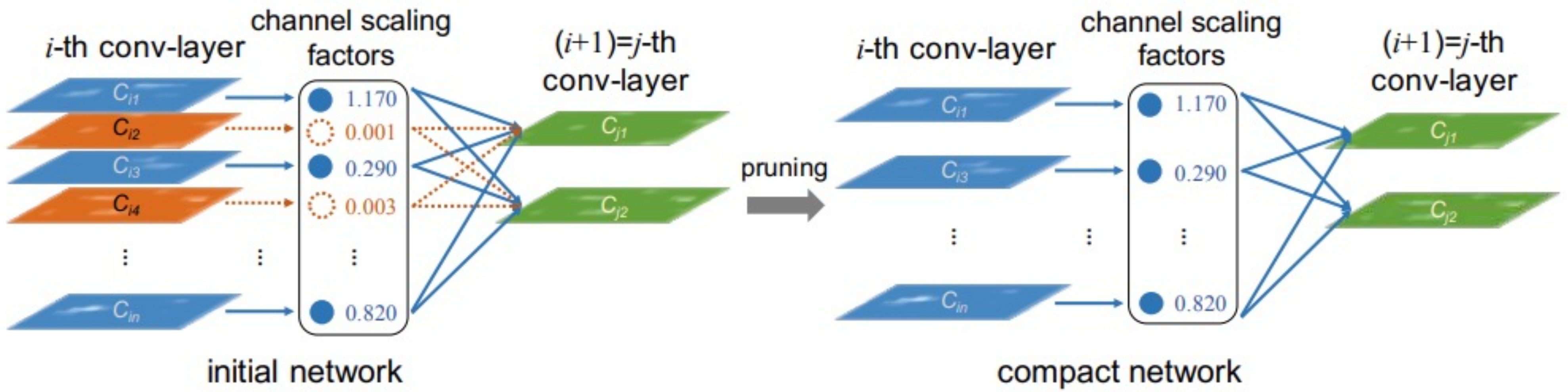
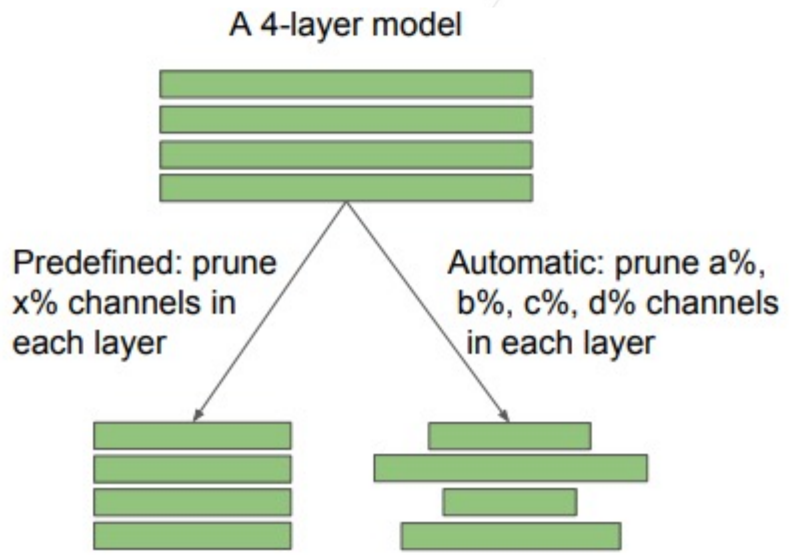
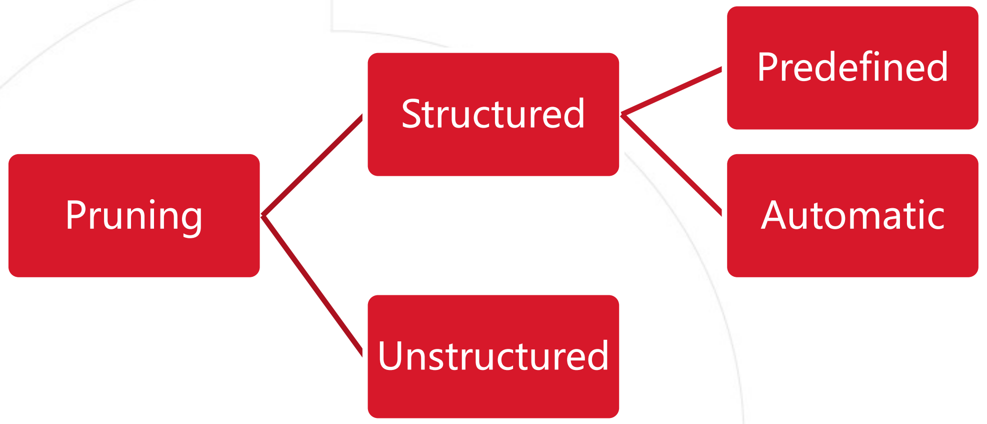


Figure 1: We associate a scaling factor (reused from a batch normalization layer) with each channel in convolutional layers. Sparsity regularization is imposed on these scaling factors during training to automatically identify unimportant channels. The channels with small scaling factor values (in orange color) will be pruned (left side). After pruning, we obtain compact models (right side), which are then fine-tuned to achieve comparable (or even higher) accuracy as normally trained full network.

Ref. [1] Liu, et al. Learning Efficient Convolutional Networks through Network Slimming

Rethinking the Value of Network Pruning


A 4-layer model
Predefined: prune x% channels in each layer
Automatic: prune a%, b%, c%, d% channels in each layer

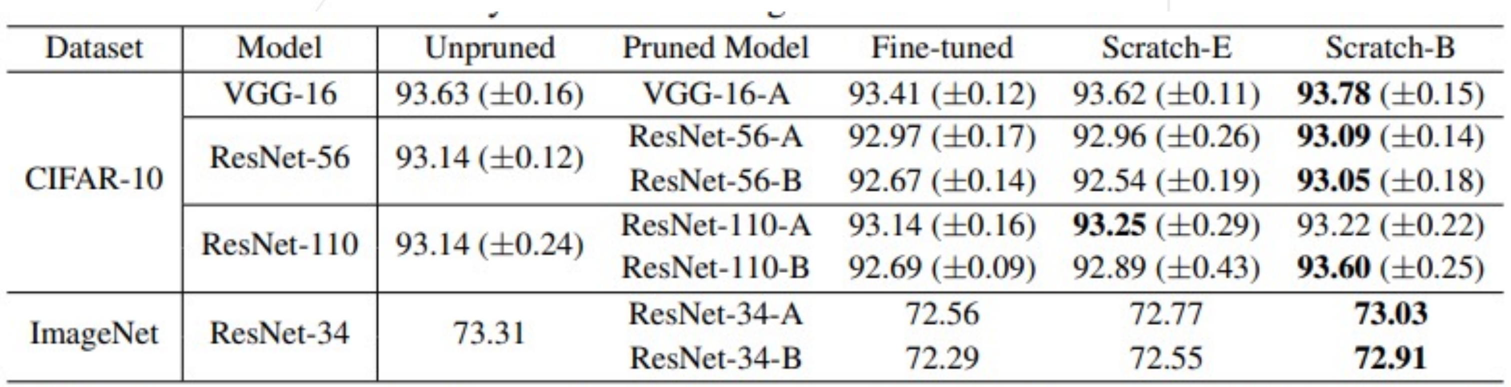


Ref. [1]] Liu, et al. Rethinking the Value of Network Pruning. ICLR2019

Rethinking the Value of Network Pruning

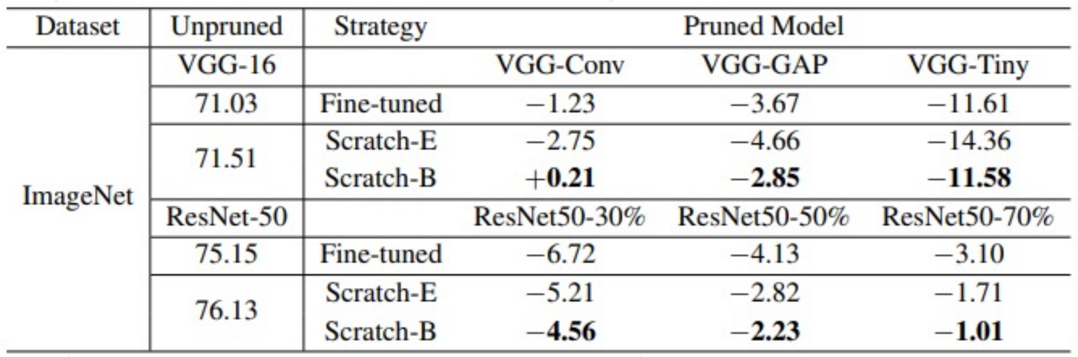
Experiments of Predefined Structured Pruning

L1-norm



Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	93.62 (±0.11)	**93.78** (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.97 (±0.17)	92.96 (±0.26)	**93.09** (±0.14)
			ResNet-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

ThiNet



Dataset	Unpruned	Strategy	Pruned Model		
ImageNet	VGG-16		VGG-Conv	VGG-GAP	VGG-Tiny
	71.03	Fine-tuned	−1.23	−3.67	−11.61
	71.51	Scratch-E	−2.75	−4.66	−14.36
		Scratch-B	**+0.21**	**−2.85**	**−11.58**
	ResNet-50		ResNet50-30%	ResNet50-50%	ResNet50-70%
	75.15	Fine-tuned	−6.72	−4.13	−3.10
	76.13	Scratch-E	−5.21	−2.82	−1.71
		Scratch-B	**−4.56**	**−2.23**	**−1.01**

Scratch-E: epochs
Scratch-B: FLOPs budget

Ref. [1]] Liu, et al. Rethinking the Value of Network Pruning. ICLR2019

Rethinking the Value of Network Pruning

Experiments of Automatic Structured Pruning

Network Slimming

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.53 (±0.16)	70%	93.60 (±0.16)	93.30 (±0.11)	**93.81** (±0.14)
	PreResNet-164	95.04 (±0.16)	40%	94.77 (±0.12)	94.70 (±0.11)	**94.90** (±0.04)
			60%	94.23 (±0.21)	94.58 (±0.18)	**94.71** (±0.21)
	DenseNet-40	94.10 (±0.12)	40%	94.00 (±0.20)	93.68 (±0.18)	**94.06** (±0.12)
			60%	**93.87** (±0.13)	93.58 (±0.21)	93.85 (±0.25)
CIFAR-100	VGG-19	72.63 (±0.21)	50%	72.32 (±0.28)	71.94 (±0.17)	**73.08** (±0.22)
	PreResNet-164	76.80 (±0.19)	40%	76.22 (±0.20)	76.36 (±0.32)	**76.68** (±0.35)
			60%	74.17 (±0.33)	75.05 (± 0.08)	**75.73** (±0.29)
	DenseNet-40	73.82 (±0.34)	40%	**73.35** (±0.17)	73.24 (±0.29)	73.19 (±0.26)
			60%	72.46 (±0.22)	72.62 (±0.36)	**72.91** (±0.34)
ImageNet	VGG-11	70.84	50%	68.62	70.00	**71.18**

Sparse Structure Selection[2]

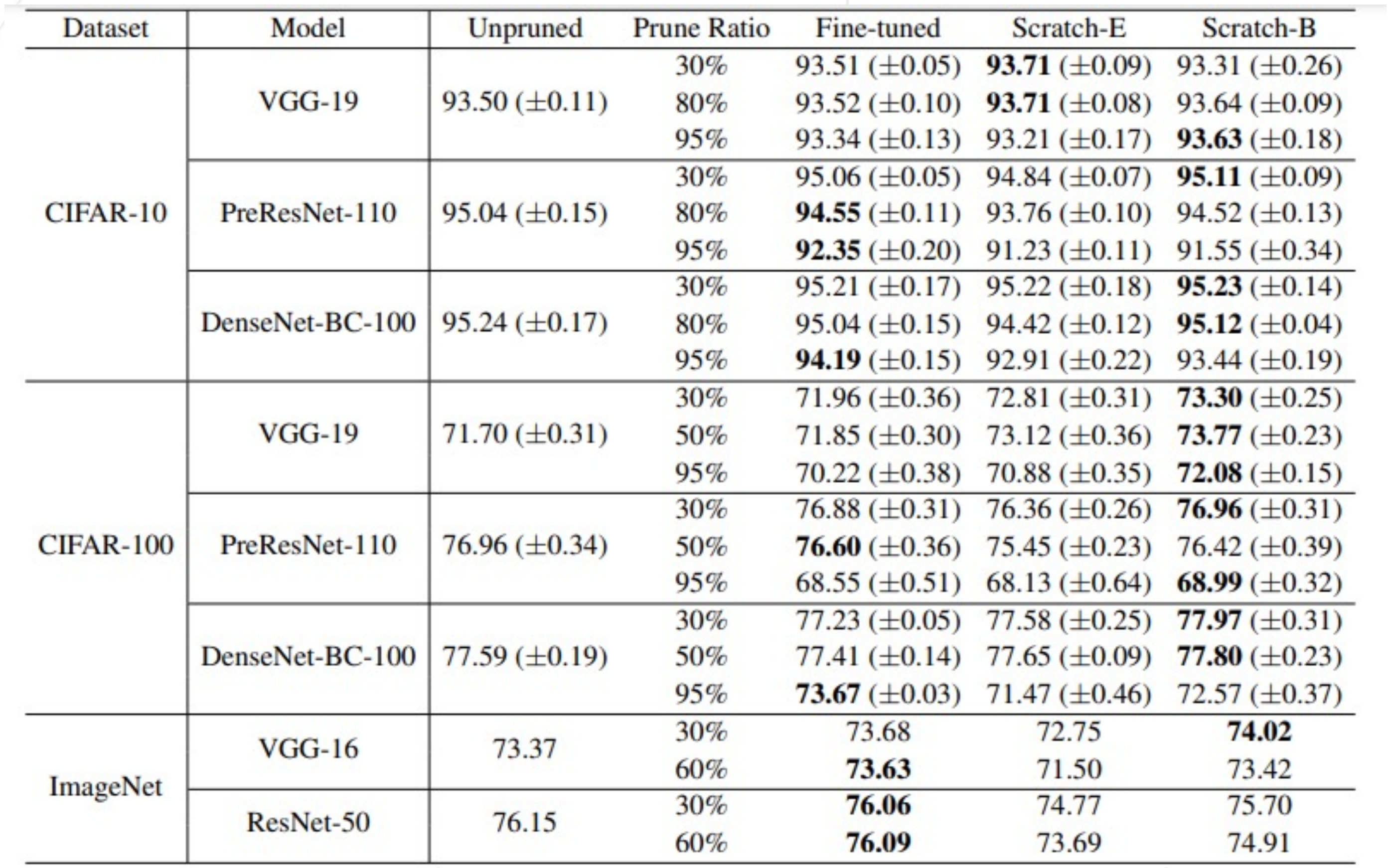
Dataset	Model	Unpruned	Pruned Model	Pruned	Scratch-E	Scratch-B
ImageNet	ResNet-50	76.12	ResNet-41	75.44	75.61	**76.17**
			ResNet-32	74.18	73.77	**74.67**
			ResNet-26	71.82	72.55	**73.41**

Ref. [1]] Liu, et al. Rethinking the Value of Network Pruning. ICLR2019
[2] Huang et al. Data-Driven Sparse Structure Selection for Deep Neural Networks. ECCV2018

Rethinking the Value of Network Pruning

Experiments of Unstructured Pruning

Magnitude-based Pruning[2]

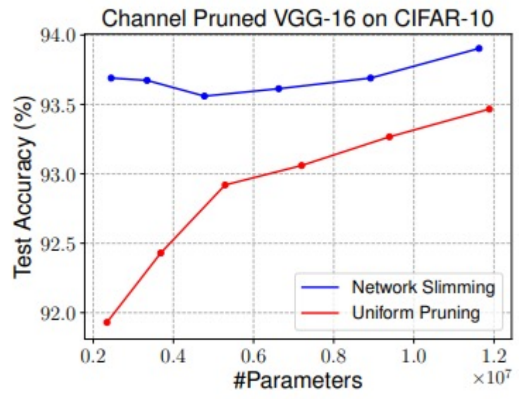


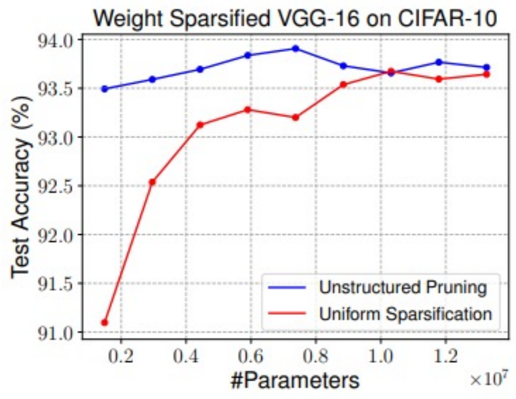
Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.50 (±0.11)	30%	93.51 (±0.05)	**93.71** (±0.09)	93.31 (±0.26)
			80%	93.52 (±0.10)	**93.71** (±0.08)	93.64 (±0.09)
			95%	93.34 (±0.13)	93.21 (±0.17)	**93.63** (±0.18)
	PreResNet-110	95.04 (±0.15)	30%	95.06 (±0.05)	94.84 (±0.07)	**95.11** (±0.09)
			80%	**94.55** (±0.11)	93.76 (±0.10)	94.52 (±0.13)
			95%	**92.35** (±0.20)	91.23 (±0.11)	91.55 (±0.34)
	DenseNet-BC-100	95.24 (±0.17)	30%	95.21 (±0.17)	95.22 (±0.18)	**95.23** (±0.14)
			80%	95.04 (±0.15)	94.42 (±0.12)	**95.12** (±0.04)
			95%	**94.19** (±0.15)	92.91 (±0.22)	93.44 (±0.19)
CIFAR-100	VGG-19	71.70 (±0.31)	30%	71.96 (±0.36)	72.81 (±0.31)	**73.30** (±0.25)
			50%	71.85 (±0.30)	73.12 (±0.36)	**73.77** (±0.23)
			95%	70.22 (±0.38)	70.88 (±0.35)	**72.08** (±0.15)
	PreResNet-110	76.96 (±0.34)	30%	76.88 (±0.31)	76.36 (±0.26)	**76.96** (±0.31)
			50%	**76.60** (±0.36)	75.45 (±0.23)	76.42 (±0.39)
			95%	68.55 (±0.51)	68.13 (±0.64)	**68.99** (±0.32)
	DenseNet-BC-100	77.59 (±0.19)	30%	77.23 (±0.05)	77.58 (±0.25)	**77.97** (±0.31)
			50%	77.41 (±0.14)	77.65 (±0.09)	**77.80** (±0.23)
			95%	**73.67** (±0.03)	71.47 (±0.46)	72.57 (±0.37)
ImageNet	VGG-16	73.37	30%	73.68	72.75	**74.02**
			60%	**73.63**	71.50	73.42
	ResNet-50	76.15	30%	**76.06**	74.77	75.70
			60%	**76.09**	73.69	74.91

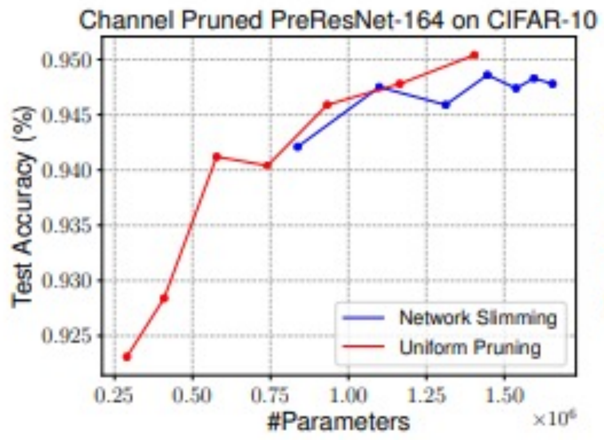
Ref. [1]] Liu, et al. Rethinking the Value of Network Pruning. ICLR2019
[2] Han, et al. Learning both Weights and Connections for Efficient Neural Networks. NIPS2015

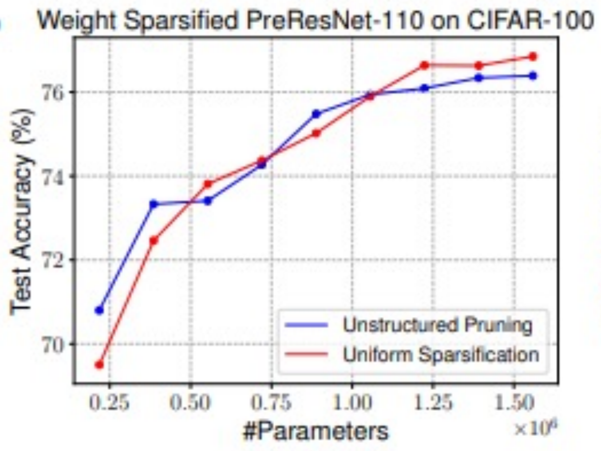
Rethinking the Value of Network Pruning

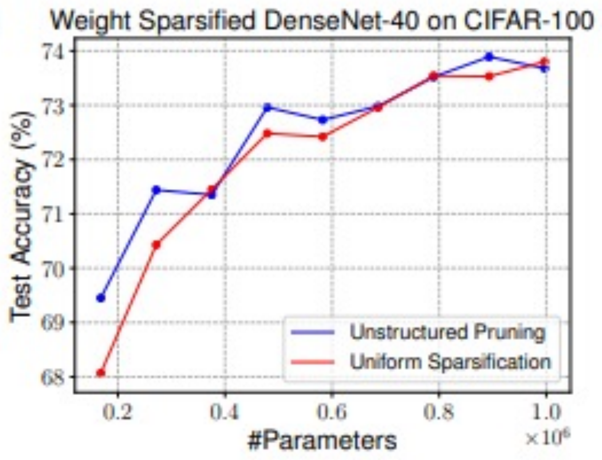
Analysis of Pruned Architectures







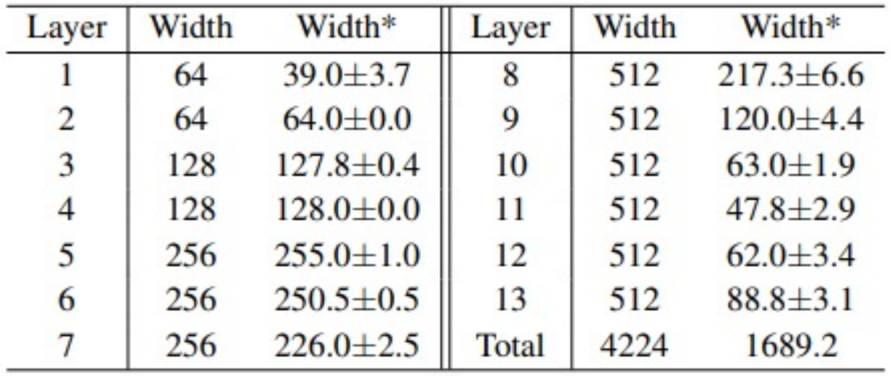




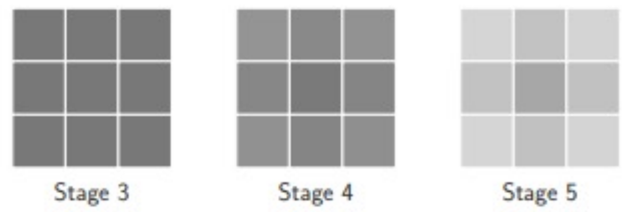
Ref. [1]] Liu, et al. Rethinking the Value of Network Pruning. ICLR2019

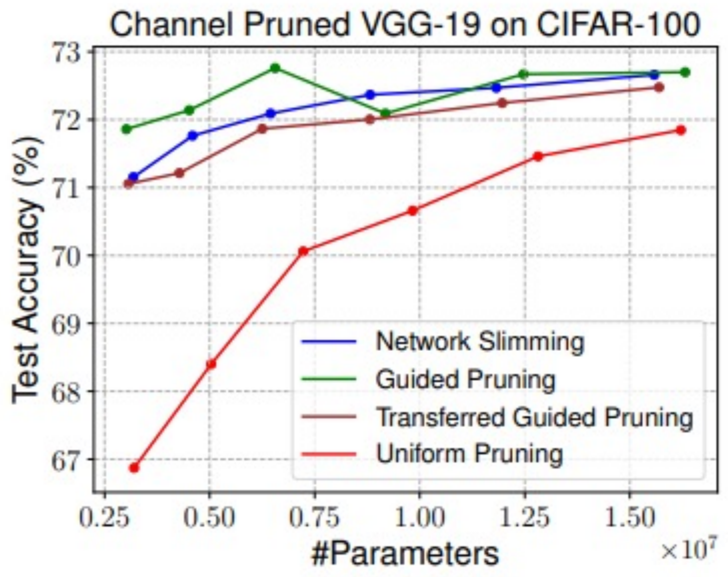
Rethinking the Value of Network Pruning

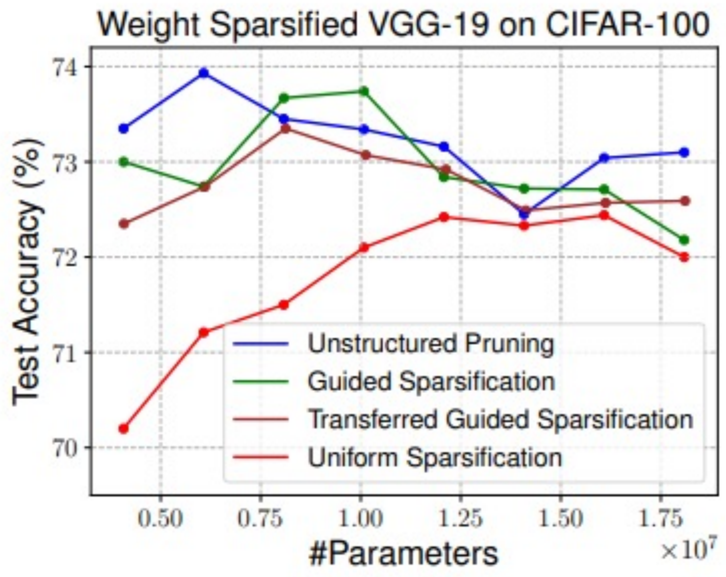
Sparsity Patterns & Guided Pruning



Layer	Width	Width*	Layer	Width	Width*
1	64	39.0±3.7	8	512	217.3±6.6
2	64	64.0±0.0	9	512	120.0±4.4
3	128	127.8±0.4	10	512	63.0±1.9
4	128	128.0±0.0	11	512	47.8±2.9
5	256	255.0±1.0	12	512	62.0±3.4
6	256	250.5±0.5	13	512	88.8±3.1
7	256	226.0±2.5	Total	4224	1689.2



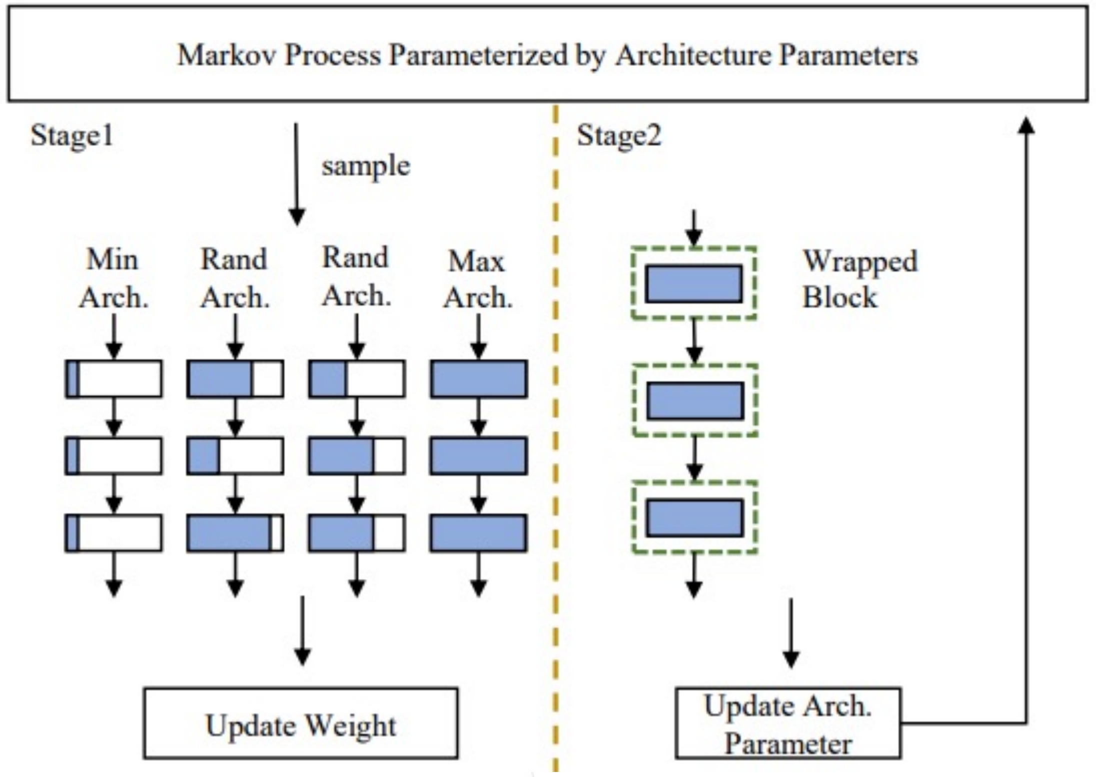




Ref. [1]] Liu, et al. Rethinking the Value of Network Pruning. ICLR2019

Pruning as NAS

DMCP[1]



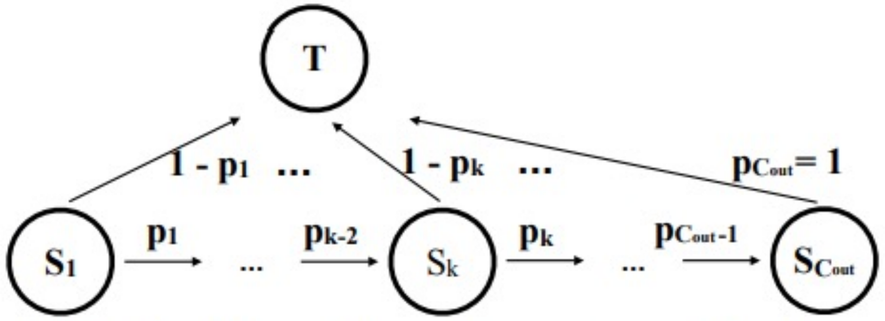
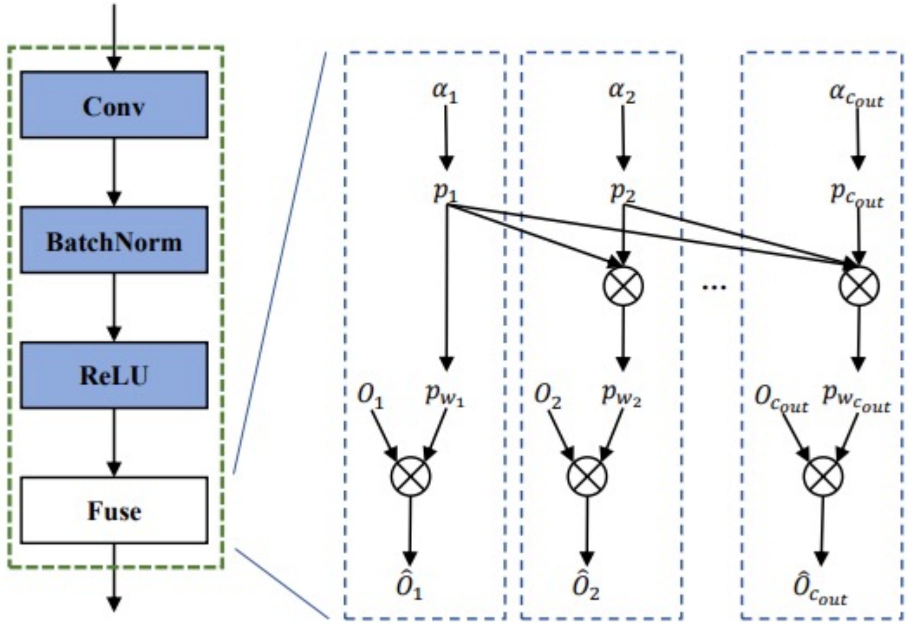


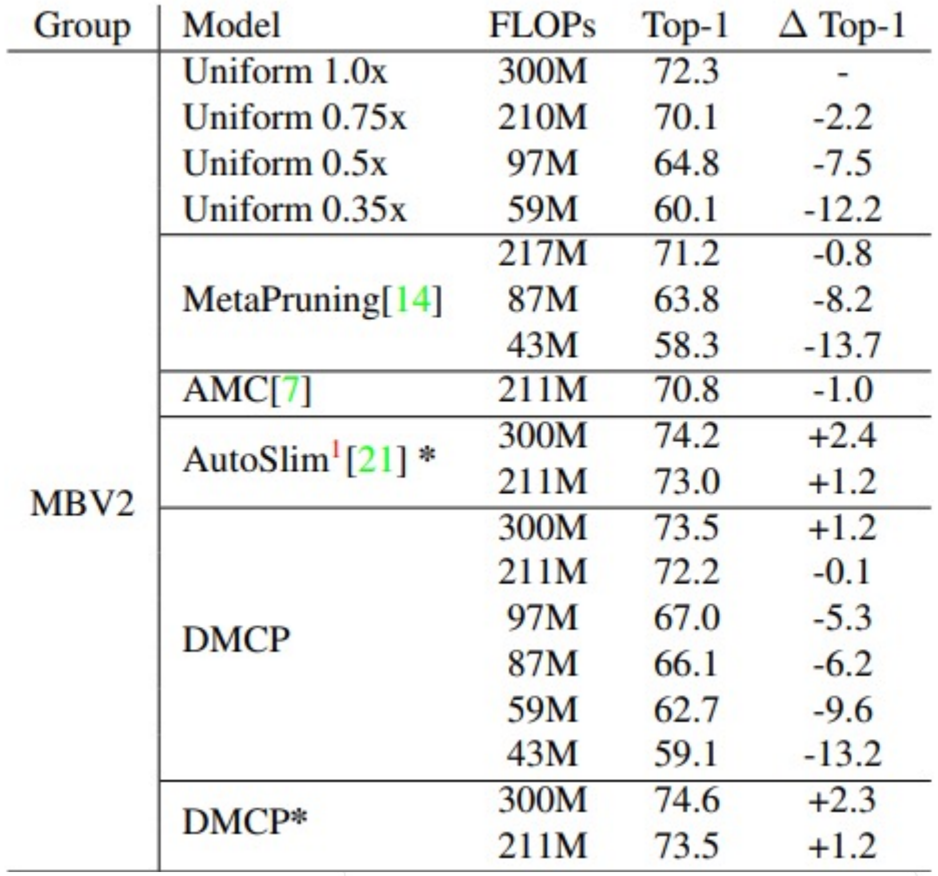
Figure 3. The Modeling of channel pruning as a Markov process.



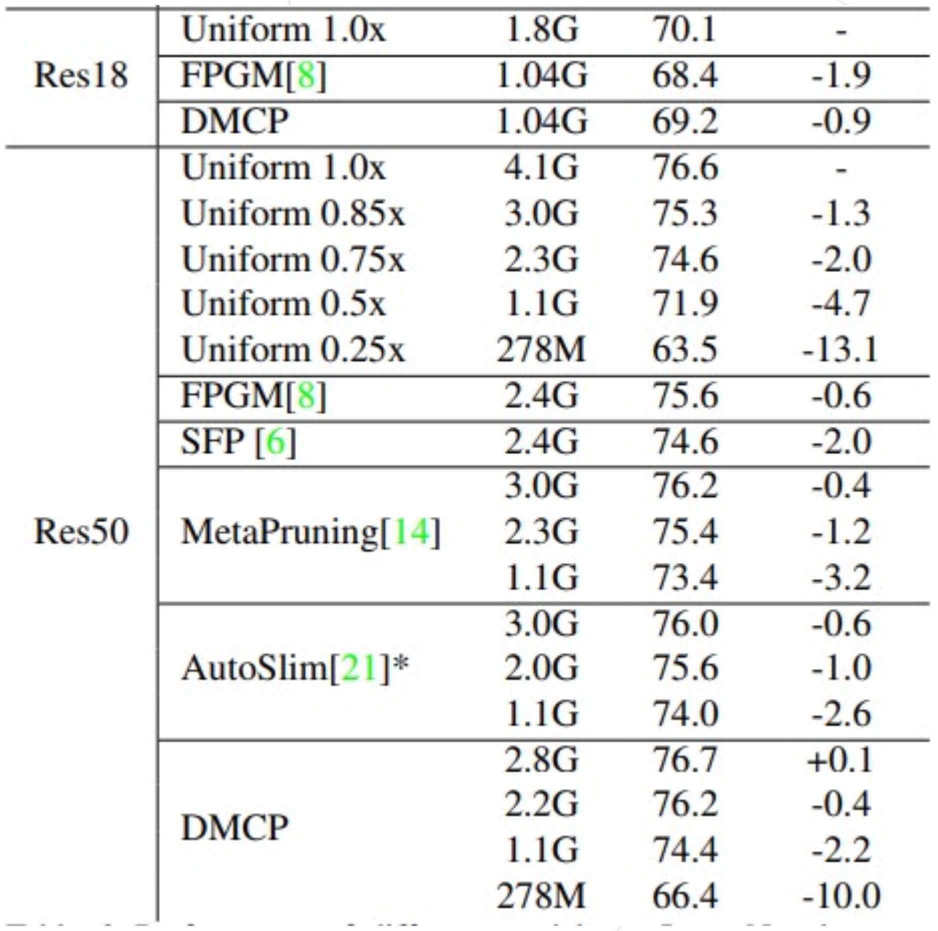
Ref. [1]] Guo, et al. DMCP: Differentiable Markov Channel Pruning for Neural Networks. CVPR2020

Pruning as NAS

DMCP[1]



Group	Model	FLOPs	Top-1	Δ Top-1
MBV2	Uniform 1.0x	300M	72.3	-
	Uniform 0.75x	210M	70.1	-2.2
	Uniform 0.5x	97M	64.8	-7.5
	Uniform 0.35x	59M	60.1	-12.2
	MetaPruning[14]	217M	71.2	-0.8
		87M	63.8	-8.2
		43M	58.3	-13.7
	AMC[7]	211M	70.8	-1.0
	AutoSlim[1] [21] *	300M	74.2	+2.4
		211M	73.0	+1.2
	DMCP	300M	73.5	+1.2
		211M	72.2	-0.1
		97M	67.0	-5.3
		87M	66.1	-6.2
		59M	62.7	-9.6
		43M	59.1	-13.2
	DMCP*	300M	74.6	+2.3
		211M	73.5	+1.2



Res18	Uniform 1.0x	1.8G	70.1	-
	FPGM[8]	1.04G	68.4	-1.9
	DMCP	1.04G	69.2	-0.9
Res50	Uniform 1.0x	4.1G	76.6	-
	Uniform 0.85x	3.0G	75.3	-1.3
	Uniform 0.75x	2.3G	74.6	-2.0
	Uniform 0.5x	1.1G	71.9	-4.7
	Uniform 0.25x	278M	63.5	-13.1
	FPGM[8]	2.4G	75.6	-0.6
	SFP [6]	2.4G	74.6	-2.0
	MetaPruning[14]	3.0G	76.2	-0.4
		2.3G	75.4	-1.2
		1.1G	73.4	-3.2
	AutoSlim[21]*	3.0G	76.0	-0.6
		2.0G	75.6	-1.0
		1.1G	74.0	-2.6
	DMCP	2.8G	76.7	+0.1
		2.2G	76.2	-0.4
		1.1G	74.4	-2.2
		278M	66.4	-10.0

Ref. [1]] Guo, et al. DMCP: Differentiable Markov Channel Pruning for Neural Networks. CVPR2020

Highlights

Basic Concept

- What is Knowledge Distillation?
  - Knowledge Distillation distill the knowledge from a larger deep neural network into a small network
  - Three key component: teacher model, student model and knowledge transfer

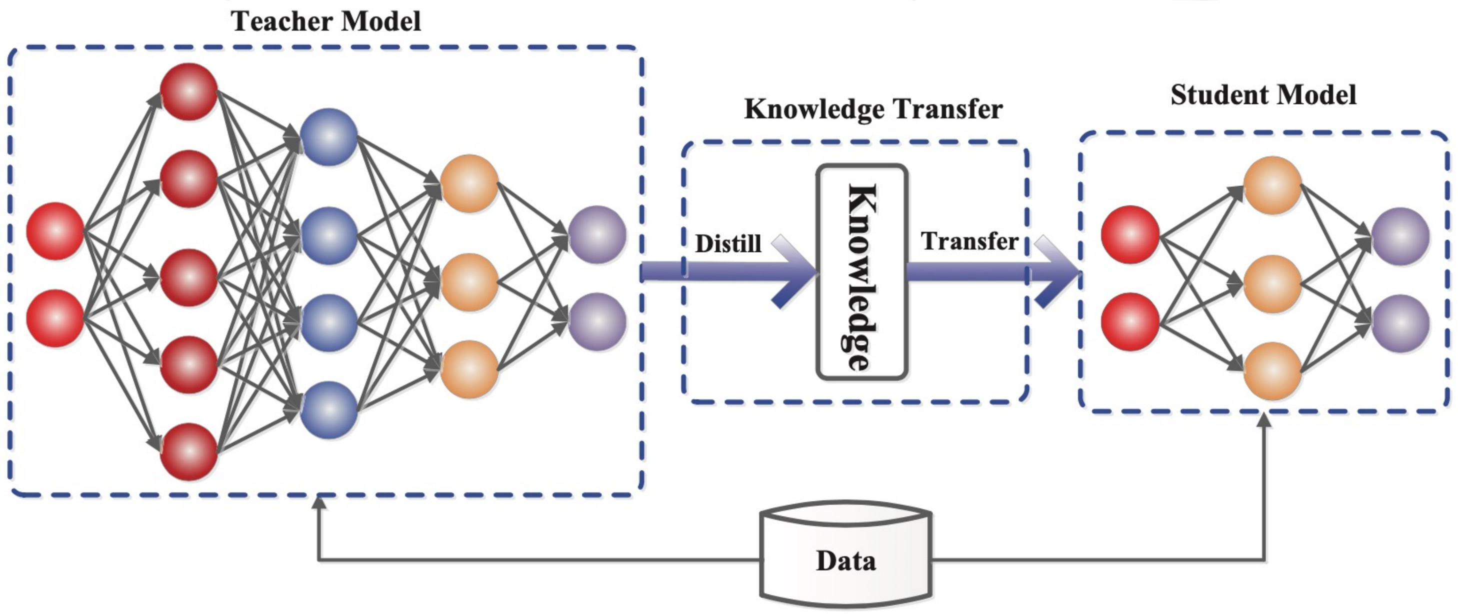
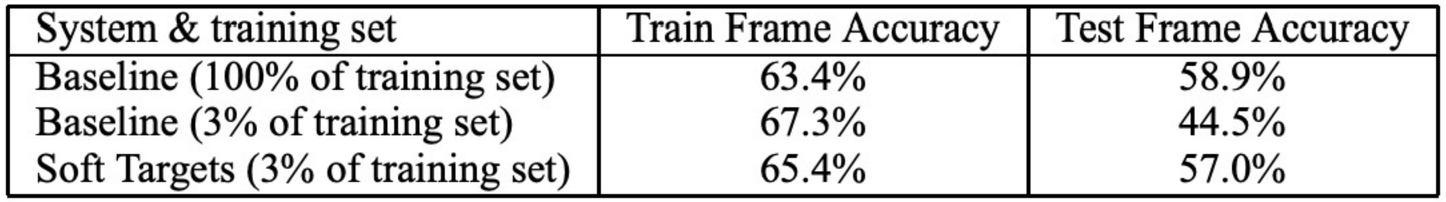


Fig. 1 The generic teacher-student framework for knowledge distillation.

[1] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531, 2015.
[2] Gou J, Yu B, Maybank S J, et al. Knowledge distillation: A survey[J]. International Journal of Computer Vision, 2021: 1-31.

Basic Concept

- What can Knowledge Distillation do?
  - compressing heavy deep neural networks
  - prevent specialists from overfitting
  - helps the training process of a smaller student network
  - improve final performance



System & training set	Train Frame Accuracy	Test Frame Accuracy
Baseline (100% of training set)	63.4%	58.9%
Baseline (3% of training set)	67.3%	44.5%
Soft Targets (3% of training set)	65.4%	57.0%

Table 5: Soft targets allow a new model to generalize well from only 3% of the training set. The soft targets are obtained by training on the full training set.

[1] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531, 2015.

Knowledge Distillation with Logits

- Knowledge Distillation Design
  - For teacher output logits t_i, student output logits s_i, one-hot label gt_i, temperature T
    - Soft logits:
      - $t_i = \frac{\exp(\frac{z_i}{T})}{\sum_i \exp(\frac{z_i}{T})}$, $s_i = \frac{\exp(\frac{z_i}{T})}{\sum_i \exp(\frac{z_i}{T})}$
    - Soft loss:
      - $L_{soft} = -\sum_i^K s_i log t_i$
    - Hard loss:
      - $L_{hard} = -\sum_i^K gt_i log s_i$
    - KD Training:
      - $\mathrm{L} = L_{hard} + \alpha L_{soft}$

[1] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531, 2015.

Knowledge Distillation with Logits

- Why does Knowledge Distillation work:
  - Soft targets contain information of inter-class distance and in-class variance than one-hot labels
  - The knowledge from the teacher expresses a more general learned information that is helpful for building up a well-performing student

- Problems
  - The parameter selection of α and temperature T should be considered
  - When the capacity of the student is too low, it is hard for the student to incorporate the logits information of the teacher successfully

Knowledge Distillation with Intermediate Features

- Feature-based distillation enables learning richer information from the teacher and provides more flexibility for performance improvement.

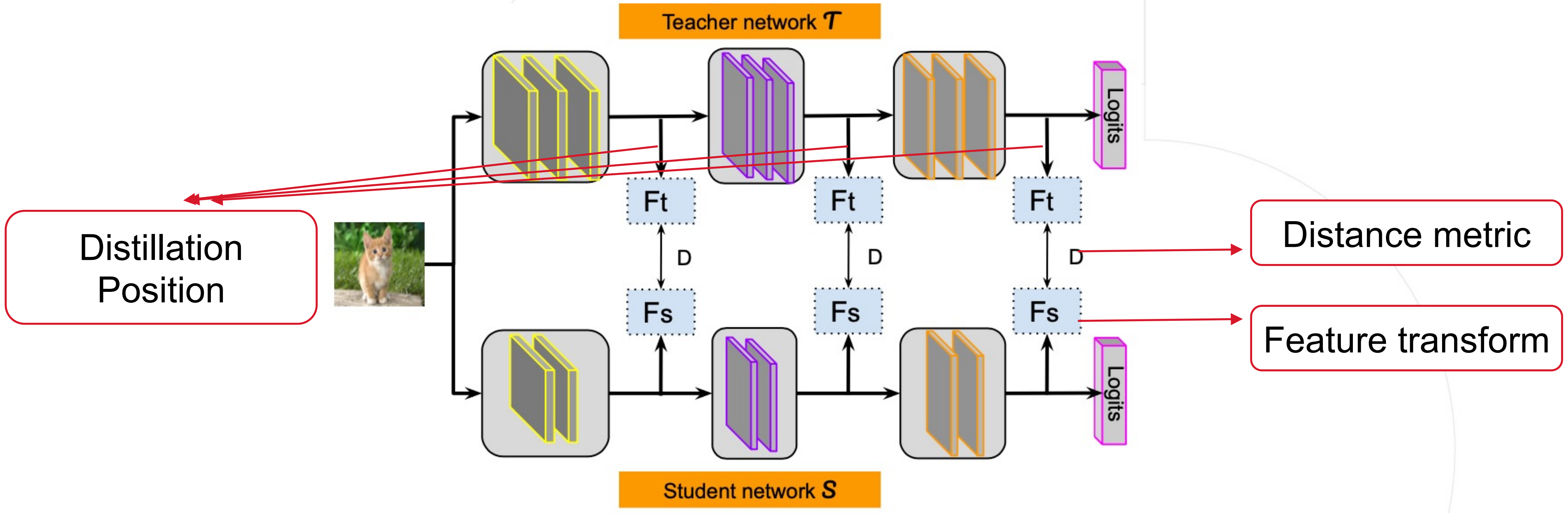


Fig. 3. An illustration of general feature-based distillation.

[1] Wang L, Yoon K J. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.
[2] Phuong M, Lampert C H. Distillation-based training for multi-exit architectures[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 1355-1364.

Knowledge Distillation with Intermediate Features

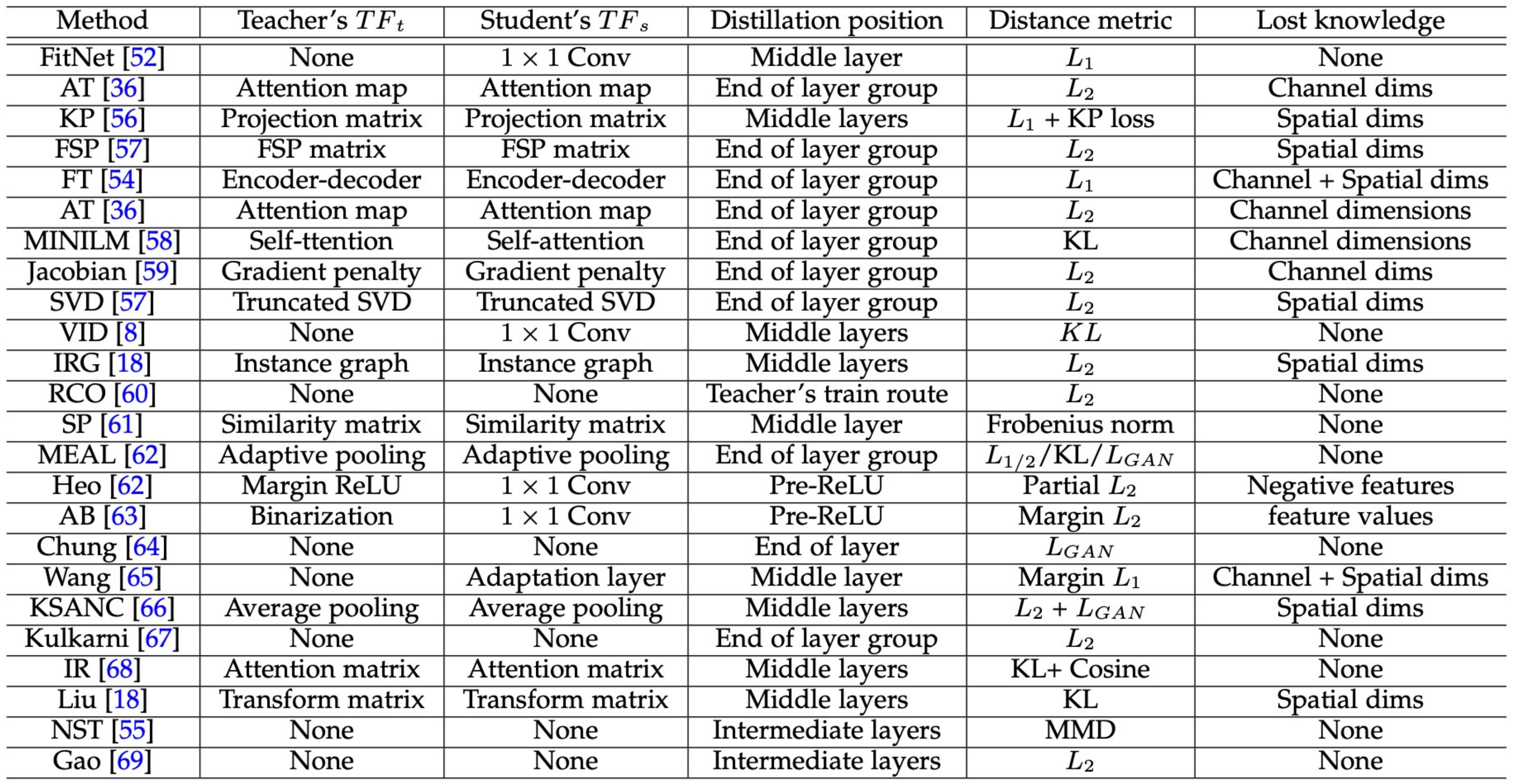
- Transformation of the guided features:
  - Teacher and student may have different size of intermediate feature maps
- Distillation positions of features:
  - The distillation position includes the feature map at the end of each block, at the end of each stage, etc.
- Distance metric for measuring distillation:
  - To measure the features after transformation, the distance metric is used to construct the kd loss

[1] Wang L, Yoon K J. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

Knowledge Distillation with Intermediate Features

- Summary

TABLE 2
A taxonomy of knowledge distillation from the intermediate layers (feature maps). KP incidates knowledge projection.



Method	Teacher's TF_t	Student's TF_s	Distillation position	Distance metric	Lost knowledge
FitNet [52]	None	1×1 Conv	Middle layer	L_1	None
AT [36]	Attention map	Attention map	End of layer group	L_2	Channel dims
KP [56]	Projection matrix	Projection matrix	Middle layers	L_1 + KP loss	Spatial dims
FSP [57]	FSP matrix	FSP matrix	End of layer group	L_2	Spatial dims
FT [54]	Encoder-decoder	Encoder-decoder	End of layer group	L_1	Channel + Spatial dims
AT [36]	Attention map	Attention map	End of layer group	L_2	Channel dimensions
MINILM [58]	Self-ttention	Self-attention	End of layer group	KL	Channel dimensions
Jacobian [59]	Gradient penalty	Gradient penalty	End of layer group	L_2	Channel dims
SVD [57]	Truncated SVD	Truncated SVD	End of layer group	L_2	Spatial dims
VID [8]	None	1×1 Conv	Middle layers	KL	None
IRG [18]	Instance graph	Instance graph	Middle layers	L_2	Spatial dims
RCO [60]	None	None	Teacher's train route	L_2	None
SP [61]	Similarity matrix	Similarity matrix	Middle layer	Frobenius norm	None
MEAL [62]	Adaptive pooling	Adaptive pooling	End of layer group	$L_{1/2}$/KL/L_{GAN}	None
Heo [62]	Margin ReLU	1×1 Conv	Pre-ReLU	Partial L_2	Negative features
AB [63]	Binarization	1×1 Conv	Pre-ReLU	Margin L_2	feature values
Chung [64]	None	None	End of layer	L_{GAN}	None
Wang [65]	None	Adaptation layer	Middle layer	Margin L_1	Channel + Spatial dims
KSANC [66]	Average pooling	Average pooling	Middle layers	$L_2 + L_{GAN}$	Spatial dims
Kulkarni [67]	None	None	End of layer group	L_2	None
IR [68]	Attention matrix	Attention matrix	Middle layers	KL+ Cosine	None
Liu [18]	Transform matrix	Transform matrix	Middle layers	KL	Spatial dims
NST [55]	None	None	Intermediate layers	MMD	None
Gao [69]	None	None	Intermediate layers	L_2	None

[1] Wang L, Yoon K J. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

Teacher-Student Architecture

- Pattern:
  - Simplified Structure :
    - Res34 & Res18
  - Quantized Structure:
    - Res18 & Int8 Res18
  - Same Structure
  - Small Structure

- Conclusion:
  - The model capacity gap between the large deep neural network and a small student neural network can degrade knowledge transfer.

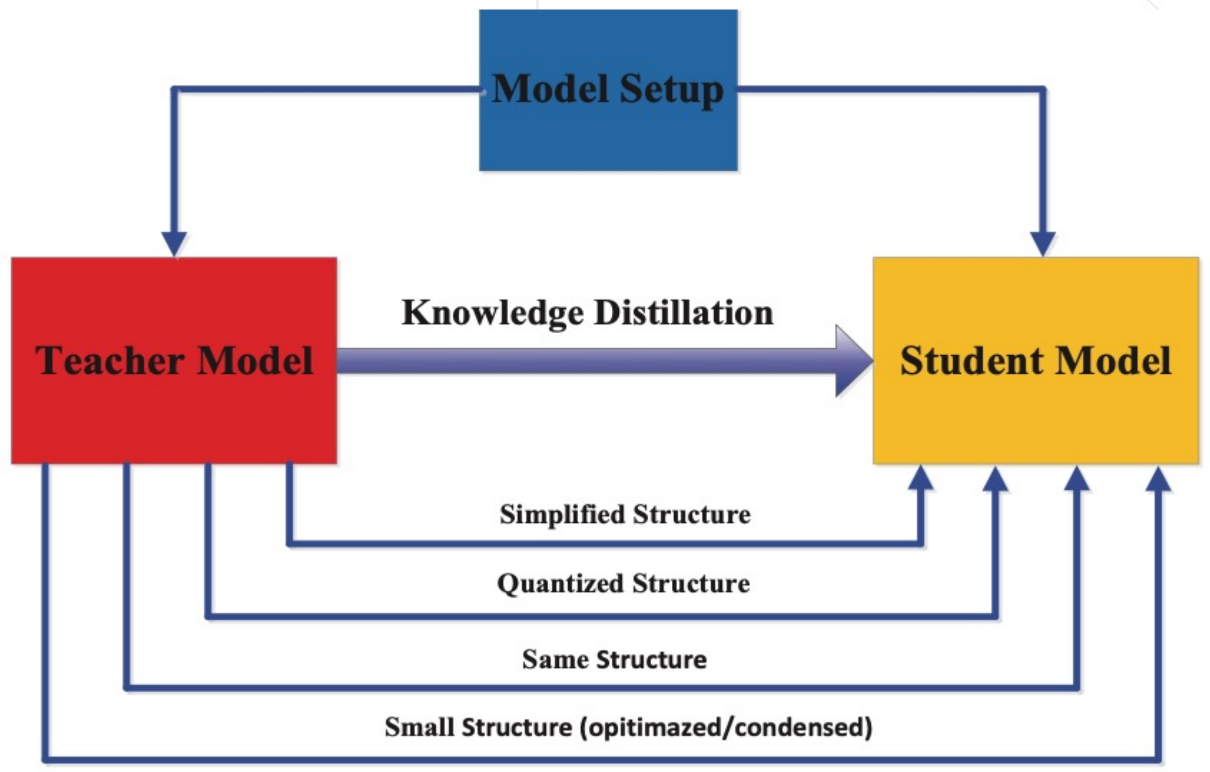


Fig. 9 Relationship of the teacher and student models.

[1] Gou J, Yu B, Maybank S J, et al. Knowledge distillation: A survey[J]. International Journal of Computer Vision, 2021: 1-31.

Teacher-Student Architecture

- Each teacher model could potentially have its own best student architecture.
- NAS can be used to discover the best student model or teacher model.

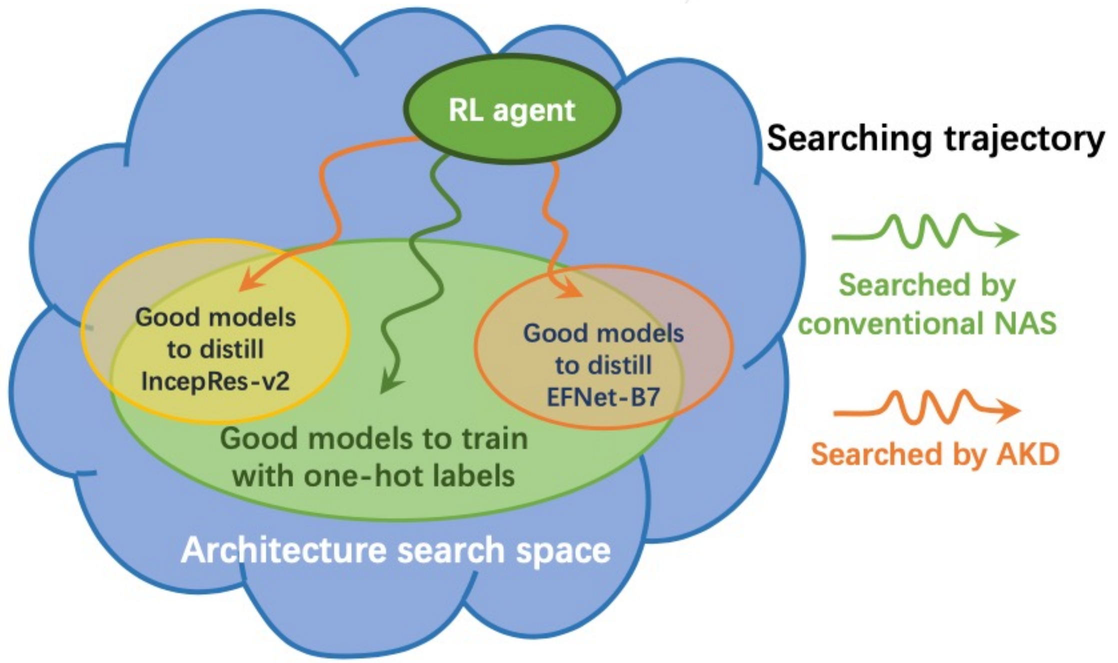
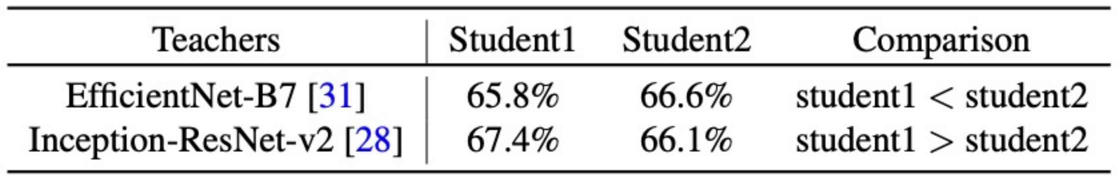

RL agent
Searching trajectory
Searched by conventional NAS
Searched by AKD
Good models to distill IncepRes-v2
Good models to distill EFNet-B7
Good models to train with one-hot labels
Architecture search space

Figure 1. Searching neural architectures by the proposed AKD and conventional NAS [30] lead to different optimal architectures.



Teachers	Student1	Student2	Comparison
EfficientNet-B7 [31]	65.8%	66.6%	student1 < student2
Inception-ResNet-v2 [28]	67.4%	66.1%	student1 > student2

Table 1. ImageNet accuracy for students with different teachers.

[1] Liu Y, Jia X, Tan M, et al. Search to distill: Pearls are everywhere but not the eyes[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 7539-7548.

More Advanced KD Methods

- Multi-Teacher: employ multiple supervision knowledge
- Date-Free Distillation: requires no training data

Multi-Teacher

Data-Free

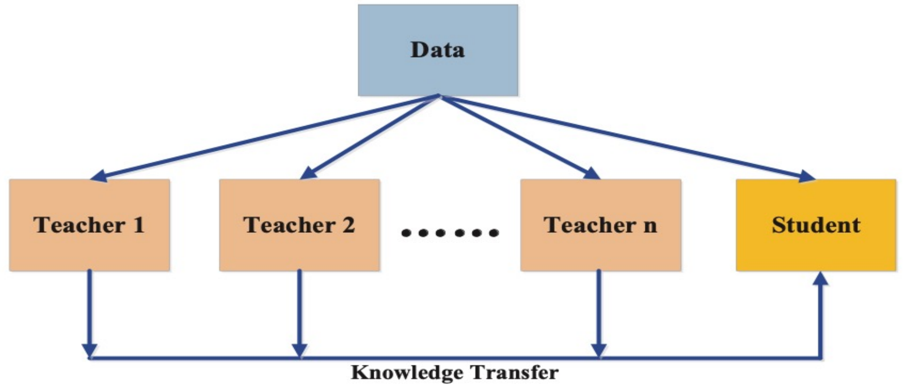
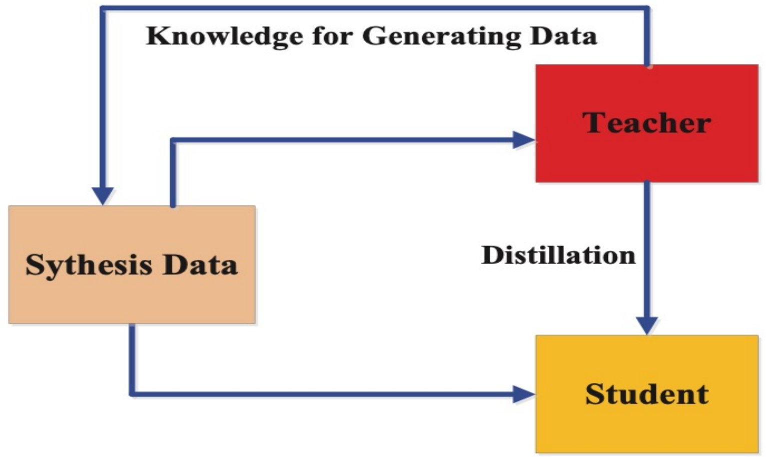


Fig. 11 The generic framework for multi-teacher distillation.



More Advanced KD Methods

- Offline Distillation: most common form, the large teacher model is first trained and 2) the teacher model is used to guide the training of the student model during distillation.
- Online Distillation: both the teacher model and the student model are updated simultaneously, and the whole knowledge distillation framework is end-to-end trainable.
- Self-Distillation: the same networks or supernet (BigNAS[1]) are used for the teacher and the student models

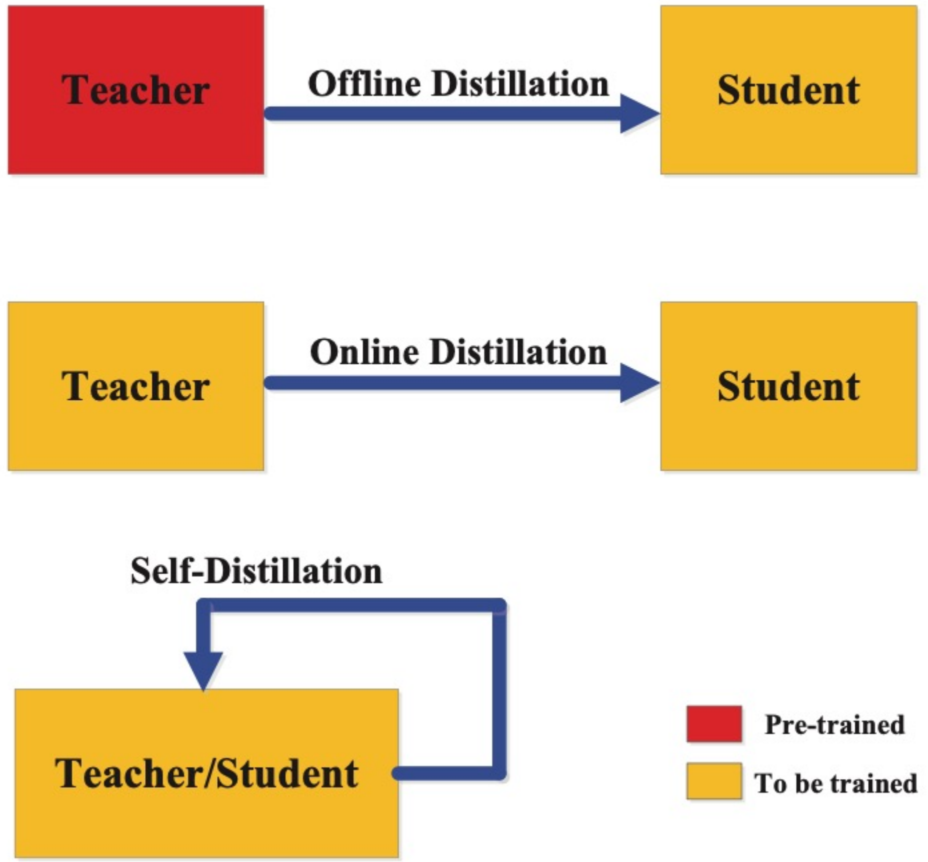


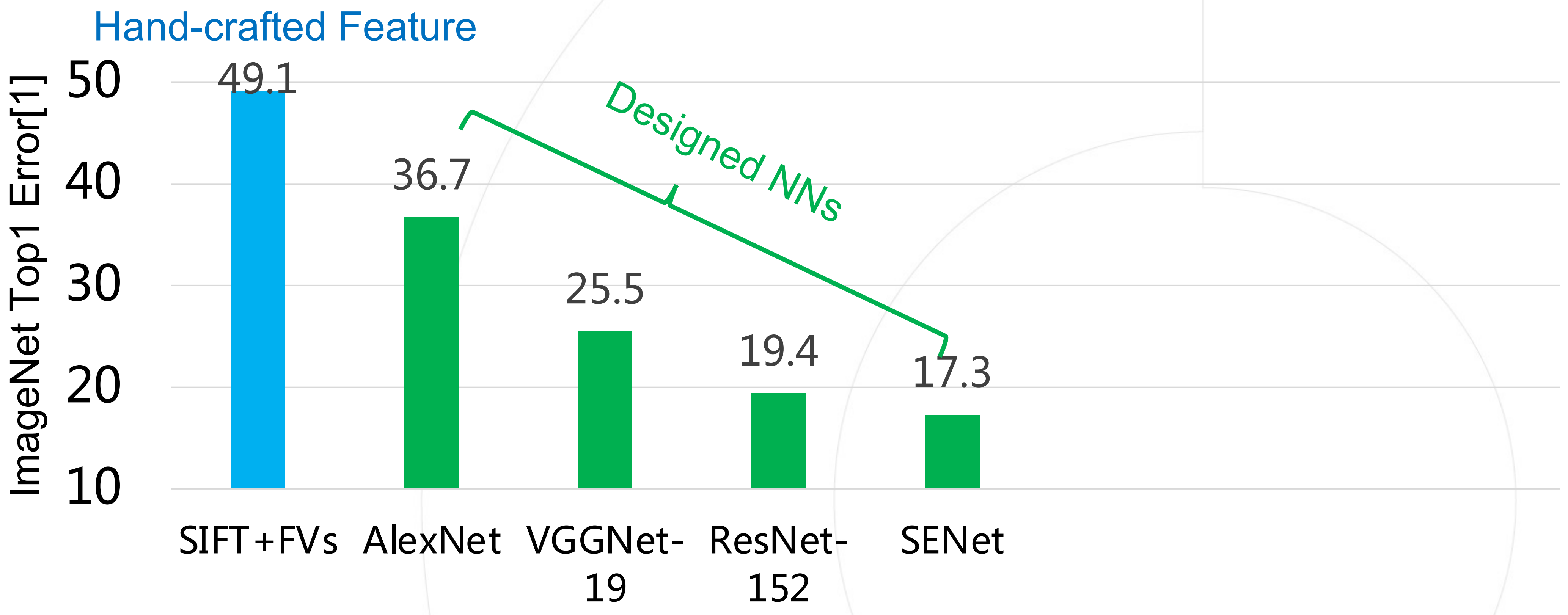
Fig. 8 Different distillations. The red color for "pre-trained" means networks are learned before distillation and the yellow color for "to be trained" means networks are learned during distillation

[1] Yu J, Jin P, Liu H, et al. Bignas: Scaling up neural architecture search with big single-stage models[C]//European Conference on Computer Vision. Springer, Cham, 2020: 702-717.

Highlights

What is NAS

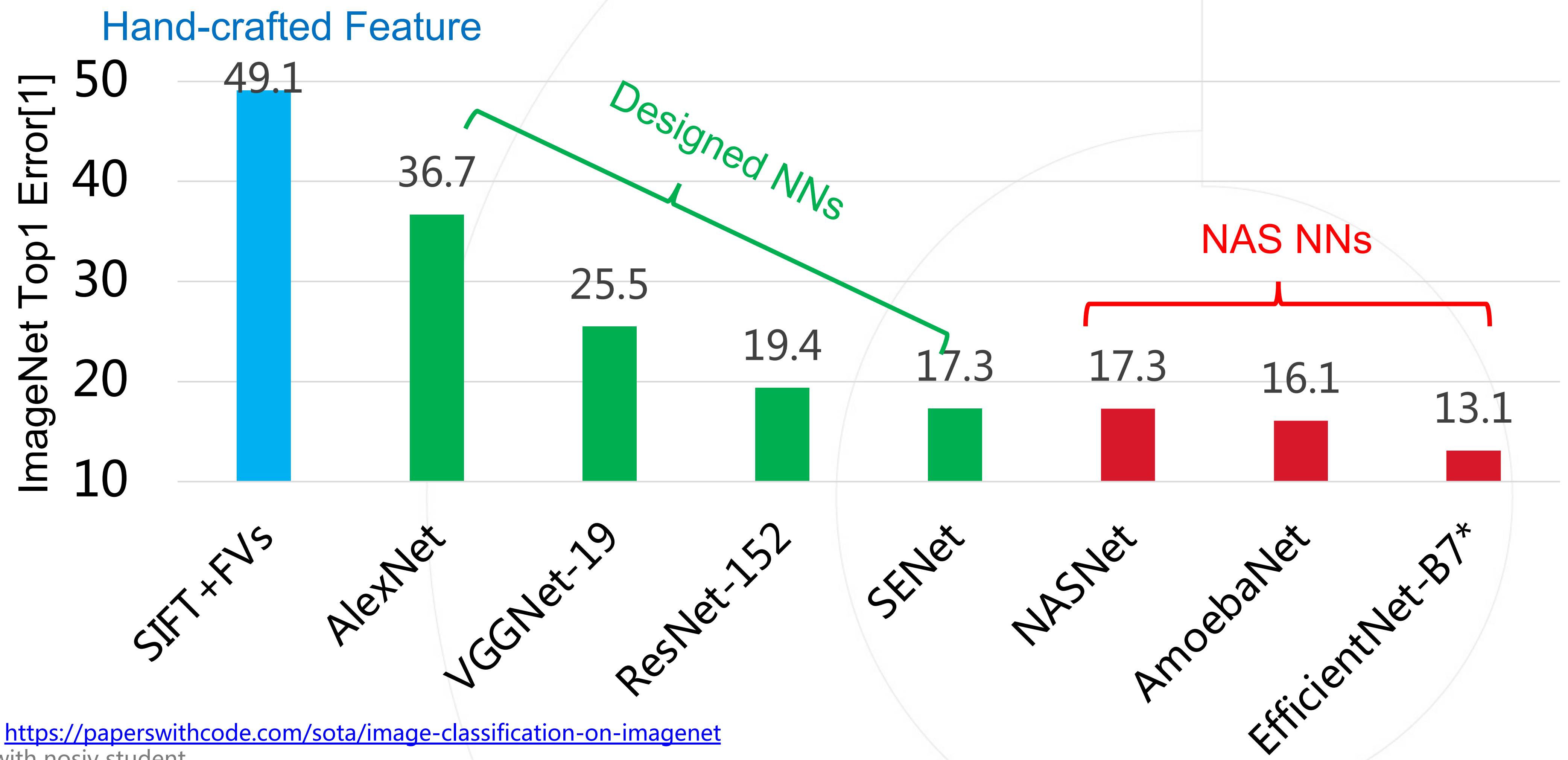
Deep Learning relays heavily on novel deep neural nets.



[1] https://paperswithcode.com/sota/image-classification-on-imagenet

What is NAS

NAS focus on automating the network architecture design.



[1] https://paperswithcode.com/sota/image-classification-on-imagenet
*: with nosiy student.

NAS Milestones

2017

NASNet

AmoebaNet
ENAS

2018

DARTS
FBNet

MNasNet
ProxylessNAS
One-Shot NAS

2019

NASFPN
DetNAS

OFA
EfficientNet

2020

BigNAS
UnNAS

Efficient
NetV2

NAS Literature 1: NASNet

Main steps:

1. RNN controller(Agent) generates child architecture A with prob p
2. Train child network A on proxy task get validation accuracy R
3. Use prob p and accuracy R to update the agent
4. Back to setp1

Key problems

- Every child needs to be trained from scratch on proxy task, which introduces prohibitive cost: thousands of GPU-days.

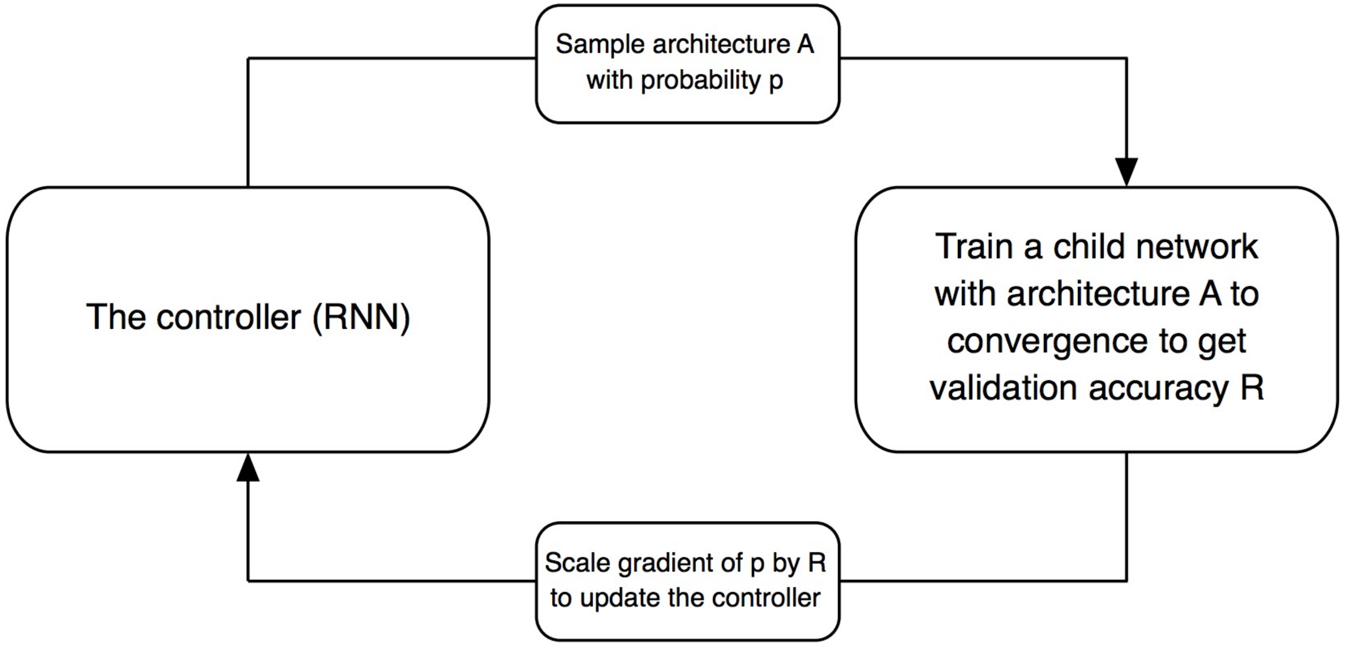

Sample architecture A with probability p
The controller (RNN)
Train a child network with architecture A to convergence to get validation accuracy R
Scale gradient of p by R to update the controller

Figure 1. Overview of Neural Architecture Search [71]. A controller RNN predicts architecture A from a search space with probability p. A child network with architecture A is trained to convergence achieving accuracy R. Scale the gradients of p by R to update the RNN controller.

[1] Zoph, Barret, and Quoc V. Le. "Neural architecture search with reinforcement learning." arXiv preprint arXiv:1611.01578 (2016).
[2] Zoph, Barret, et al. "Learning transferable architectures for scalable image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

NAS Literature 2: ENAS

Contributions

- ENAS proposes sharing parameters strategy, i.e. reusing partial weights from the former trained child network. ENAS significantly reduces the overall cost.
- ENAS can also be viewed as a weight-sharing supernet approach.

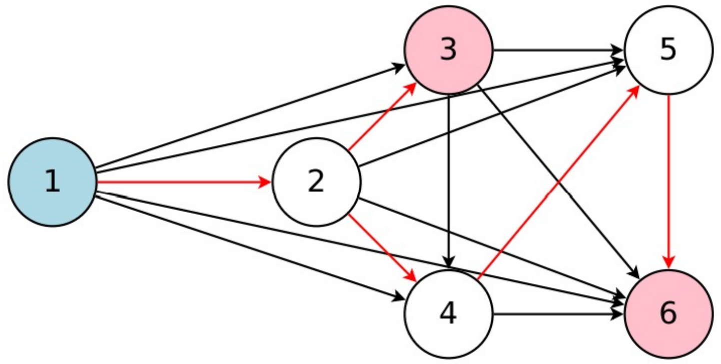


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

[1] Pham, Hieu, et al. "Efficient neural architecture search via parameters sharing." International Conference on Machine Learning. PMLR, 201

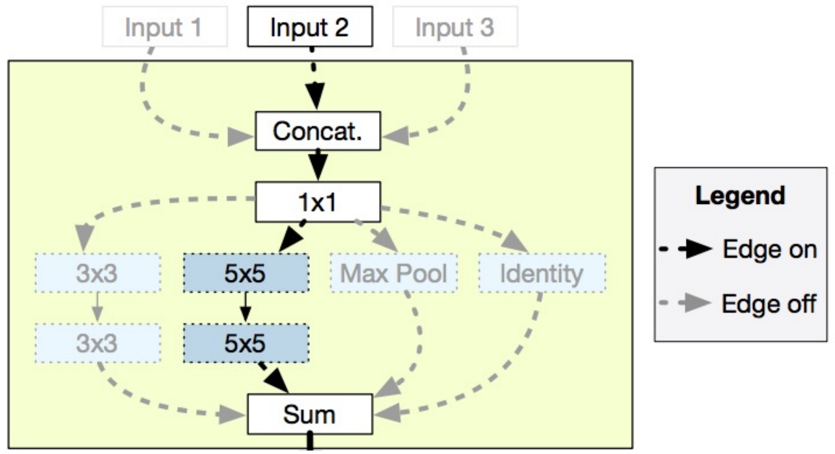
NAS Literature 3: One-shot NAS

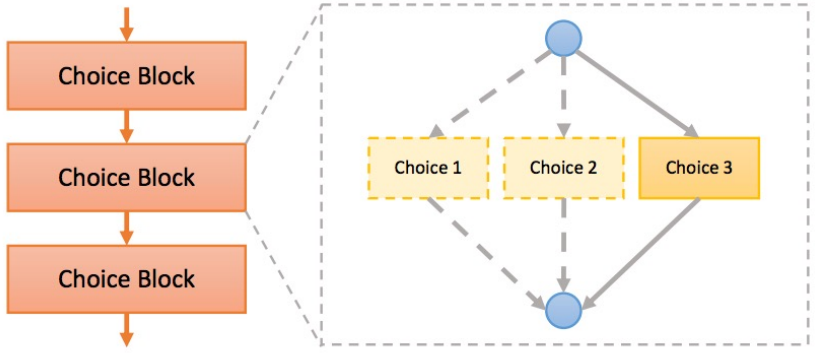
Contributions

- One-shot NAS builds a weight-sharing supernet in which each subnet can be viewed as a candidate architecture.
- One-shot NAS trains the supernet properly and uses the subnet validation accuracy to estimate the final candidate performance.
- Supernet training is a once cost, so it orderly reduces the cost.

Drawbacks

- Unreliable performance ranking in supernet.





(a)

[1] Bender, Gabriel, et al. “Understanding and simplifying one-shot architecture search.” International Conference on Machine Learning. PMLR, 2018.
[2] Stamoulis, Dimitrios, et al. "Single-path nas: Designing hardware-efficient convnets in less than 4 hours." Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham, 2019.
[3] Guo, Zichao, et al. "Single path one-shot neural architecture search with uniform sampling." European Conference on Computer Vision. Springer, Cham, 2020.

NAS Literature 4: MNasNet

Contributions

- Searching directly on large dataset ImageNet.
- Integrating platform latency to the searching reward calculation, which helps to find a architecture that achieves the best latency-accuracy tradeoff.

Drawbacks

- Following NASNet costly RL-based searching algorithm: Thousands of TPU days

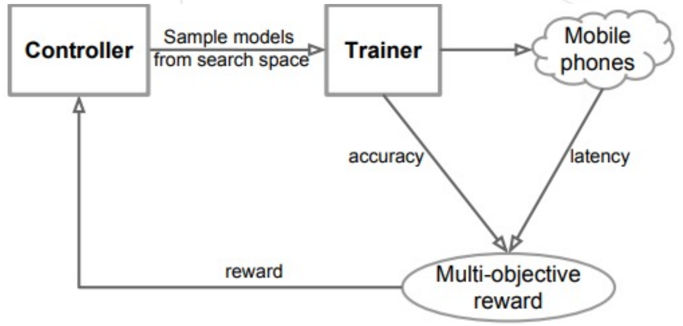


Figure 1: **An Overview of Platform-Aware Neural Architecture Search for Mobile.**

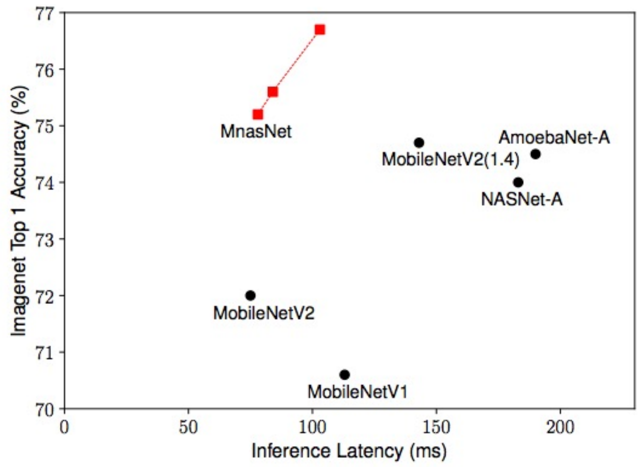


Figure 2: **Accuracy vs. Latency Comparison** – Our MnasNet models significantly outperforms other mobile models [29, 36, 26] on ImageNet. Details can be found in Table 1.

[1] Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." arXiv preprint arXiv:1806.09055 (2018).
[2] Chen, Xin, et al. "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

NAS Literature 5: DARTS

Contributions

- Building a fully-connected supernet, each path contains several operations.
- Using bi-level optimization method to update the architecture parameters and weights.

Drawbacks

- Because every operator is maintained in the computation graph, DARTS is memory hungry. Typically, DARTS searches the cell on CIFAR10 and then transfers it to ImageNet.
- DARTS is one of the most well-known NAS baselines. There are many good follow-up papers, like P-DARTS[2], PCDARTS[3]

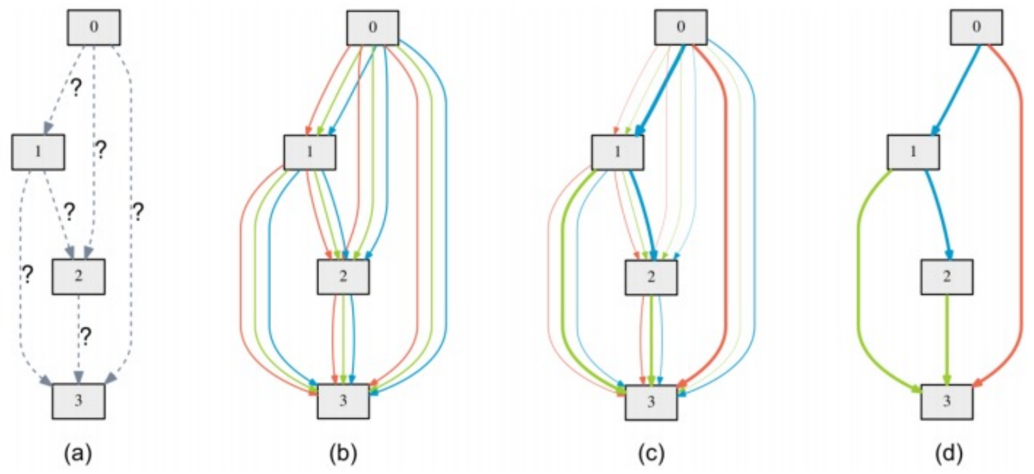


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

[1] Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." arXiv preprint arXiv:1806.09055 (2018).
[2] Chen, Xin, et al. “Progressive differentiable architecture search: Bridging the depth gap between search and evaluation.” Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.
[3] Xu, Yuhui, et al. "PC-DARTS: Partial channel connections for memory-efficient architecture search." arXiv preprint arXiv:1907.05737 (2019).

NAS Literature 6: ProxylessNAS

Contributions

- Searching directly on large dataset ImageNet.
- Utilizing real hardware latency as a constraint factor.
- Proposing binary gate method, which maintains only O(1) operator in the computation graph, to solve the memory issue.

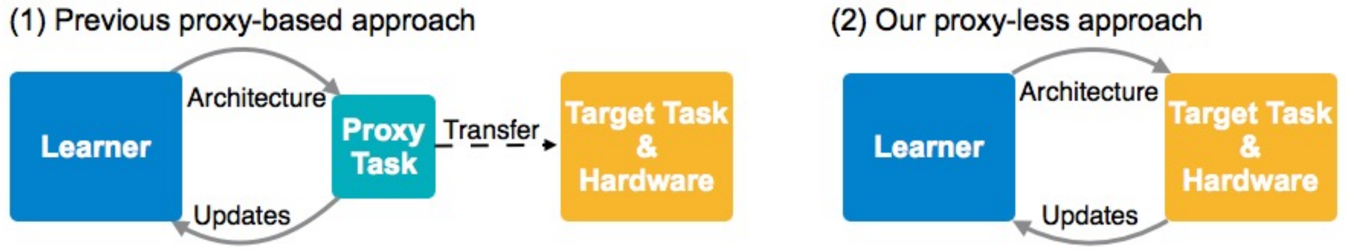


Figure 1: ProxylessNAS directly optimizes neural network architectures on target task and hardware. Benefiting from the directness and specialization, ProxylessNAS can achieve remarkably better results than previous proxy-based approaches. On ImageNet, with only 200 GPU hours (200 × fewer than MnasNet (Tan et al., 2018)), our searched CNN model for mobile achieves the same level of top-1 accuracy as MobileNetV2 1.4 while being 1.8× faster.

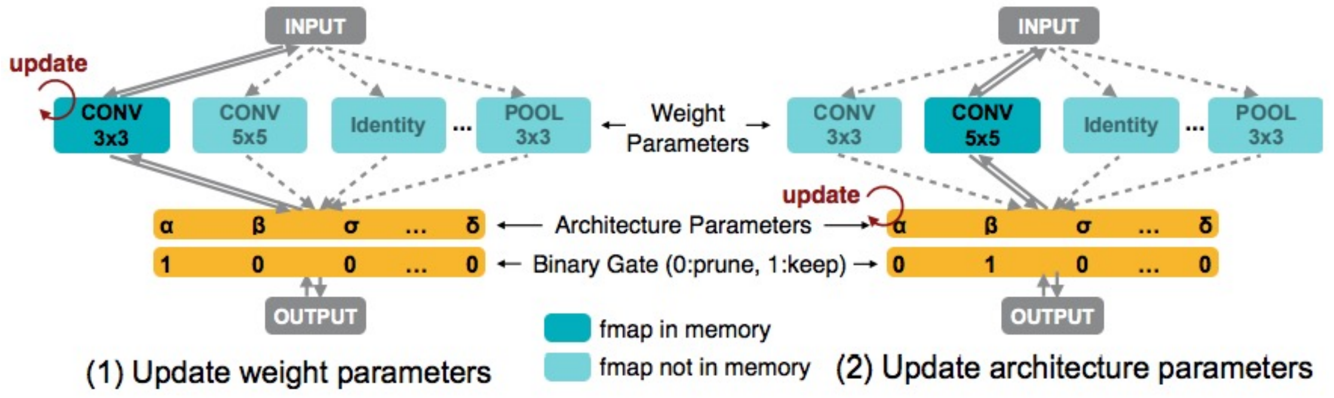


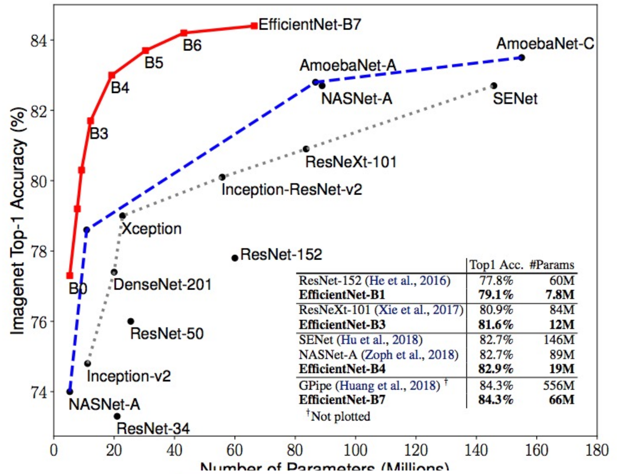
Figure 2: Learning both weight parameters and binarized architecture parameters.

[1] Cai, Han, Ligeng Zhu, and Song Han. "Proxylessnas: Direct neural architecture search on target task and hardware." arXiv preprint arXiv:1812.00332 (2018).

NAS Literature 7: EfficientNet

Contributions

- The traditional scaling-up method is to increase a single dimension. EfficientNet proposes a compound scaling method which increases width, depth and resolution simultaneously.
- A set of good scaling up parameters is found by grid search, and the result of SOTA is obtained by scale up from MNasNet(EfficientNet-B0)



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

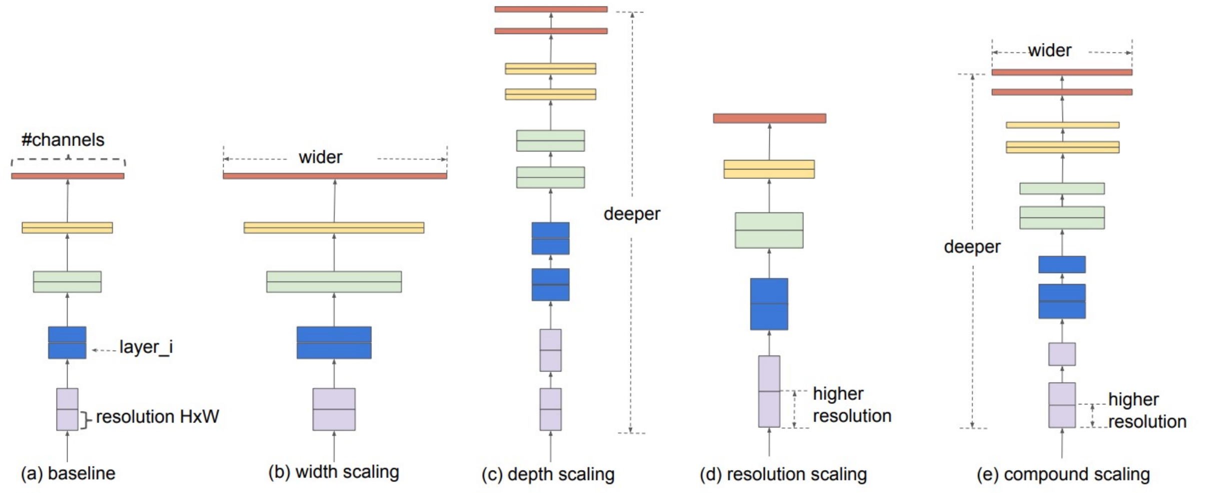


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

[1] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." International Conference on Machine Learning. PMLR, 2019.
[2] Tan, Mingxing, and Quoc V. Le. "EfficientNetV2: Smaller Models and Faster Training." arXiv preprint arXiv:2104.00298 (2021).

NAS Literature 8: OFA/ BigNAS

Contributions

- Former one-shot NAS methods need to retrain the found architecture from scratch to obtain the final accuracy. OFA/BigNAS can directly deploy the subnet without further retraining.
- Supernet training is a once cost. We can sample and deploy a series of different architectures under different constraint.

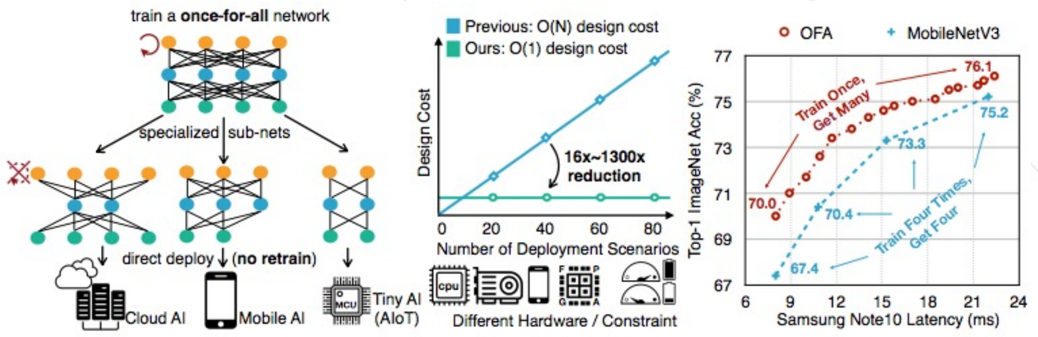


Figure 1: Left: a single once-for-all network is trained to support versatile architectural configurations including depth, width, kernel size, and resolution. Given a deployment scenario, a specialized sub-network is directly selected from the once-for-all network without training. Middle: this approach reduces the cost of specialized deep learning deployment from O(N) to O(1). Right: once-for-all network followed by model selection can derive many accuracy-latency trade-offs by training only once, compared to conventional methods that require repeated training.

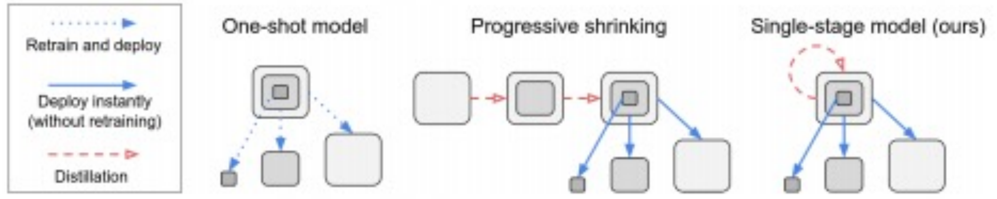


Fig. 1: Comparison with several existing workflows. We use nested squares to denote models with shared weights, and use the size of the square to denote the size of each model. Workflow in the middle refers the concurrent work from [5], where submodels are sequentially induced through progressive distillation and channel sorting. We simultaneously train all child models in a single-stage model with proposed modifications, and deploy them without retraining or finetuning.

[1] Cai, Han, et al. "Once-for-all: Train one network and specialize it for efficient deployment." arXiv preprint arXiv:1908.09791 (2019).
[2] Yu, Jiahui, et al. "Bignas: Scaling up neural architecture search with big single-stage models." European Conference on Computer Vision. Springer, Cham, 2020.

A taxonomy of NAS

Search Strategy	Important Work
Individual – Reinforcement	NASNet, PNAS, Block-QNN, MNasNet, EfficientNet, NAS-FPN
Individual – Evolutionary	AmoebaNet, Genetic cnn, Evolved transformer
Weight-Sharing Heuristic	ENAS, Smash, SPOS, FairNAS, OFA, BigNAS
Weight-Sharing Differentiable	DARTs, PDARTs, PCDARTs, NAO, SNAS, ProxylessNAS,
Predictor-based Search	Chamnet, Peephole

[1] Xie, Lingxi, et al. "Weight-Sharing Neural Architecture Search:\\A Battle to Shrink the Optimization Gap." arXiv preprint arXiv:2008.01475 (2020).

NAS + Different CV tasks

NAS + XX Task?

Object Detection:

DetNAS
NASFPN
EfficientDet
...

Semantic Segmentation:

AutoDeepLab
...

Generative Models:

AutoGan
AdversarialNAS
...

NAS + Different ML algorithms

NAS + XX Learning?

Unsupervised Learning[1]
Domain Adaptation[2]
Transfer Learning[3]
Multi-Task Learning[4]
Meta Learning[5]

[1] ECCV2020. Liu, Chenxi, et al. "Are Labels Necessary for Neural Architecture Search?."
[2] NeurIPS 2020. Li, Yanxi, et al. "Adapting neural architectures between domains." -> AdaptNAS
[3] NeurIPS 2020. Cai, Han, et al. "Tiny Transfer Learning: Towards Memory-Efficient On-Device Learning."
[4] CVPR 2020. Gao, Yuan, et al. "Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning."
[5] ICLR 2019. Lian, Dongze, et al. "Towards fast adaptation of neural architectures with meta learning." -> T-NAS

NAS + Unsupervised Learning

Unsupervised NAS: We can achieve comparable NAS results without labels.

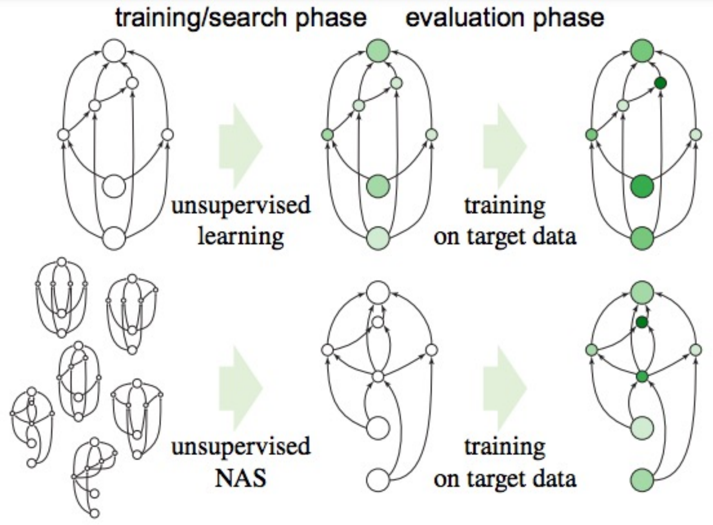


Figure 1: **Unsupervised neural architecture search**, or UnNAS, is a new problem setup that helps answer the question: are labels necessary for neural architecture search? In traditional unsupervised learning (top panel), the *training phase* learns the weights of a fixed architecture; then the *evaluation phase* measures the quality of the weights by training a classifier (either by fine-tuning the weights or using them as a fixed feature extractor) using supervision from the target dataset. Analogously, in UnNAS (bottom panel), the *search phase* searches for an architecture without using labels; and the *evaluation phase* measures the quality of the architecture found by an UnNAS algorithm by training the architecture's weights using supervision from the target dataset.

[1] ECCV2020. Liu, Chenxi, et al. "Are Labels Necessary for Neural Architecture Search?."

Highlights

Summary

Model Compression

- Quantization: utilizing integer only arithmetic to speed up the inference
- Pruning: removing unnecessary connections to get smaller models
- KD: distilling teacher models’ knowledge into smaller ones
- NAS: designing efficient models in an automatic way

推荐阅读

Quantization:

- https://arxiv.org/pdf/1806.08342.pdf
- https://arxiv.org/abs/2103.13630
- https://arxiv.org/abs/2004.09602
- https://github.com/ModelTC/MQBench

Pruning:

- https://arxiv.org/abs/2102.00554
- https://arxiv.org/abs/2007.00864

KD:

- https://arxiv.org/pdf/2004.05937.pdf

NAS:

- https://www.automl.org/automl/literature-on-neural-architecture-search/