Deep Learning Optimization - Neural Architecture Search

April 24, 2023

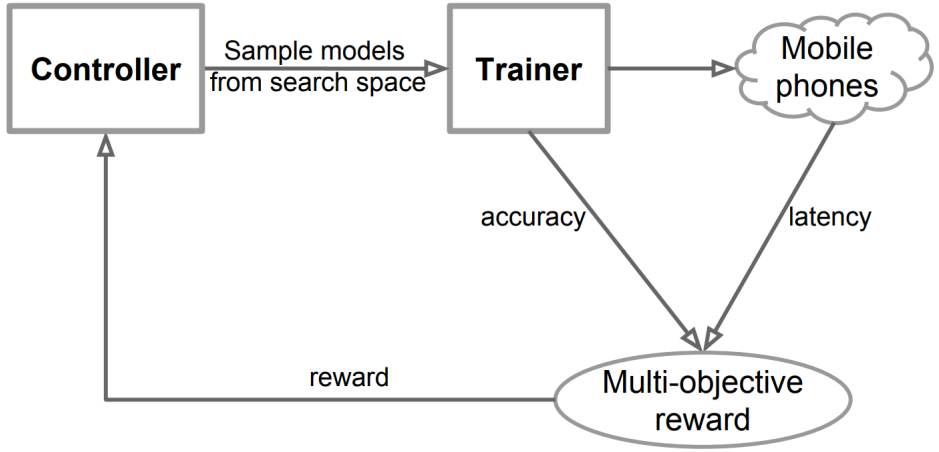
Eunhyeok Park

MnasNet: Platform-Aware Neural Architecture Search for Mobile

Platform-Aware NAS

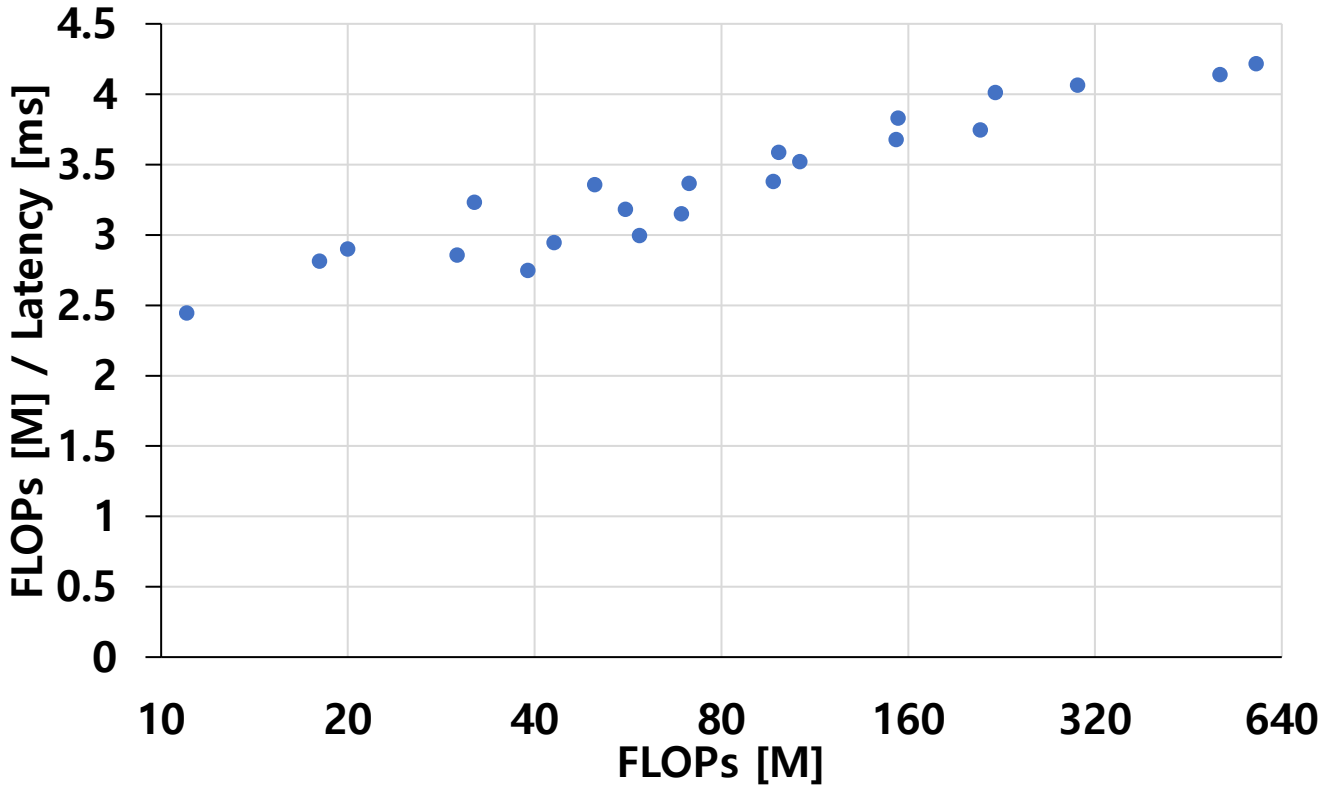
- Preivous studies focus on finding out the network having the highest accuracy
  - In real life, not only accuracy but also performance is matter
  - How can we make the AutoML algorithm consider the performance metric?

- The simplest approach?
  - Give FLOPs or MACs information as a additional reward
  - However, $latency \neq FLOPs\ or\ MACs$ on mobile device



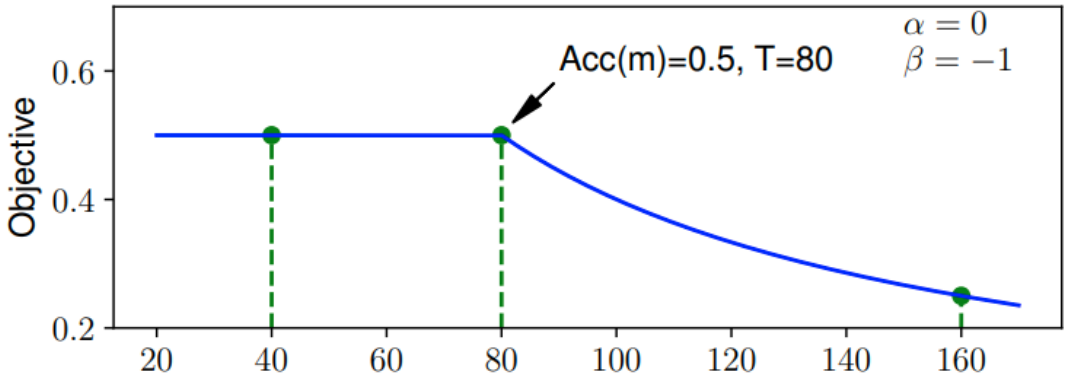
FLOPs vs Latency

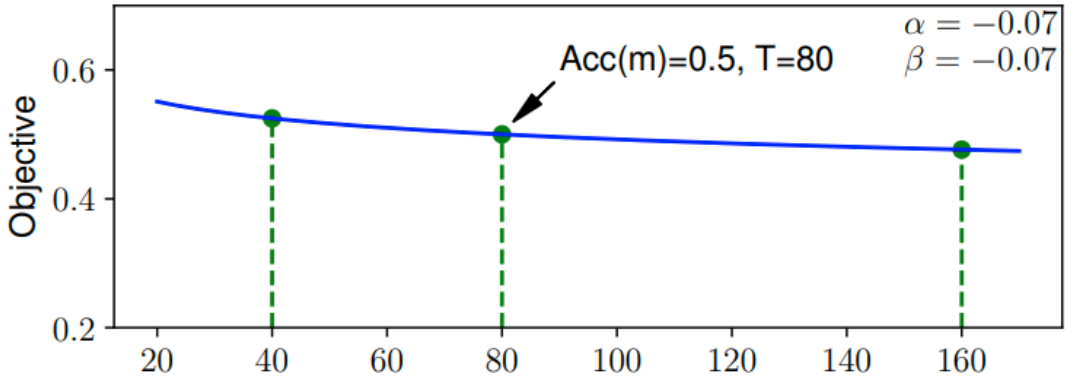
- Measured for MobileNet-v2 on Mobile CPU
  - Change channel scale / input resolution
  - Larger networks tend to be faster than the smaller networks



Latency Modeling

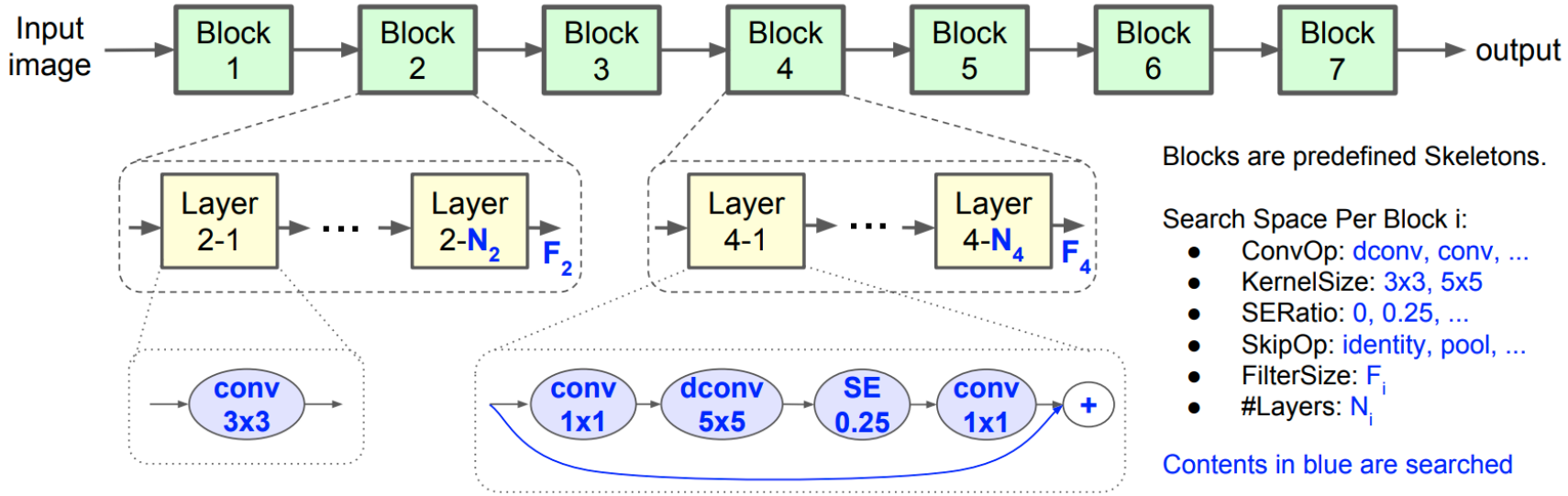
- Measure the latency of the model by executing it on the real device
- Hard constraint
  - $maximize_m\ ACC(m)\ subject\ to\ LAT(m) \leq T$
  - It is not possible to find out the multiple Pareto optimal solutions
- Soft constraint
  - $maximize_m\ ACC(m) \times \left[\frac{LAT(m)}{T}\right]^w, where\ w = \begin{cases} \alpha, if\ LAT(m) \leq T \\ \beta, otherwise \end{cases}$
  - Multiple solutions can be found based on the trade-off by picking α & β





Search Space Design

- Problem of cell-based search space
  - Limited diversity is critical for achieving both high accuracy and lower latency
- A novel factorized hierarchical search space
  - Factorize a CNN model into unique blocks
  - Search for the operations and connections per block separately
  - Allow different layer architectures in different blocks.



Search Pipeline

- Use policy-gradient method to maximize the expected reward

$$J = E_{P(a_{1:T};\theta)}[R(m)]$$

- Use proximal policy optimization (PPO)

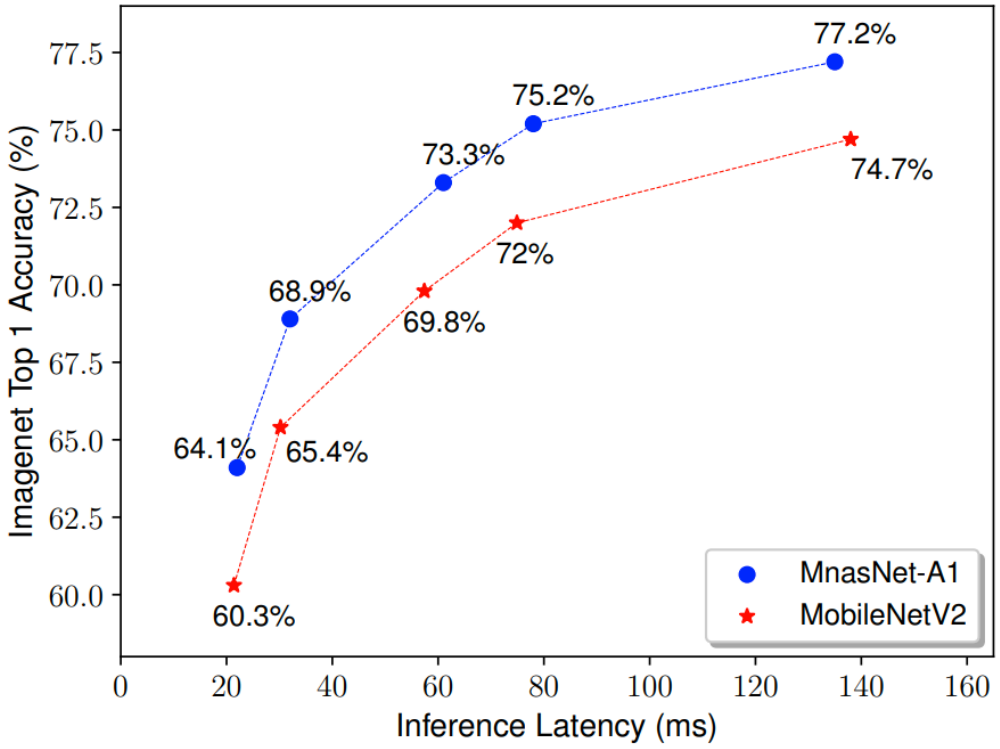
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

- 1. For each sampled model m, train it on the target task to get the accuracy $ACC(m)$
- 2. Run the model on real phones to get the inference latency $LAT(m)$
- 3. Calculate the reward $R(m)$ and update the controller with PPO

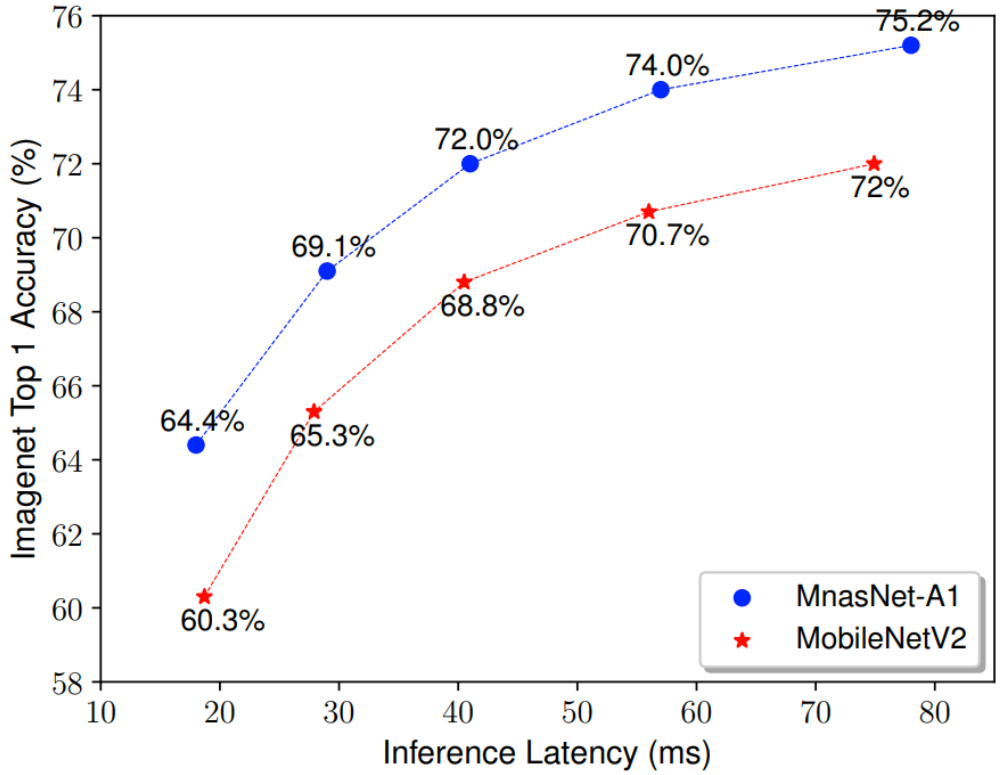
Results

Model	Type	#Params	#Mult-Adds	Top-1 Acc. (%)	Top-5 Acc. (%)	Inference Latency
MobileNetV1 [11]	manual	4.2M	575M	70.6	89.5	113ms
SqueezeNext [5]	manual	3.2M	708M	67.5	88.2	-
ShuffleNet (1.5x) [33]	manual	3.4M	292M	71.5	-	-
ShuffleNet (2x)	manual	5.4M	524M	73.7	-	-
ShuffleNetV2 (1.5x) [24]	manual	-	299M	72.6	-	-
ShuffleNetV2 (2x)	manual	-	597M	75.4	-	-
CondenseNet (G=C=4) [14]	manual	2.9M	274M	71.0	90.0	-
CondenseNet (G=C=8)	manual	4.8M	529M	73.8	91.7	-
MobileNetV2 [29]	manual	3.4M	300M	72.0	91.0	75ms
MobileNetV2 (1.4x)	manual	6.9M	585M	74.7	92.5	143ms
NASNet-A [36]	auto	5.3M	564M	74.0	91.3	183ms
AmoebaNet-A [26]	auto	5.1M	555M	74.5	92.0	190ms
PNASNet [19]	auto	5.1M	588M	74.2	91.9	-
DARTS [21]	auto	4.9M	595M	73.1	91	-
MnasNet-A1	**auto**	**3.9M**	**312M**	**75.2**	**92.5**	**78ms**
MnasNet-A2	**auto**	**4.8M**	**340M**	**75.6**	**92.7**	**84ms**
MnasNet-A3	**auto**	**5.2M**	**403M**	**76.7**	**93.3**	**103ms**

Results

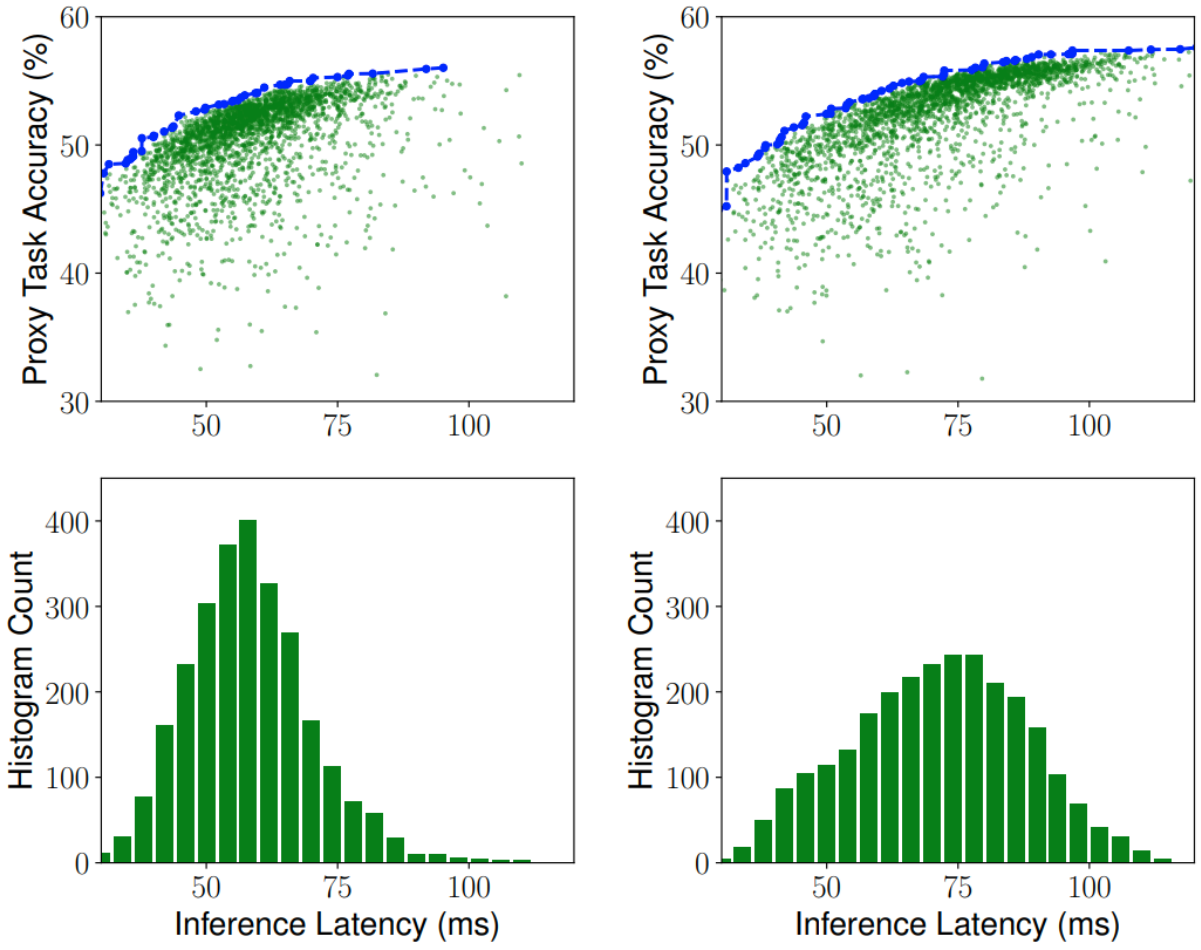


(a) Depth multiplier = 0.35, 0.5, 0.75, 1.0, 1.4, corresponding to points from left to right.

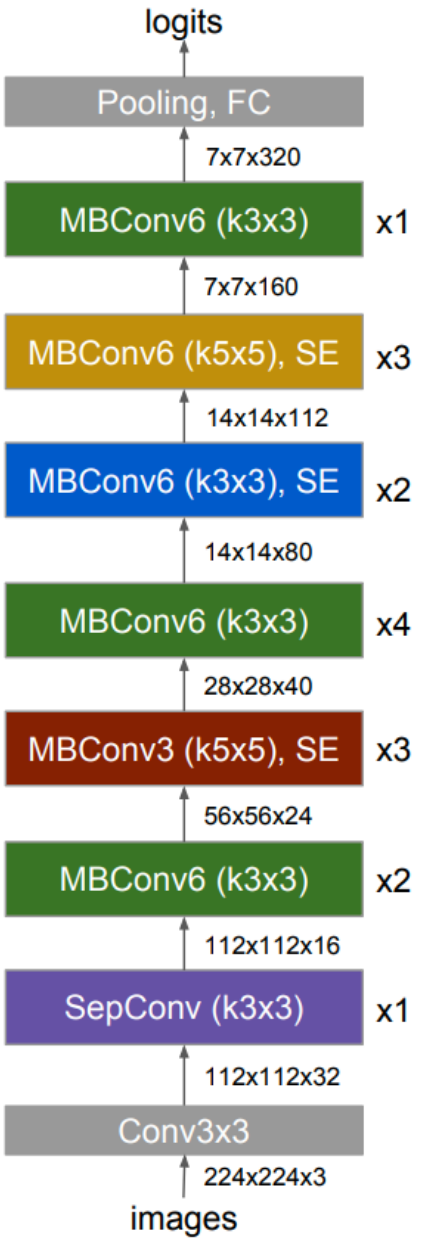


(b) Input size = 96, 128, 160, 192, 224, corresponding to points from left to right.

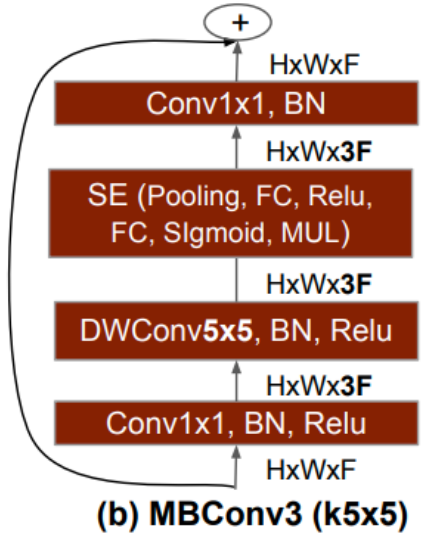
Results



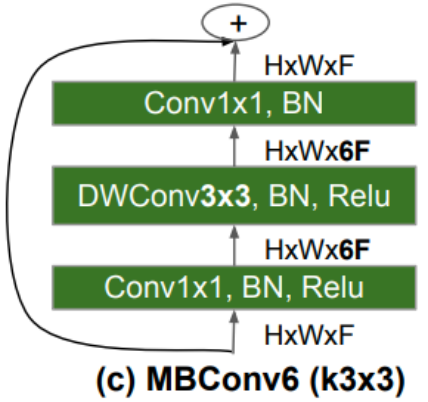
(a) $\alpha = 0, \beta = -1$ (b) $\alpha = \beta = -0.07$



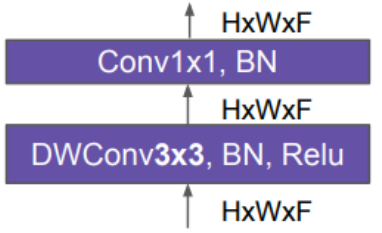
(a) MnasNet-A1



(b) MBConv3 (k5x5)



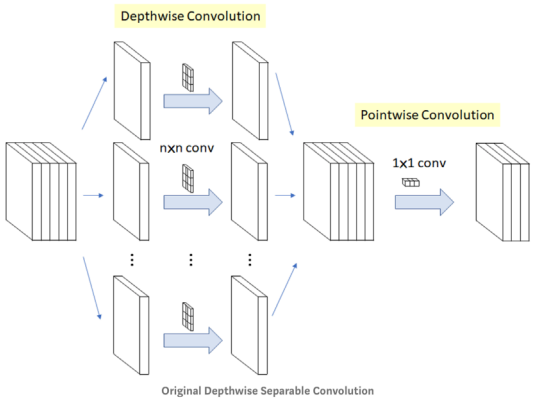
(c) MBConv6 (k3x3)



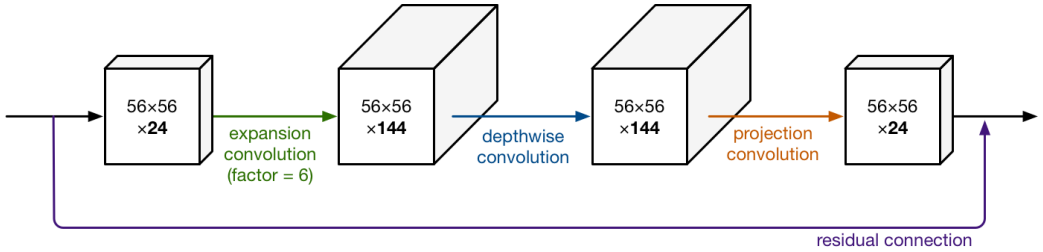
(d) SepConv (k3x3)

Searching for MobileNetV3

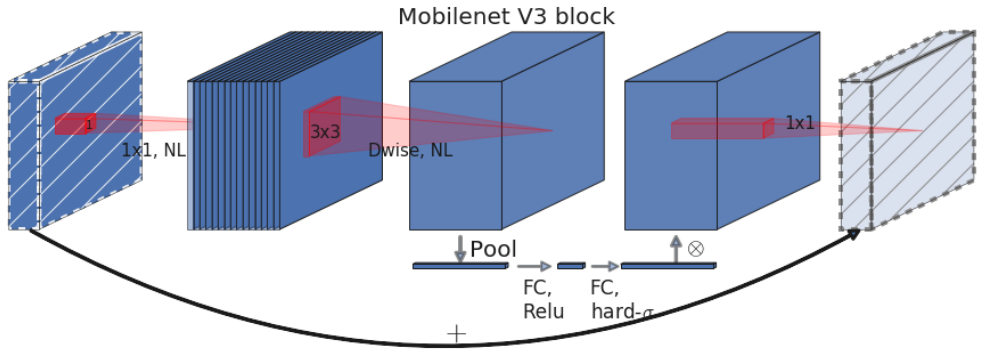
Evolution of MobileNets



MobileNet-v1
Depthwise-seperable convolution



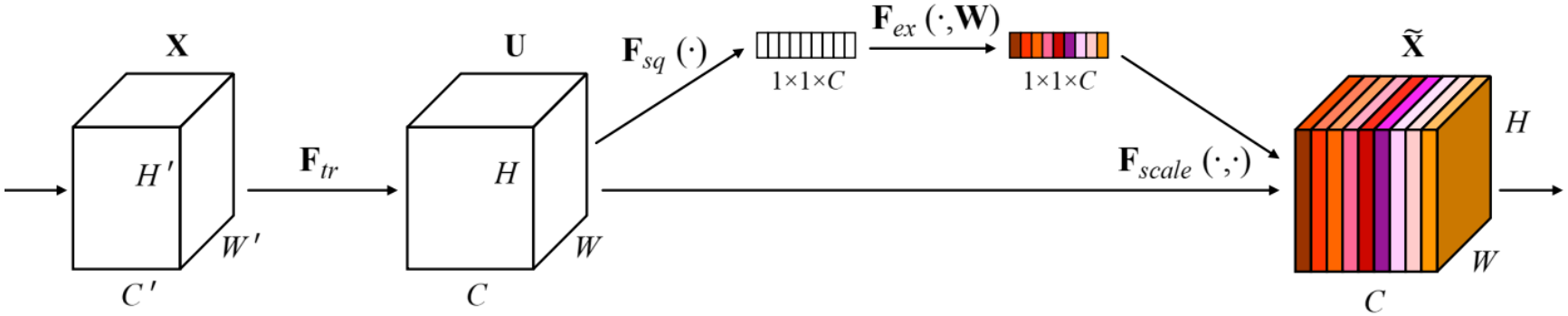
MobileNet-v2
Inverted-residual connection



MobileNet-v3
Squeeze-excitation layer
H-swish activation
Layer-wise tuning with AutoML

New Component: Squeeze-excitation Layer

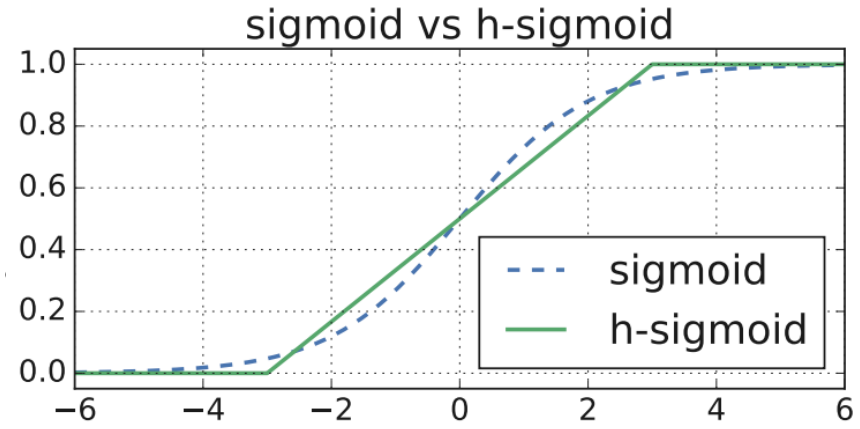
- Initially proposed at "Squeeze-and-Excitation Networks"
  - Channel-wise scaling
    - Global average pooling – bottleneck MLP – channel-wise scaling with sigmoid
    - Can be seen as self-attention

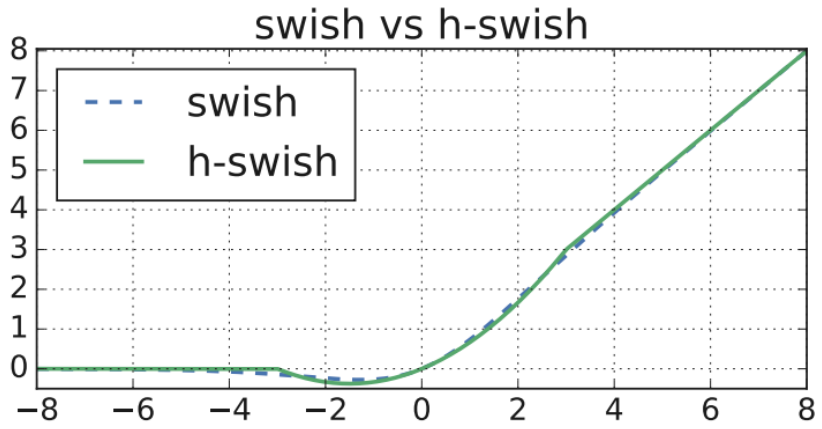


    - Emphasize class/context information

New Component: Swish Nonlinearities

- Unlike ReLU, swish activation function allows gradient flow for the small negative values
  - $f(x) = sigmoid(x) \cdot x$
  - However, swish function is expensive, so accuracy improvement over performance improvement is highly limited
  - Instead of using the original form, adopt hard-swish function
    - $f(x) = h - sigmoid(x) \cdot x, where\ h - simoid(x) = clip\left(\frac{x+3}{6}, 0, 1\right)$

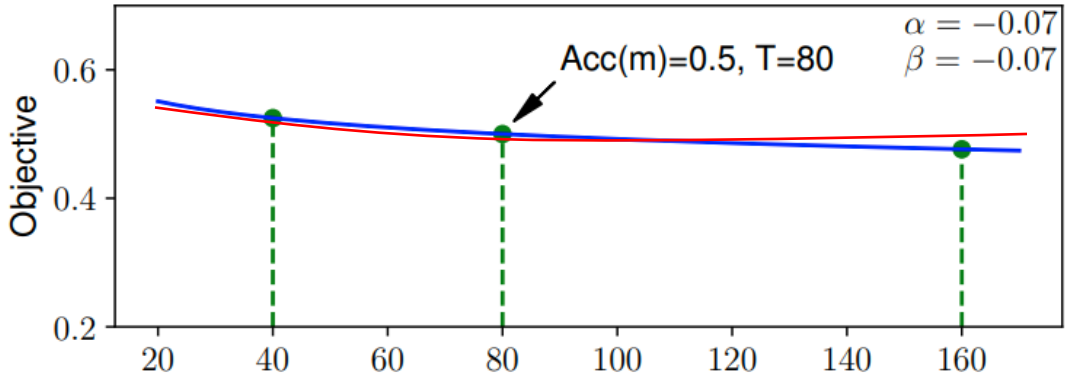




Architecture Search

- Step 1: Platform-Aware NAS for Block-wise Search
  - Author's own implementation show similar results to MnasNet
  - Large Model
    - reuse MnasNet-A1 as a initial seed
  - Smaller model
    - Perform MnasNet search once again, but with smaller weight $w = -0.15$ (vs-0.07)
    - The accuracy of smaller model is more vulnerable to latency optimization for accuracy

- $maximize_m\ ACC(m) \times \left[\frac{LAT(m)}{T}\right]^w, where\ w = \begin{cases} \alpha, if\ LAT(m) \le T \\ \beta, otherwise \end{cases}$



Architecture Search - 2

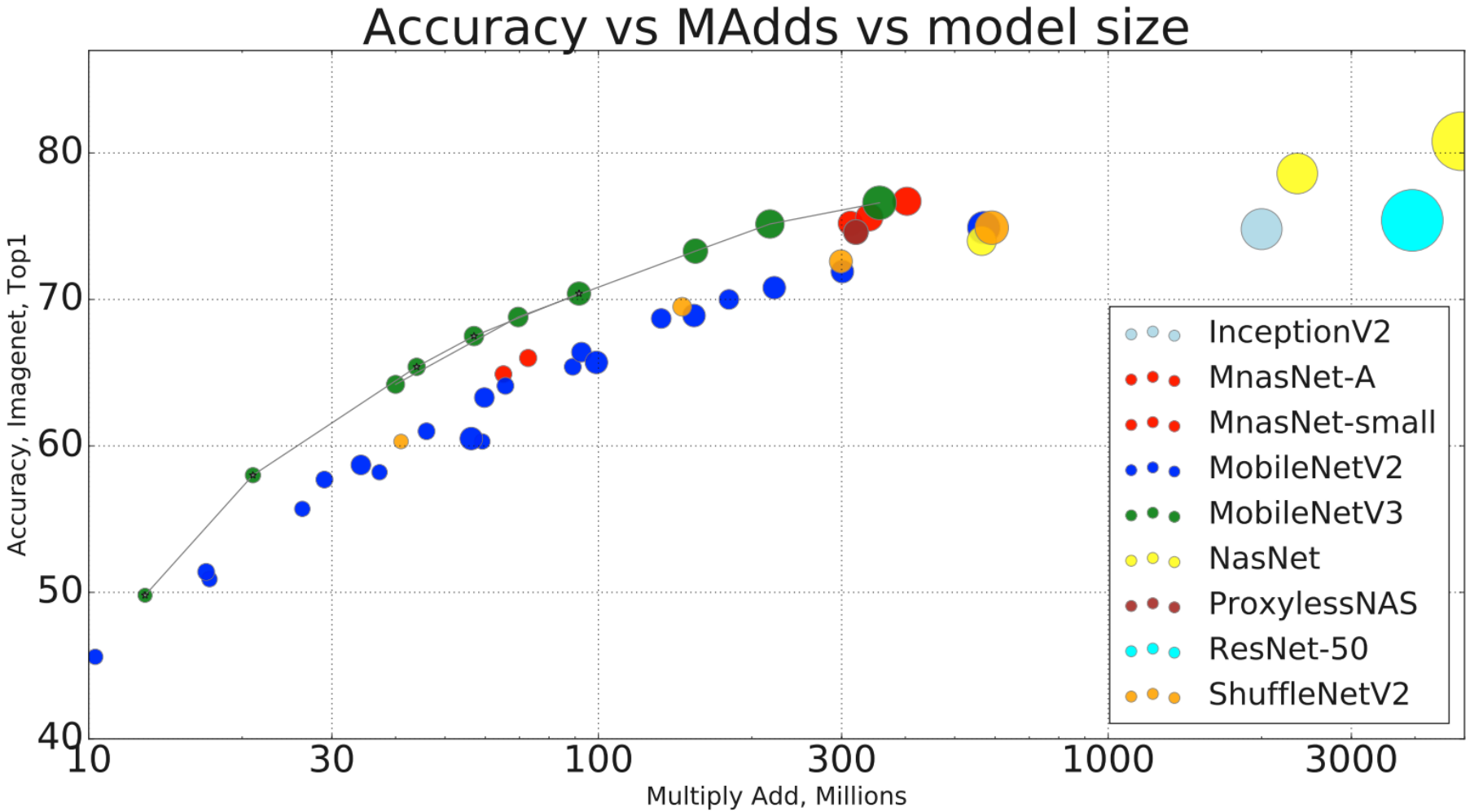
- Step 2: NetAdapt for layer-wise search
  1. Starts with a seed network architecture found by platform-aware NAS.
  2. For each step:
     1. Generate a set of new proposals. Each proposal represents a modification of an architecture that generates at least δ reduction in latency compared to the previous step.
     2. For each proposal we use the pre-trained model from the previous step and populate the new proposed architecture, truncating and randomly initializing missing weights as appropriate. Finetune each proposal for T steps to get a coarse estimate of the accuracy.
     3. Selected best proposal according to some metric.
  3. Iterate previous step until target latency is reached.

  - Pick the layer that maximize $\frac{\Delta Acc}{\Delta Latency}$

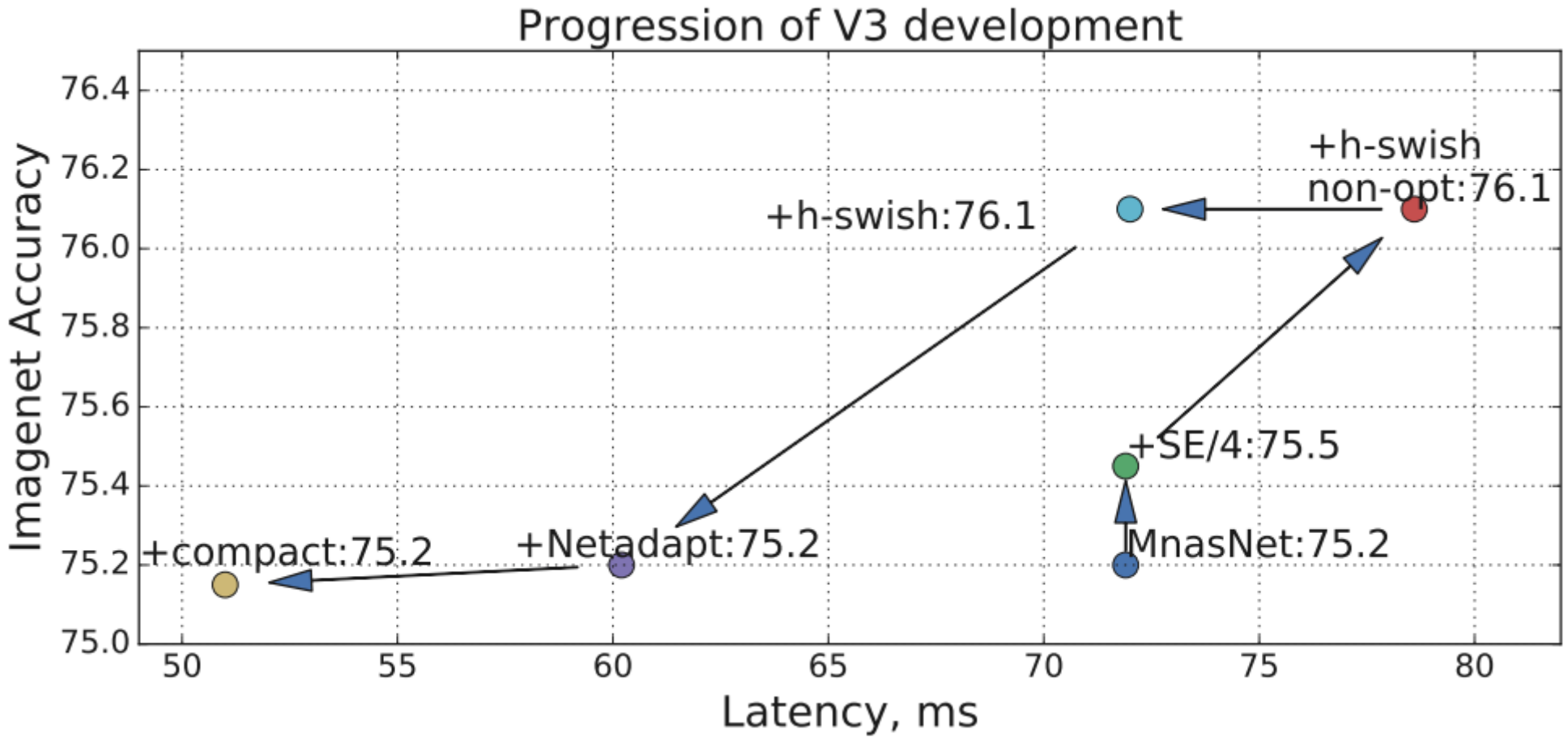
Results

Network	Top-1	MAdds	Params	P-1	P-2	P-3
V3-Large 1.0	**75.2**	**219**	5.4M	**51**	**61**	**44**
V3-Large 0.75	73.3	155	4.0M	39	46	40
MnasNet-A1	75.2	315	3.9M	71	86	61
Proxyless[5]	74.6	320	4.0M	72	84	60
V2 1.0	72.0	300	3.4M	64	76	56
V3-Small 1.0	**67.4**	**56**	2.5M	**15.8**	**19.4**	**14.4**
V3-Small 0.75	65.4	44	2.0M	12.8	15.6	11.7
Mnas-small [43]	64.9	65.1	1.9M	20.3	24.2	17.2
V2 0.35	60.8	59.2	1.6M	16.6	19.6	13.9

Results


Accuracy vs MAdds vs model size
Accuracy, Imagenet, Top1
80
70
60
50
40
10
30
100
300
1000
3000
Multiply Add, Millions
InceptionV2
MnasNet-A
MnasNet-small
MobileNetV2
MobileNetV3
NasNet
ProxylessNAS
ResNet-50
ShuffleNetV2

Results


Progression of V3 development
Imagenet Accuracy
76.4
76.2
76.0
75.8
75.6
75.4
75.2
75.0
50
55
60
65
70
75
80
Latency, ms
+h-swish
non-opt:76.1
+h-swish:76.1
+SE/4:75.5
MnasNet:75.2
+Netadapt:75.2
+compact:75.2

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Scaling Up of ConvNets

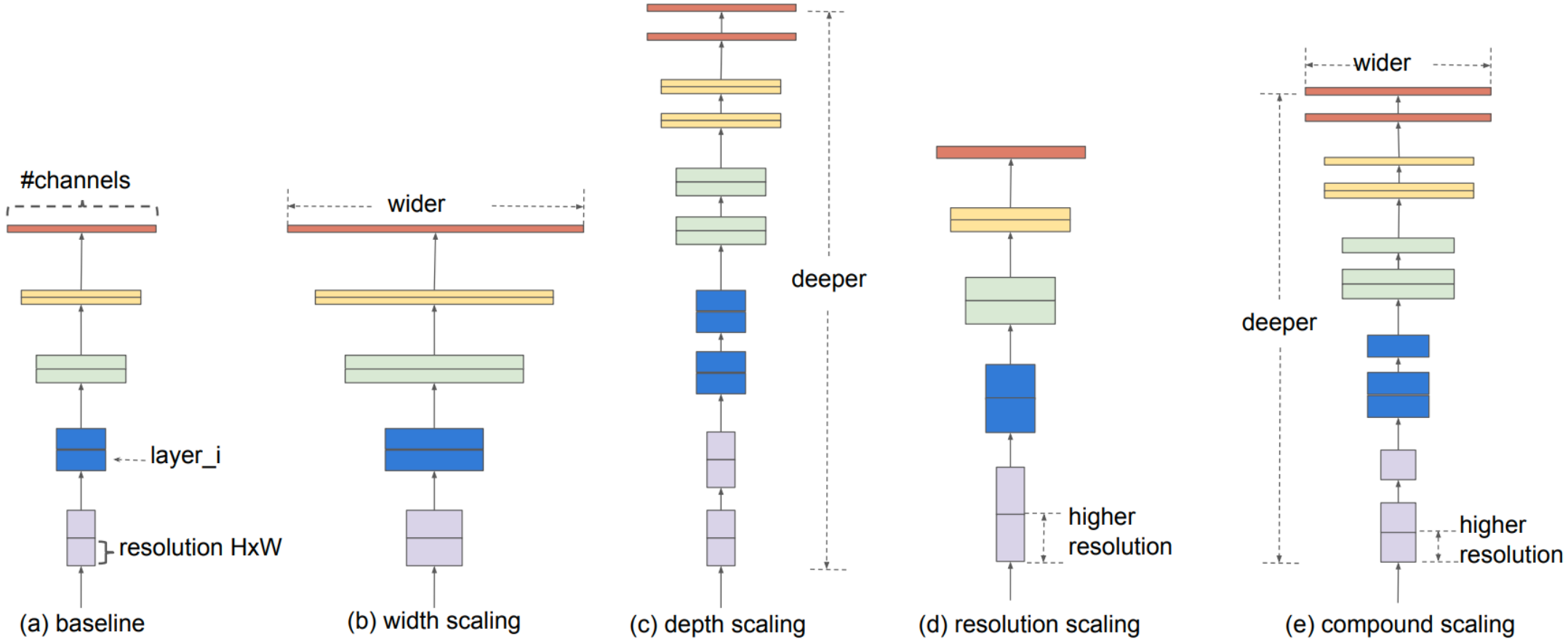


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Problem Formulation

- Basic ConvNets
  - $N = \bigodot_{i=1,\ldots,s} F_i^{L_i}\left(X_{(H_i,W_i,C_i)}\right)$
  - Scaling candidates
    - The network length L_i
    - The channel width C_i
    - Input resolution (H_i, W_i)
  - Target: maximize the model accuracy with given resource constraints
    - $\max Accuracy\left(N(d,w,r)\right)\ s.t.$

$$N(d,w,r) = \bigodot_{i=1,\ldots,s} \widehat{F_i}^{d\cdot\widehat{L_i}}\left(X_{(r\cdot\widehat{H_i},r\cdot\widehat{W_i},w\cdot\widehat{C_i})}\right)$$

    - Where d, w, r is the coefficients for scaling network

Observations

- 1. Scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models
- 2. In order to pursue better accuracy and efficiency, it is critical to balance all dimensions of network width, depth, and resolution during ConvNet scaling

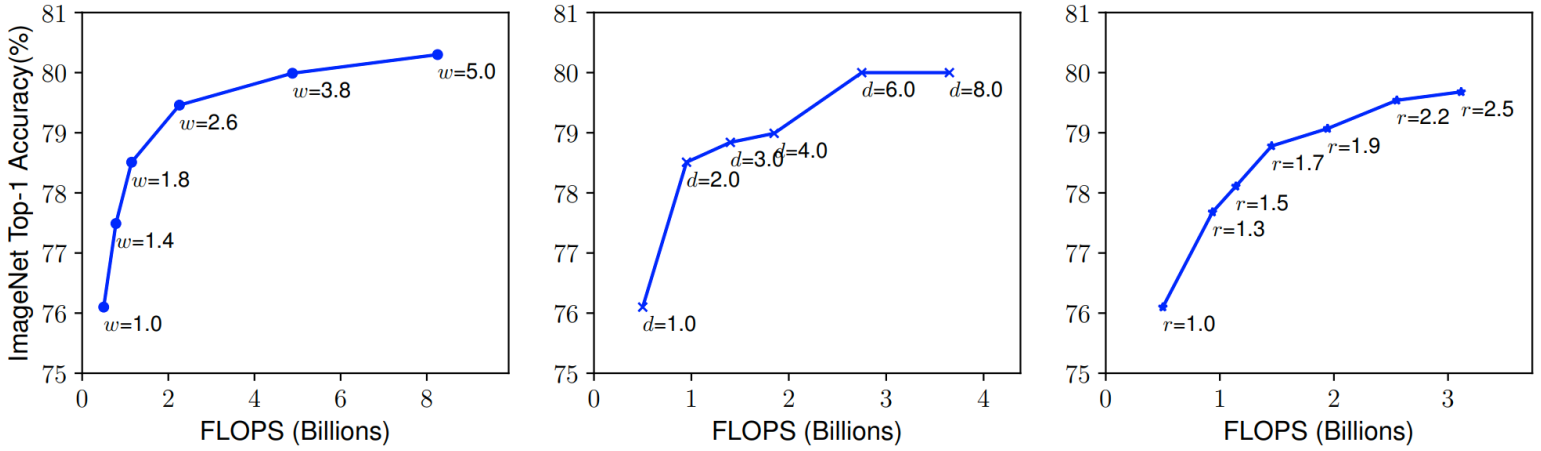


Figure 3. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling. Baseline network is described in Table 1.

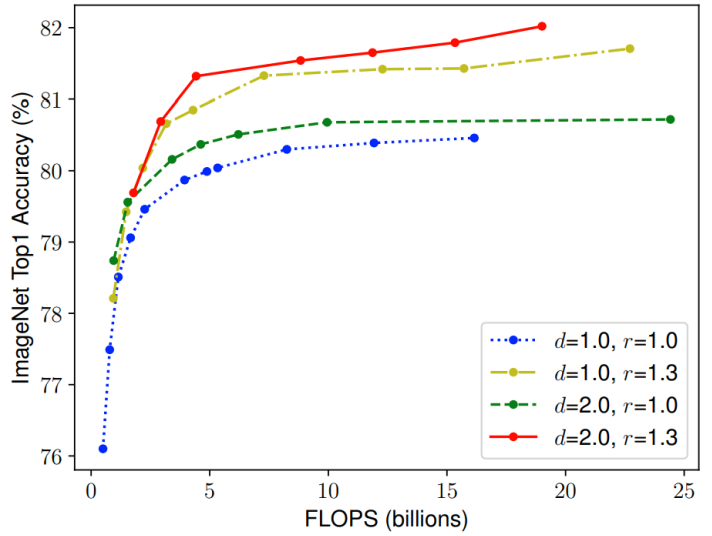


Figure 4. **Scaling Network Width for Different Baseline Networks**. Each dot in a line denotes a model with different width coefficient (w). All baseline networks are from Table 1. The first baseline network (d=1.0, r=1.0) has 18 convolutional layers with resolution 224x224, while the last baseline (d=2.0, r=1.3) has 36 layers with resolution 299x299.

Compound Scaling Method

- Use a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way

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

- Where α, β, γ are constants that can be determined by a small grid search

EfficientNet Architecture

- Exploit the search method of MnasNet, but optimize FLOPs instead of latency
  - $maximize_m\ ACC(m) \times \left[\frac{FLOPs(m)}{T}\right]^w$

- Starting from B0, apply compound scaling method
  - Step 1: fix $\phi = 1$ and do a grid search of α, β, γ. $\alpha = 1.2, \beta = 1.1$, $\gamma = 1.15$ under constraint of $\alpha\beta^2\gamma^2 \approx 2$
  - Step 2: fix α, β, γ and scale up the baseline to obtain B1 to B7

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Results

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Results

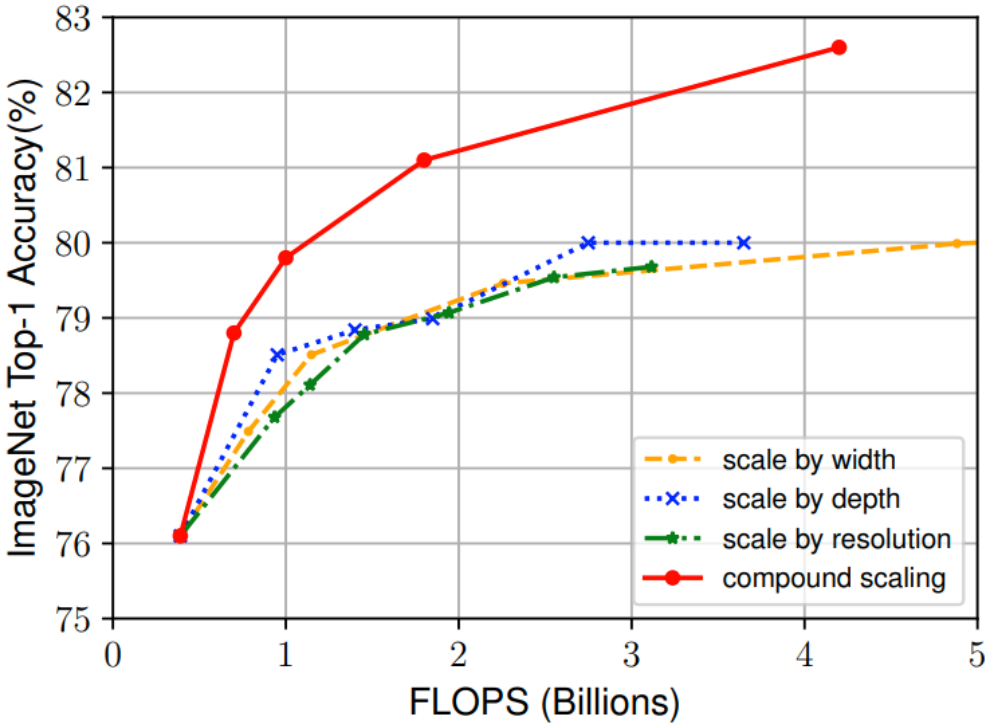


Figure 8. **Scaling Up EfficientNet-B0 with Different Methods.**

Table 3. **Scaling Up MobileNets and ResNet.**

Model	FLOPS	Top-1 Acc.
Baseline MobileNetV1 (Howard et al., 2017)	0.6B	70.6%
Scale MobileNetV1 by width (w=2)	2.2B	74.2%
Scale MobileNetV1 by resolution (r=2)	2.2B	72.7%
compound scale (d=1.4, w=1.2, r=1.3)	**2.3B**	**75.6%**
Baseline MobileNetV2 (Sandler et al., 2018)	0.3B	72.0%
Scale MobileNetV2 by depth (d=4)	1.2B	76.8%
Scale MobileNetV2 by width (w=2)	1.1B	76.4%
Scale MobileNetV2 by resolution (r=2)	1.2B	74.8%
MobileNetV2 compound scale	**1.3B**	**77.4%**
Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth (d=4)	16.2B	78.1%
Scale ResNet-50 by width (w=2)	14.7B	77.7%
Scale ResNet-50 by resolution (r=2)	16.4B	77.5%
ResNet-50 compound scale	**16.7B**	**78.8%**

Once For All: Train One Network and Specialize It for Efficient Deployment

Multiple Targets for Diverse Devices

- Every computation unit have divergent design principles
  - Utilization of the units are different depending on the network architecture
    - CPU: SIMD-based computation, diverse functionality
    - GPU: 1000s of cores, large parallelism, moderate functionality
    - NPU: 100~10ks of PEs, limited functionality

- Finding out the optimal network per accelerator is prohibitively expensive
  - OFA tries to optimize the multiple candidates at once

Possible Solutions and Problems

- We may train diverse networks as sub networks of a large base network based on parameter-sharing idea
  - Sample multiple networks from the base network and jointly train the sub networks sharing the same parameters
  - Can be seen as an operator-wise dropout and ensemble learning

- Problem?
  - Different sub-networks are interfering with each other
  - Making the training process of the network inefficient
    - Spend longer time to converge
    - Quality of the trained networks are suboptimal

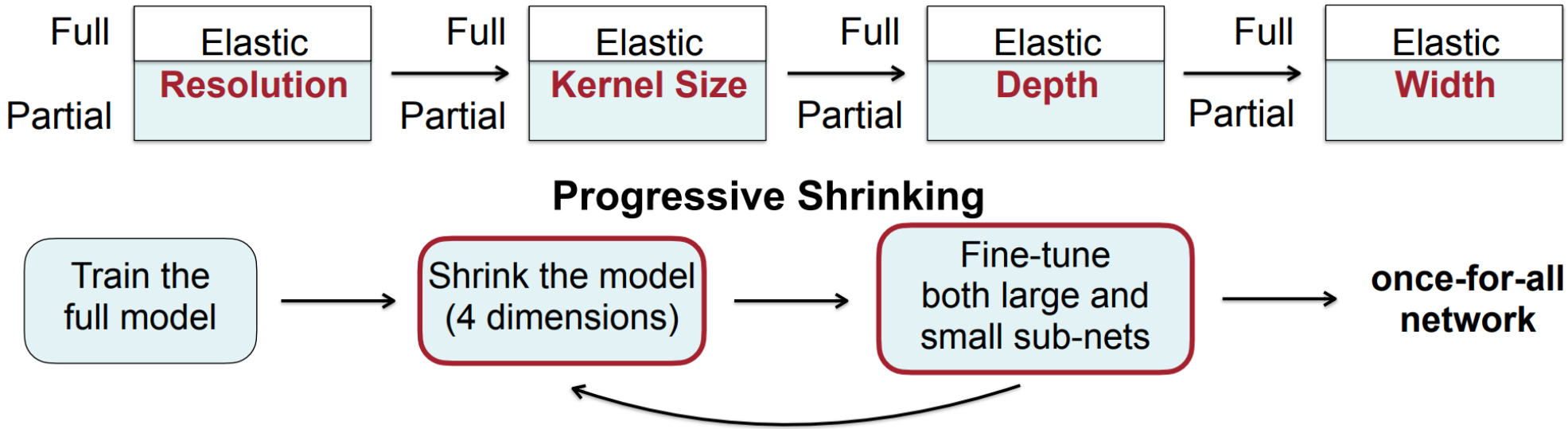
Once-for-all Network Space

- Problem formulation
  - $\min_{W_0} \Sigma_{arch} L_{val}\big(C(W_0, arch_i)\big)$, where $C(W_0, arch_i)$ is a selection scheme from the once-for-all network W_0 to form a subnetwork with configuration $arch_i$

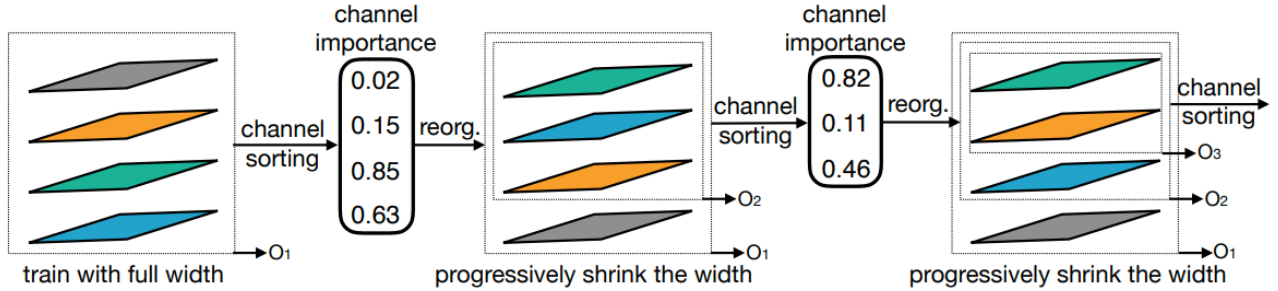
  - Sub-network options
    - Input resolution 128 to 224 with a stride 4
    - Depth of unit 2, 3, 4
    - Width expansion ration 3, 4, 6
    - Kernel size 3, 5, 7
    - With 5 units, roughly 2×10^{19} different architectures, 25 different input resolutions

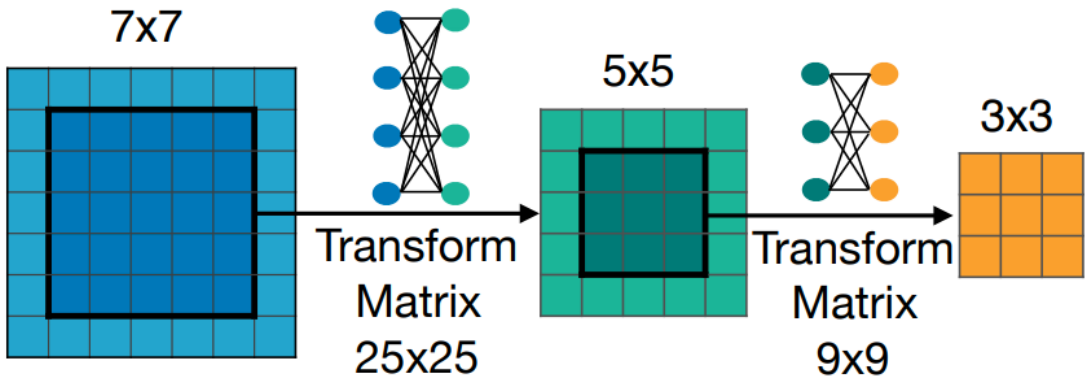
Training Method

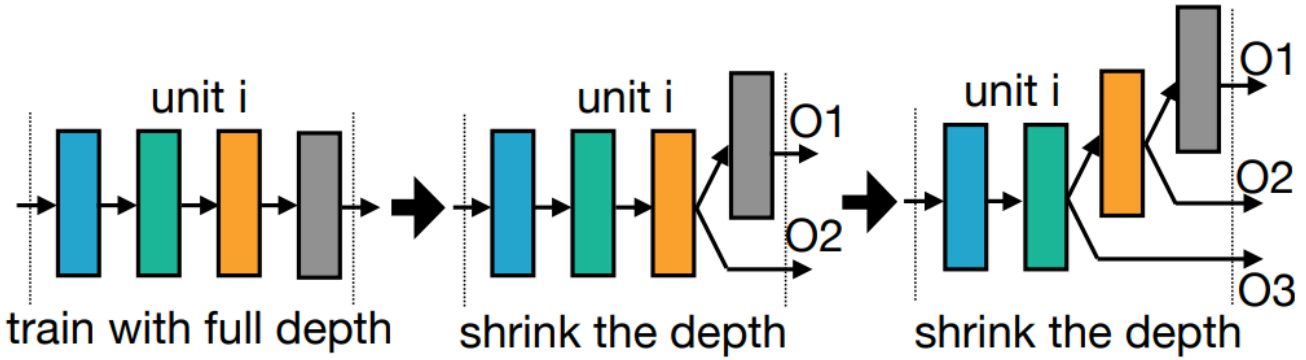
- Progressive Shrinking
  - Enforce a training order from large sub-networks to small sub-networks in a progressive manner
  - Start with training the largest neural network with the maximum kernel size (e.g., 7), depth (e.g., 4), and width (e.g., 6)
  - Progressively fine-tune the network to support smaller sub-networks by gradually adding them into the sampling space



Elasticity of Network







- Kernel size
  - Center of kernel weights are shared for the smaller convolution
- Depth
  - First D layers are shared
- Width
  - Calculate L_1 norm of each channel and prune the less important one

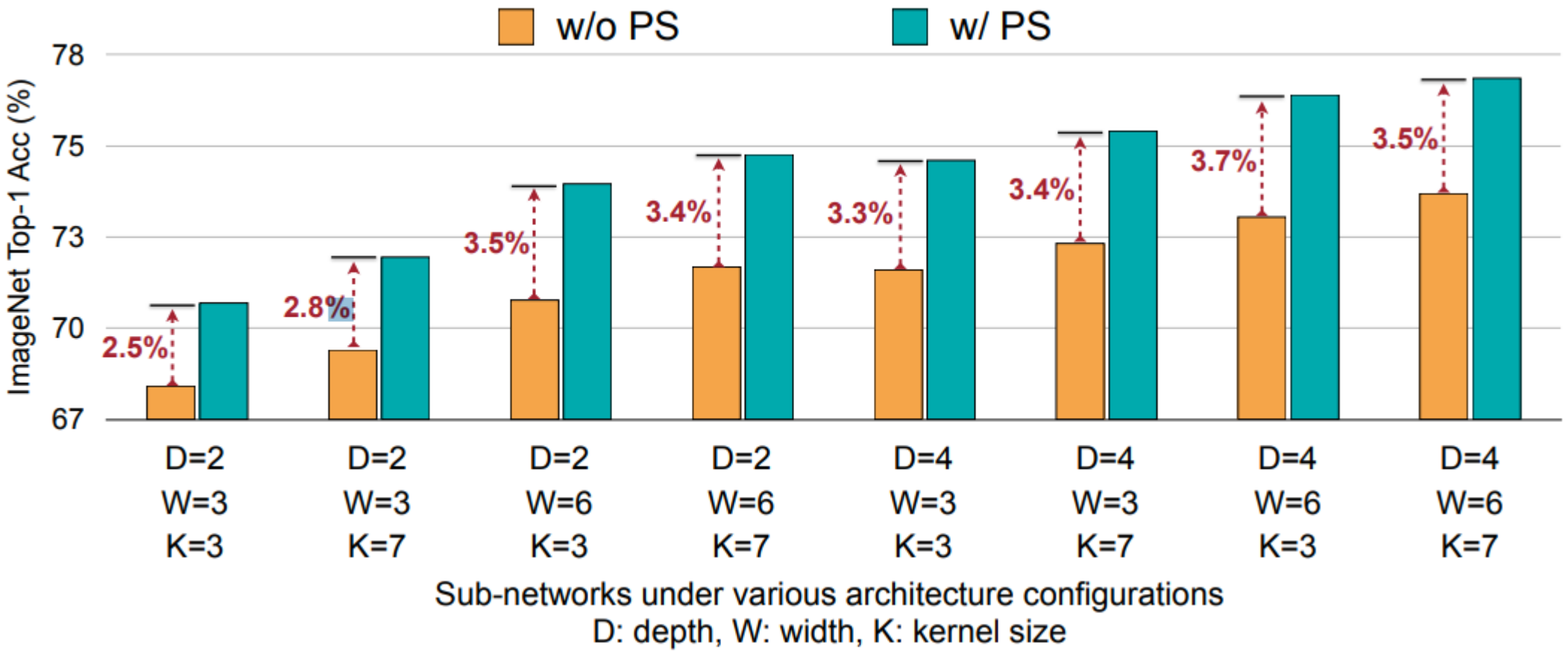
Training Pipeline

- Train the largest base network
- Train the sub-network with progressive shrinking
  - Please note that teacher-student is applied
    - The based network is used as a teacher

- Train the performance predictor based on the sub-networks
  - Sample 16K sub-networks with different architectures and input image size
    - Their accuracy is measured on 10K validation images on the graining set

- Based on the trained predictor, apply evolutionary search to get a specialized network

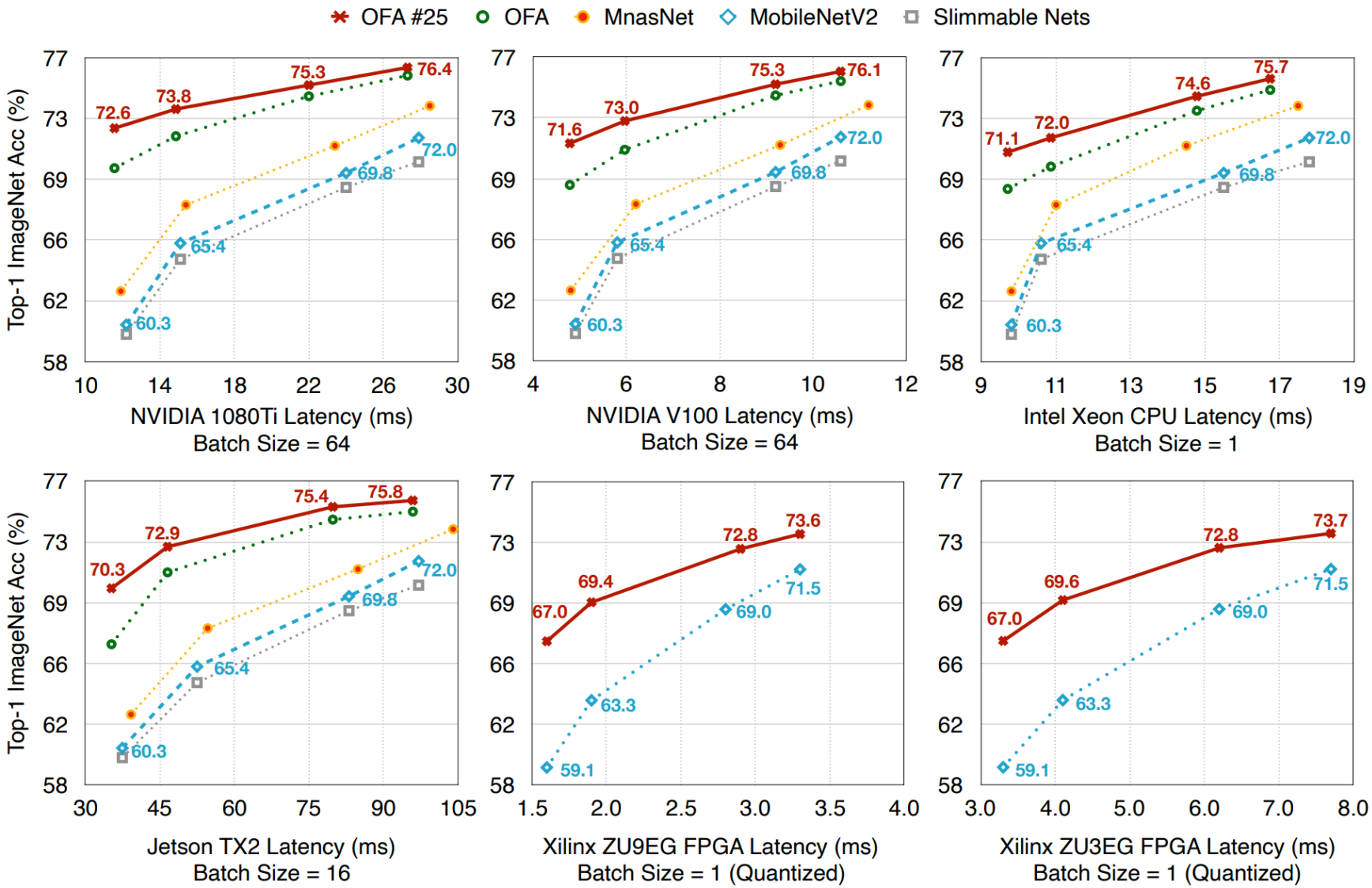
Results


w/o PS
w/ PS
ImageNet Top-1 Acc (%)
78
75
73
70
67
2.5%
2.8%
3.5%
3.4%
3.3%
3.4%
3.7%
3.5%
D=2 W=3 K=3
D=2 W=3 K=7
D=2 W=6 K=3
D=2 W=6 K=7
D=4 W=3 K=3
D=4 W=3 K=7
D=4 W=6 K=3
D=4 W=6 K=7
Sub-networks under various architecture configurations
D: depth, W: width, K: kernel size

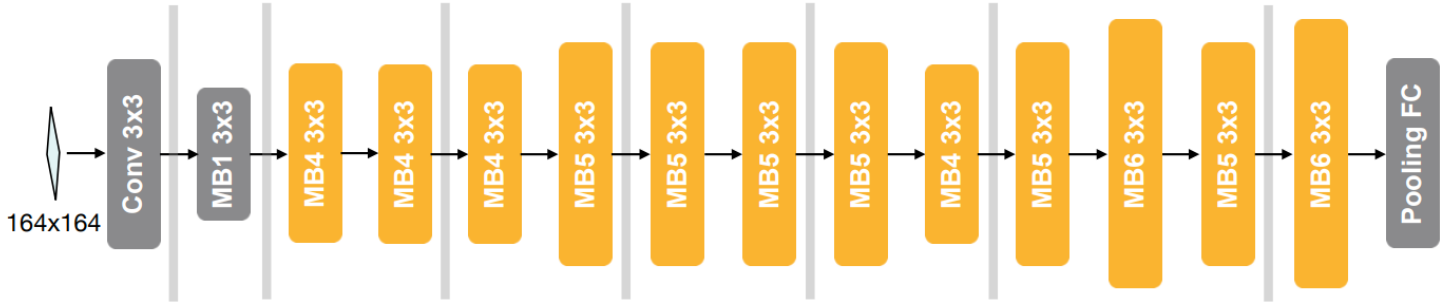
Results

Model	ImageNet Top1 (%)	MACs	Mobile latency	Search cost (GPU hours)	Training cost (GPU hours)	Total cost ($N = 40$)		
						GPU hours	CO_2e (lbs)	AWS cost
MobileNetV2 [31]	72.0	300M	66ms	0	$150N$	6k	1.7k	$18.4k
MobileNetV2 #1200	73.5	300M	66ms	0	$1200N$	48k	13.6k	$146.9k
NASNet-A [44]	74.0	564M	-	$48{,}000N$	-	1,920k	544.5k	$5875.2k
DARTS [25]	73.1	595M	-	$96N$	$250N$	14k	4.0k	$42.8k
MnasNet [33]	74.0	317M	70ms	$40{,}000N$	-	1,600k	453.8k	$4896.0k
FBNet-C [36]	74.9	375M	-	$216N$	$360N$	23k	6.5k	$70.4k
ProxylessNAS [4]	74.6	320M	71ms	$200N$	$300N$	20k	5.7k	$61.2k
SinglePathNAS [8]	74.7	328M	-	$288 + 24N$	$384N$	17k	4.8k	$52.0k
AutoSlim [38]	74.2	305M	63ms	180	$300N$	12k	3.4k	$36.7k
MobileNetV3-Large [15]	75.2	219M	58ms	-	$180N$	7.2k	1.8k	$22.2k
OFA w/o PS	72.4	235M	59ms	40	1200	1.2k	0.34k	$3.7k
OFA w/ PS	**76.0**	230M	58ms	40	1200	1.2k	0.34k	$3.7k
OFA w/ PS #25	**76.4**	230M	58ms	40	$1200 + 25N$	2.2k	0.62k	$6.7k
OFA w/ PS #75	**76.9**	230M	58ms	40	$1200 + 75N$	4.2k	1.2k	$13.0k
$\text{OFA}_{\text{Large}}$ w/ PS #75	**80.0**	595M	-	40	$1200 + 75N$	4.2k	1.2k	$13.0k

Results


OFA #25
OFA
MnasNet
MobileNetV2
Slimmable Nets
Top-1 ImageNet Acc (%)
NVIDIA 1080Ti Latency (ms)
Batch Size = 64
NVIDIA V100 Latency (ms)
Batch Size = 64
Intel Xeon CPU Latency (ms)
Batch Size = 1
Jetson TX2 Latency (ms)
Batch Size = 16
Xilinx ZU9EG FPGA Latency (ms)
Batch Size = 1 (Quantized)
Xilinx ZU3EG FPGA Latency (ms)
Batch Size = 1 (Quantized)

Results



(a) 4.1ms latency on Xilinx ZU3EG (batch size = 1).

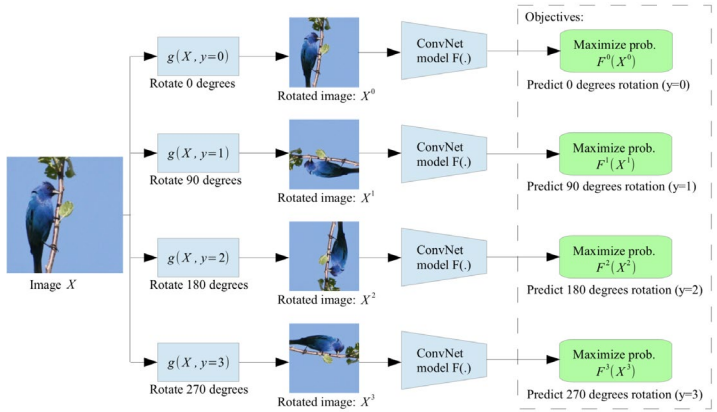
(b) 10.9ms latency on Intel Xeon CPU (batch size = 1).

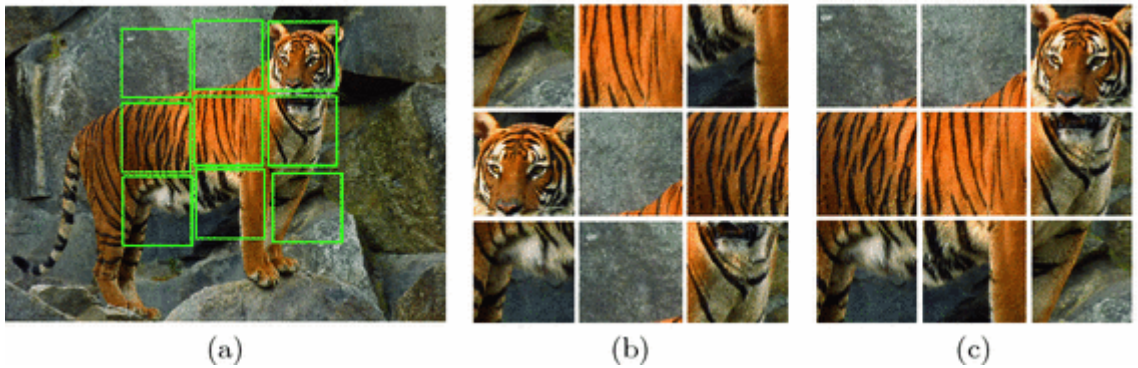
(c) 14.9ms latency on NVIDIA 1080Ti (batch size = 64).

Are Labels Necessary for Neural Architecture Search?

Unsupervised Training for CV

- Rotation, colorization, jigsaw, similarity, ..

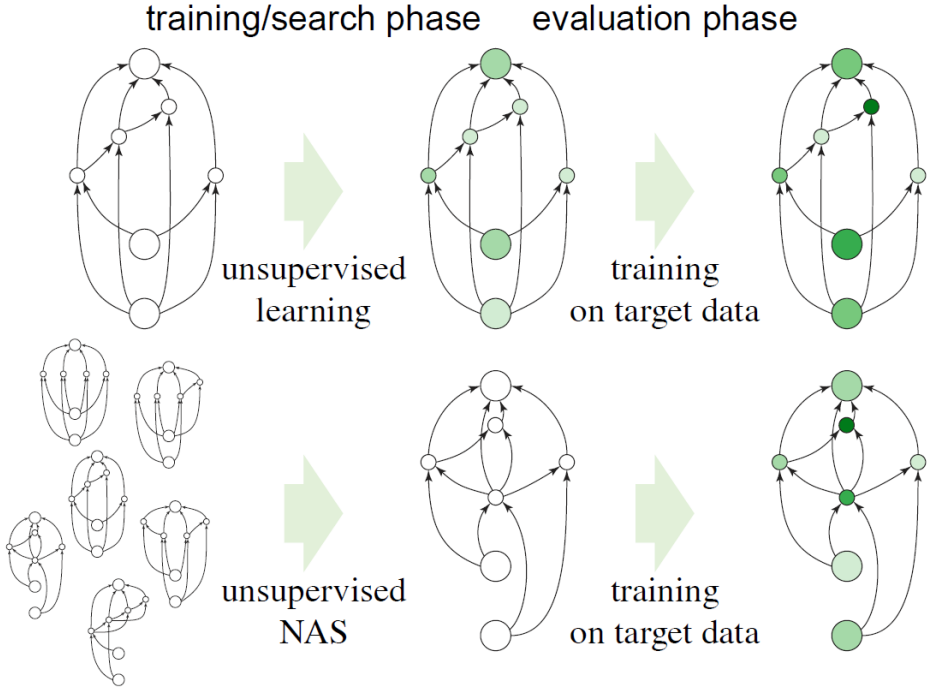




Unsupervised NAS

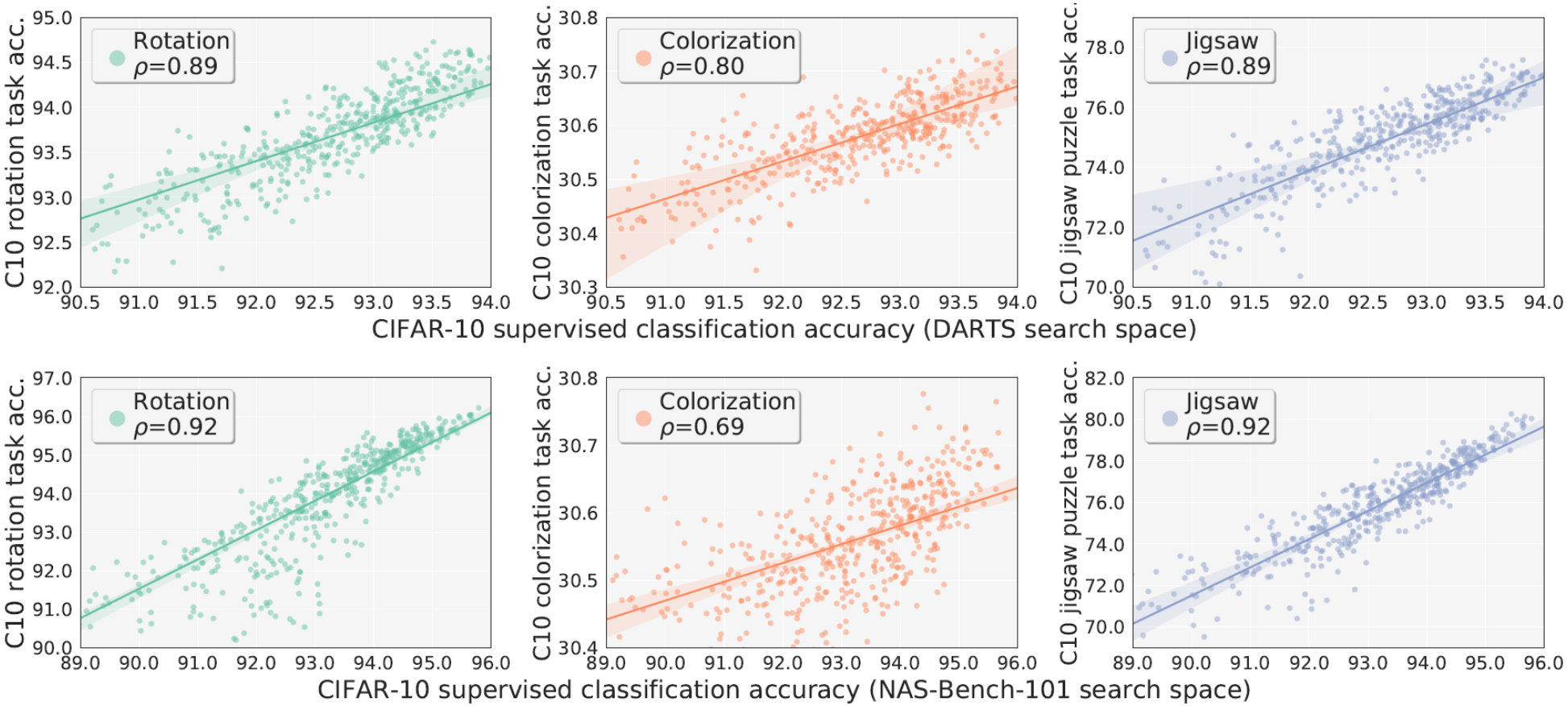
- Unsupervised(self-supervised) problems can be used to measure the network quality
  - Rotation, colorization and jigsaw puzzle use cross-entropy loss
  - These functionalities can replace classification loss

- Perform NAS without label → evaluation using the target dataset



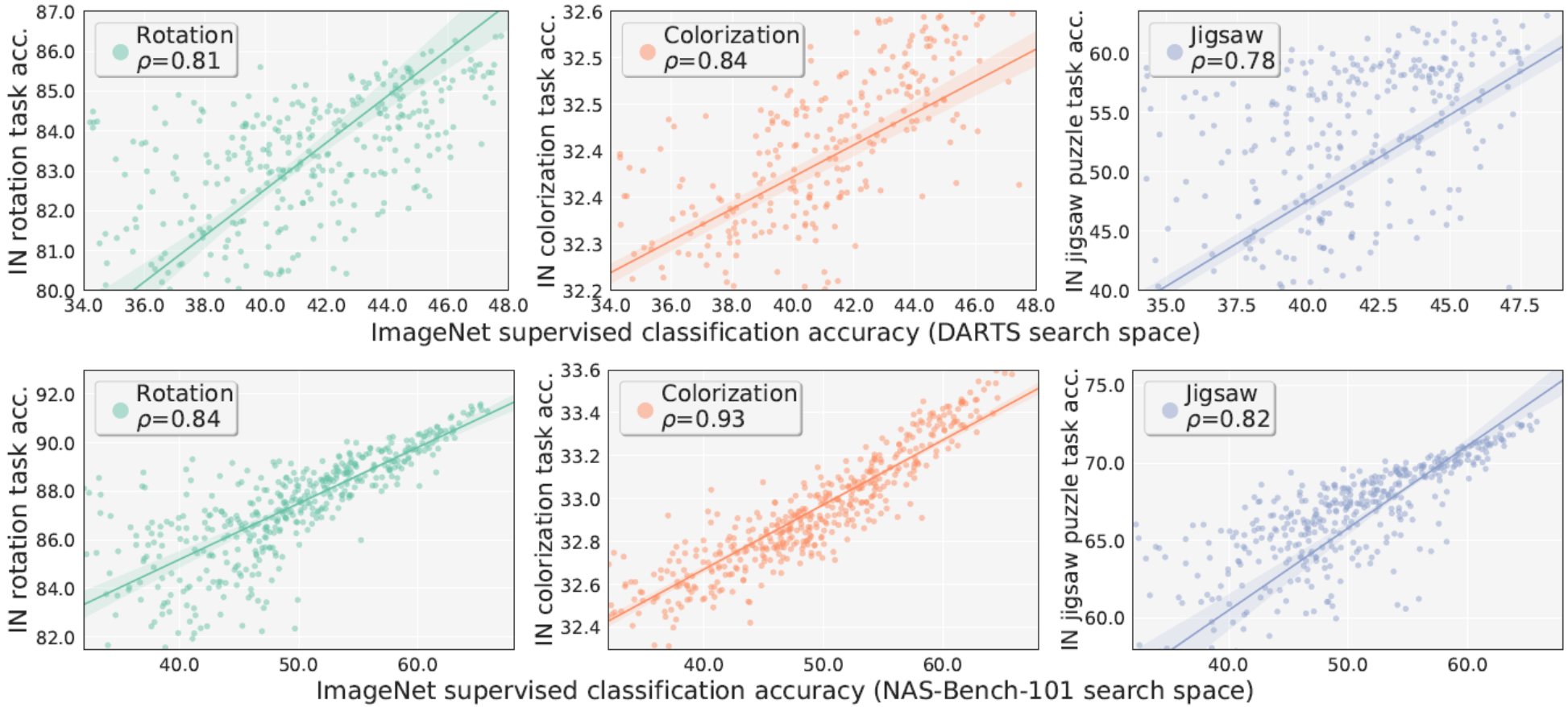
Correlation between Supervised vs Unsupervised – Cifar10

- While colorization show slightly lower correlation, other methods is highly correlated to the classification task accuracy



Correlation between Supervised vs Unsupervised – ImageNet1k

- Interestingly, colorization works better in the case of the imagenet dataset
  - The small images in Cifar-10 make the colorization difficult



Results

method	search dataset & task	top-1 acc.	FLOPs (M)	params (M)
NAS-DARTS [20]	CIFAR-10 Supv.Cls	73.3	574	4.7
NAS-P-DARTS [6]	CIFAR-10 Supv.Cls	75.6	557	4.9
NAS-PC-DARTS [33]	CIFAR-10 Supv.Cls	74.9	586	5.3
NAS-PC-DARTS [33]	IN1K Supv.Cls	75.8	597	5.3
NAS-DARTS†	CIFAR-10 Supv.Cls	$74.9_{\pm 0.08}$	538	4.7
NAS-DARTS	IN1K Supv.Cls	$76.3_{\pm 0.06}$	590	5.3
UnNAS-DARTS	IN1K Rot	$75.8_{\pm 0.18}$	558	5.1
UnNAS-DARTS	IN1K Color	$75.7_{\pm 0.12}$	547	4.9
UnNAS-DARTS	IN1K Jigsaw	$75.9_{\pm 0.15}$	567	5.2
NAS-DARTS	IN22K Supv.Cls	$75.9_{\pm 0.09}$	585	5.2
UnNAS-DARTS	IN22K Rot	$75.7_{\pm 0.23}$	549	5.0
UnNAS-DARTS	IN22K Color	$75.9_{\pm 0.21}$	547	5.0
UnNAS-DARTS	IN22K Jigsaw	$75.9_{\pm 0.31}$	559	5.1
NAS-DARTS	Cityscapes Supv.Seg	$75.8_{\pm 0.13}$	566	5.1
UnNAS-DARTS	Cityscapes Rot	$75.9_{\pm 0.19}$	554	5.1
UnNAS-DARTS	Cityscapes Color	$75.2_{\pm 0.15}$	594	5.1
UnNAS-DARTS	Cityscapes Jigsaw	$75.5_{\pm 0.06}$	566	5.0

Table 1: **ImageNet-1K classification results of the architectures searched by NAS and UnNAS algorithms**. Rows in gray correspond to invalid UnNAS configurations where the search and evaluation datasets are the same. † is our training result of the DARTS architecture released in [20].

Results

method	search dataset & task	mIoU	FLOPs (B)	params (M)
NAS-DARTS†	CIFAR-10 Supv.Cls	$72.6_{\pm 0.55}$	121	9.6
NAS-DARTS	IN1K Supv.Cls	$73.6_{\pm 0.31}$	127	10.2
UnNAS-DARTS	IN1K Rot	$73.6_{\pm 0.29}$	129	10.4
UnNAS-DARTS	IN1K Color	$72.2_{\pm 0.56}$	122	9.7
UnNAS-DARTS	IN1K Jigsaw	$73.1_{\pm 0.17}$	129	10.4
NAS-DARTS	IN22K Supv.Cls	$72.4_{\pm 0.29}$	126	10.1
UnNAS-DARTS	IN22K Rot	$72.9_{\pm 0.23}$	128	10.3
UnNAS-DARTS	IN22K Color	$73.6_{\pm 0.41}$	128	10.3
UnNAS-DARTS	IN22K Jigsaw	$73.1_{\pm 0.59}$	129	10.4
NAS-DARTS	Cityscapes Supv.Seg	$72.4_{\pm 0.15}$	128	10.3
UnNAS-DARTS	Cityscapes Rot	$73.0_{\pm 0.25}$	128	10.3
UnNAS-DARTS	Cityscapes Color	$72.5_{\pm 0.31}$	122	9.5
UnNAS-DARTS	**Cityscapes Jigsaw**	$\mathbf{74.1_{\pm 0.39}}$	**128**	**10.2**

Table 2: **Cityscapes semantic segmentation results of the architectures searched by NAS and UnNAS algorithms**. These are trained from scratch: there is no fine-tuning from ImageNet checkpoint. Rows in gray correspond to an illegitimate setup where the search dataset is the same as the evaluation dataset. † is our training result of the DARTS architecture released in [20].