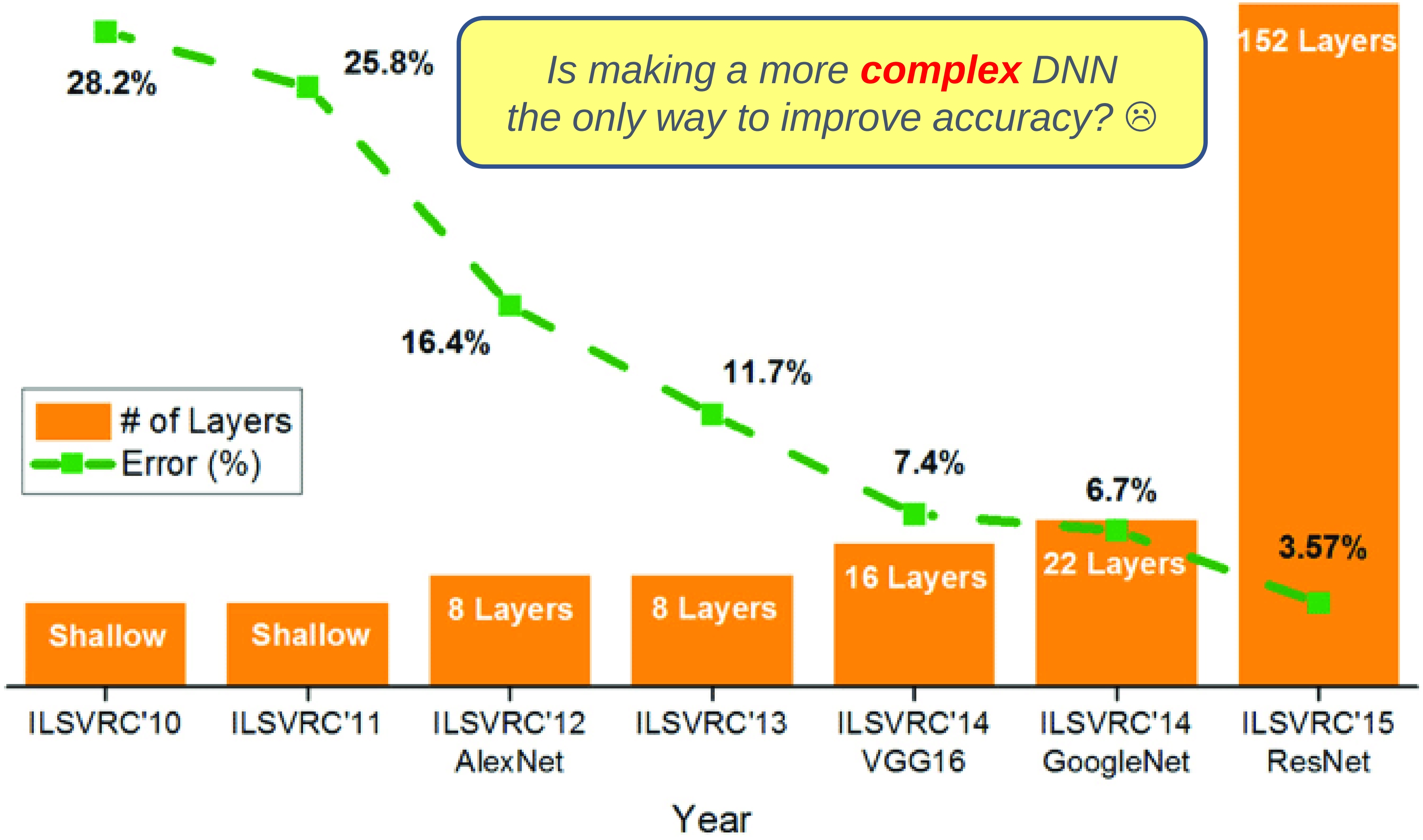
Review

- **CNN**: Conv layer and pooling layer, <u>sparsity of connections</u>, <u>parameter sharing</u>
- **AlexNet [2012]**: <u>Pioneering work</u> but complex architecture
- **VGG [2015]**: A deeper NN with <u>unified architecture</u> but heavy computation
- **Inception [2015]**: A deeper NN with <u>Inception module</u> (all-in-one!) including <u>bottleneck layer</u> to go deeper efficiently
- **ResNet [2016]**: A super deep NN with <u>skip connection</u> to train deep NN robustly

Review

*Is making a more **complex** DNN the only way to improve accuracy?* ☹



Lightweight CNN for 2D Obejct Classification

Lecture 3

Hyung-Sin Kim

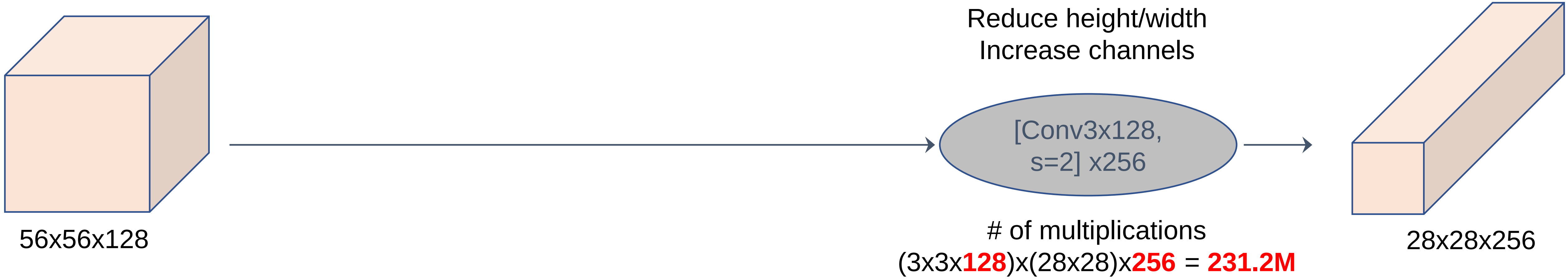
SNU Graduate School of Data Science

We’ve already seen that using bottleneck layer reduces computation of a convolutional layer

*Today we will learn **MobileNet** that uses another technique for reducing computation and memory of a convolutional layer*
– Depth-wise Separable Convolution –

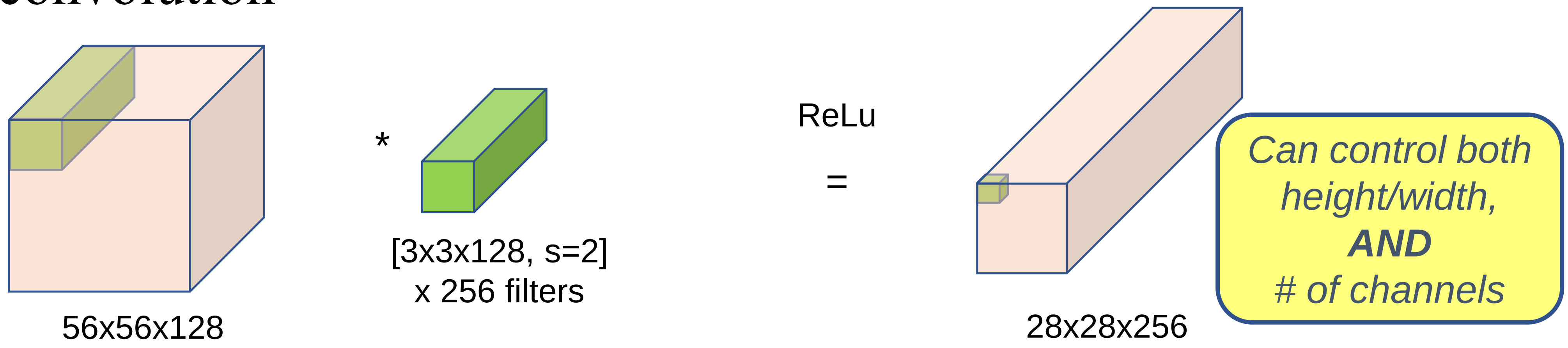
MobileNet [arxiv'17] – Motivation

- Again, a standard conv layer requires heavy computation when # of channels for the input (or output) is large

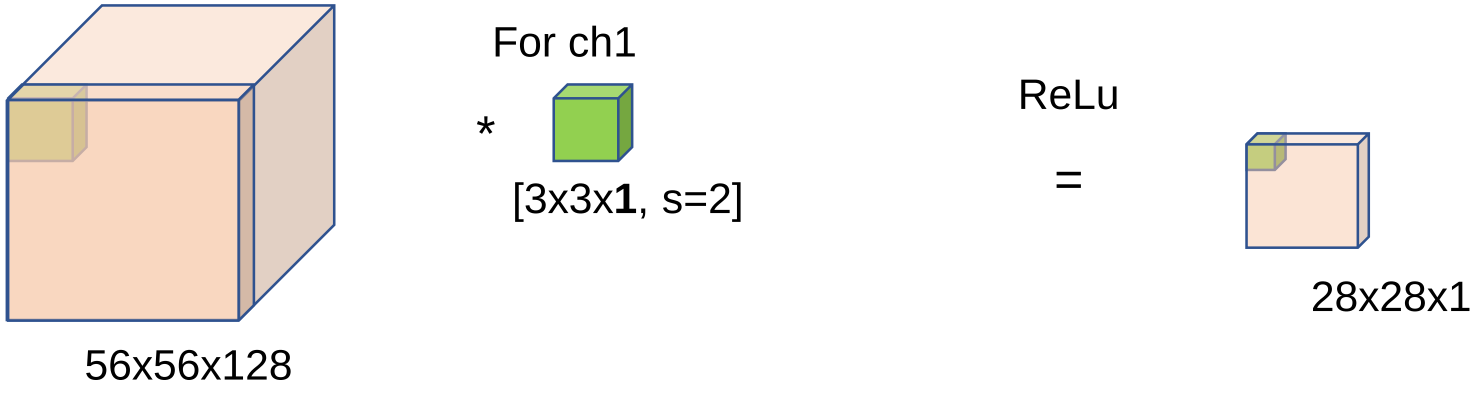


MobileNet [arxiv'17] – Depthwise Convolution

- Standard convolution

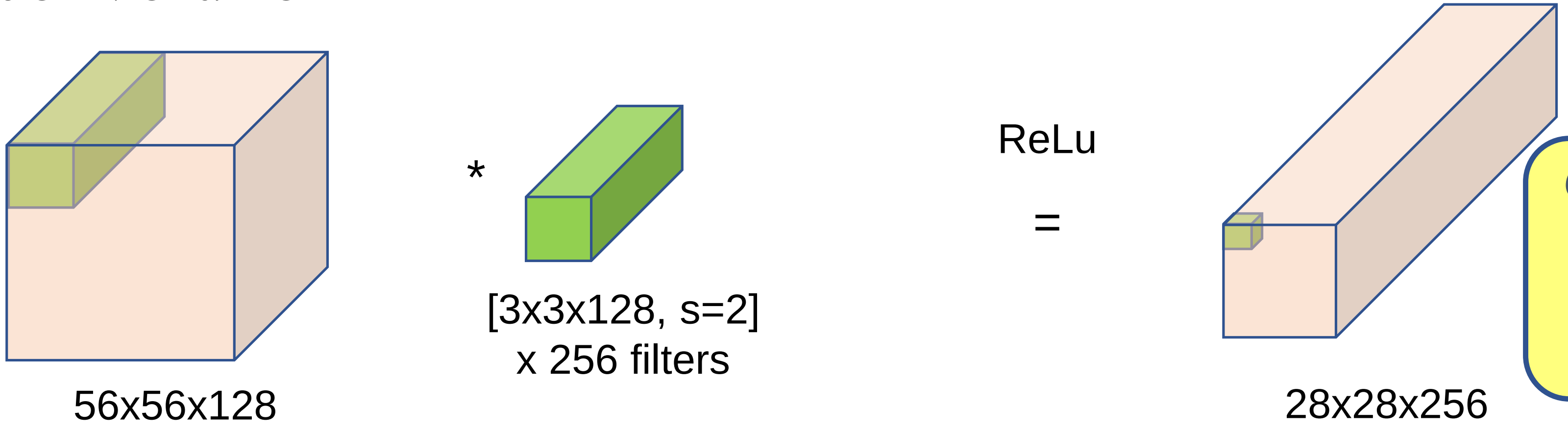


- Depthwise convolution



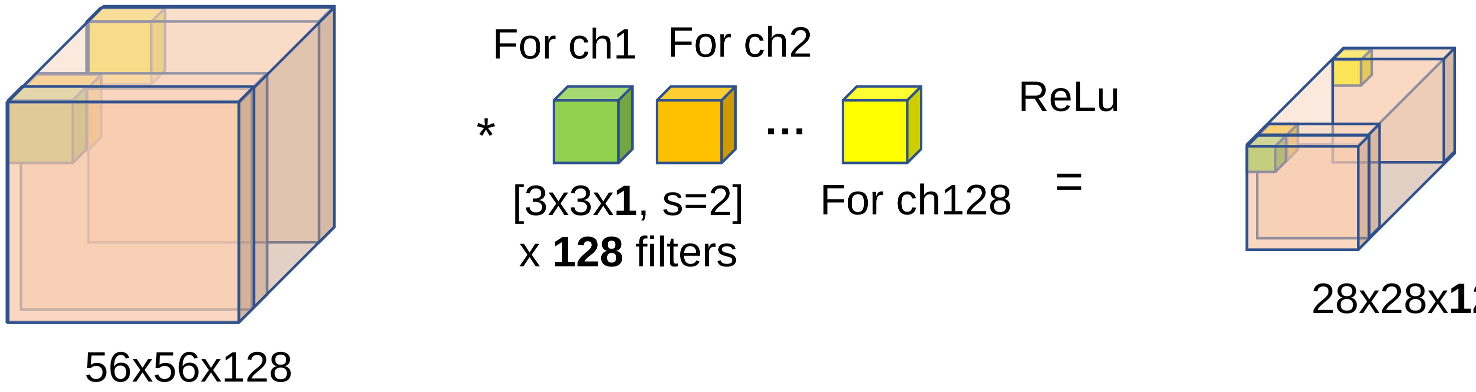
MobileNet [arxiv'17] – Depthwise Convolution

- Standard convolution



Can control both height/width,
AND
# of channels

- Depthwise convolution



Can control height/width,
*but **NOT***
# of channels

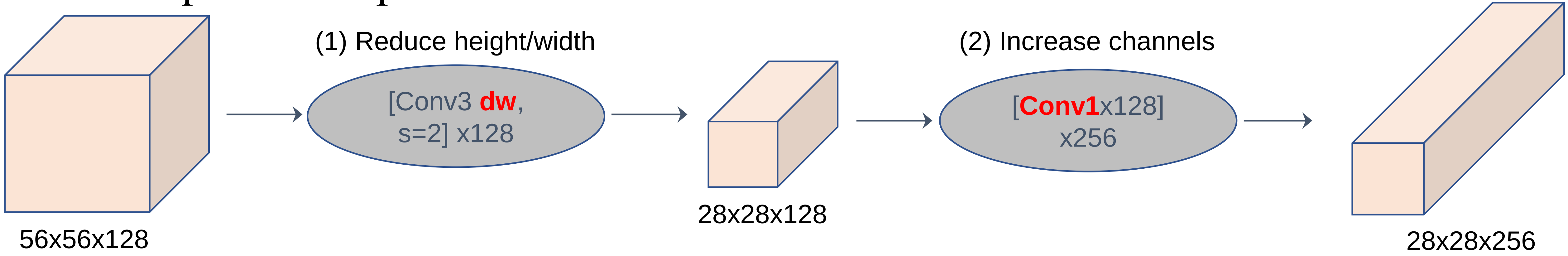
How about # of channels then?

Don’t worry.
*We have **1x1 Conv layer** which is useful for channel control*

MobileNet [arxiv’17] – Depthwise Separable Conv

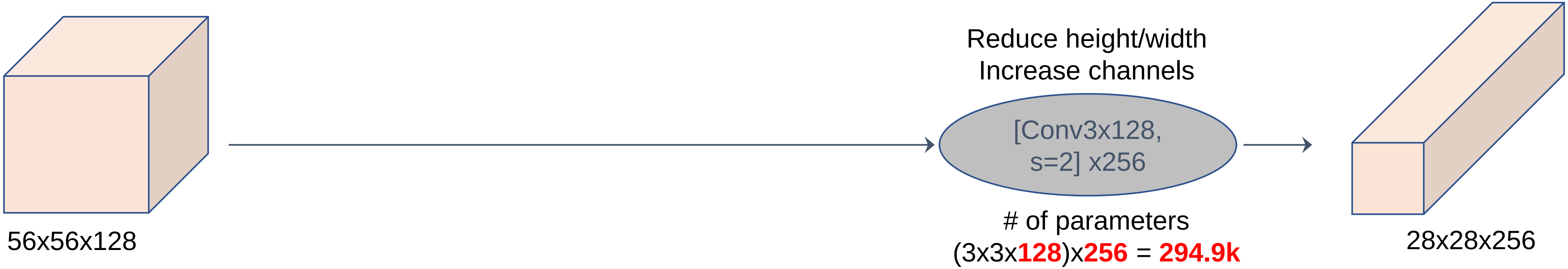
- Standard convolution

- Depthwise separable convolution

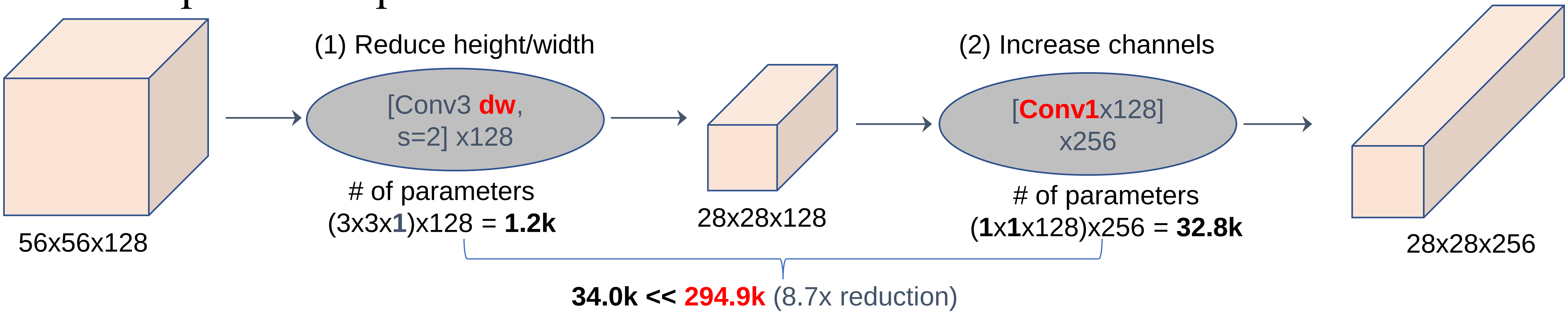


MobileNet [arxiv'17] – Parameter Reduction

- Standard convolution



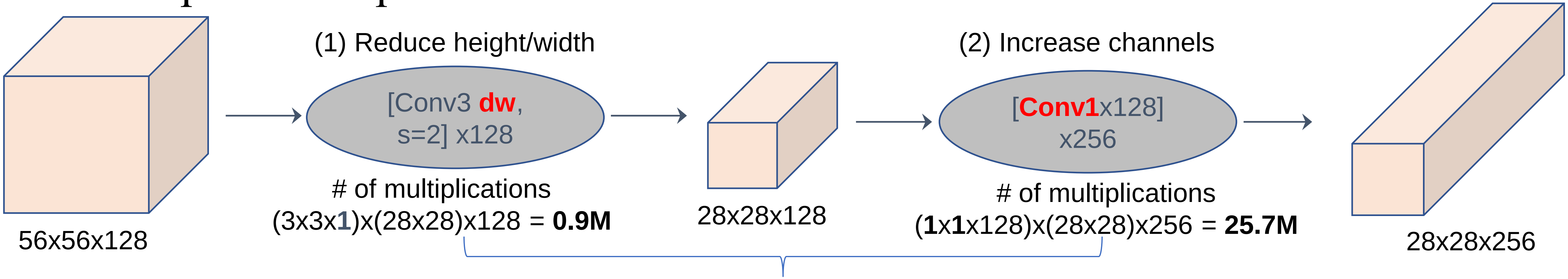
- Depthwise separable convolution



MobileNet [arxiv'17] – Computation Reduction

- Standard convolution

- Depthwise separable convolution



34.6M << 231.2M (6.7x reduction)

MobileNet [arxiv'17] – Computation Reduction

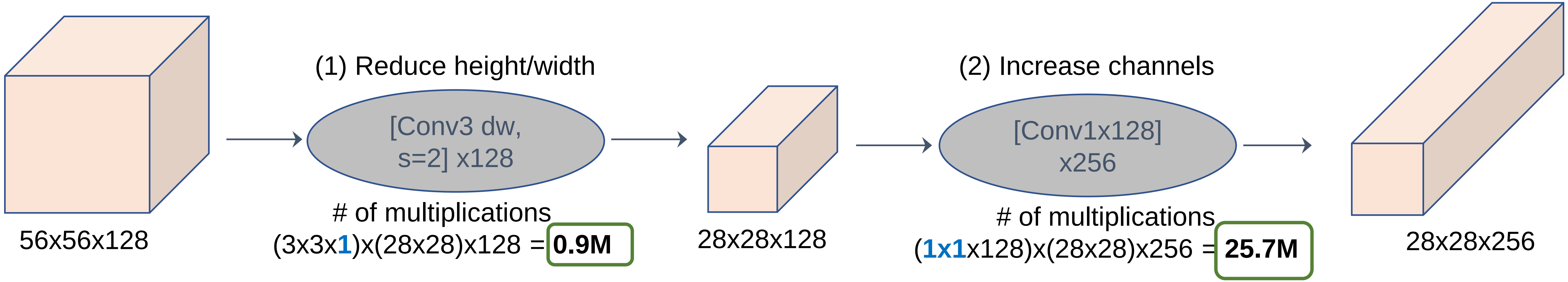
- Standard convolution
  - # of multiplications = $(D_F$ x D_F x $C_{in})$ x $(D_{out}$ x $D_{out})$ x C_{out}

- Depthwise separable convolution
  - # of multiplications = $(D_F$ x D_F x 1) x $(D_{out}$ x $D_{out})$ x C_{in} + (1 x 1 x $C_{in})$ x $(D_{out}$ x $D_{out})$ x C_{out}

- Gain
  - $\frac{1}{C_{out}} + \frac{1}{D_F^2}$

MobileNet [arxiv’17] – Computation Efficiency

- NxN Conv vs. 1x1 Conv



***96.7%** of multiplications are for 1x1 Conv!*

Remember!
1x1 Conv is more computationally efficient than f x f Conv

MobileNet [arxiv'17] – The Whole Architecture

- A **28-layer** but lightweight DNN

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 4. Depthwise Separable vs Full Convolution MobileNet

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

Only 1% accuracy loss
Huge reduction of parameters and computation

[The tables are from AG. Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications."]

MobileNet [arxiv'17] – Make it Even Lighter

- Width multiplier – Reduce # of channels
  - # of multiplications = $(D_F \text{ x } D_F \text{ x } 1) \text{ x } (D_{out} \text{ x } D_{out}) \text{ x } \boldsymbol{\alpha} C_{in} + (1 \text{ x } 1 \text{ x } \boldsymbol{\alpha} C_{in}) \text{ x } (D_{out} \text{ x } D_{out}) \text{ x } \boldsymbol{\alpha} C_{out}$
  - Reduce computation $\boldsymbol{\alpha}^2$ times

Table 6. MobileNet Width Multiplier

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

- Resolution multiplier – Reduce # of width/height
  - # of multiplications = $(D_F \text{ x } D_F \text{ x } 1) \text{ x } (\boldsymbol{\rho} D_{out} \text{ x } \boldsymbol{\rho} D_{out}) \text{ x } \boldsymbol{\alpha} C_{in} + (1 \text{ x } 1 \text{ x } \boldsymbol{\alpha} C_{in}) \text{ x } (\boldsymbol{\rho} D_{out} \text{ x } \boldsymbol{\rho} D_{out}) \text{ x } \boldsymbol{\alpha} C_{out}$
  - Reduce computation $\boldsymbol{\rho}^2$ times

Table 7. MobileNet Resolution

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

[The tables are from AG. Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications."]

MobileNet [arxiv'17] – Summary

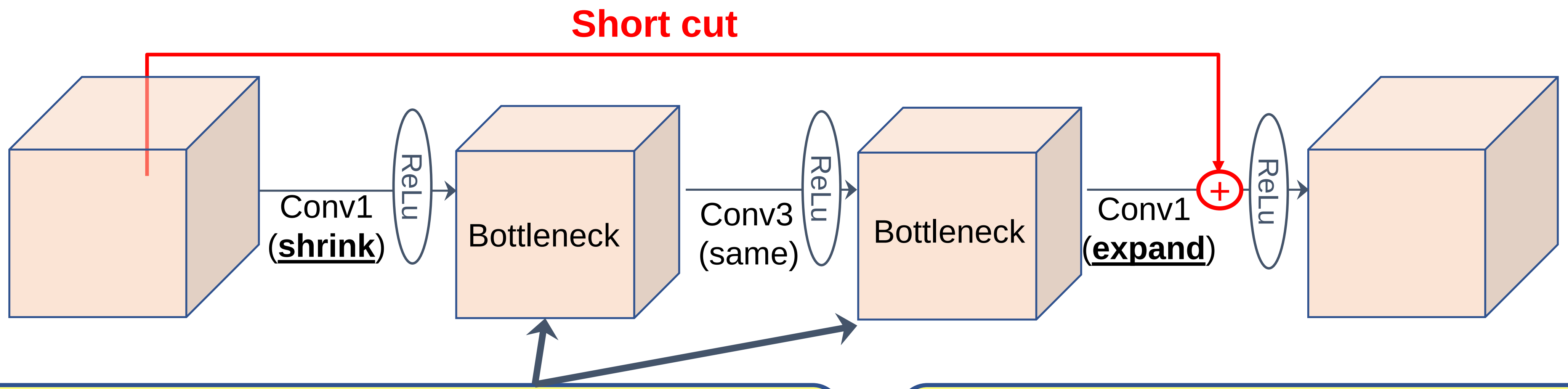
- Depth-wise separable convolution (2 steps)
  - Depthwise convolution to control height/width
  - 1x1 convolution to control # of channels
- Reduce # of parameters and computation significantly
- Most of the computation is from 1x1 convolution, which is even better
- Two multipliers for width and resolution
  - Achieve trade-off between accuracy and computation

How can we combine depthwise separable convolution with residual connection?

MobileNet v2!

MobileNet v2 [CVPR'18]

- Residual block in ResNet

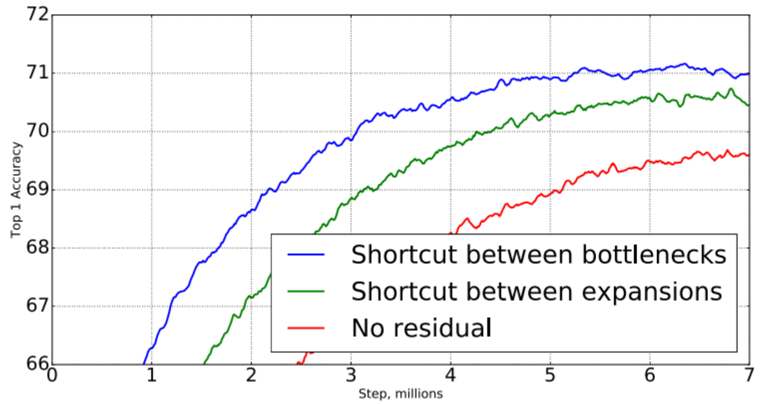


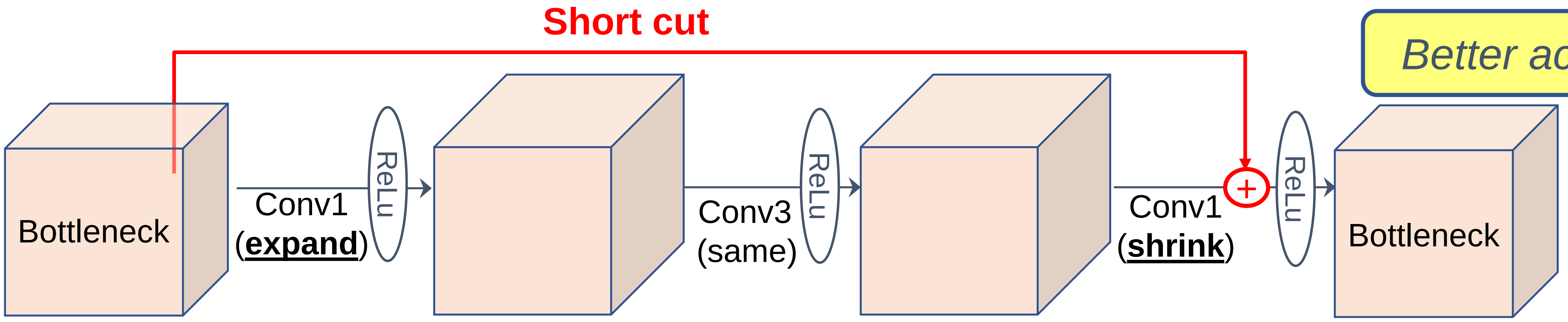
***Assumption**: Bottleneck layers have all the necessary information, while an expansion layer is just an implementation detail*

***Approach**: Make short cuts directly between bottleneck layers instead of expansion layers!*

MobileNet v2 [CVPR'18] – Ideas

- **Inverted** residual block



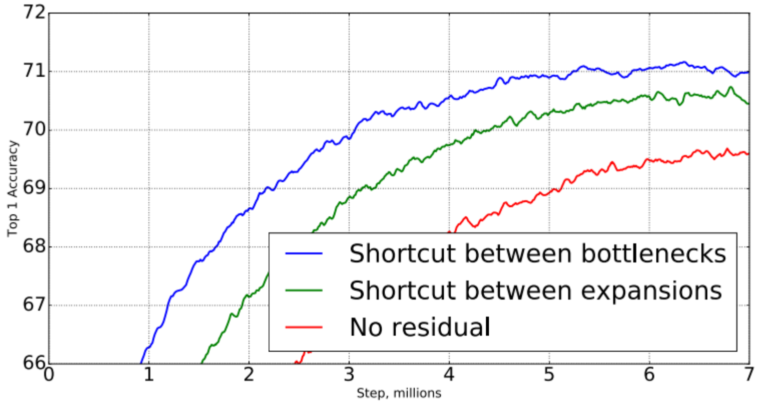


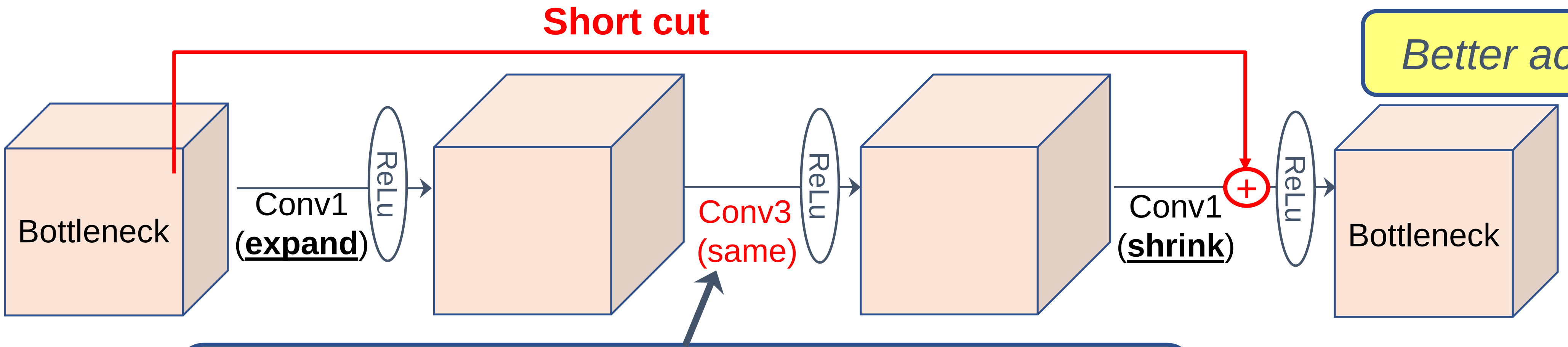
Better accuracy!

[The figure is from M. Sandler et al., "Mobilenetsv2: Inverted residuals and linear bottlenecks."]

MobileNet v2 [CVPR'18] – Ideas

- **Inverted** residual block





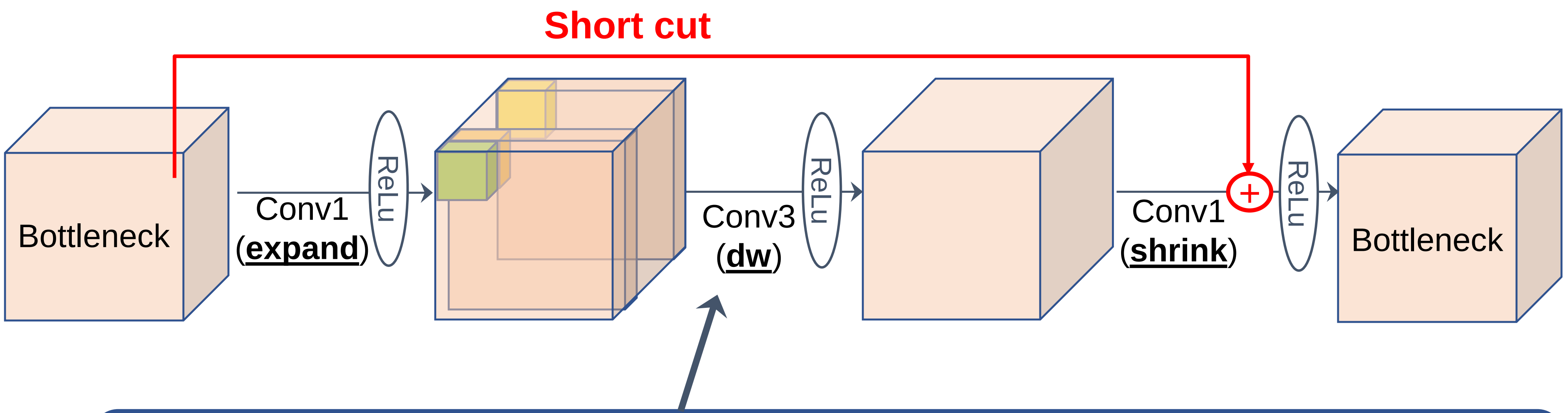
Better accuracy!

*f x f conv layer for an **expanded** input.*
How to deal with the increased computation?

[The figure is from M. Sandler et al., "Mobilenetsv2: Inverted residuals and linear bottlenecks."]

MobileNet v2 [CVPR'18] – Ideas

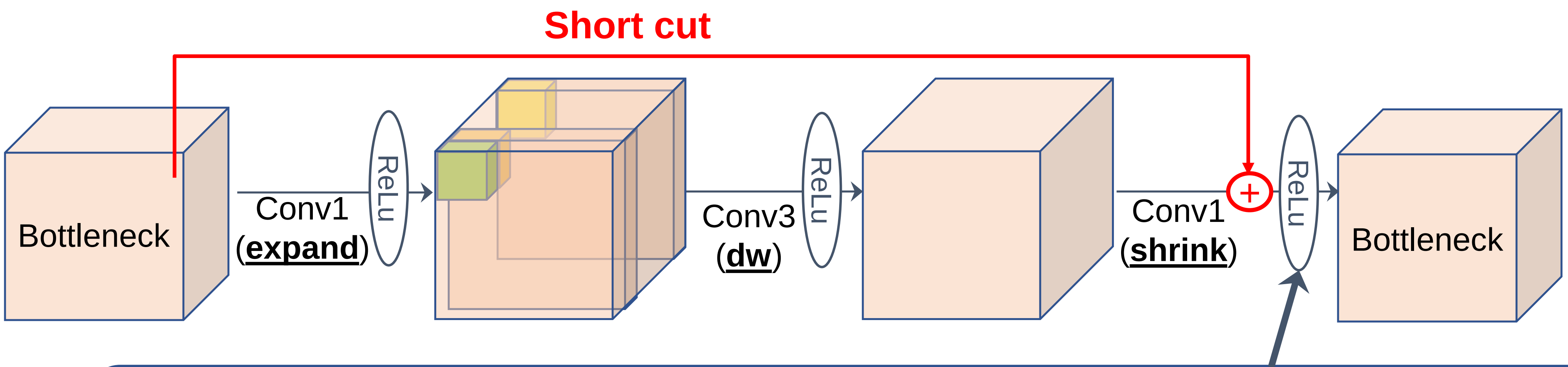
- **Inverted** residual block
- **Depthwise** convolution between expansion layers



*MobileNet uses **depthwise separable convolution** for f x f conv layer.*
Let's do that again here to reduce computation overhead!

MobileNet v2 [CVPR'18] – Ideas

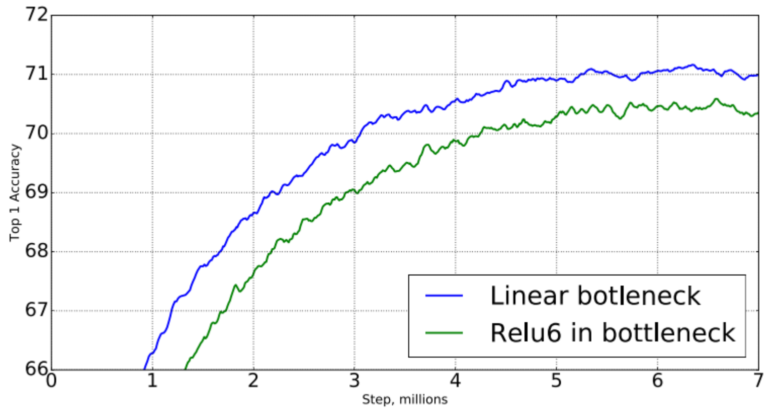
- **Inverted** residual block
- **Depthwise** convolution between expansion layers



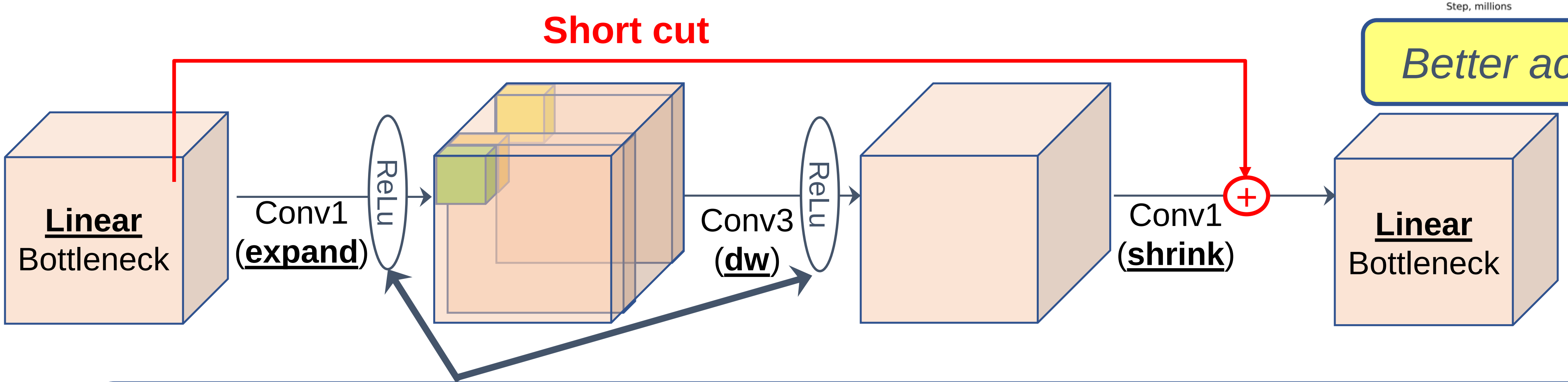
*ReLu non-linearity can **lose lots of information** (a lot of zeros).*
*This loss is more significant at a bottleneck layer having **dense** information.*
*Let's **remove** this part!*

MobileNet v2 [CVPR'18] – Ideas

- **Inverted** residual block
- **Depthwise** convolution between expansion layers
- **Linear** bottleneck



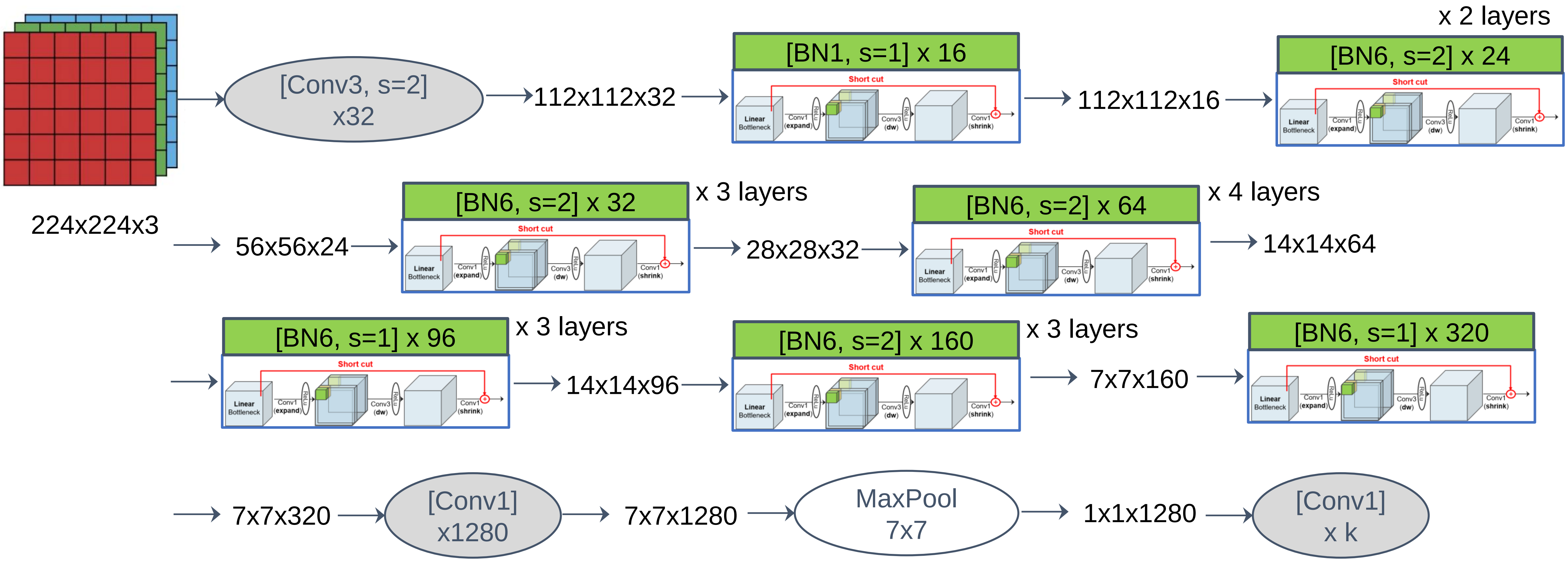
Better accuracy!



*ReLu is still needed for **expressiveness** (to do more delicate processing). Also, using it for **expansion layers** does not lose much information.*

[The figure is from M. Sandler et al., "Mobilenetsv2: Inverted residuals and linear bottlenecks."]

MobileNet v2 [CVPR’18] – The Whole Architecture



MobileNet v2 [CVPR'18] – Performance

- ImageNet classification performance
  - Improve accuracy compared to MobileNet v1
  - Reduce # of parameters and computation compared to MobileNet v1
  - MobileNet v2 (1.4) uses width multiplier 1.4

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

[The table is from M. Sandler et al., "Mobilenetsv2: Inverted residuals and linear bottlenecks."]

MobileNet v2 [CVPR’18] – Summary

- **Short cut between bottleneck layers**
  - Since bottlenecks have all the necessary information

- **Depthwise convolution** between expansion layers to reduce computation

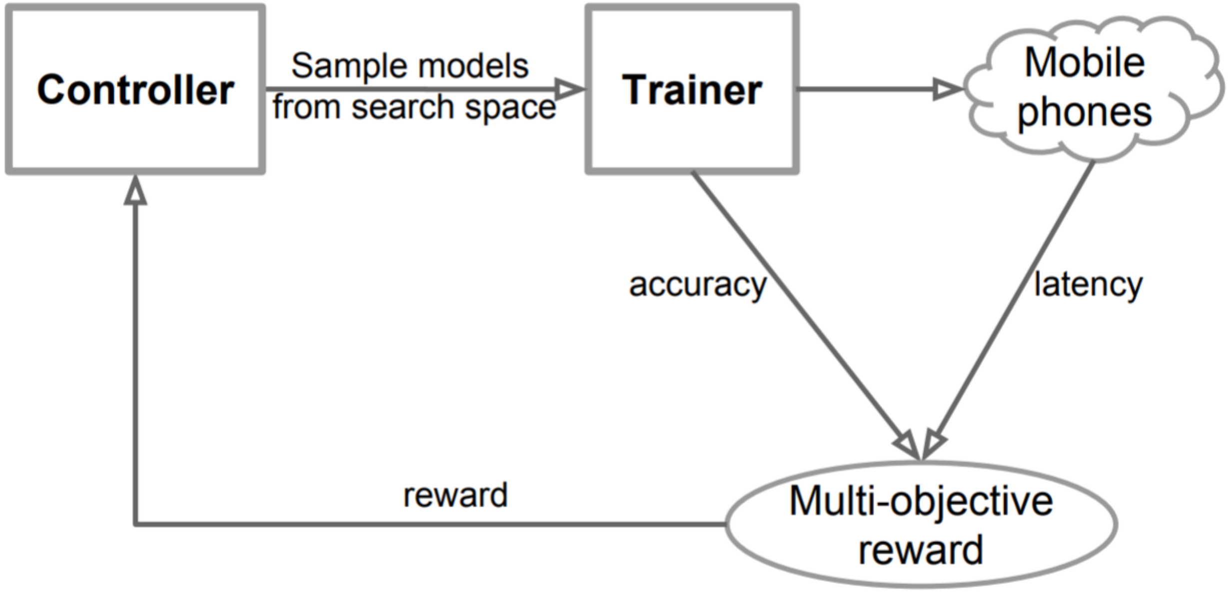
- Remove ReLu before a bottleneck layer, making **linear bottlenecks**
  - To not lose much information

Tired of ***handcrafting*** *model architectures…*
There are so many options and trials and errors

Let our computer do the architecture search
while you are drinking coffee!

MnasNet [CVPR'19]

- Neural architecture search for designing a **lightweight** CNN
- NAS typically needs reinforcement learning formulation
  - **Objective function** design
  - **Search space** design: layer diversity vs. search space size



MnasNet: NAS to MNAS!

[The figure is from M. Tan et al., "MnasNet: Platform-Aware Neural Architecture for Mobile."]

MnasNet [CVPR'19] – Objective Function

- In contrast to previous NAS approaches, it considers not only accuracy but also **latency** to search a lightweight architecture

Latency of model m
(not an indirect metric, such as FLOPs, but latency on a ***real platform****)*

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^w$$

Accuracy of model m

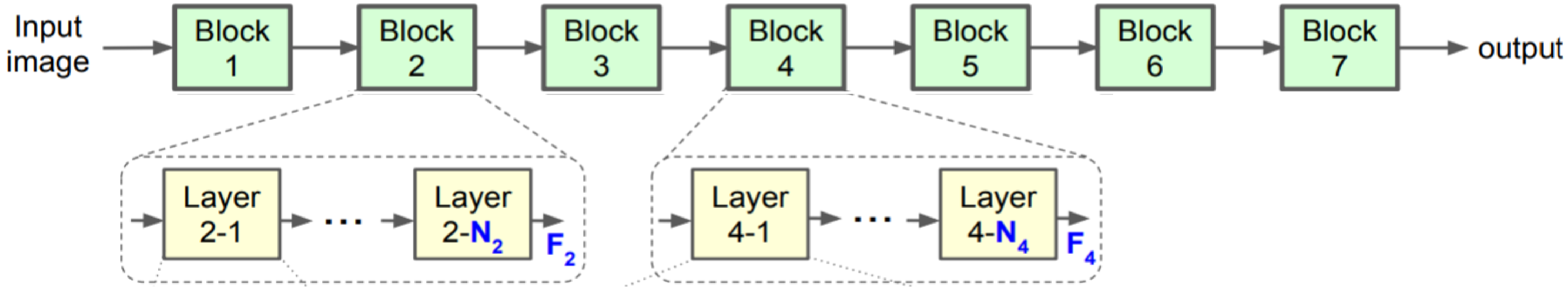
Latency target

A negative coefficient (-0.07)

MnasNet [CVPR'19] – Search Space

- Divide a model into several **blocks** that can have **different architectures**
- Let each block has a variable number of repeated **identical layers**
- For each block, find the best architecture within a **per-block search space**
  - Conv options: conv, dw conv, or inverted bottleneck conv
  - Skip options: pooling, residual, or no skip
  - Kernel (filter) size: 3x3 or 5x5
  - Number of layers, output size …

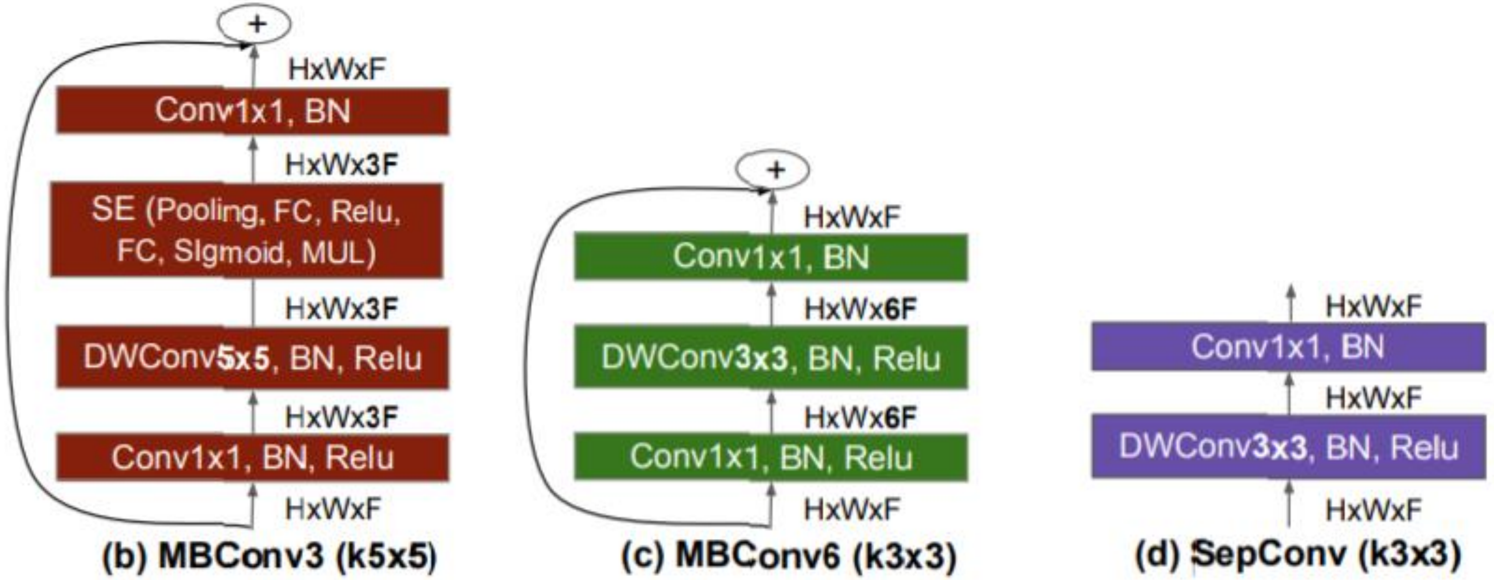
***Balance** between*
layer diversity and total search space

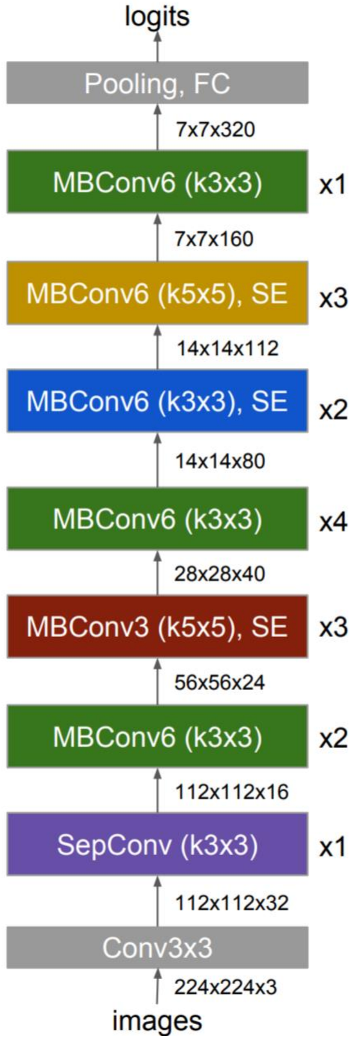


[The figures are from M. Tan et al., "MnasNet: Platform-Aware Neural Architecture for Mobile."]

MnasNet [CVPR'19] – Architecture

- MnasNet uses not only 3x3 but also **5x5** conv
  - In contrast to previous lightweight DNNs that use only 3x3 conv
- Each block ends up having a different architecture
  - Layer diversity

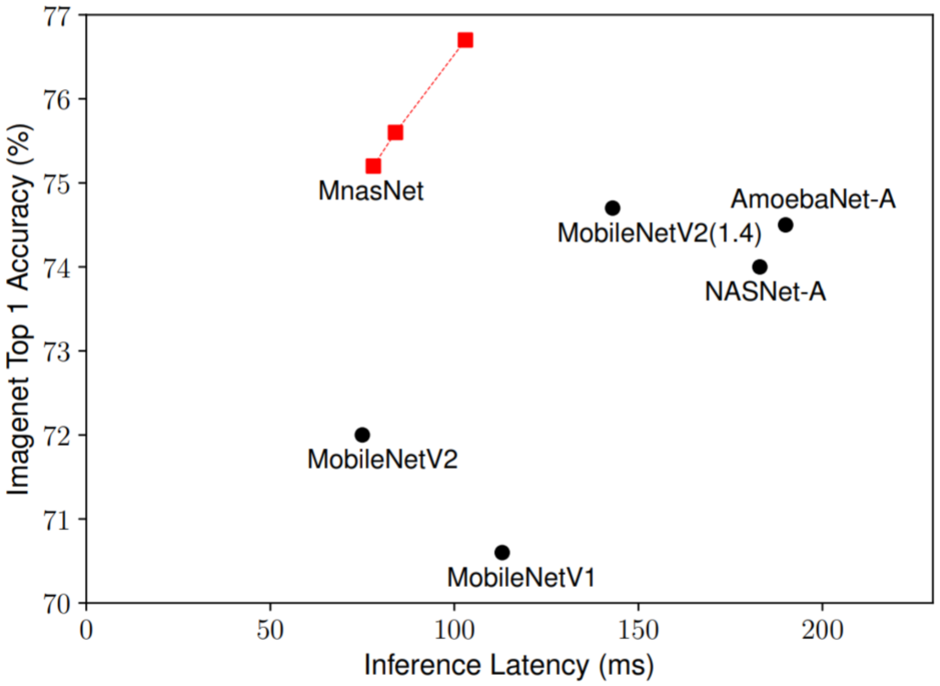


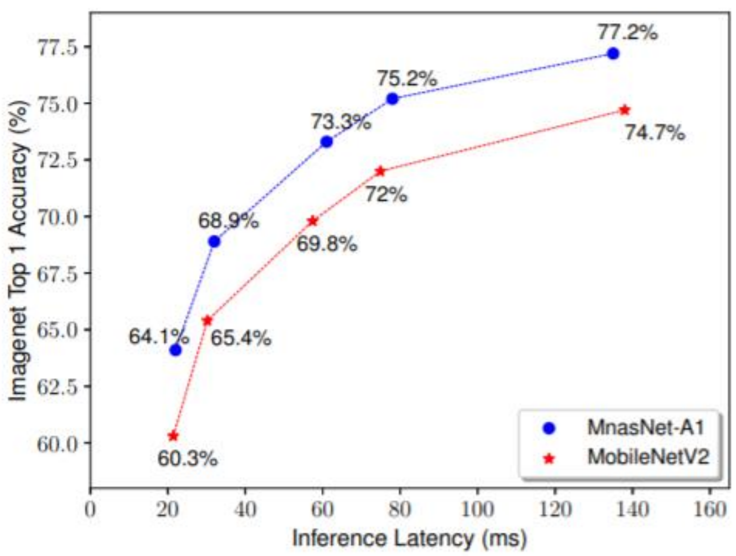


[The figures are from M. Tan et al., "MnasNet: Platform-Aware Neural Architecture for Mobile."]

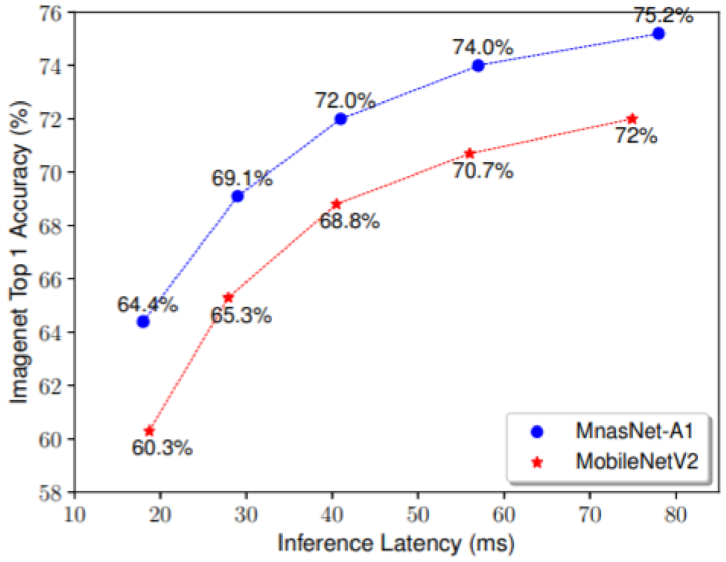
MnasNet [CVPR'19] – Performance

- Outperform other lightweight models in terms of both latency and accuracy
- Always better than MobileNetV2 regardless of scaling factors





(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.

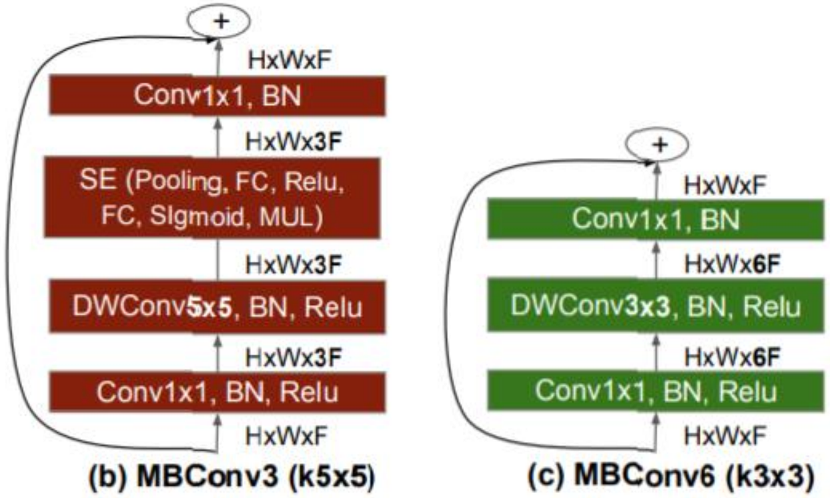
[The figures are from M. Tan et al., "MnasNet: Platform-Aware Neural Architecture for Mobile."]

Now we want to take care of how to scale a model efficiently!

EfficientNet

EfficientNet [ICML'19] – Baseline

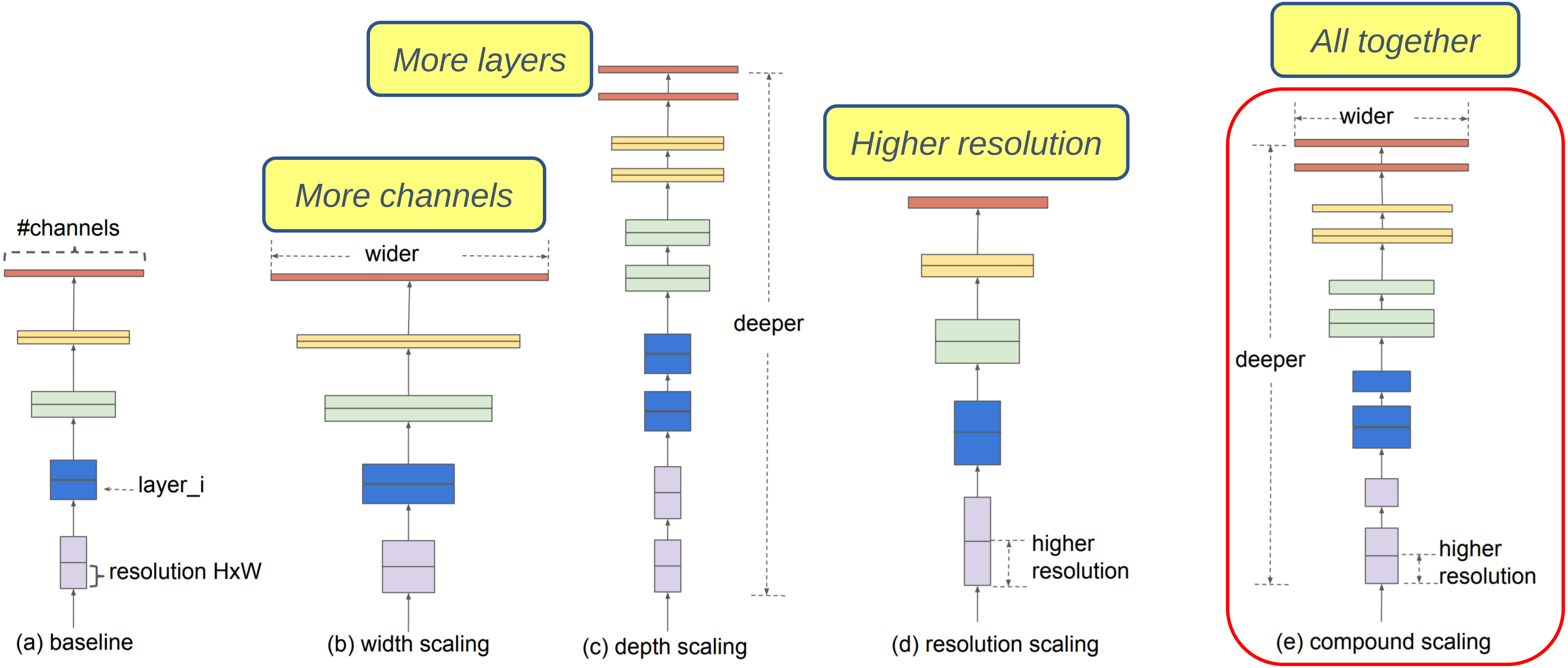
- Neural architecture search again…
  - **Objective**: FLOPS instead of latency since it does not target a specific hardware
  - **Search space**: Same as MnasNet
- The result (EfficientNet-B0) is similar to MnasNet



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	28×28	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

[The figures are from M. Tan et al., "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks."]

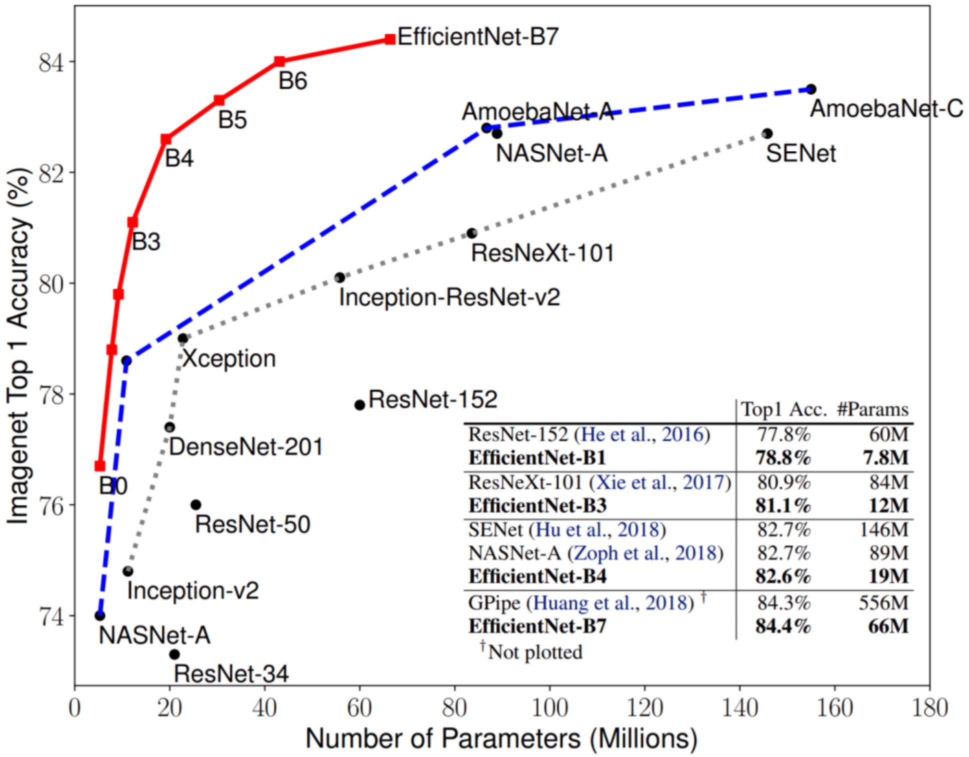
EfficientNet [ICML'19] – Scaling



[The figures are from M. Tan et al., "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks."]

EfficientNet [ICML'19] – Performance

- EfficientNet family outperform other DNNs (good baseline and scaling)
- The scaling method works well with other DNNs



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

† Not plotted

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%**

[The figures are from M. Tan et al., "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks."]

MnasNet and EfficientNet – Summary

- **Neural architecture search**
  - Considering both accuracy and computational cost (latency or FLOPs)
- **Hierarchical search space**
  - Block-wise search to balance layer diversity and computational cost
- **Compound scaling**
  - Scale width, depth, resolution all together
- Core ideas of MobileNets still survive!
  - Inverted bottleneck, depthwise separable convolution, linear bottleneck

Thanks!