Computer Vision
Week6

2025-2

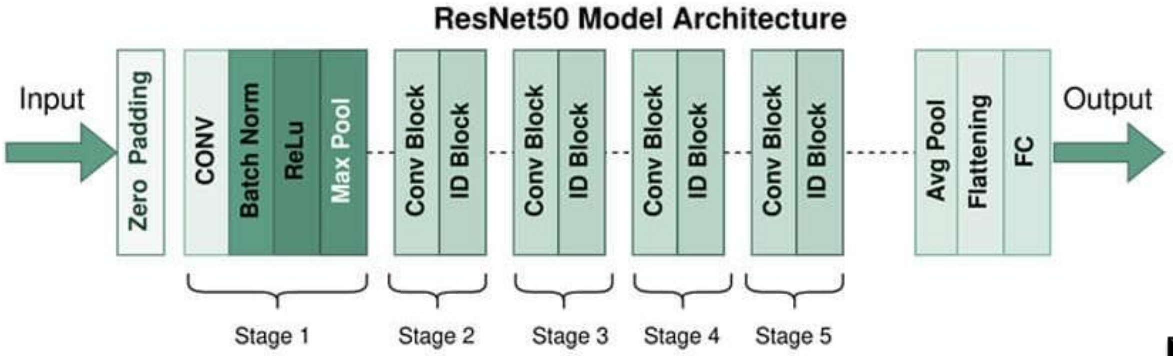
Mobile Systems Engineering

Dankook University

From ResNet to Lightweight CNNs: MobileNet and EfficientNet

▪ Why Do We Need Lightweight Models?

- ResNet and deeper models improve accuracy but are **too large** for real-time/mobile use.
- **Mobile apps, drones, AR, and robotics** require fast and lightweight models.
- Need a **trade-off between accuracy, speed, and size**.



Model	2D-CNN	3D-CNN
	Params	Params
VGG-16	134.7 M	179.1 M
ResNet-18	11.4 M	33.3 M
ResNet-34	21.5 M	63.6 M
ResNet-50	23.9 M	46.4 M
ResNet-101	42.8 M	85.5 M
ResNet-152	58.5 M	117.6 M

"Low memory, real-time requirement"

From ResNet to Lightweight CNNs: MobileNet and EfficientNet

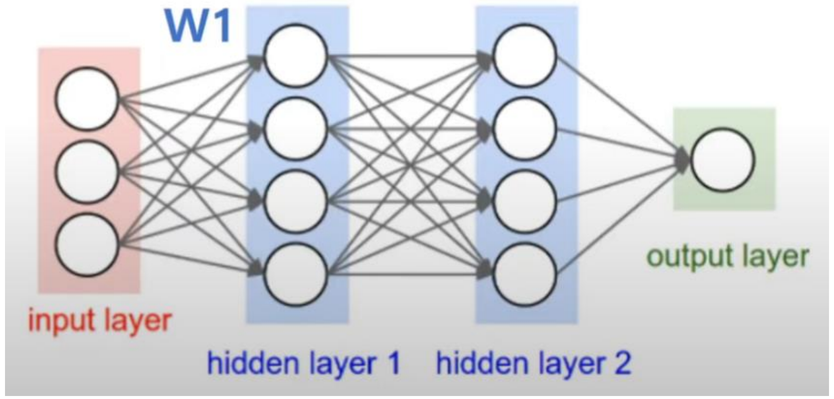
- **Three Directions in CNN Efficiency**
  - We can reduce the computational cost of CNNs by adjusting three key design axes.

Axis	What it Means	Effect on Computation
Depth	Number of layers	Linear increase in FLOPs
Width	Number of channels per layer	Quadratic increase in FLOPs
Resolution	Size of input images	Quadratic increase in FLOPs

  - Efficient design must **balance all three** rather than scaling just one.
  - This is the idea behind **EfficientNet's compound scaling**.

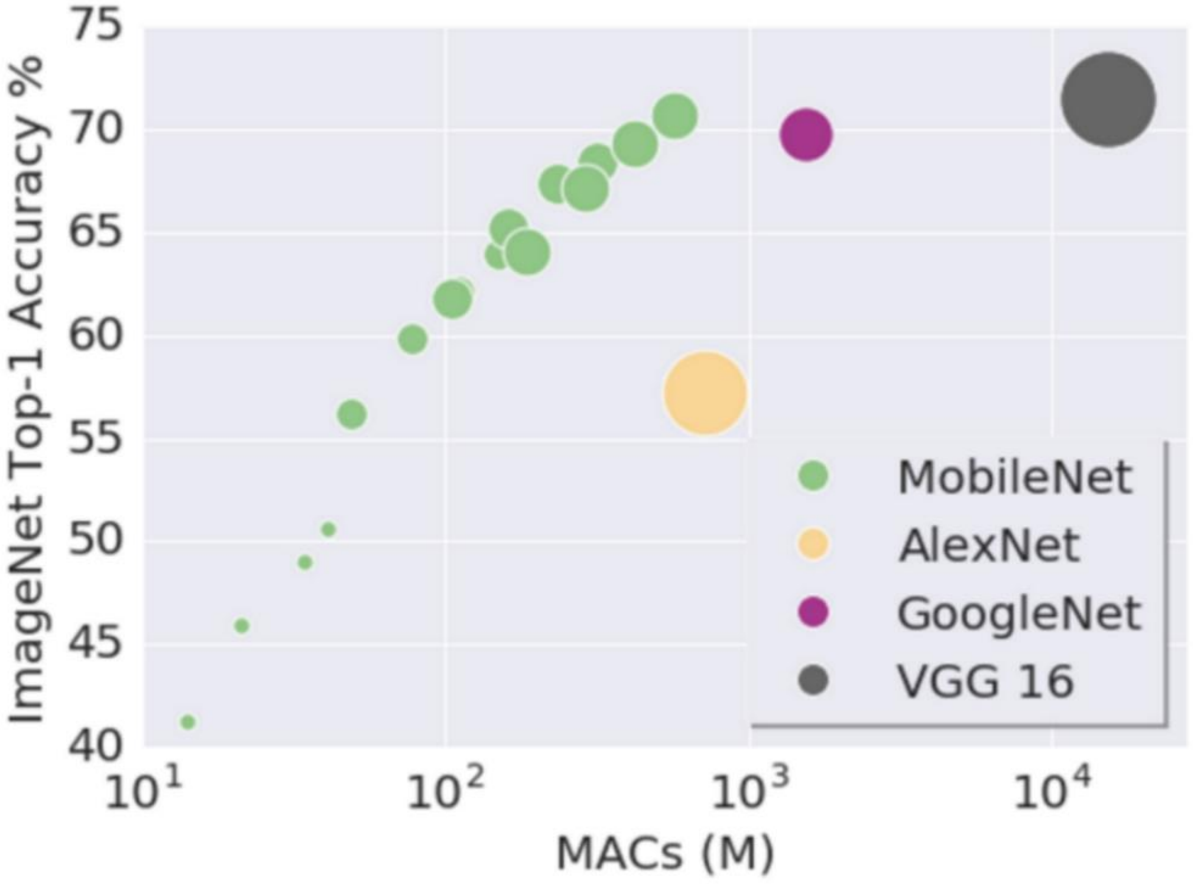
From ResNet to Lightweight CNNs: MobileNet and EfficientNet

- **What is a FLOP?**
  - **Understanding FLOPs – What Do We Actually Count?**
    - **FLOP = Floating Point Operation**
    - A single **multiplication** or **addition** is counted as 1 FLOP
    - In deep learning, FLOPs are used to estimate the **computational cost** of models
  - **Example 1: Dot Product**
    - $Y =$
      - ✓**3** multiplications and **2** additions → **Total: 5 FLOPs → In general: ________________ FLOPs for n elements**
  - **Example 2: Dense (Fully Connected) Layer**
    - **Input vector:** 3-dimensional / **Output vector:** 4-dimensional
      → **Weight matrix:**
    - **4 dot products performed** → 5×4=20 FLOPs
      - ✓**12 multiplications** (4 rows × 3 elements)
      - ✓**8 additions** (each dot product needs n-1 additions)



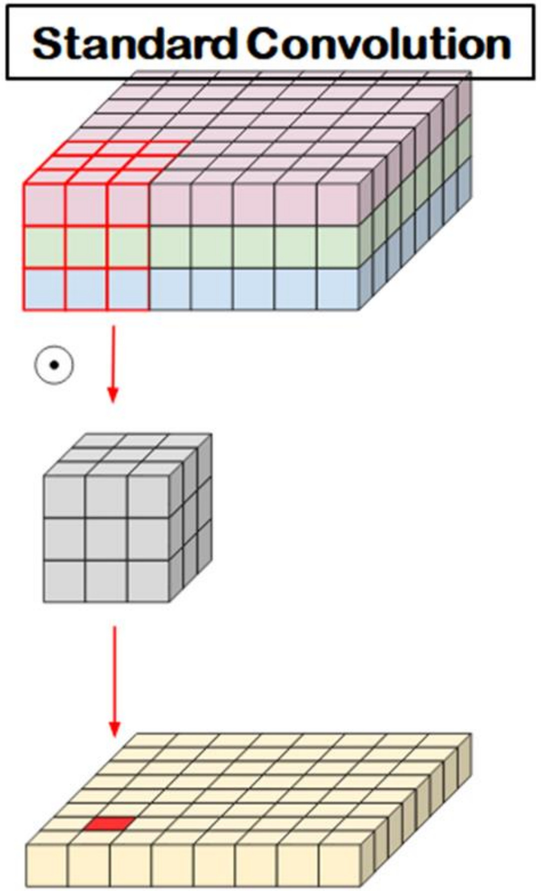
What is MobileNet?

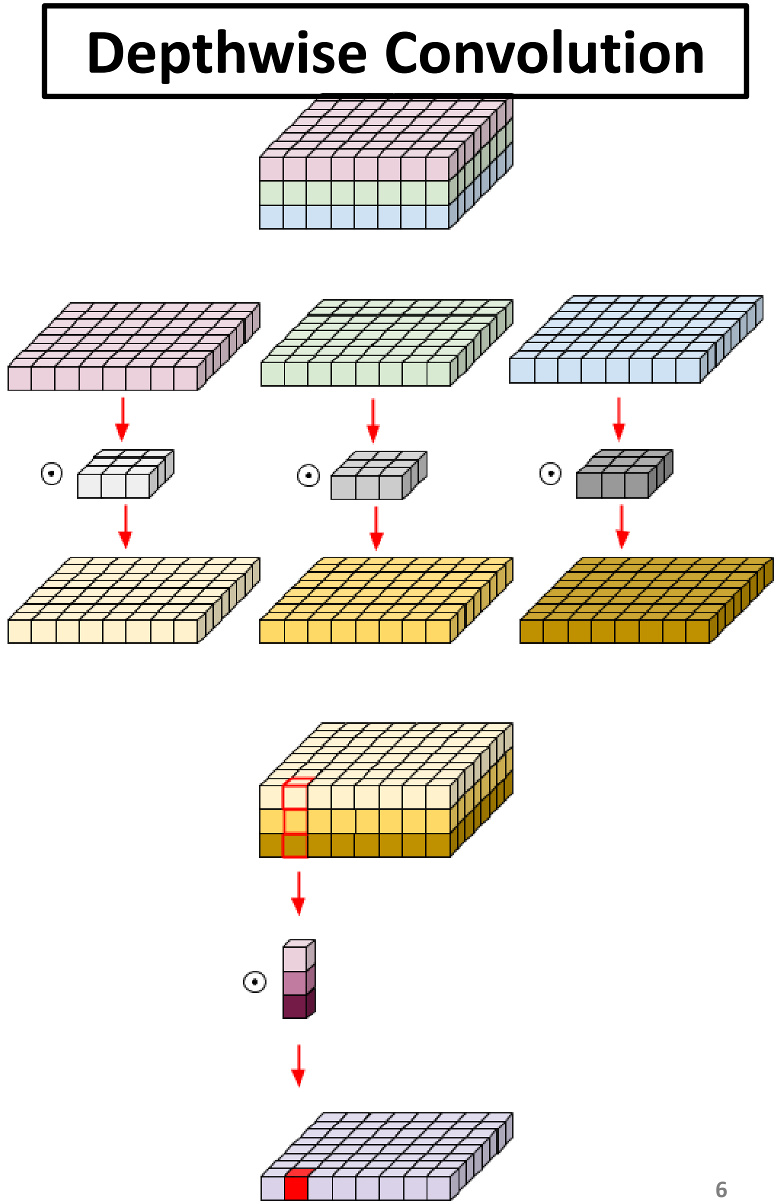
- **Introducing MobileNet – Efficient CNNs for Mobile Vision**
  - Proposed by Google in 2017
  - Designed for **mobile and embedded vision applications**
  - **Key ideas**
    - ***Depthwise Separable Convolution***
    - ***Two hyperparameters*** for flexible scaling
  - Used in real-world tasks
    - object detection, face recognition, geo-localization



Standard Convolution vs. Depthwise Separable Convolution

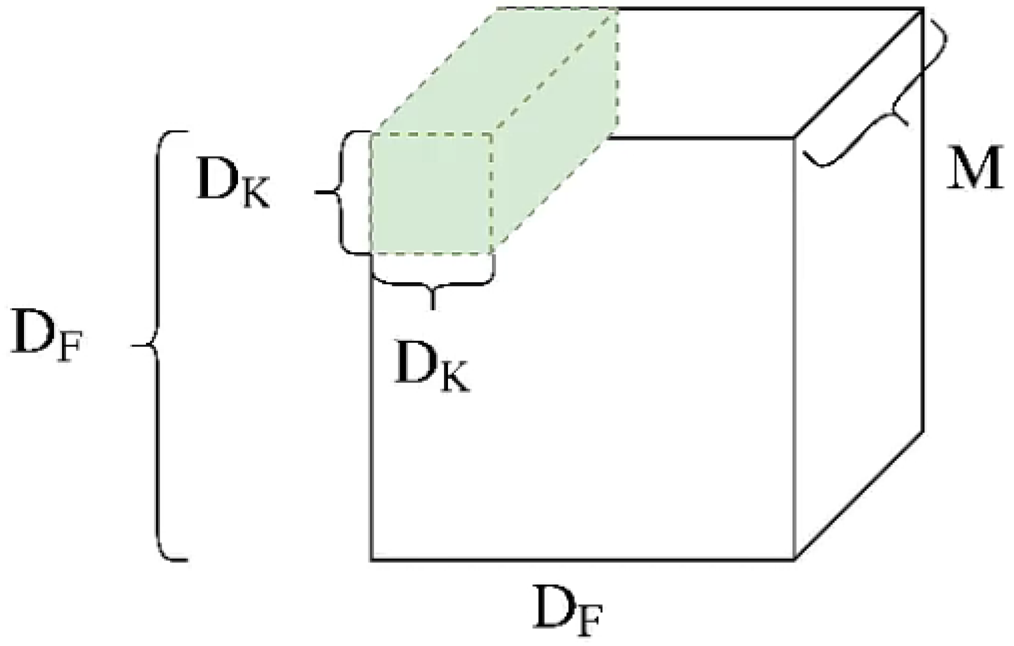
- **Rethinking Convolution: Lighter and Faster**
  - **Standard convolution**
    - Applies multiple filters across all channels simultaneously
  - **Depthwise separable convolution**
    - Depthwise
      - ✓Applies one filter per input channel
    - Pointwise
      - ✓1×1 convolution to combine outputs
  - Result
    - **~8–9x less computation**





How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - **How to Compute FLOPs in Standard Convolution**
    - **Key Definitions**
      - ✓ D_K: Kernel size (e.g., 3 → 3×3)
      - ✓ M: Number of input channels
      - ✓ N: Number of output channels (i.e., number of filters)
      - ✓ D_F: Input feature map spatial dimension
      - ✓ D_G: Output feature map spatial dimension
        - ➢ Usually, $D_G \approx D_F$ (same padding, stride 1)



How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - **How to Compute FLOPs in Standard Convolution**
    - **Step-by-Step FLOP Count**

✓**Step 1. Single Location, Single Filter**

➢Each filter performs: multiplications

✓**Step 2. All Spatial Locations in One Filter**

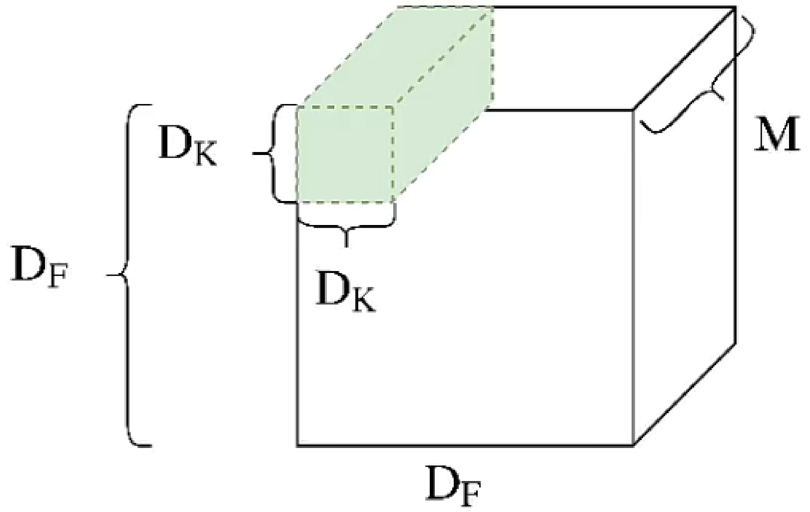
➢Spatial positions: (assuming square output)
→ Typically $D_G = D_F$ for stride 1 & padding

➢FLOPs per filter:

✓**Step 3. All N Filters (Output Channels)**

➢Multiply by N

✓Total FLOPs=



Mults once =

Mults per Kernel =

Mults N Kernels =

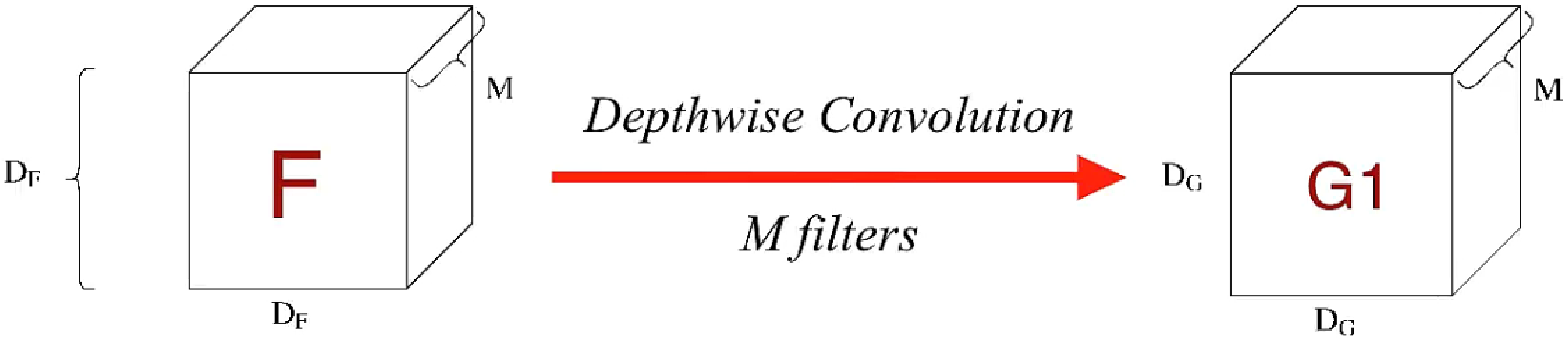
How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - What is Depthwise Separable Convolution?
    - **Key Concepts**
      - ✓Standard convolution combines **spatial filtering + channel mixing** in one operation.
      - ✓Depthwise separable convolution splits this into two steps
        - ➢**Step1. Depthwise Convolution**: applies spatial filtering **per input channel**
        - ➢**Step 2. Pointwise Convolution**: mixes the output channels using **1×1 conv**

      → This separation significantly **reduces computation and parameters**.

How Efficient is Depthwise Separable Conv?

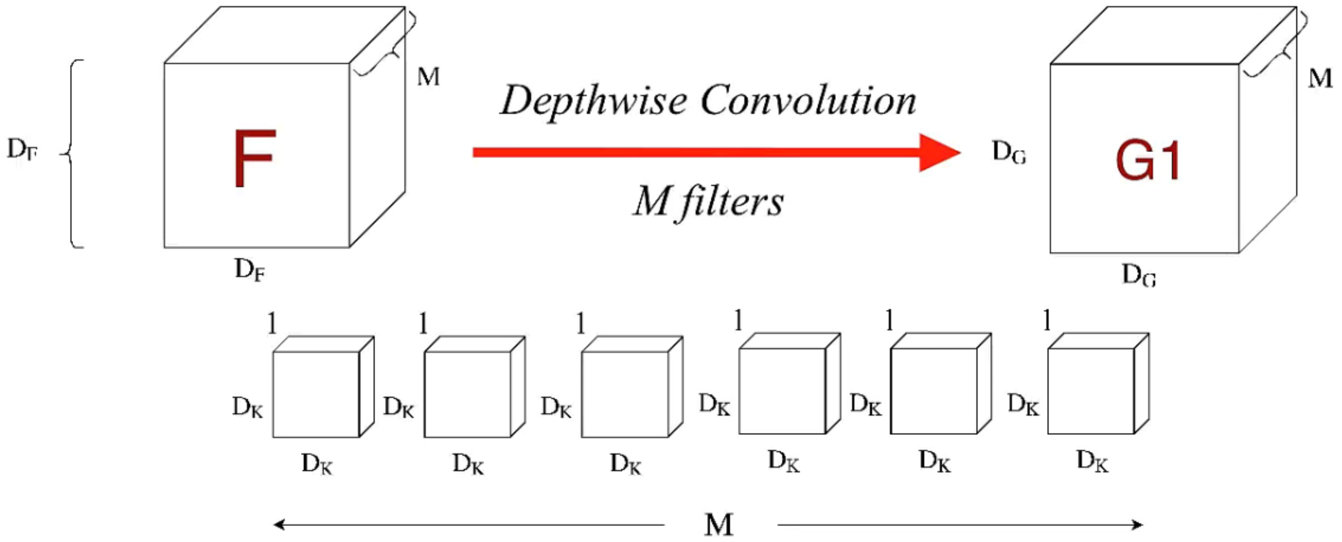
- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - FLOPs Breakdown of Depthwise Separable Convolution
    - **Key Definition**



✓$\boldsymbol{D_K}$: kernel size (e.g., 3 for 3×3)
✓$\boldsymbol{D_F}$: input spatial resolution
✓$\boldsymbol{M}$: number of input channels
✓$\boldsymbol{N}$: number of output channels
✓Assume $\boldsymbol{D_G} = \boldsymbol{D_F}$ for simplicity

How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - FLOPs Breakdown of Depthwise Separable Convolution
    - Step-by-step FLOP Calculation
      - ✓ **Step 1. Depthwise Convolution (per channel)**



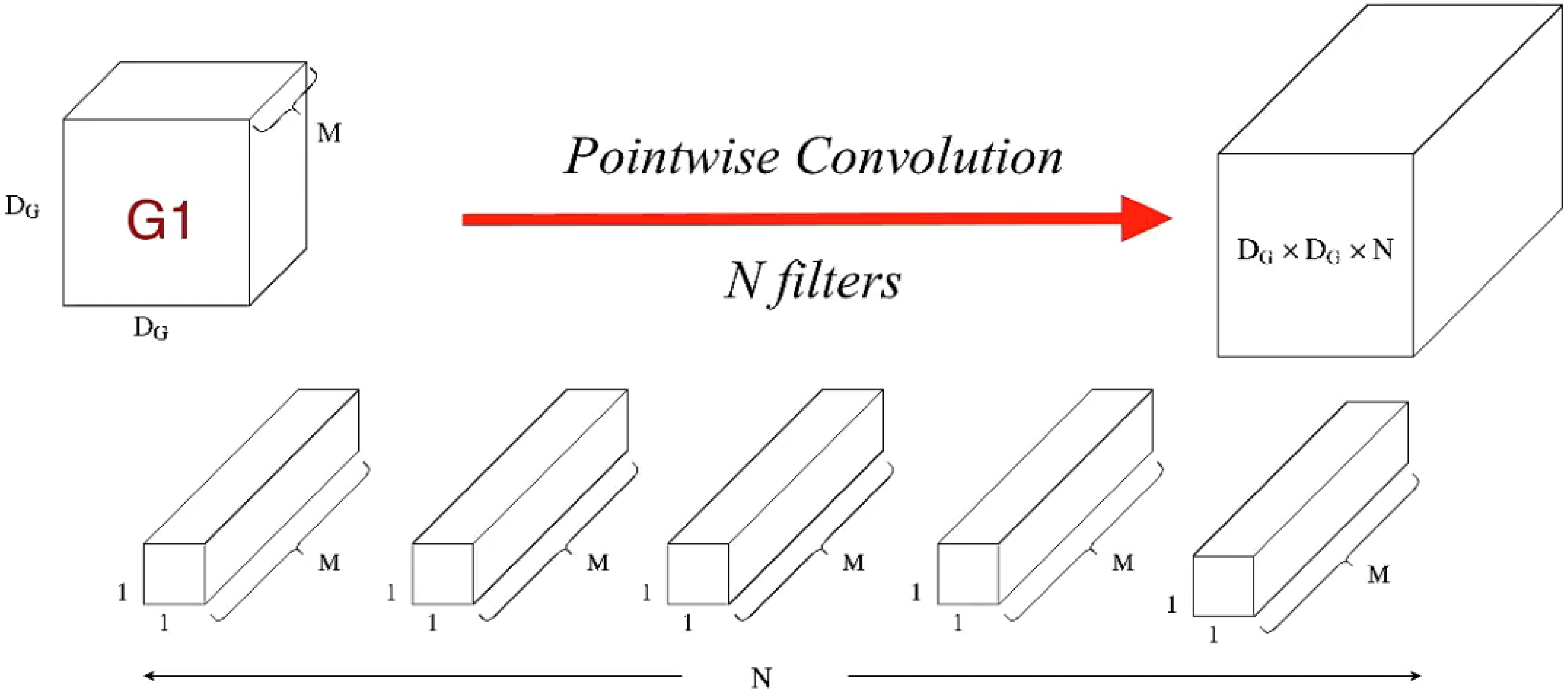
➢ Each of the **M** channels gets a **filter**

➢ For each channel: operations

→ Total:

How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - FLOPs Breakdown of Depthwise Separable Convolution
    - Step-by-step FLOP Calculation
      - ✓**Step 2. Pointwise Convolution (1×1 conv)**



➢Each of the $\boldsymbol{D_F} \times \boldsymbol{D_F}$ positions applies **N** 1×1 filters across **M** channels

→ Total:

How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - **Total FLOPs**
    - $\boldsymbol{FLOPs =}$
  - Compare with **Standard Convolution**
    - $\boldsymbol{FLOPs_{standard} =}$
  - **Efficiency Gain**

$$\frac{\boldsymbol{Depthwise\ Separable\ Convolution}}{\boldsymbol{Standard\ Convolution}} = \qquad =$$

$$=$$

How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - **Example – Efficiency Gain**
    - Efficiency Gain

$$\frac{Depthwise\ Separable\ Convolution}{Standard\ Convolution} =$$

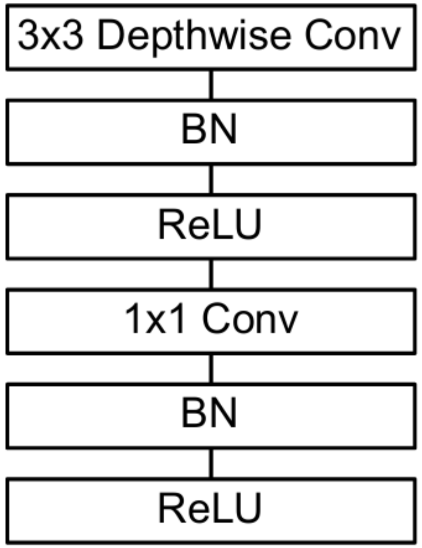
    - For $D_K = 3$ (size of filter), $N = 256$ (number of channels)

      ✓Relative Cost = $\quad = \quad = \quad =$

      ✓The calculation cost is reduced by about 9 times.

MobileNet Architecture

- **Building Blocks of MobileNet**
  - **All layers (except the first) use depthwise separable conv**
  - **Each block**
    - Depthwise conv → BN → ReLU → Pointwise conv $(\mathbf{1} \times \mathbf{1})$ → BN → ReLU
  - **Total: 28 layers**



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	$14 \times 14 \times 512$
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	$1 \times 1 \times 1000$

MobileNet Architecture

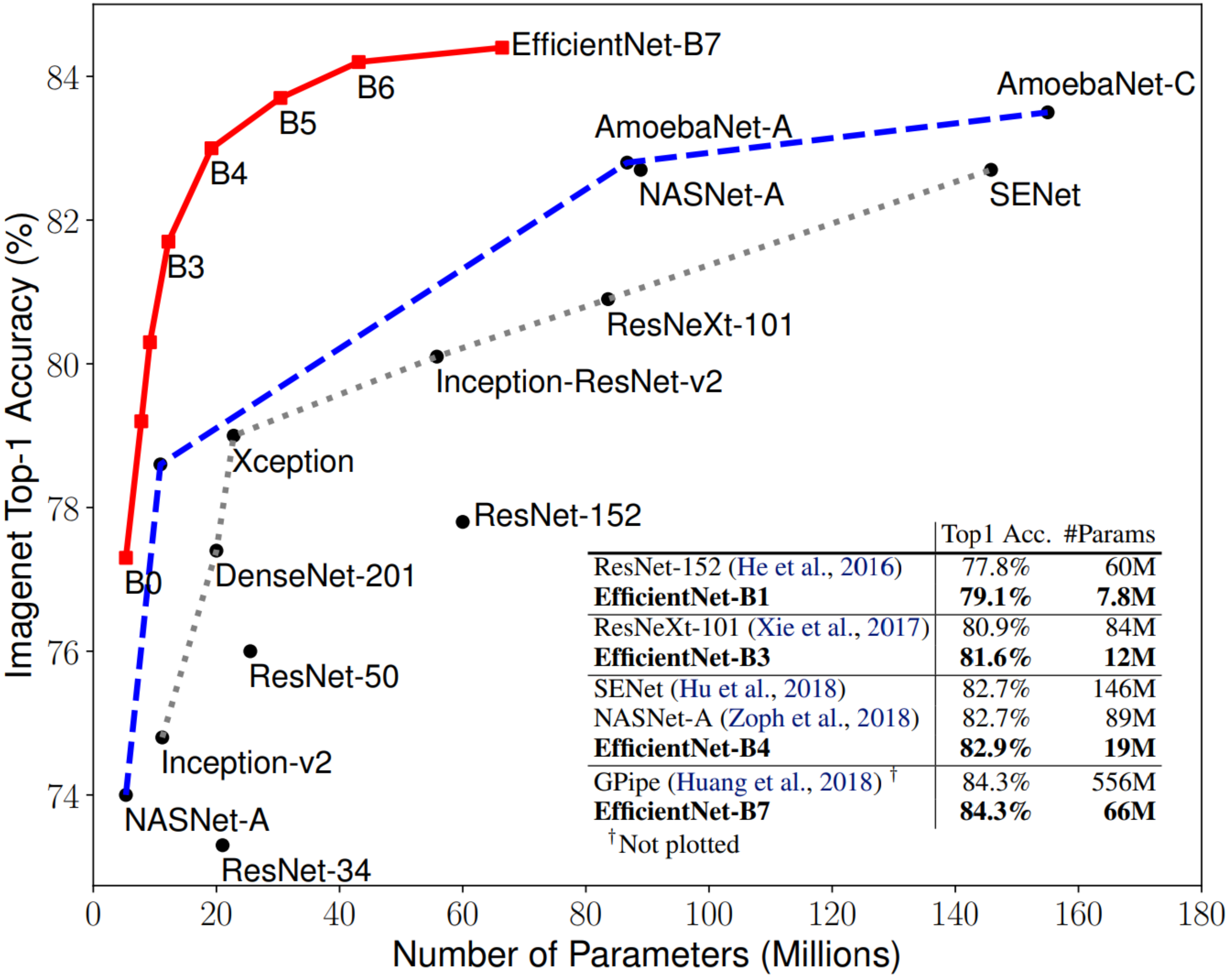
- **Width Multiplier & MobileNet's Efficiency**
  - **MobileNet Trade-offs: Width Multiplier**
    - Scaling with Width Multiplier $\boldsymbol{\alpha}$
      - ✓ Controls number of channels at each layer
    - Input/output channels → multiplied by $\boldsymbol{\alpha}$ ($0 < \boldsymbol{\alpha} \leq 1$)
      - ✓ Reduces FLOPs and parameters by approximately $\boldsymbol{\alpha}^2$
    - Common choices
      - ✓ $\boldsymbol{\alpha} \in \{1.0, 0.75, 0.5, 0.25\}$

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

    - **Key Observations**
      - ✓ As **width multiplier** decreases
        - ➢ **(1)** Accuracy drops / **(2)** FLOPs and parameters drop **significantly**
      - ✓ Enables scaling model size for devices with limited resources

What is EfficientNet?

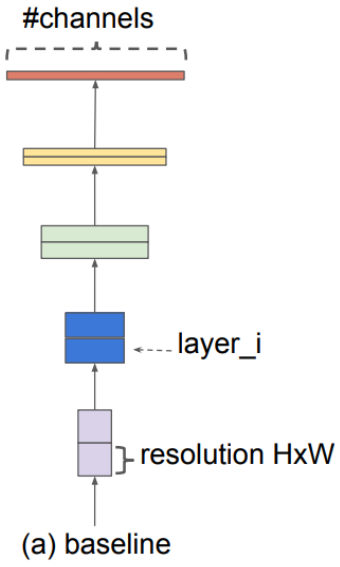
- **EfficientNet: A New Way to Scale CNNs**
  - Proposed by Google (2019)
  - Goal
    - **better accuracy + fewer parameters**
  - **Key idea**
    - ***Compound Scaling***
      - ✓ Instead of manually scaling **depth/width/resolution, scale all together**
  - Achieves state-of-the-art accuracy with **high efficiency**

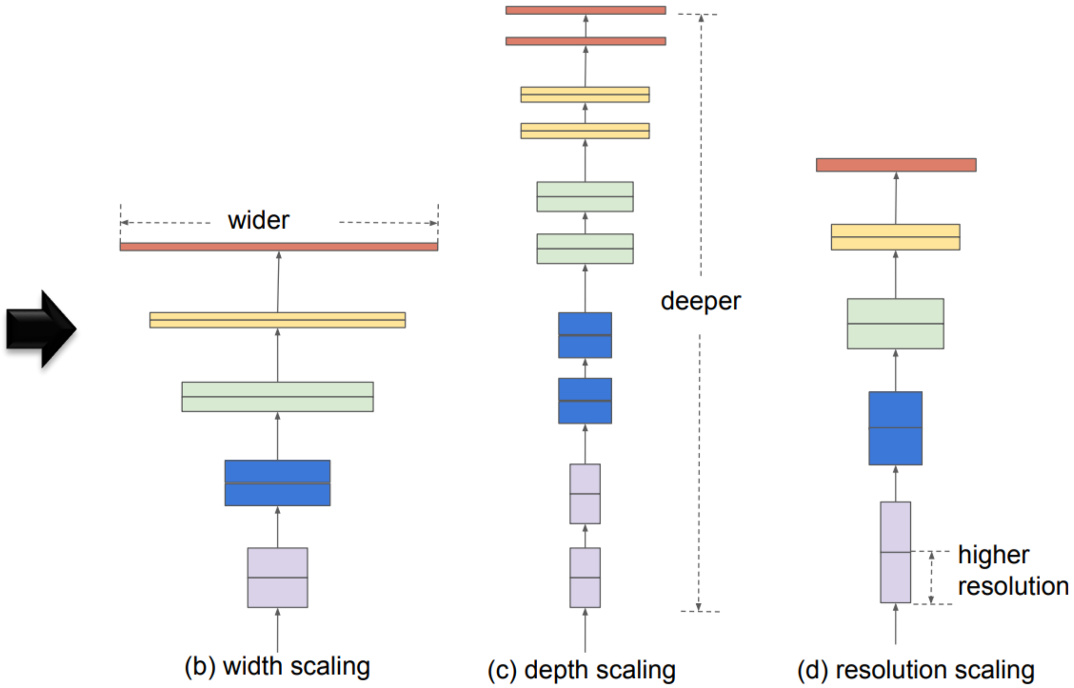


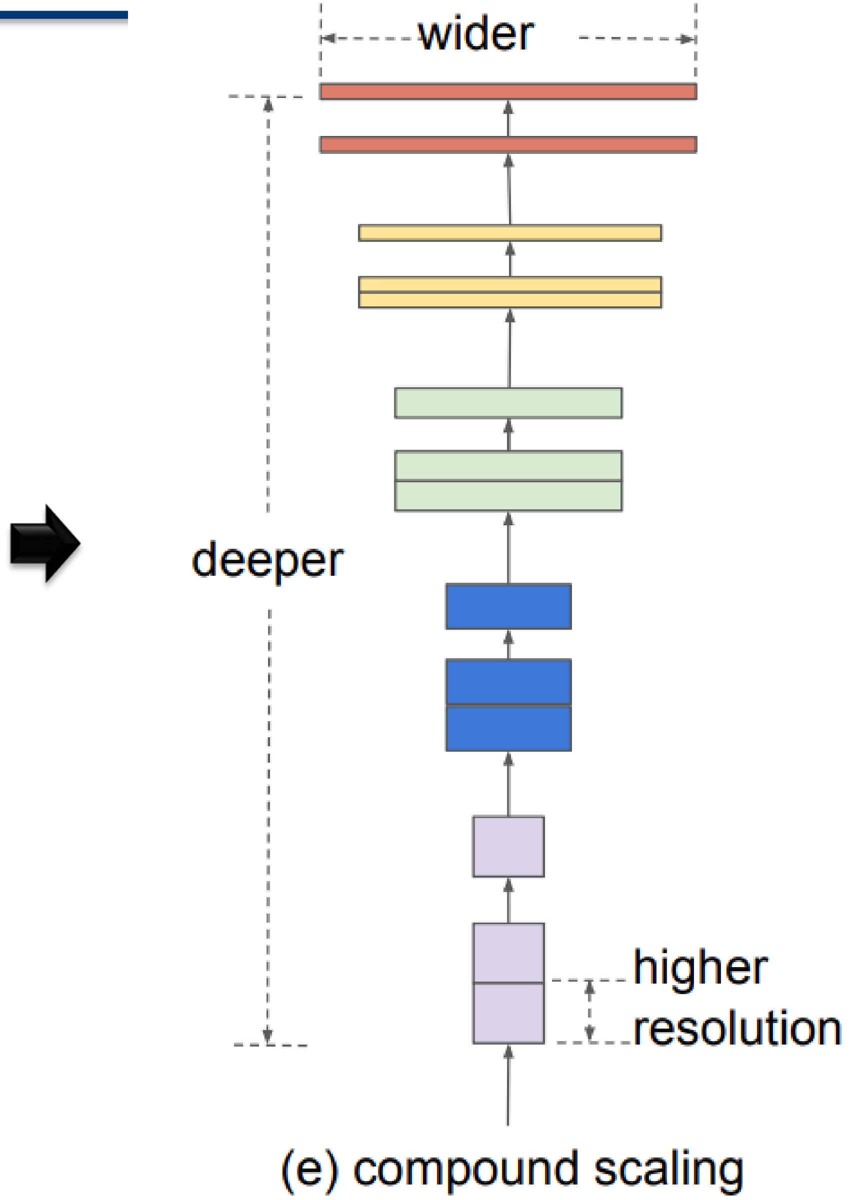
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018)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

Compound Scaling Explained

- **How EfficientNet Scales Models Efficiently**
  - Previous models scale only one aspect (depth **or** width **or** resolution)







  - **EfficientNet**
    - Scale all 3 in a **balanced** way using a compound coefficient ϕ
    - **Scaling formula**
      - ✓**Depth**: $\boldsymbol{d = \alpha^{\phi}}$, **Width**: $\boldsymbol{w = \beta^{\phi}}$, **Resolution**: $\boldsymbol{r = \gamma^{\phi}}$
    - **Constraint**
      - ✓$\boldsymbol{\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2}$

EfficientNet's Compound Scaling – Mathematical View

- **How EfficientNet Scales Architectures under Constraints**
  - EfficientNet assumes a **fixed layer structure** (pre-searched via NAS)
  - Rather than redesigning the layers, it scales
    - **Depth** (number of layers): $\boldsymbol{d}$
    - **Width** (number of channels): $\boldsymbol{w}$
    - **Resolution** (input size): $\boldsymbol{r}$
  - The goal is to **maximize accuracy** under **(1)** Memory constraint and **(2)** FLOPs constraint
  - **Optimization Objective**
    - $\max_{\boldsymbol{d},\boldsymbol{w},\boldsymbol{r}} \boldsymbol{Accuracy}\big(\boldsymbol{\mathcal{N}}(\boldsymbol{d},\boldsymbol{w},\boldsymbol{r})\big); \boldsymbol{where}\ \boldsymbol{\mathcal{N}}\ \boldsymbol{is\ a\ model}$
    - $\boldsymbol{subject\ to}: \begin{cases} \boldsymbol{Memory}(\boldsymbol{\mathcal{N}}) \leq \\ \boldsymbol{FLOPs}(\boldsymbol{\mathcal{N}}) \leq \end{cases}$

EfficientNet's Compound Scaling – Mathematical View

- **How EfficientNet Scales Architectures under Constraints**
  - **Optimization Objective**
    - $\max_{d,w,r} Accuracy(\mathcal{N}(d,w,r))$
    - $subject\ to: \begin{cases} Memory(\mathcal{N}) \leq \\ FLOPs(\mathcal{N}) \leq \end{cases}$

  - **Interpretation to Optimization Objective**
    - $\mathcal{N}$: CNN model with depth d, width w, and resolution r
    - Scaling is applied to input shape (H_i, W_i, C_i) as

      ✓ ______________________________ while increasing model depth ___________

EfficientNet Architecture

- **Building Blocks of EfficientNet**
  - Based on **MobileNetV2 MBConv blocks**
  - ***Key components***
    - **1. MBConv**: Depthwise separable conv + expansion + projection
    - **2. SE block** (Squeeze-and-Excitation): channel-wise attention
    - **3. Swish activation**: smooth and non-monotonic
  - Progressive stage-wise scaling of the backbone

EfficientNet Architecture

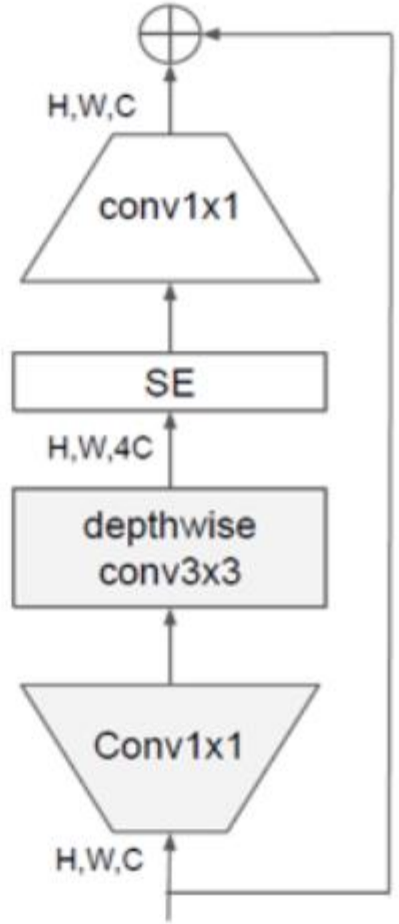
- **Building Blocks of EfficientNet**
  - Based on **MobileNetV2 MBConv blocks**

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

  - Key components
    - **MBConv**: Depthwise separable conv + expansion + projection
    - **SE block** (Squeeze-and-Excitation): channel-wise attention
    - **Swish activation**: smooth and non-monotonic
  - Progressive stage-wise scaling of the backbone

EfficientNet Architecture

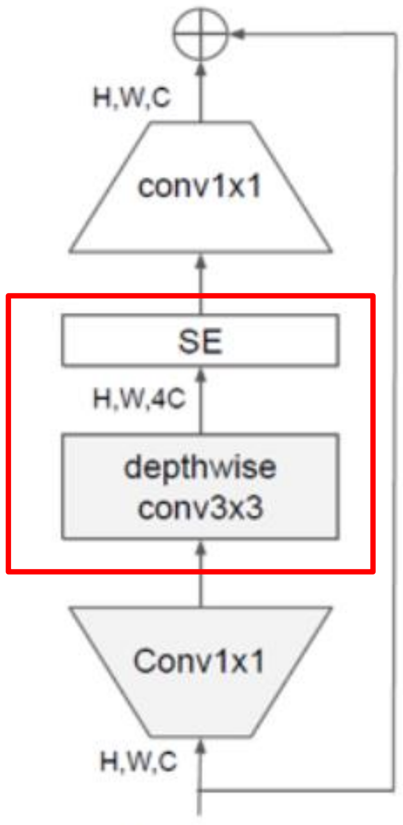
- **MBConv – Mobile Inverted Bottleneck**
  - **MBConv: Efficient Block for Lightweight CNNs**
    - Introduced in **MobileNetV2**, reused in EfficientNet
    - **Inverted bottleneck**
      - ✓Expand → Depthwise Conv → SE Block → Project (1x1 conv)
    - Includes residual connection if input and output shapes match
  - **MBConv Structure**
    - **1. Expansion (1×1 conv):** increases channel size
    - **2. Depthwise Conv (3×3):** lightweight spatial feature extraction
      - ✓Discussed in MobileNet (applying convolution to each channel)
    - **3. SE Block:** Squeeze and excitation
    - **4. Projection (1×1 conv):** reduces channels back
    - **5. Residual connection** (optional)
      - ✓Efficient gradient propagation



MBConv

EfficientNet Architecture

- **SE Block – Squeeze and Excitation in MBconv**
  - **Motivation – Why Channel Attention?**
    - Traditional convolutions primarily focus on **spatial information** within local regions (receptive field).
    - Each **channel is treated independently**, without considering **inter-channel dependencies**.
    - In **MBConv** (used in EfficientNet), **depthwise convolution** is applied per channel
      - ✓This means there is ***no interaction across channels*** during spatial filtering.
      - ✓The model cannot learn which channels are more important or how they relate to each other.

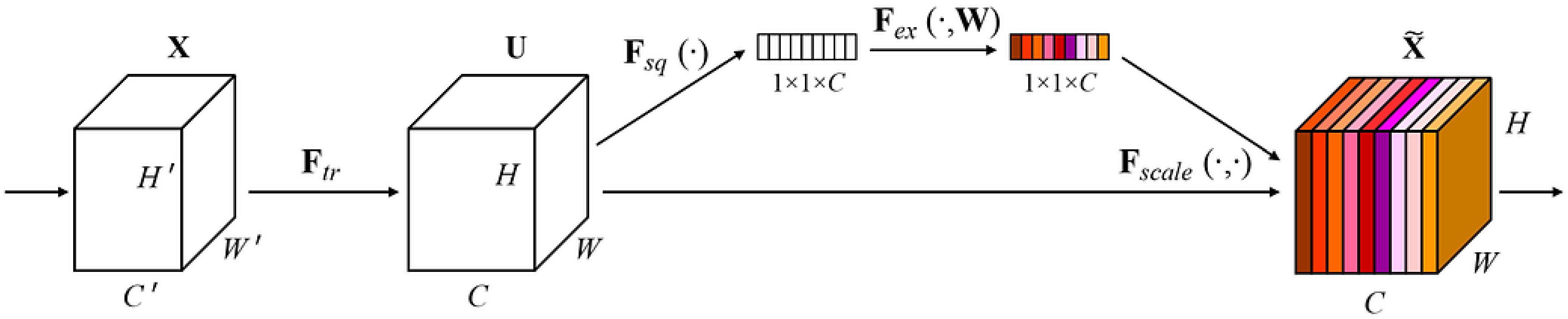


MBConv

"Depthwise convolutions extract spatial features **independently for each channel**, but **do not capture relationships between channels.**"

EfficientNet Architecture

- **SE Block – Squeeze and Excitation in MBconv**
  - **Overview – SE Block Structure**



○ **Step 1.** $\boldsymbol{F_{tr}}$: Input feature map $\boldsymbol{X} \rightarrow \boldsymbol{U}$ via convolution

○ **Step 2.** SE block applies

✓ **(1) Squeeze** $\boldsymbol{F_{sq}}$:

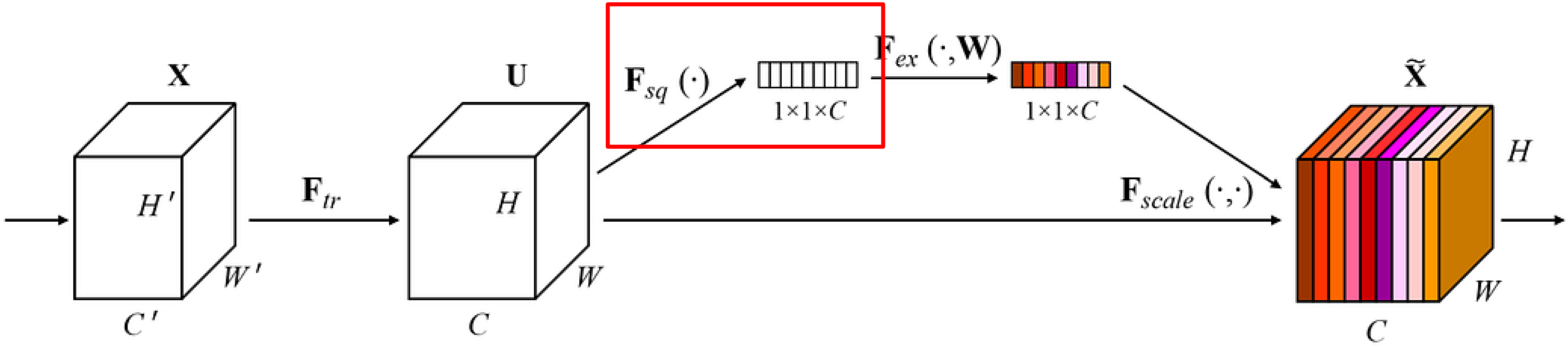
✓ **(2) Excitation** $\boldsymbol{F_{ex}}$:

✓ **(3) Scale** $\boldsymbol{F_{scale}}$:

○ **Step 3.** output: recalibrated feature map $\widetilde{\boldsymbol{X}}$

EfficientNet Architecture

- SE Block – Squeeze and Excitation in MBconv
  - Squeeze – Global Information Embedding
    - Applies **Global Average Pooling** to each channel

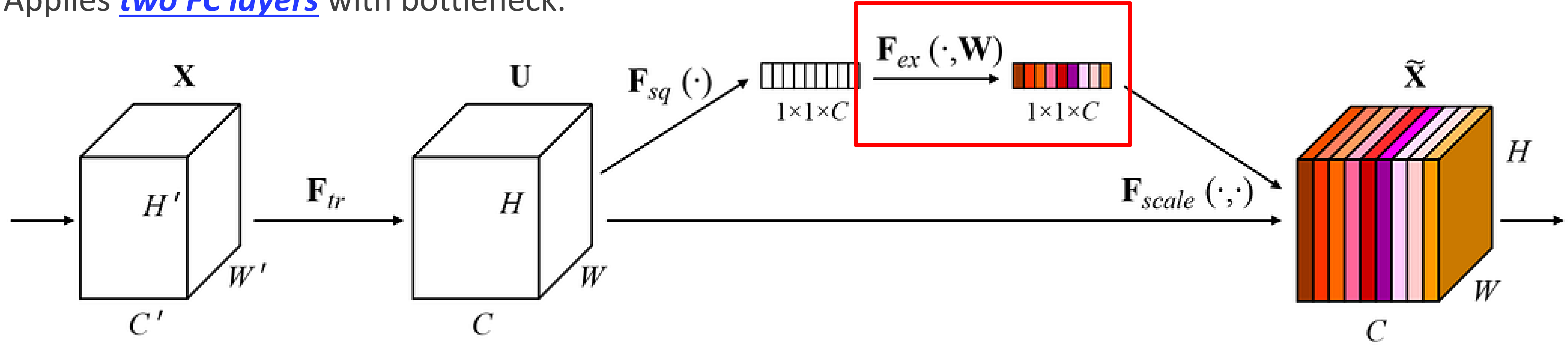


✓ $z_c =$

✓Each channel is compressed into a scalar representing its global activation.

EfficientNet Architecture

- **SE Block – Squeeze and Excitation in MBconv**
  - **Excitation – Learn Channel Dependencies (i.e., Learning What to Emphasize)**
    - Applies ***two FC layers*** with bottleneck.



✓ $\boldsymbol{s} =$

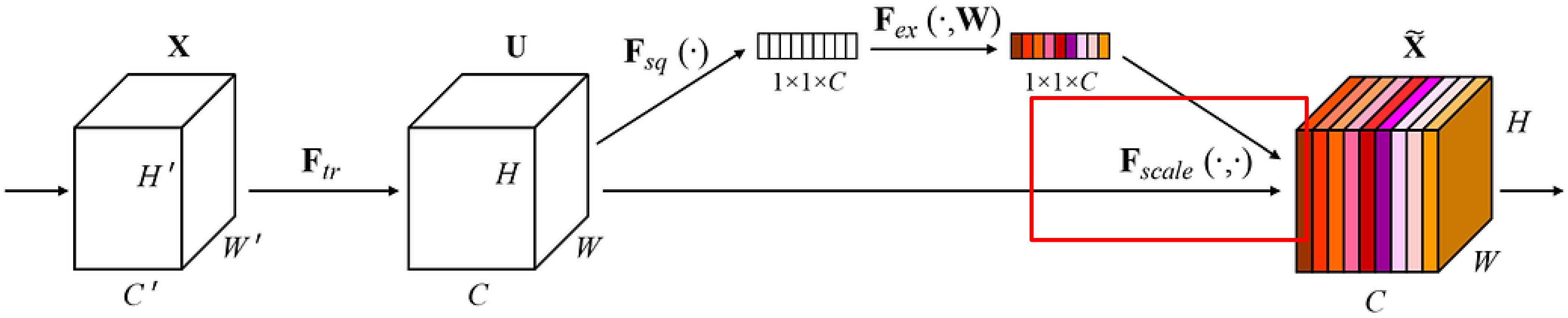
- $\boldsymbol{z}$: input vector (obtained form the squeeze step)
- $\boldsymbol{W_1} \in \mathbb{R}^{\frac{C}{r}\times C}$: reduce channel dimension from $\boldsymbol{C} \rightarrow \boldsymbol{C/r}$
- $\boldsymbol{W_2} \in \mathbb{R}^{C\times\frac{C}{r}}$: expand back from $\boldsymbol{C/r} \rightarrow \boldsymbol{C}$
- $\boldsymbol{r}$: **reduction ratio** (e.g., 16, 4) – a hyperparameter

- $\boldsymbol{\delta}$: ReLU for FC1
- $\boldsymbol{\sigma}$: Sigmoid for FC2

$$\boldsymbol{FC1}: \boldsymbol{W_1}\cdot\boldsymbol{z} \rightarrow \boldsymbol{ReLU}(\boldsymbol{\delta}) \rightarrow \mathbb{R}^{\frac{C}{r}}$$
$$\boldsymbol{FC2}: \boldsymbol{W_2}\cdot\boldsymbol{z} \rightarrow \boldsymbol{Sigmoid}(\boldsymbol{\sigma}) \rightarrow \mathbb{R}^{C}$$

EfficientNet Architecture

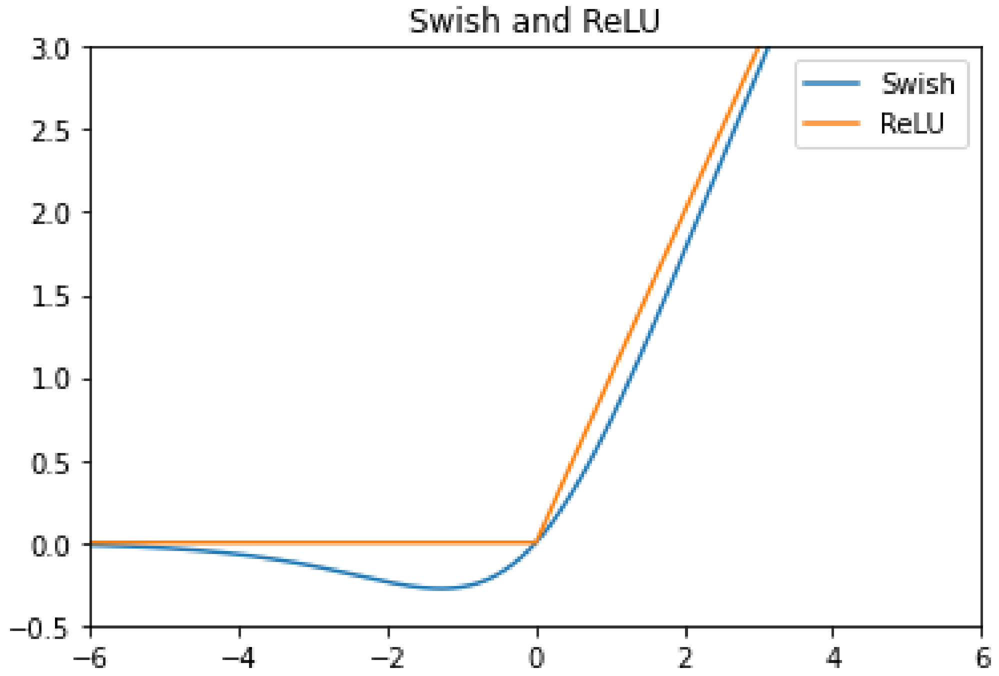
- **SE Block – Squeeze and Excitation in MBconv**
  - **Scale – Channel-wise Recalibration (i.e., Reweighting the Feature Map)**



- Output weights ______________________ are used to rescale ___
- *Channel-wise multiplication*
  - ✓ $\widetilde{X_c} = F_{scale}(u_c, s_c) = s_c \cdot u_c$
- This enhances the informative channels and suppresses less useful ones.

EfficientNet Architecture

- **Swish Activation Function**
  - **A Smooth Activation for Better Gradients**
    - Proposed by Google researchers
    - Formula
      - ✓$\boldsymbol{Swish(x) =}$ (where $\boldsymbol{\beta}$ is often set to 1)
    - **Non-monotonic**, **smooth**, and **avoids dying neurons**
    - A Smooth Activation for Better Gradients



Function	Formula	Smooth?	Monotonic?	Used in
ReLU		No	Yes	Most CNNs
Swish		Yes	No	EfficientNet

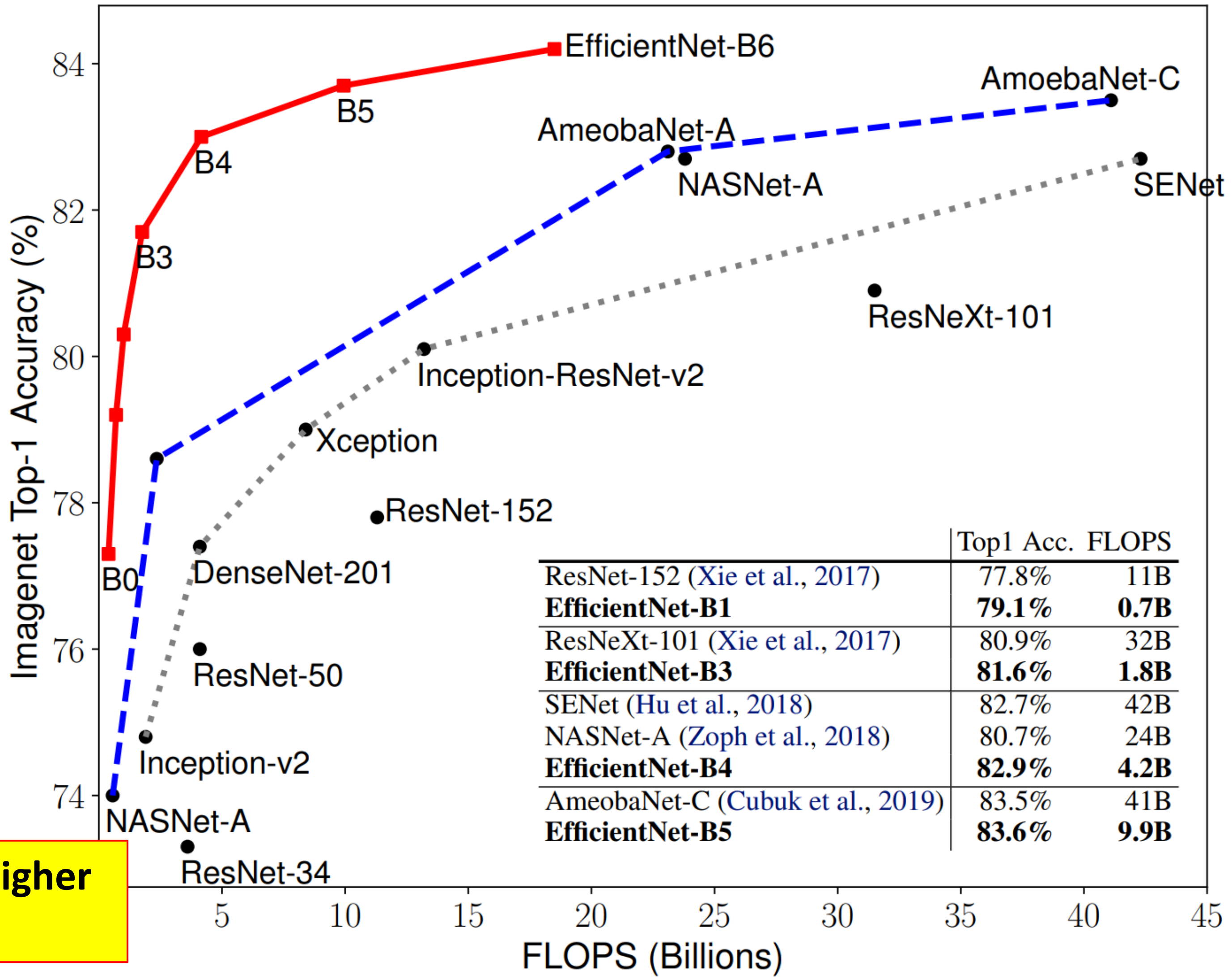
EfficientNet Architecture

- **Swish Activation Function**
  - Activation Map Comparison

- Activation maps from a 6-layer network output.
- **ReLU**: Sharp edges (star-shaped regions) → ***sudden activation changes*** → ***harder optimization***
- **Swish**: Smoother transition → **stable and gradual activation response**
  → *This makes it easier for optimizers to* ***follow smooth loss surfaces*** *and* ***avoid local traps***

EfficientNet Performance vs. Other Models

- **EfficientNet Achieves Better Accuracy with Fewer FLOPs**
  - **Top-1 Accuracy vs. FLOPs on ImageNet**
    - **Red Line**
      - ✓EfficientNet family (B0–B6)
    - **Blue Line**
      - ✓NAS-based models (NASNet, AmoebaNet)
    - **Gray Dotted Line**
      - ✓Other conventional models (ResNet, SENet, DenseNet, etc.)
  - **Meaning of B0 to B6**
    - B0: Base model
    - B1–B6: Versions scaled from B0 using compound scaling



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

With **lower computational cost**, EfficientNet achieves **higher accuracy** than larger models

MobileNet vs EfficientNet

- **Comparing MobileNet and EfficientNet**

Feature	MobileNet	EfficientNet
Design Strategy		
Building Blocks		
Accuracy		
Params / FLOPs		
Use Cases		