ComputerVision

Week6

2025-2

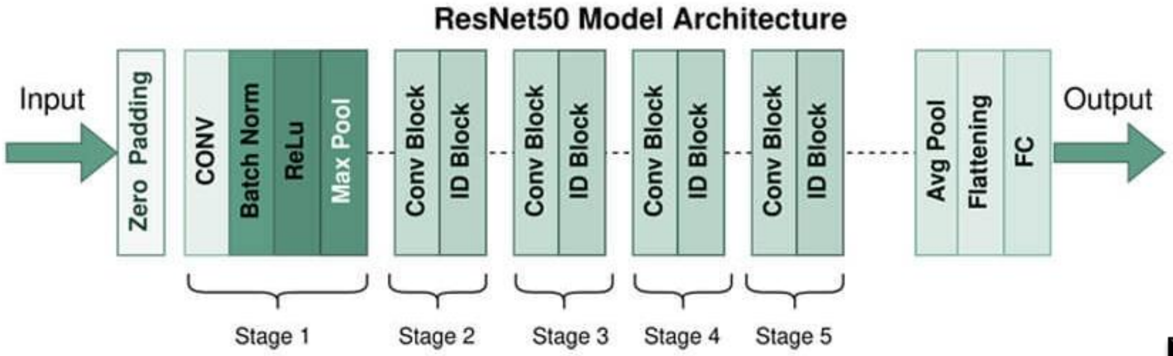
Mobile Systems Engineering

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**.

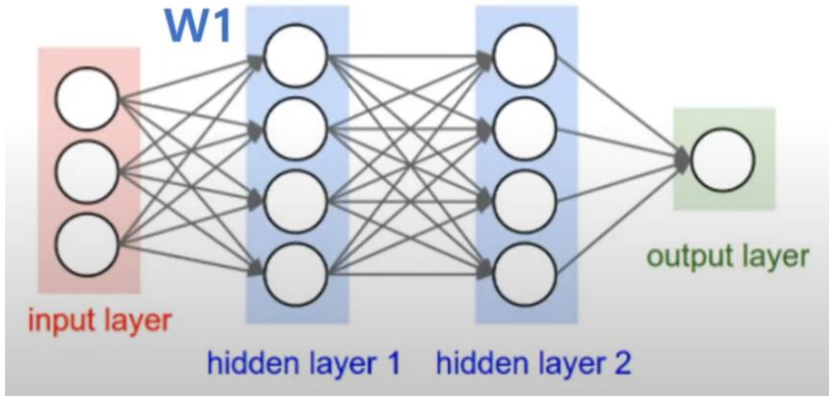
From ResNet to Lightweight CNNs: MobileNet and EfficientNet

▪ 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

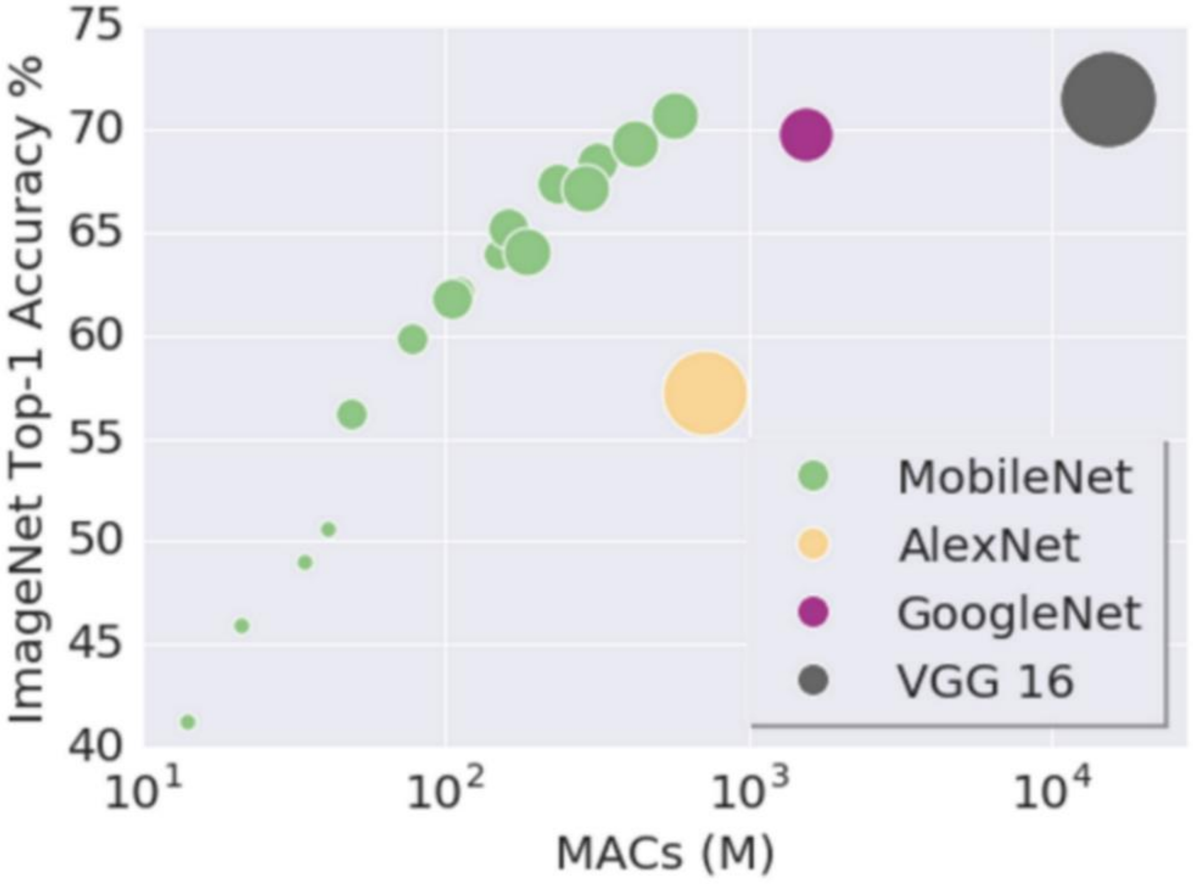
- **Example 1: Dot Product**
  - $\boldsymbol{Y} = \boldsymbol{w}[0] \cdot \boldsymbol{x}[0] + \boldsymbol{w}[1] \cdot \boldsymbol{x}[1] + \boldsymbol{w}[2] \cdot \boldsymbol{x}[2]$
    - ✓**3** multiplications and **2** additions → **Total: 5 FLOPs → In general: 2n−1 FLOPs for n elements**

- **Example 2: Dense (Fully Connected) Layer**
  - **Input vector:** 3-dimensional / **Output vector:** 4-dimensional
    → **Weight matrix:** $\boldsymbol{W1} \in \mathbb{R}^{4\times 3}$
  - **4 dot products performed** → 5×4=20 FLOPs
    - ✓**12 multiplications** (4 rows × 3 elements)
    - ✓**8 additions** (each dot product needs n-1 additions)



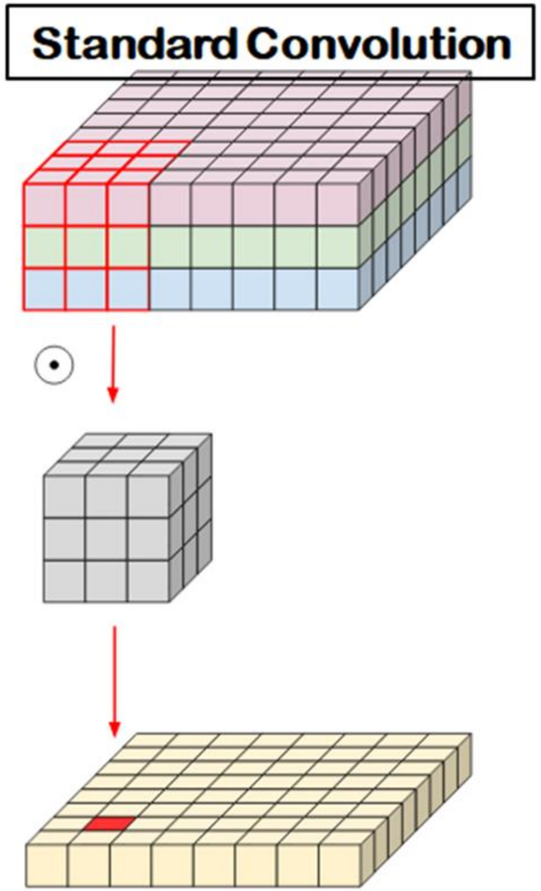
What is MobileNet?

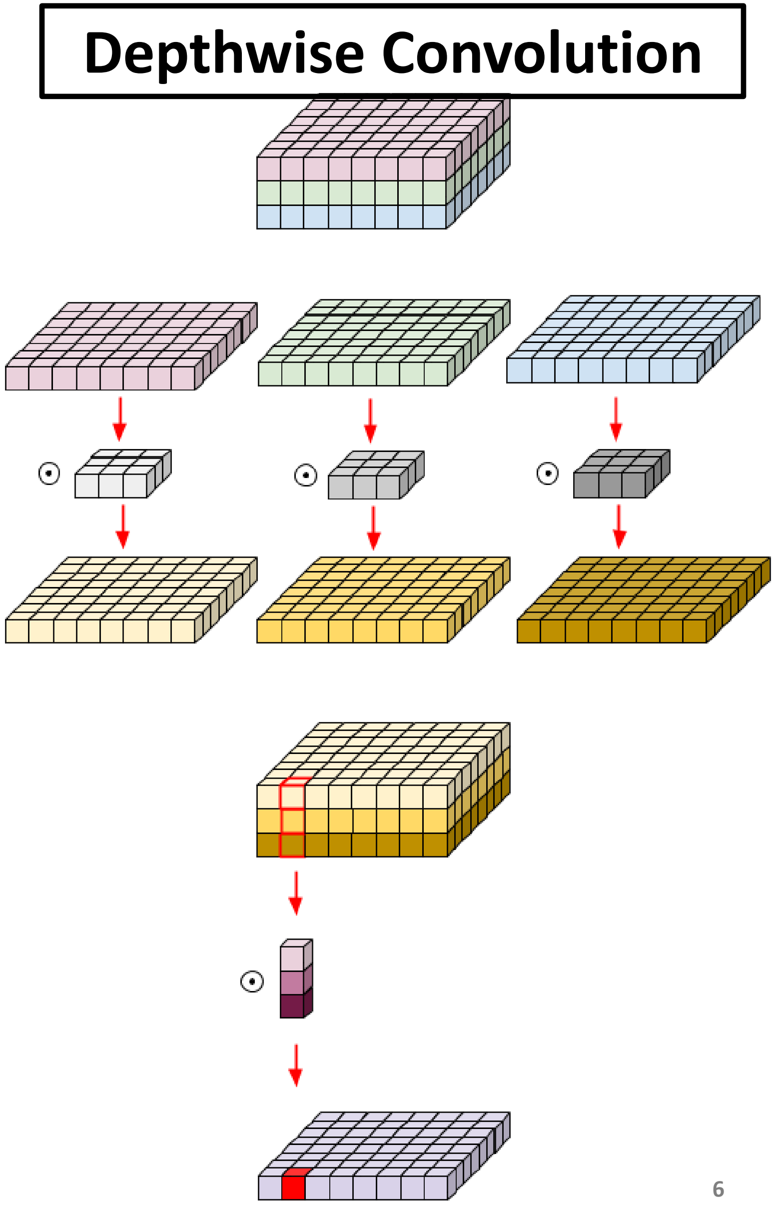
- **Introducing MobileNet – Efficient CNNs for Mobile Vision**
  - Proposed by Google in 2017
  - Designed for **mobile and embedded vision applications**
  - **Key ideas**
    - o **Depthwise Separable Convolution**
    - o **Two hyperparameters** for flexible scaling
  - Used in real-world tasks
 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



Standard Convolution vs. Depthwise Separable Convolution

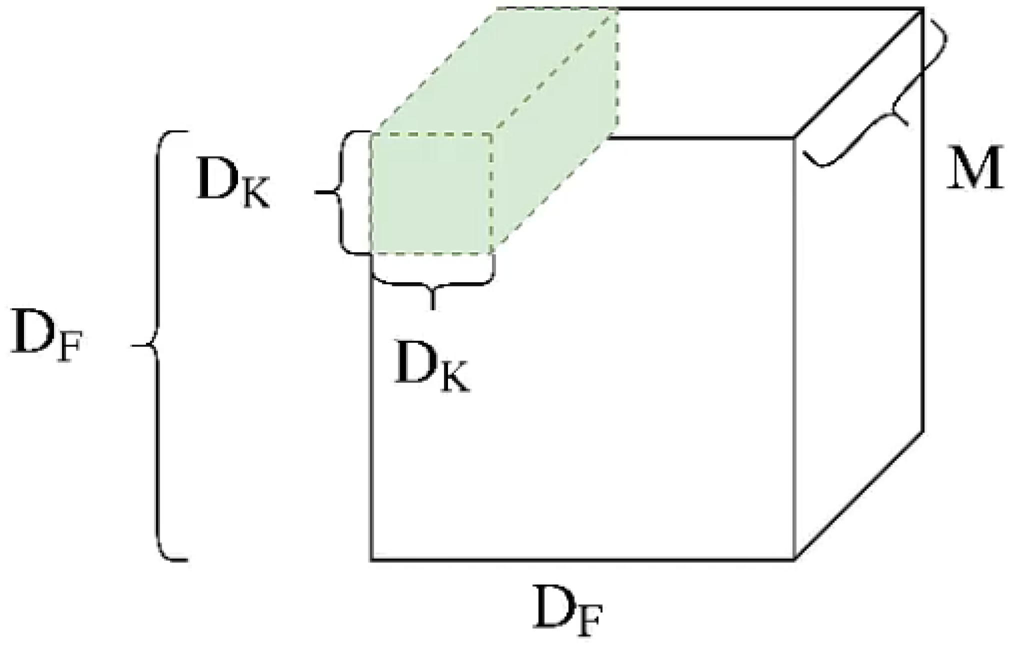
- **Rethinking Convolution: Lighter and Faster**
  - **Standard convolution**
    - Applies multiple filters across all channels simultaneously
  - **Depthwise separable convolution**
    - Depthwise
      - ✓Applies one filter per input channel
    - Pointwise
      - ✓1×1 convolution to combine outputs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





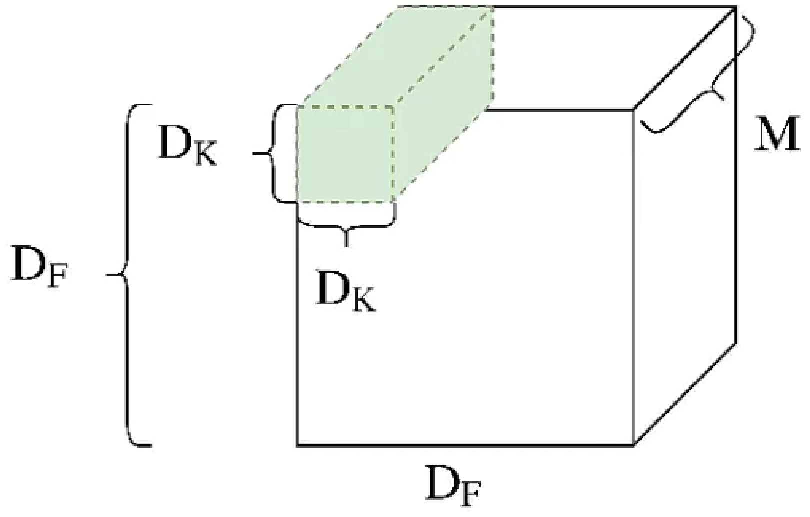
How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - **How to Compute FLOPs in Standard Convolution**
    - **Key Definitions**
      - ✓ $\boldsymbol{D_K}$: Kernel size (e.g., 3 → 3×3)
      - ✓ $\boldsymbol{M}$: Number of input channels
      - ✓ $\boldsymbol{N}$: Number of output channels (i.e., number of filters)
      - ✓ $\boldsymbol{D_F}$: Input feature map spatial dimension
      - ✓ $\boldsymbol{D_G}$: Output feature map spatial dimension
        - ➢ Usually, $\boldsymbol{D_G} \approx \boldsymbol{D_F}$ (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: $\boldsymbol{D_K^2} \times \boldsymbol{M}$ multiplications
      - ✓**Step 2. All Spatial Locations in One Filter**
        - ➢Spatial positions: $\boldsymbol{D_G^2}$ (assuming square output)
          → Typically $\boldsymbol{D_G} = \boldsymbol{D_F}$ for stride 1 & padding
        - ➢FLOPs per filter: $\boldsymbol{D_G^2} \times \boldsymbol{D_K^2} \times \boldsymbol{M}$
      - ✓**Step 3. All N Filters (Output Channels)**
        - ➢Multiply by $\boldsymbol{N}$
      - ✓Total FLOPs=$\boldsymbol{D_G^2} \times \boldsymbol{D_K^2} \times \boldsymbol{M} \times \boldsymbol{N}$



Mults once = $D_K^2 \times M$

Mults per Kernel = $D_G^2 \times D_K^2 \times M$

Mults N Kernels = $N \times D_G^2 \times D_K^2 \times M$

How Efficient is Depthwise Separable Conv?

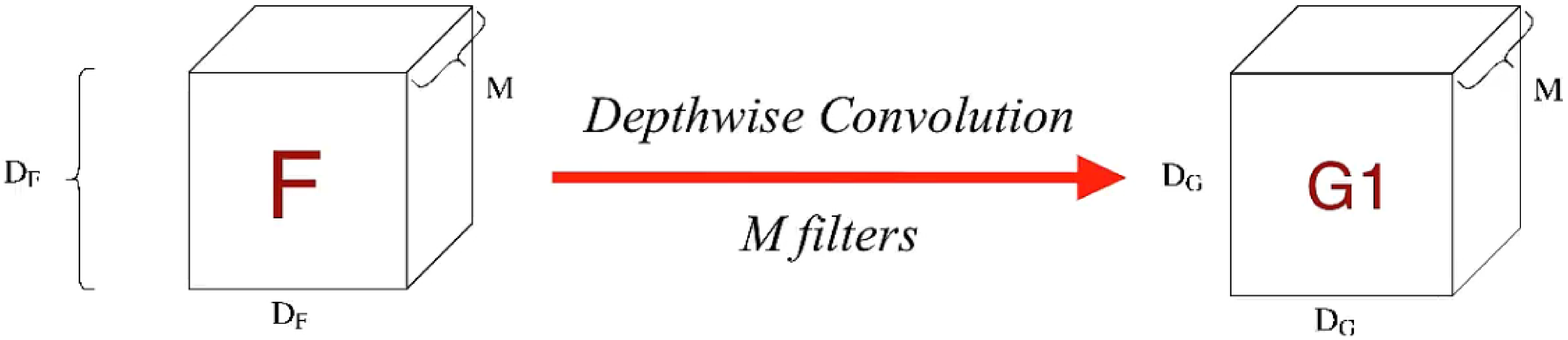
- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - What is Depthwise Separable Convolution?
    - **Key Concepts**
      - ✓Standard convolution combines **spatial filtering + channel mixing** in one operation.

      - ✓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

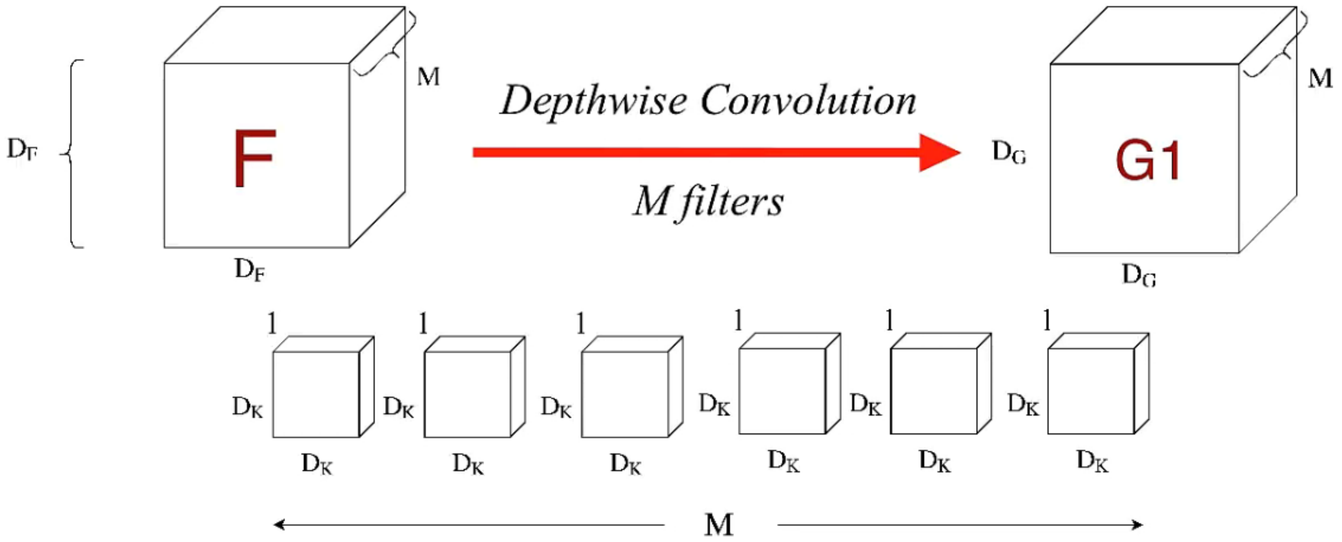
✓$\boldsymbol{M}$: number of input channels

✓$\boldsymbol{N}$: 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)**



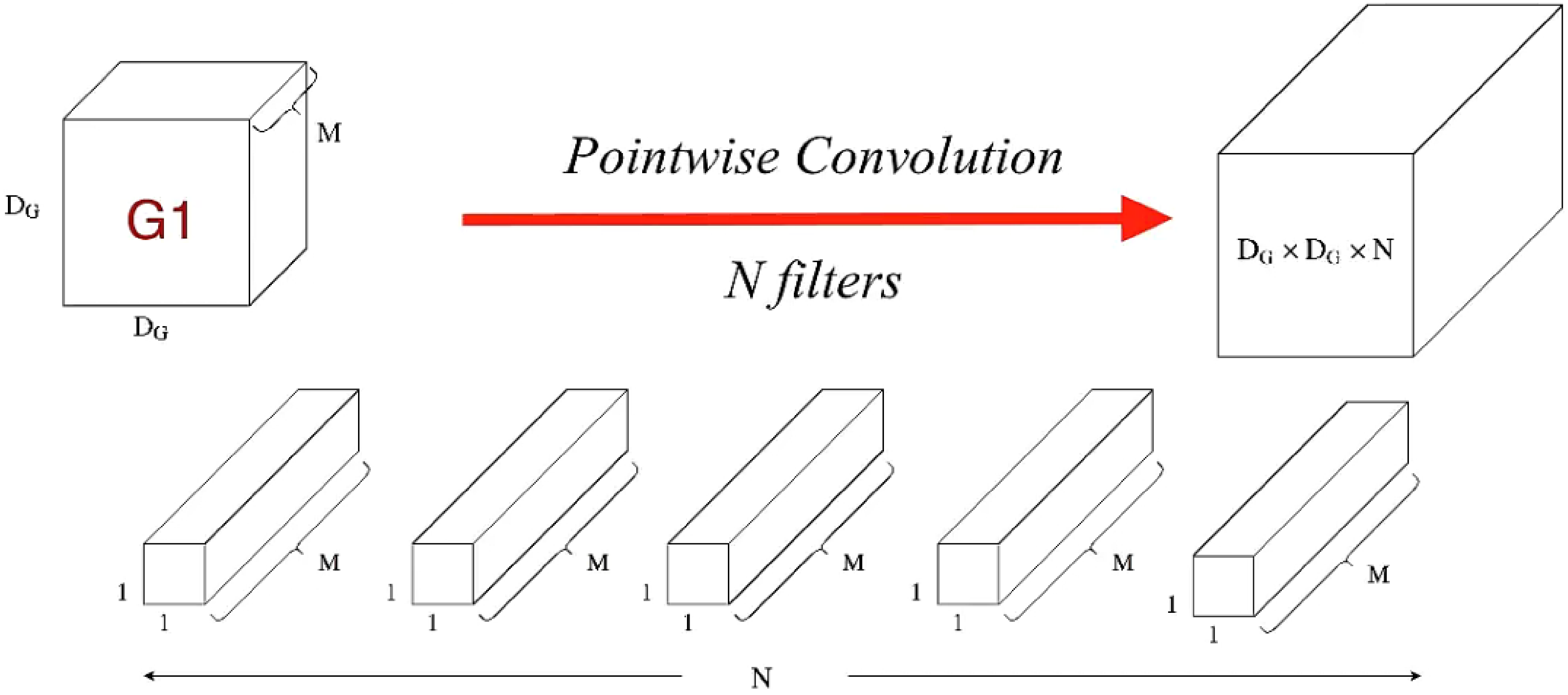
➢Each of the **M** channels gets a **filter**

➢For each channel: $D_F^2 \times D_K^2$ operations

→ Total: $D_F^2 \times D_K^2 \times M$

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: $\boldsymbol{M} \times \boldsymbol{N} \times \boldsymbol{D_F^2}$

How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - **Total FLOPs**
    - $FLOPs = D_F^2 \times\ D_K^2 \times M\ (i.e., Step 1) + M \times\ N \times\ D_F^2\ (i.e., Step\ 2)$
  - Compare with **Standard Convolution**
    - $FLOPs_{standard} = D_G^2 \times D_K^2 \times M \times N$
  - **Efficiency Gain**
    - $\dfrac{Depthwise\ Separable\ Convolution}{Standard\ Convolution} = \dfrac{D_F^2 \times D_K^2 \times M + M \times N \times D_F^2}{D_G^2 \times D_K^2 \times M \times N} = \dfrac{M \times D_F^2 (D_K^2 + N)}{D_G^2 \times D_K^2 \times M \times N}$

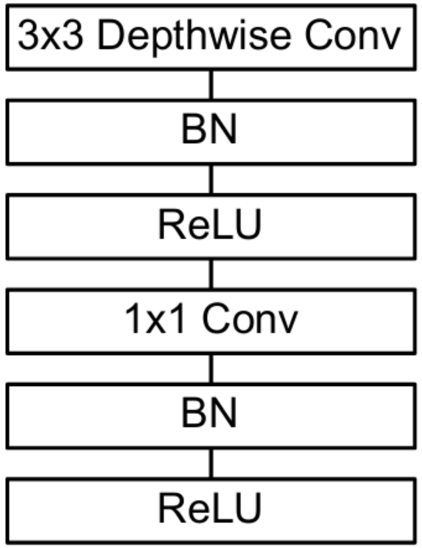
$$= \frac{M \times\ D_G^2 (D_K^2 + N)}{D_G^2 \times D_K^2 \times M \times N} = \frac{D_K^2 + N}{D_K^2 \times N} = \frac{1}{N} + \frac{1}{D_K^2}$$

How Efficient is Depthwise Separable Conv?

- **FLOPs Comparison: Standard vs. Depthwise Separable**
  - **Example – Efficiency Gain**
    - Efficiency Gain: $\frac{1}{N} + \frac{1}{D_K^2}$
    - For $D_K = 3$ (size of filter), $N = 256$ (number of channels)
      - ✓Relative Cost = $\frac{1}{256} + \frac{1}{3^2} = \frac{1}{256} + \frac{1}{9} = \frac{265}{2304} = 0.11501\ldots$
      - ✓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 α
      - ✓Controls number of channels at each layer
    - Input/output channels → multiplied by α ($0 < \alpha \leq 1$)
      - ✓Reduces FLOPs and parameters by approximately α^2
    - Common choices
      - ✓$\alpha \in \{1.0, 0.75, 0.5, 0.25\}$

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