# Residual Network:

**ResNet** (short for **Residual Network**) is a type of deep neural network architecture that was introduced by **Kaiming He et al.** in the paper **"Deep Residual Learning for Image Recognition"** in 2015. It was a breakthrough in deep learning and won the **ImageNet 2015 competition**.

### 🧠 Why ResNet Was Created

Before ResNet, very deep networks (e.g., 50+ layers) were difficult to train due to the **vanishing/exploding gradient problem**. As a result, adding more layers actually made performance worse. ResNet solved this by introducing the concept of **"residual learning."**

**ResNet** (short for **Residual Network**) is a type of deep neural network architecture that was introduced by **Kaiming He et al.** in the paper **"Deep Residual Learning for Image Recognition"** in 2015. It was a breakthrough in deep learning and won the **ImageNet 2015 competition**.

### Why ResNet Was Created

Before ResNet, very deep networks (e.g., 50+ layers) were difficult to train due to the **vanishing/exploding gradient problem**. As a result, adding more layers actually made performance worse. ResNet solved this by introducing the concept of **"residual learning."**

### Common ResNet Variants

**ResNet-18**: 18 layers

**ResNet-34**: 34 layers

**ResNet-50**: 50 layers (uses bottleneck blocks)

**ResNet-101** and **ResNet-152**: deeper versions

These are widely used in **image classification**, **object detection**, **face recognition**, and more.

### What does ****ResNet-50**** mean?

**ResNet-50** is a specific version of the **ResNet (Residual Network)** architecture that has **50 layers** deep. It’s one of the most popular versions used in computer vision tasks due to its good balance between performance and computational cost.

### 🧱 ResNet-50 Architecture Overview

ResNet-50 uses a special type of building block called a **bottleneck block**, which is a stack of three convolutional layers:

Note: see practicale implementatio of ResNet-50 in github “deeplearning specialization”

## What is Transfer Learning?

**Transfer Learning** is a technique where you **take a model trained on one task** (usually on a large dataset like ImageNet) and **reuse it for a different but related task**.

Instead of training a model **from scratch**, you **leverage the learned features** of an already trained model. This saves **time, data, and compute**.

### Example: ResNet + Transfer Learning

Let’s say ResNet-50 is trained on ImageNet (millions of images, 1000 classes). You want to classify **medical images** (e.g., X-rays with just 2 classes: healthy or pneumonia).

With transfer learning:

You **load the pretrained ResNet-50**.

You **remove the last layer** (1000-class output).

You **add your own classifier** (e.g., for 2 classes).

You **fine-tune** the model on your dataset (fully or partially).

# What is MobileNet:

**MobileNet** is a class of efficient neural networks developed by **Google** for **real-time computer vision** on devices with **limited computational power**.

The key idea is to **reduce computation and model size** while maintaining good accuracy.

## **MobileNet Family**

MobileNet is a mobile neural network architecture, firstly developed by Google in 2017. The main feature of this model is a high speed, combined with a high performance.

Currently, there are 3 versions of MobileNet architecture.

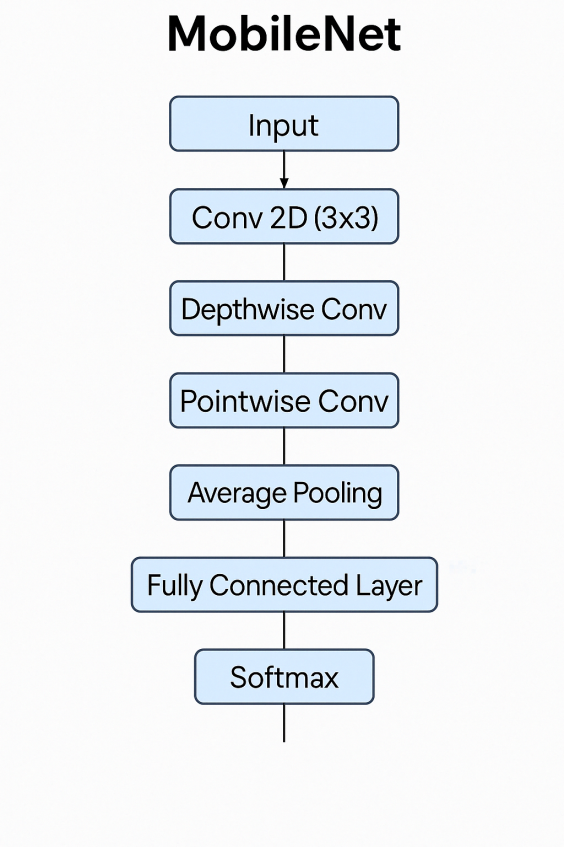
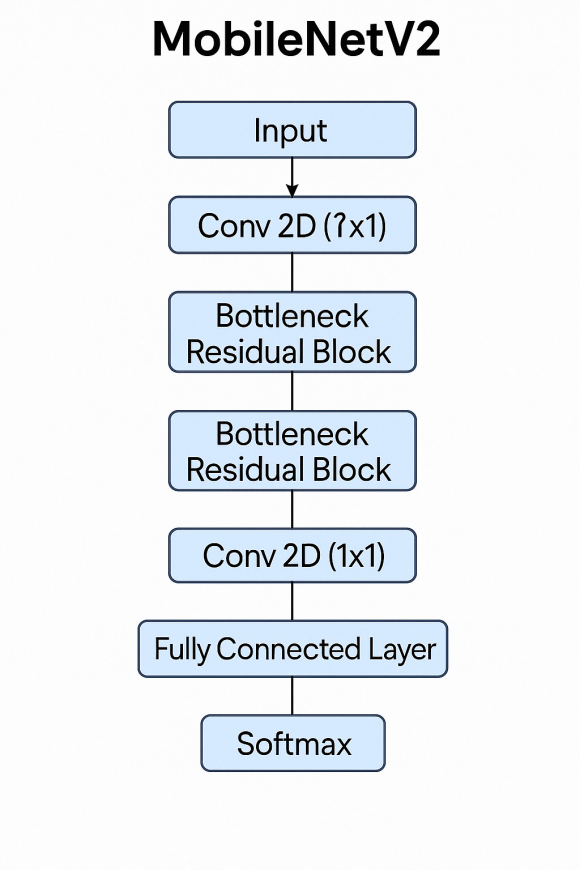
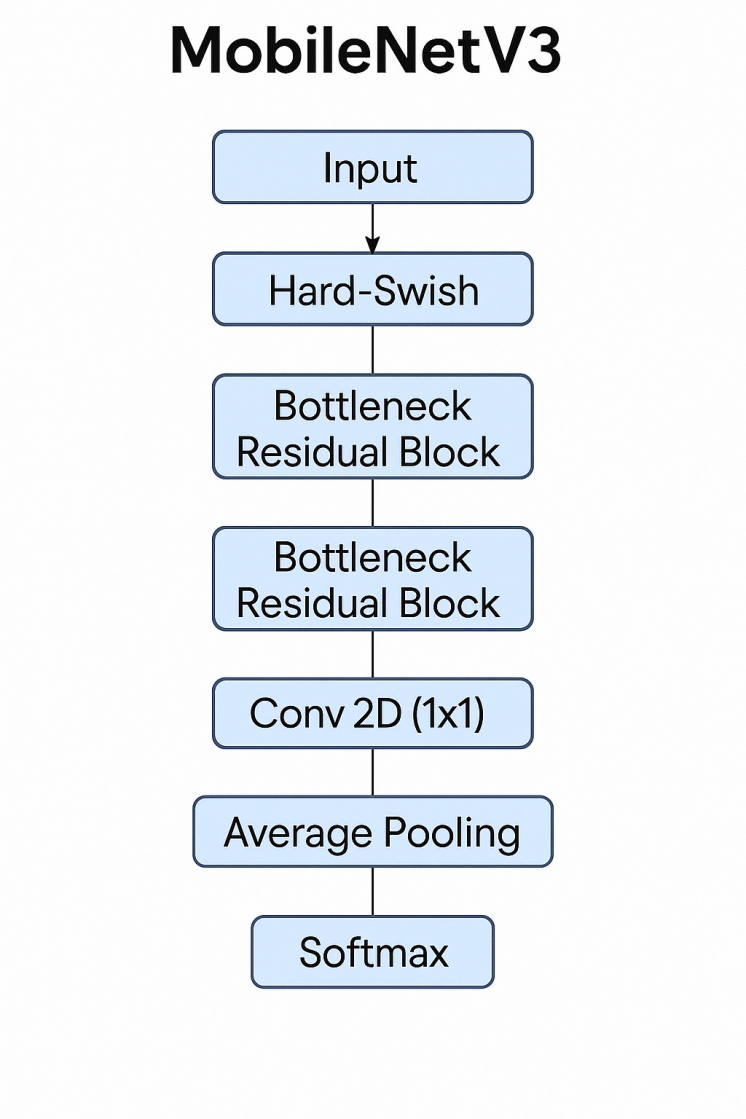
* [MobileNet v1 (2017)](https://arxiv.org/abs/1704.04861" \t "https://medium.com/@pandrii000/_blank)
* [MobileNet v2 (2018)](https://arxiv.org/abs/1801.04381" \t "https://medium.com/@pandrii000/_blank)
* [MobileNet v3 (2019)](https://arxiv.org/abs/1905.02244" \t "https://medium.com/@pandrii000/_blank)

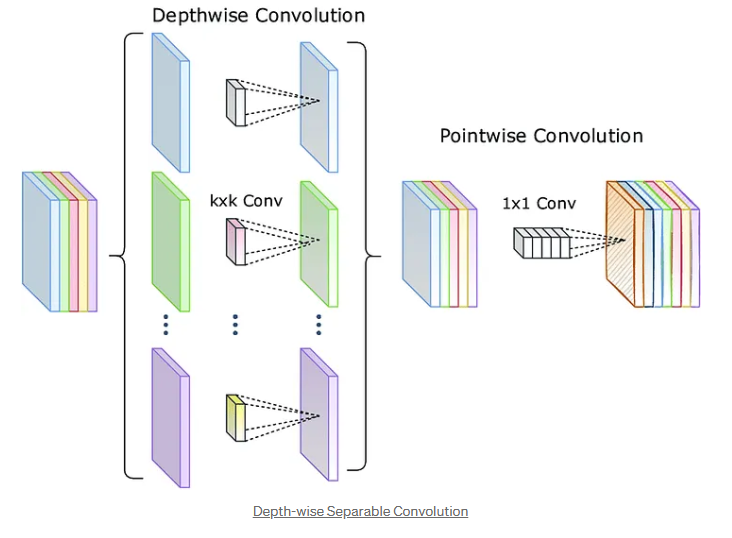
### 🌟 Motivation

Traditional CNNs like VGG, ResNet, etc., are powerful but computationally expensive. MobileNetV2 was designed to:

**Run efficiently on devices with limited resources** (like smartphones, Raspberry Pi, etc.).

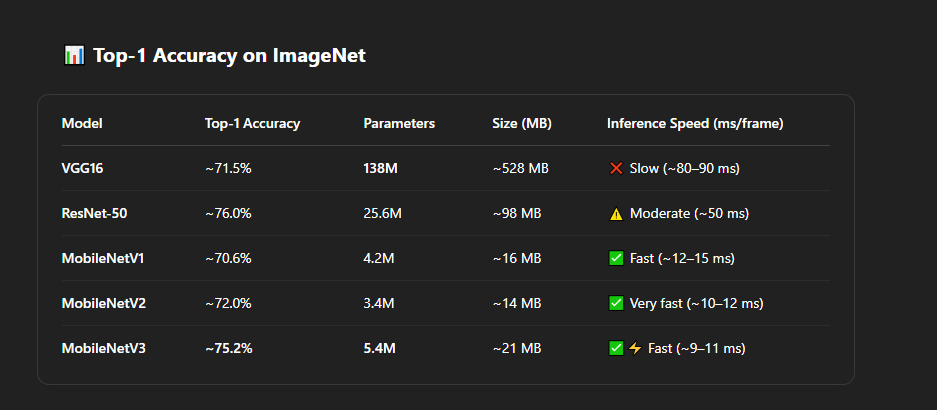
**Maintain high accuracy** despite reduced computation





Note :

https://medium.com/@pandrii000/mobilenet-architectures-17fe7406d794#id\_token=eyJhbGciOiJSUzI1NiIsImtpZCI6ImM3ZTA0NDY1NjQ5ZmZhNjA2NTU3NjUwYzdlNjVmMGE4N2FlMDBmZTgiLCJ0eXAiOiJKV1QifQ..wKRGh195W6LxBh\_azWu9LtviNCqSve1mRaVk1IN-J2LyciK6I9pGDNlRNqnNjyURwrIAxxbiIcYhnVv2g9V3GGj2K6xkzy2oRB1GG9x1-B3pPM9QLpd9S8VB84GMviX\_Q8ZmvjHIktBp9W1u\_Y4wQ5g8K0hZI8ETt0hnlfqjBbTsbFKIUVEElliHwt5CDj2LM6CviKJlREhjwNDKn\_OUVumKcDHKAoE2qO4YdowhTRajHXS1NHmbhxLuwZF4fiPLkvb4Hf2X4otsgbUydXdPwaXI2P9xjB\_EryLl9PhhdjerzmiIQZ\_4ZOoApB0yEEJPXs9a5zgo9S2ys6-sKyK7Hg



Details explanation

## Standard Convolution (For Reference)

In a standard convolution:

You apply **one filter per output channel**, and each filter spans **all input channels**.

Very **computationally expensive**, especially with large input sizes.

For example:

Input: 32x32x3 (Height x Width x Channels)

Kernel: 3x3

Number of filters: 64

Output: 32x32x64  
Each filter does a full 3x3x3 operation and we have 64 such filters.

**Total computations:**

CopyEdit

64 filters × (3×3×3) = 1,728 weights

## Depthwise Separable Convolution (Used in MobileNet)

It’s a two-step process:

### ****Step 1: Depthwise Convolution****

Applies **one filter per input channel**.

Doesn't mix information **between** channels.

If input has C channels, then you apply C separate filters

Example

Input: 32x32x3

Apply one 3x3 filter to **each** of the 3 channels → You still have 3 channels after this.

✅ This drastically **reduces computation**.

### ****Step 2: Pointwise Convolution (1x1 Convolution)****

Now we apply **1x1 convolutions** across **all channels** to **combine** the features.

This is where the **inter-channel mixing** happens.

You can apply as many 1x1 filters as needed for the desired output channels.

Example:

Input from depthwise conv: 32x32x3

You want output channels = 64 → Use 64 1x1x3 filters

### ****Object Localization****

**Goal:**  
Find **where** a specific object is located in an image.

**Output:**  
A **bounding box** around the object + the class label (e.g., "cat").

**Example Use Case:**  
You know there is only one object in the image, and you want to locate it.

**Illustration:**  
An image of a dog → Output: "dog" with coordinates of the box around the dog.

### 🎯 ****Object Detection****

**Goal:**  
Detect **multiple objects** in an image and **classify** them as well.

**Output:**  
Multiple bounding boxes + class labels for each detected object.

**Example Use Case:**  
Images with **more than one object**, like cars, people, traffic lights, etc.

**Illustration:**  
A street scene → Output: 3 bounding boxes labeled "car," "person," and "traffic light."

