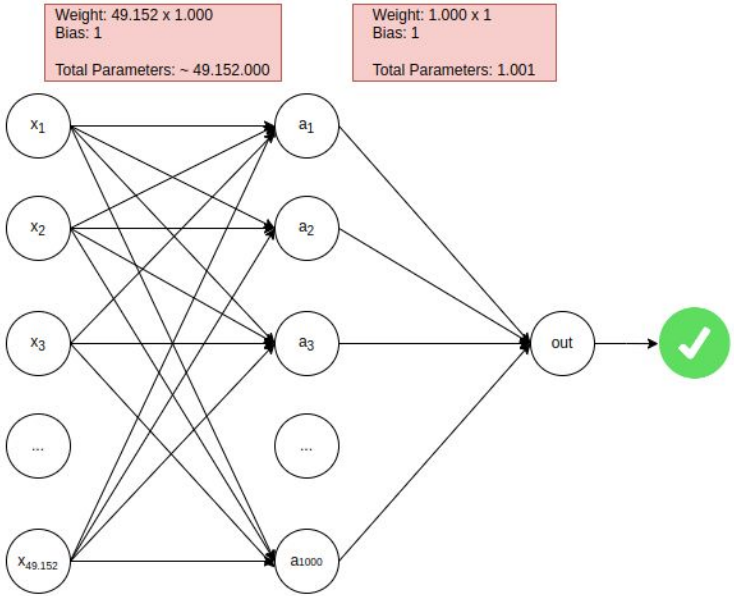
Day 2: Introduction to Convolutional Neural Networks (CNNs)

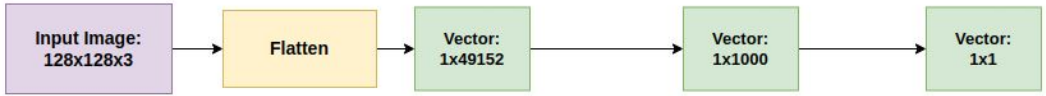
Lesson of Content

1. What is the problem with Neural Network ?
2. Image processing
3. Convolutional Operation
4. Some blocks
5. ImageNet challenge
6. Some Architecture
7. CNN Applications

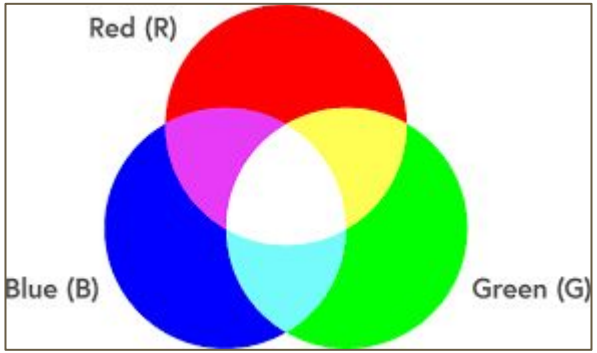
1. What problem with Neural Network ?



Với ảnh kích thước nhỏ: 128x128x3
Số lớp Hidden: 1
=> Mạng tương đối đơn giản mà số parameters đã đến gần 50 M. (Too big)



2. Image Processing

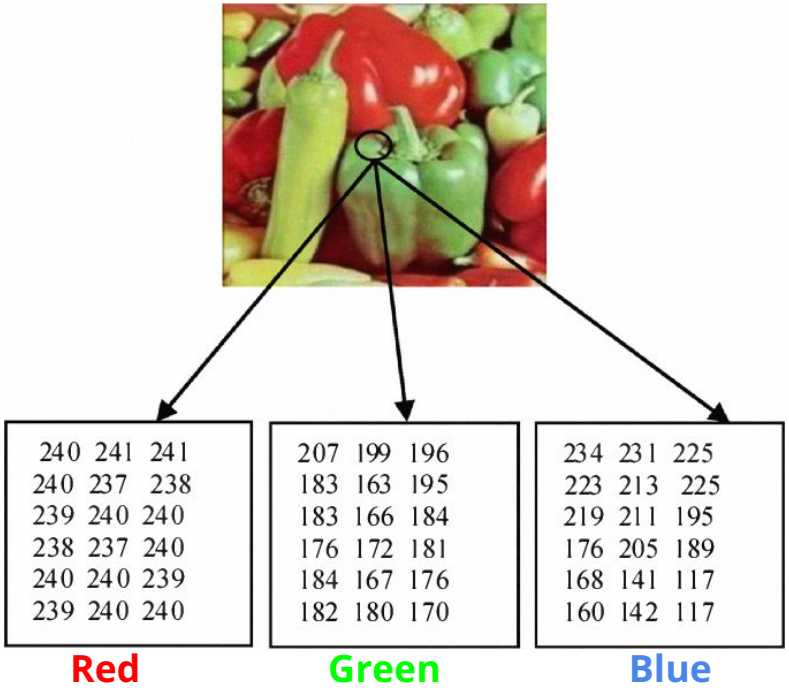


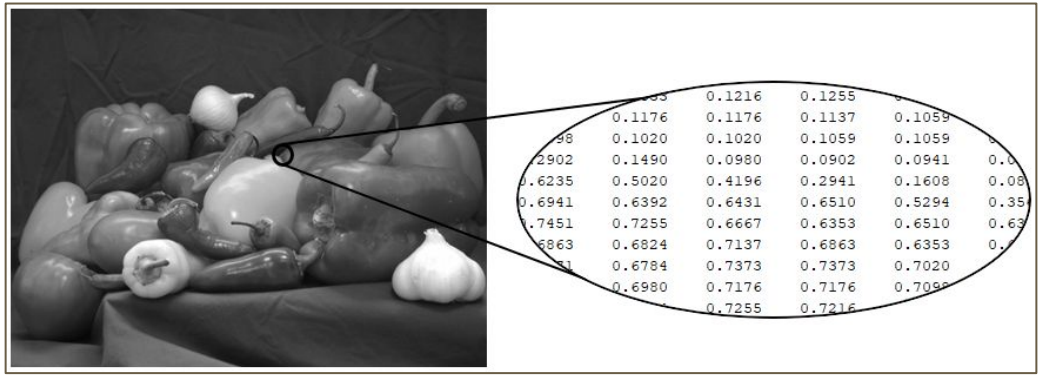
- **Color:** Được tạo từ ba kênh màu.
  Shape: **32 x 80 x3**

- **GrayScale:** Được tạo từ 1 kênh màu range từ 0 -> 255.
  Shape: **32 x 80 x 1**

- **Black-and-white:** Được bởi hai giá trị 1 và 0.
  Shape : **32 x 80 x 1**

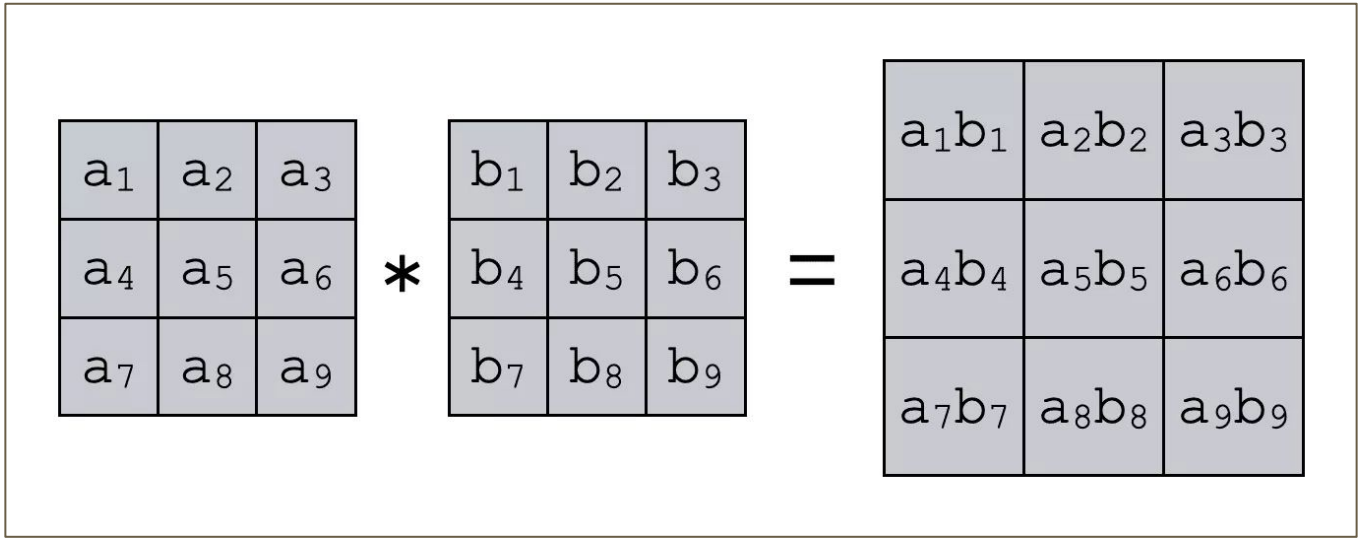
2. Image Processing



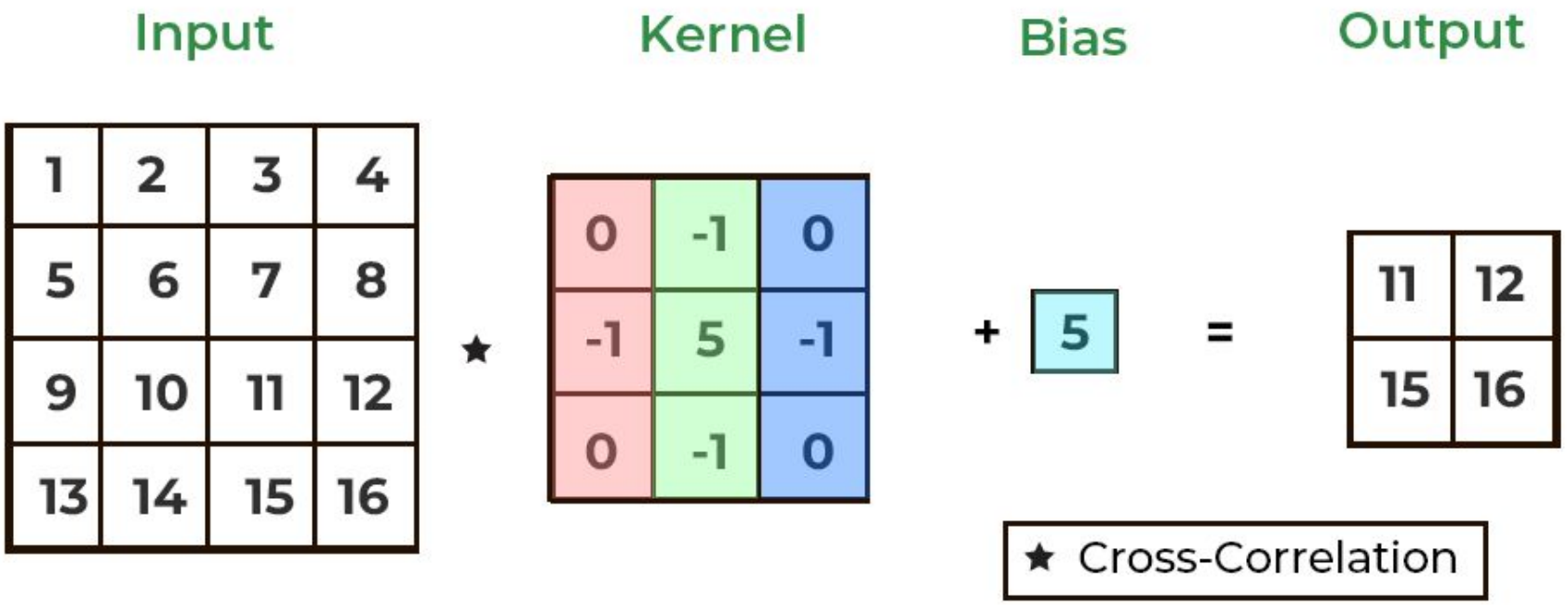


Element-wise multiplication matrix

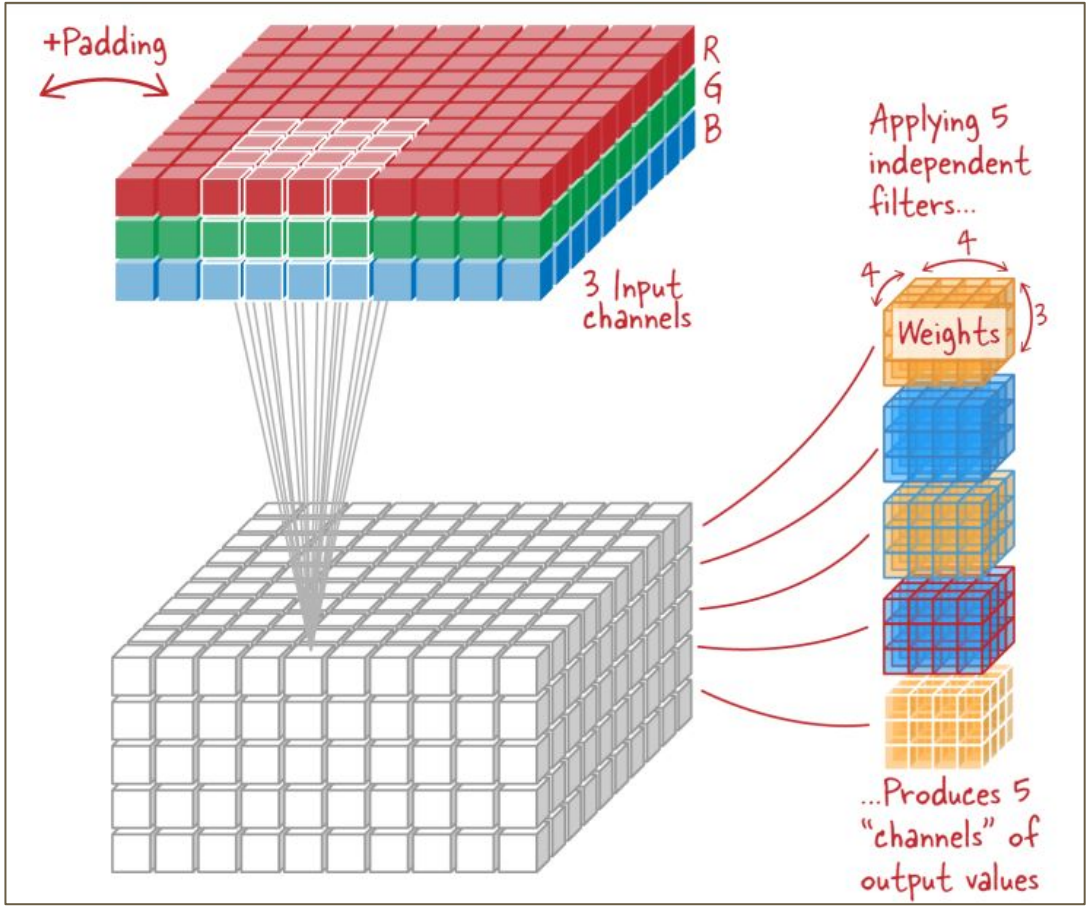
Đây là một phép toán được sử dụng nhiều nhất trong CNNs.



3. Convolutional Operation



3. Convolutional Operation



Input: 256x256x3
Kernel: 4x4x3 , có **5** cái kernel,
Padding = 0 , Stride = 2
Output: 127x127x**5**

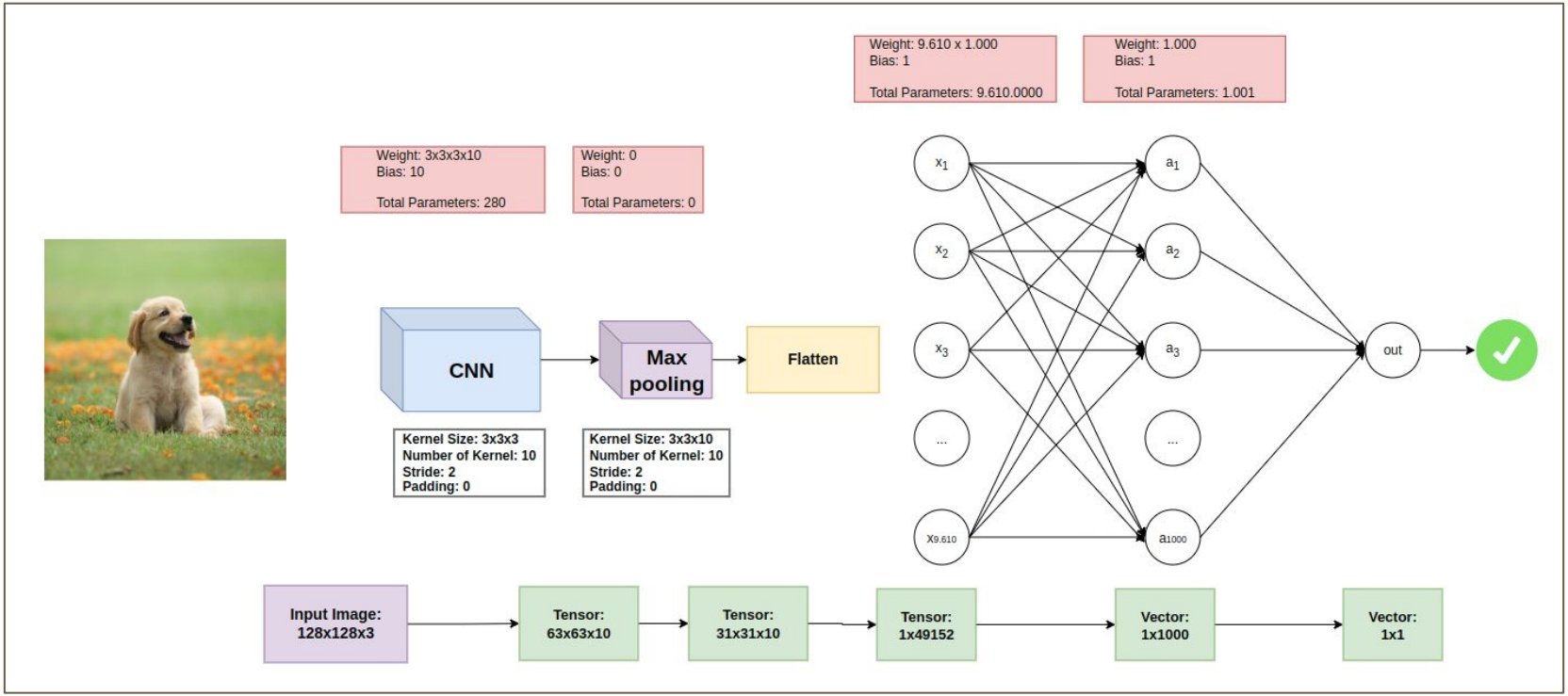
- Trích xuất đặc trưng: Giúp học và phát hiện được các đặc trưng của dữ liệu như (Màu, góc, cạnh, ...).

- **Hiểu được cấu trúc không gian:** của dữ liệu.

- Tăng tính phi tuyến: CNN có khả năng phát hiện các đặc trưng phức tạp -> nó có thể tạo thành các hàm phức tạp.

- Giảm số lượng tham số: Giúp inference nhanh và training nhanh (Do CNN không phụ thuộc vào kích thước ảnh đầu vào mà phụ thuộc vào kernel size)

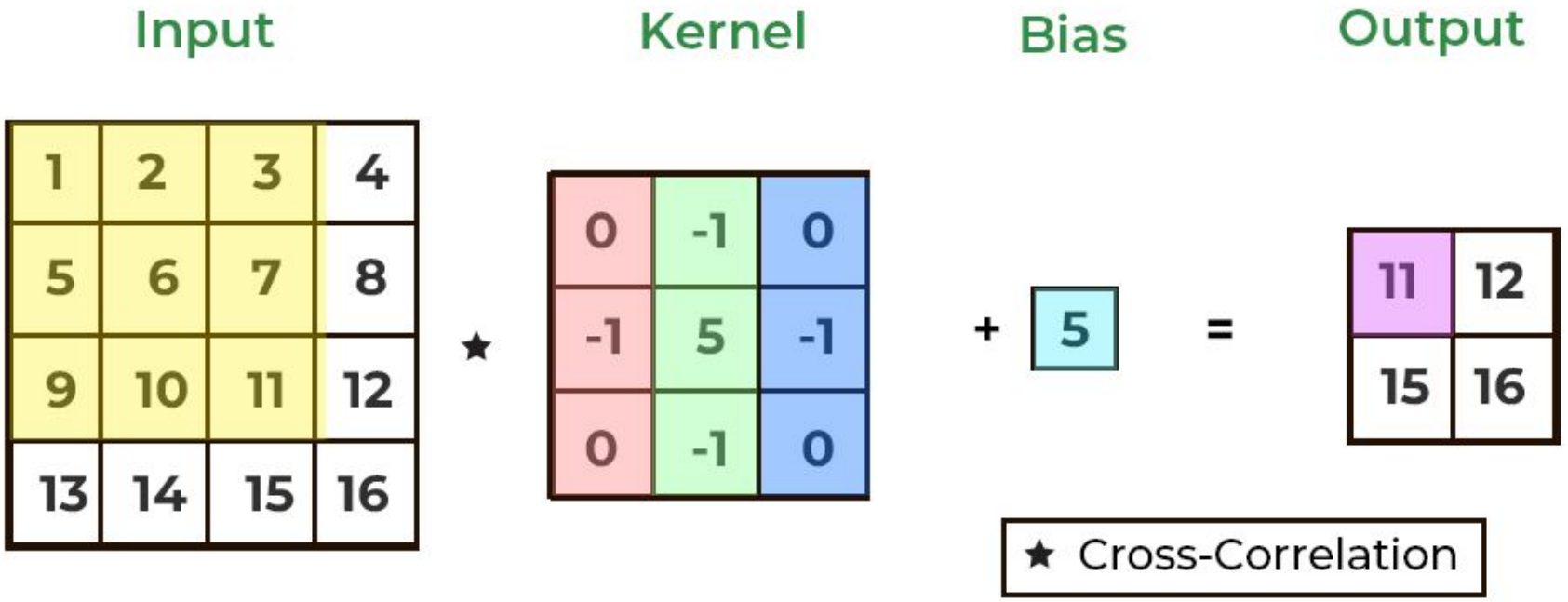
3. Convolutional Operation



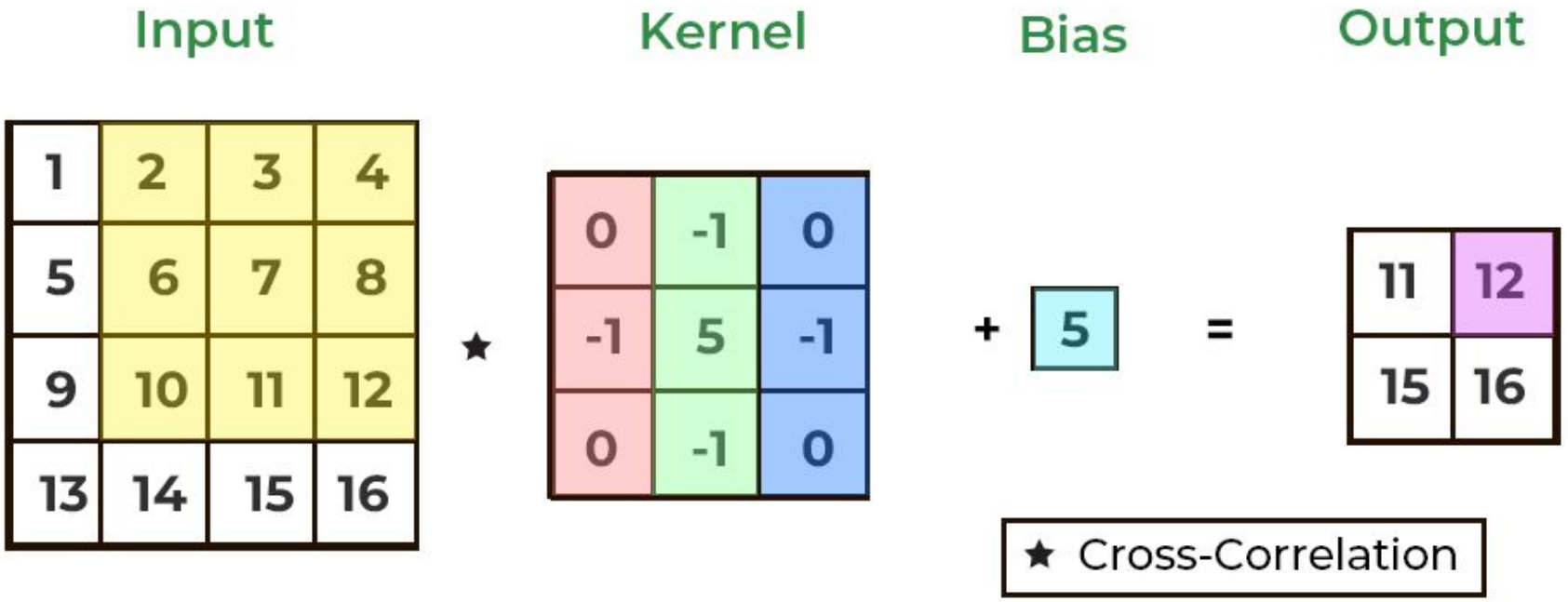
Total Parameter:

- **Neural Network: 50M**
- **CNNs: 10M**

3. Convolutional Operation

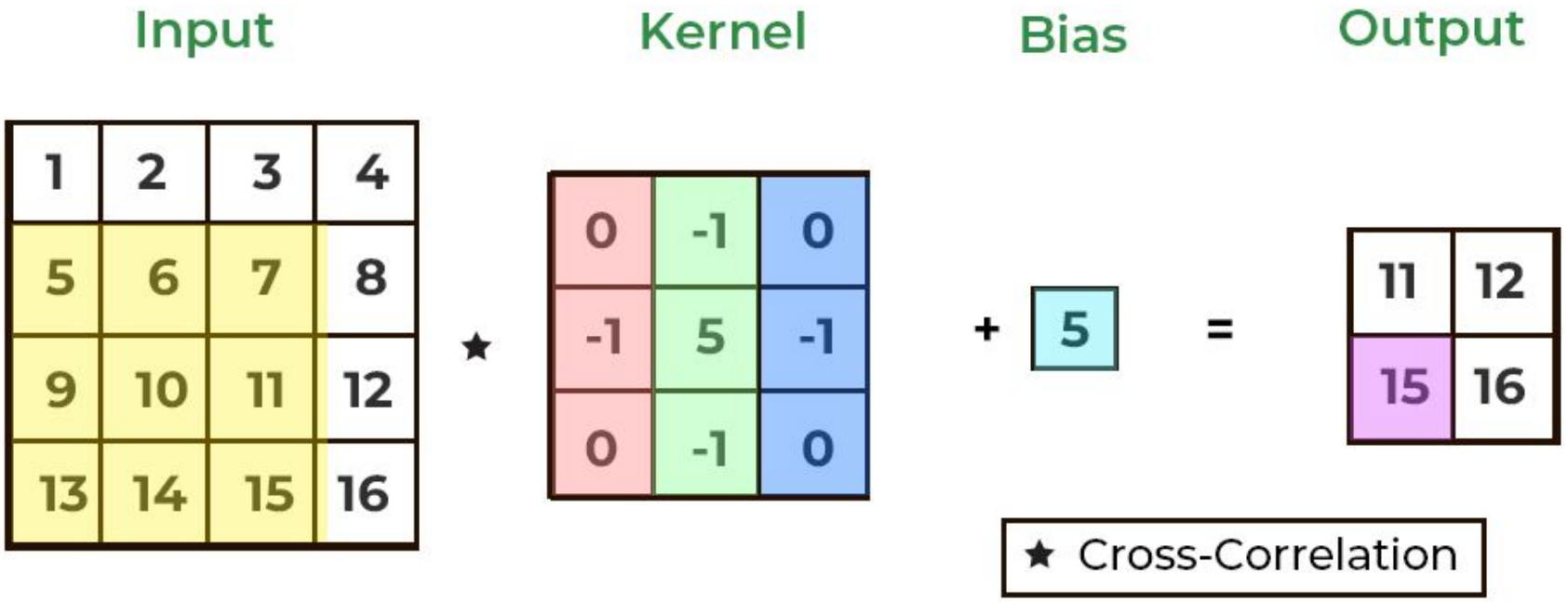


1 X 0 + 5 X -1 + 9 X 0 + 2 X -1 + 6 X 5 + 10 X -1 + 3 X 0 + 7 X -1 + 11 X 0 + 5 = 11



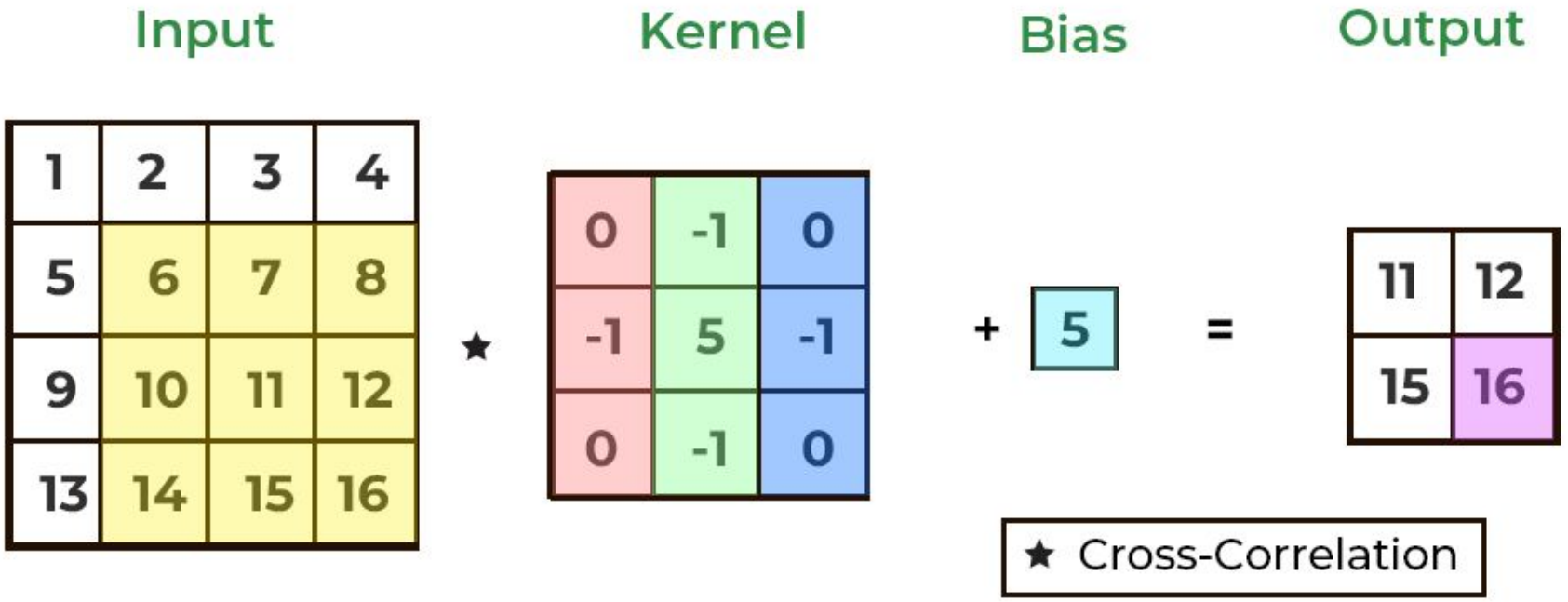
2 X 0 + 6 X -1 + 10 X 0 + 3 X -1 + 7 X 5 + 11 X -1 + 4 X 0 + 8 X-1 + 12 X 0 + 5 = 12

3. Convolutional Operation



5 X 0 + 9 X -1 + 13 X 0 + 6 X -1 + 10 X 5 + 14 X -1 + 7 X 0 + 11 X -1 + 15 X 0 + 5 = 15

3. Convolutional Operation



6 X 0 + 10 X -1 + 14 X 0 + 7 X -1 + 11 X 5 + 15 X -1 + 8X0 + 12X-1 + 16X0 + 5 = 16

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

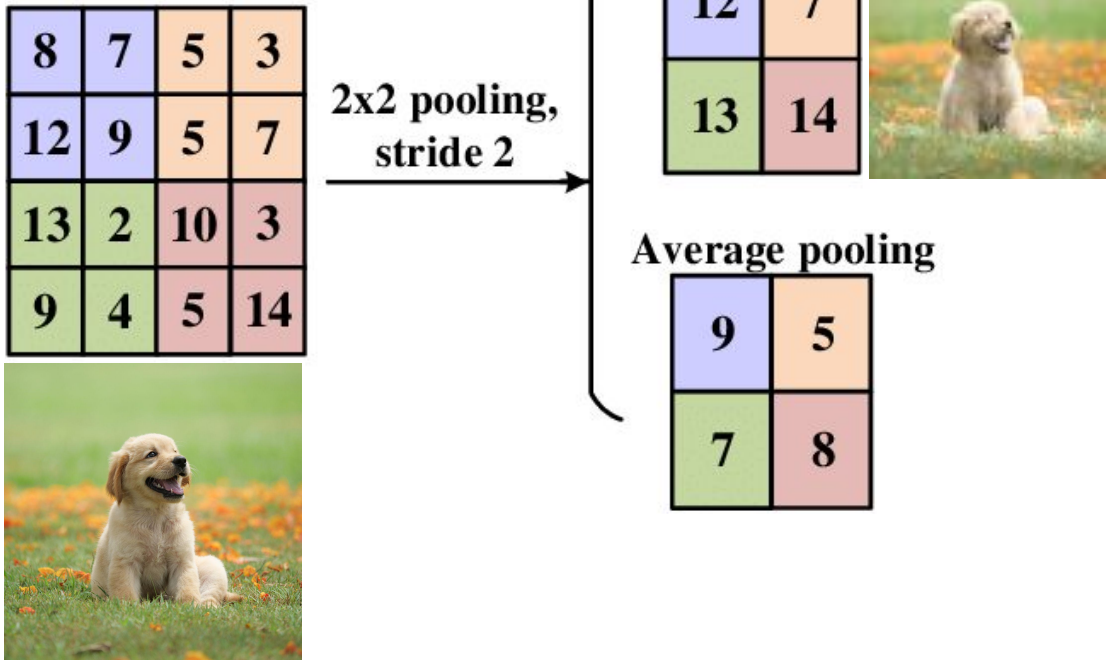
$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

source: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

N: Số lượng sample
C_in/out : Số lượng Channel ngõ vào/ra
H, W: Lần lượt là Width và Height

Padding: Kích thước phần được thêm vào (Theo chiều Width và chiều Height)
Stride: Kích thước trượt của Kernel (Theo chiều Width và chiều Height)
Kernel_size: Kích thước của Kernel

Pooling



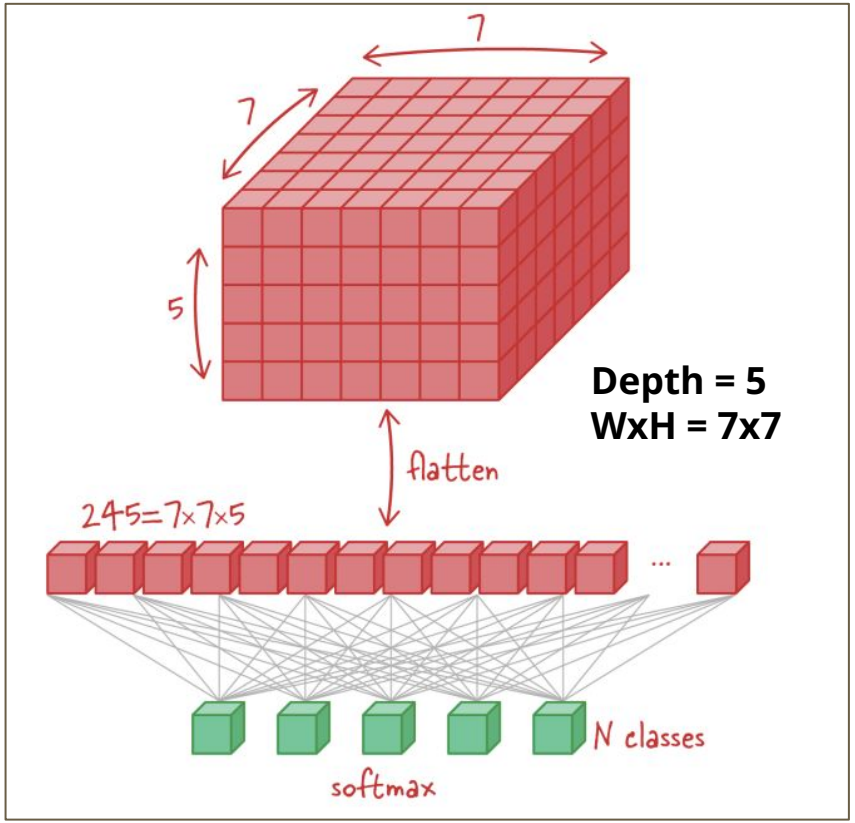
Giảm kích thước không giảm : Giảm số lượng tham số và phép tính, giúp mô hình nhanh và ít bộ nhớ hơn.

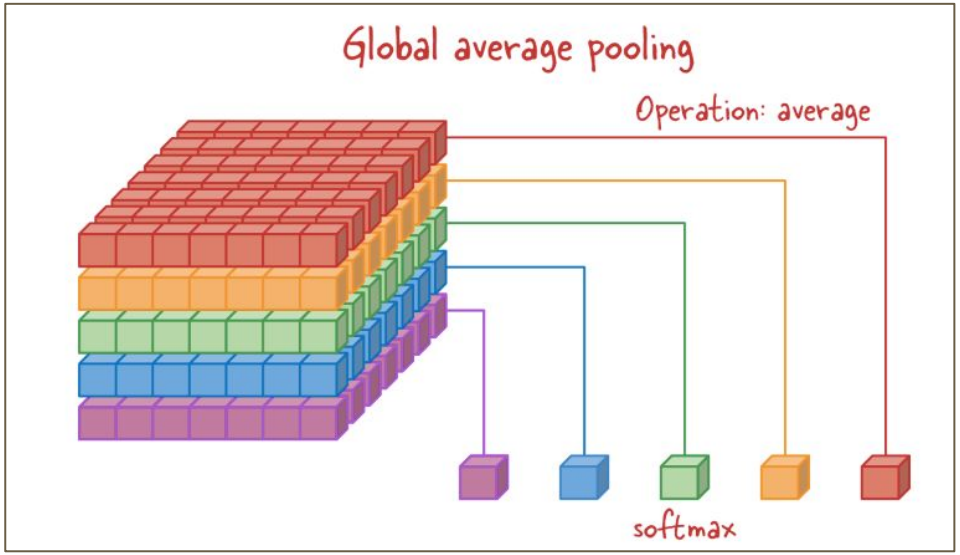
Trích xuất đặc trưng cục bộ

Tăng tính phi tuyến: Giúp thêm một chút tính phi tuyến vào mô hình.

Mất thông tin: Mỗi lần pooling mất một số thông tin. Nếu dùng MaxPooling thì chỉ giữ những pixel có giá trị max, còn AveragePooling thì giữ lại giá trị trung bình.

Flatten and Global Average Pooling (GAP)

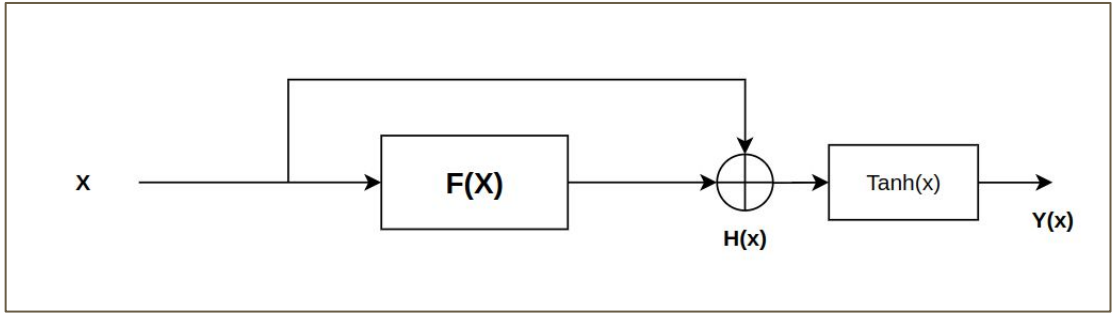




Flatten thường dùng cho mạng CNN ít lớp (LeNet, AlexNet)

GAP thường dùng cho mạng CNN hiện đại như ResNet

Skip Connection (Residual Connection)

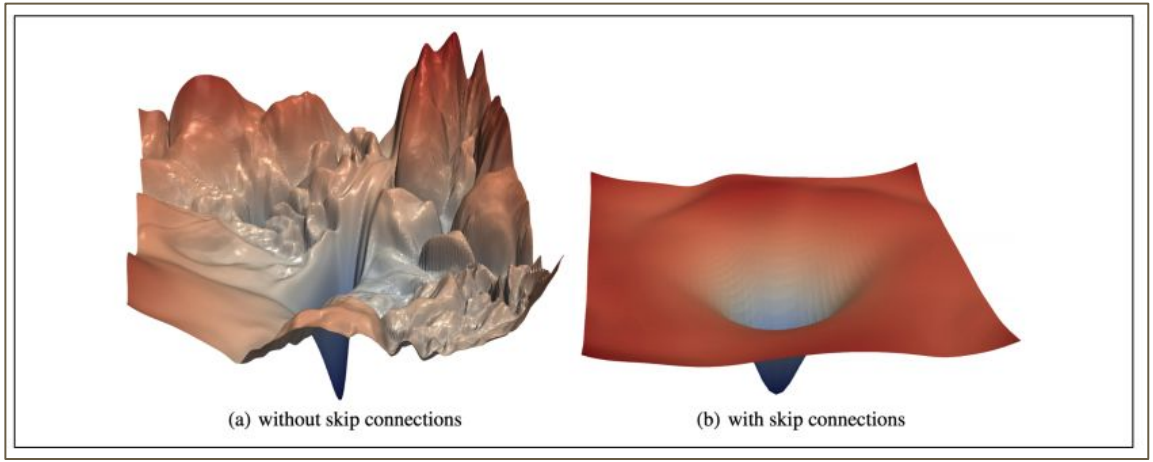


Giảm Vanishing gradient

Cho phép mạng sâu hơn: Minh chứng cho thấy mạng ResNet thường có số lớp sâu nhưng vẫn hiệu quả.

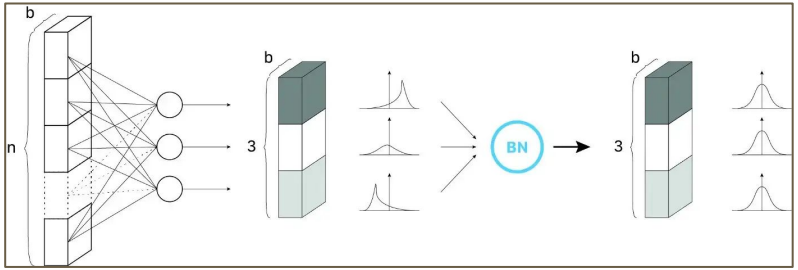
Khả năng hội tụ nhanh hơn: Trên thực nghiệm cho thấy mô hình có skip connection sẽ hội tụ nhanh hơn.

$$H(x) = F(x) + x$$
$$Y(x) = \tanh(H(x))$$
$$\frac{dY}{dx} = \frac{dY}{dH} \cdot \frac{dH}{dx}$$
$$\frac{d\tanh(z)}{dz} = 1 - \tanh^2(z)$$
$$\frac{dY}{dH} = 1 - \tanh^2(H(x))$$
$$\frac{dH}{dx} = \frac{dF}{dx} + 1$$
$$\frac{dY}{dx} = (1 - \tanh^2(H(x))) \cdot (\frac{dF}{dx} + 1)$$



Batch Normalization

- Thường đặt ở sau hàm kích hoạt để chuẩn hóa đưa phổ dữ liệu về vùng bão hòa. Giúp **giảm Vanishing/ Exploding Gradient.**
- **Đồng nhất phân phối:** Giúp việc huấn luyện nhanh và ổn định hơn.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

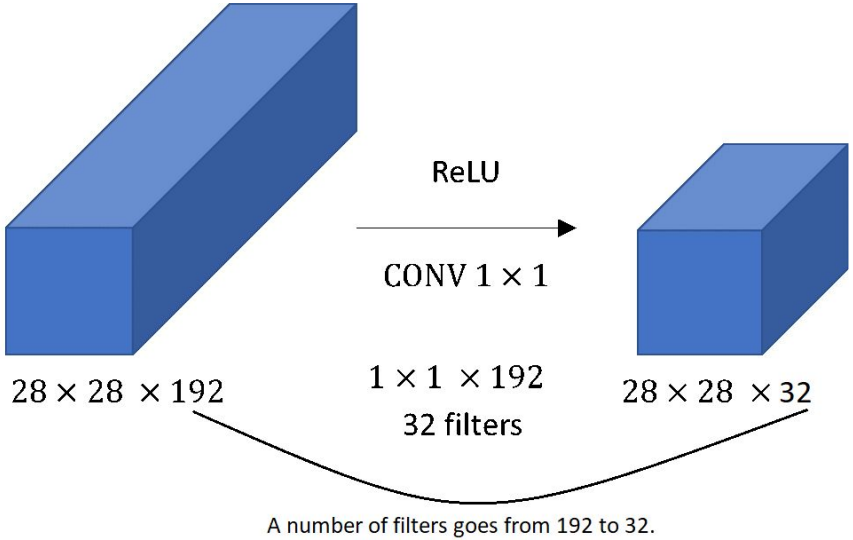
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

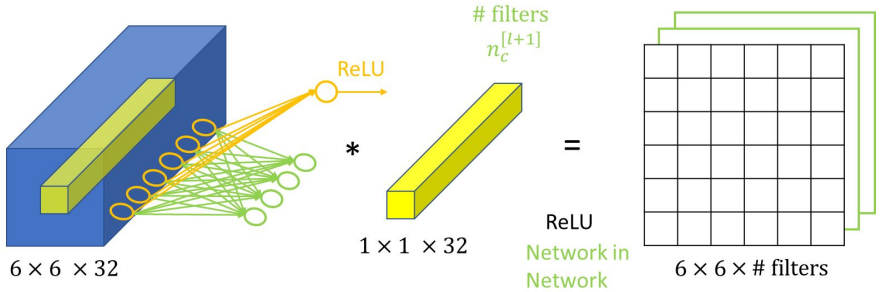
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

1x1 Convolutions (Pointwise Convolutions)





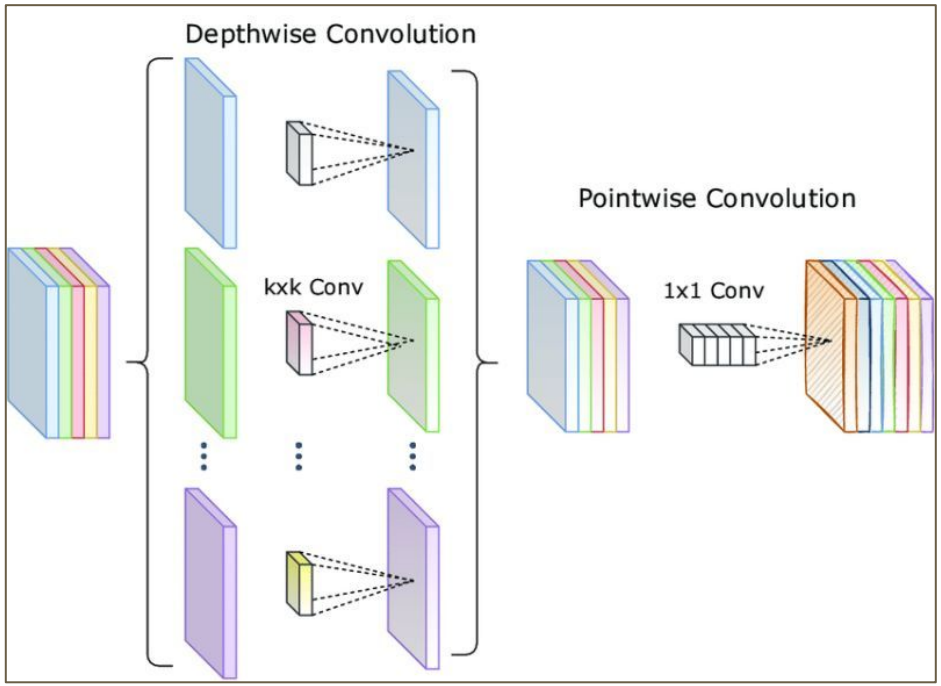
Thường dùng để **Giảm/Tăng chiều** của một feature map mà không thay đổi chiều cao, chiều rộng.

- **Tính toán phi không gian:** Giúp mô hình có sự học hiểu giữa các kênh của tích chập.

- **Giảm chi phí tính toán:** Thường đặt 1x1 Conv trước khi vào 3x3 hoặc 5x5 Conv , giúp giảm số lượng tham số tính toán.

- Tăng cường tính phi tuyến: Giúp mô hình có những biểu diễn phức tạp hơn.

Depthwise Convolutions



Depth-Separable Convolutions

Giảm số lượng tham số: Mỗi kênh đầu vào có một kernel riêng, nên số lượng tham số giảm đáng kể.

Giảm tính toán: Số lượng tính toán chỉ phụ thuộc vào số lượng channel.

Tích hợp thông tin đặc trưng và không gian (Nếu dùng theo sau là Pointwise Convolutions)

Không thể học mối quan hệ phức tạp như CNN truyền thống.

Các architecture chứa loại này: **Xception**, **MobileNet**, **EfficientNet**, **SqueezeNet**

CONV:

Parameters:

(5x5x32 + 1)x64 = 51.264

Computational Complexity:

(126x126)x(5x5x32)x64 = 812.851.200 (Phép tính)

Depth-Separable Convolutions:

Parameters:

w1 = 5x5x32 = 800

w2 = (1x1x32 +1)x64 = 2.112

w1 + w2 = 2.912

Computational Complexity:

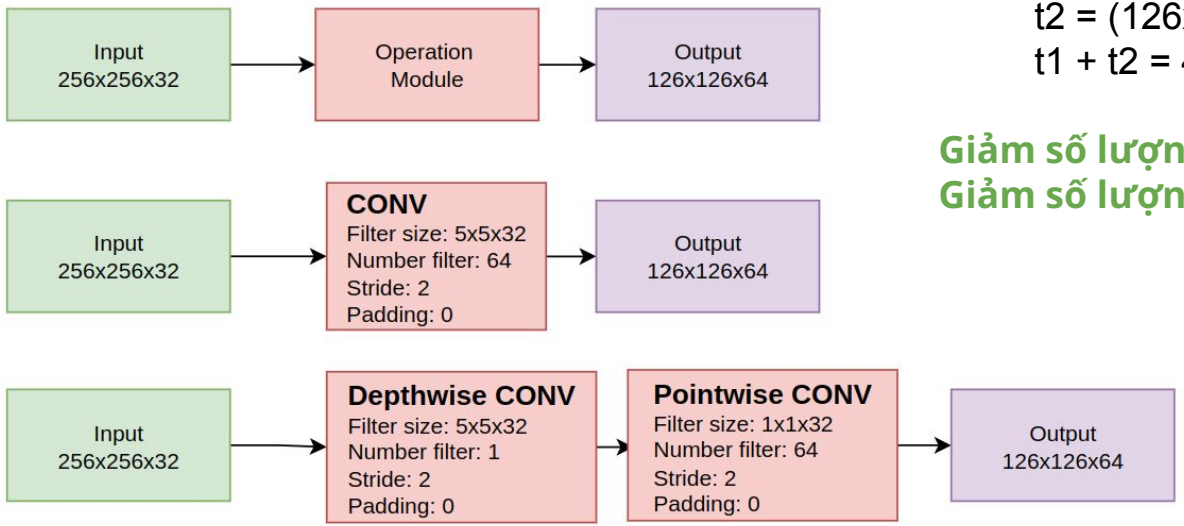
t1 = (126x126)x(5x5x32)x1 = 12.700.800

t2 = (126x126)x(1x1x32)x64 = 32.514.048

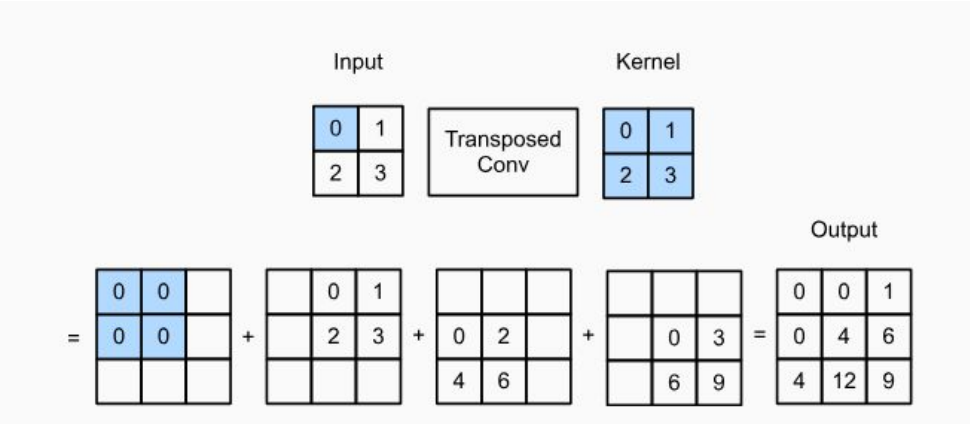
t1 + t2 = 45.214.848 (Phép tính)

Giảm số lượng tính toán đi: 17.9 lần

Giảm số lượng tham số đi: 17.6 lần



Transposed Convolution




```
import torch
from torch import nn
X = torch.tensor([[0.0, 1.0], [2.0, 3.0]])
K = torch.tensor([[0.0, 1.0], [2.0, 3.0]])

X, K = X.reshape(1, 1, 2, 2), K.reshape(1, 1, 2, 2)
tconv = nn.ConvTranspose2d(1, 1, kernel_size=2, bias=False)
tconv.weight.data = K
tconv(X)
```

[8] ✓ 0.0s

```
tensor([[[[ 0.,  0.,  1.],
          [ 0.,  4.,  6.],
          [ 4., 12.,  9.]]]], grad_fn=<ConvolutionBackward0>)
```

Thường dùng trong Image Segmentation , Generative Adversarial Networks (GANs) và Super Resolution.

- **Khôi phục thông tin:** Do các ConV thường làm mất đi thông tin, thì TConv sẽ là khôi phục thông tin của feature đã mất.

- Phân đoạn hình ảnh: Nó là bước trung gian từ 1 pixel tăng lên rồi giảm về lại kích thước ban đầu, từ đó đưa ra dự đoán mức độ pixel

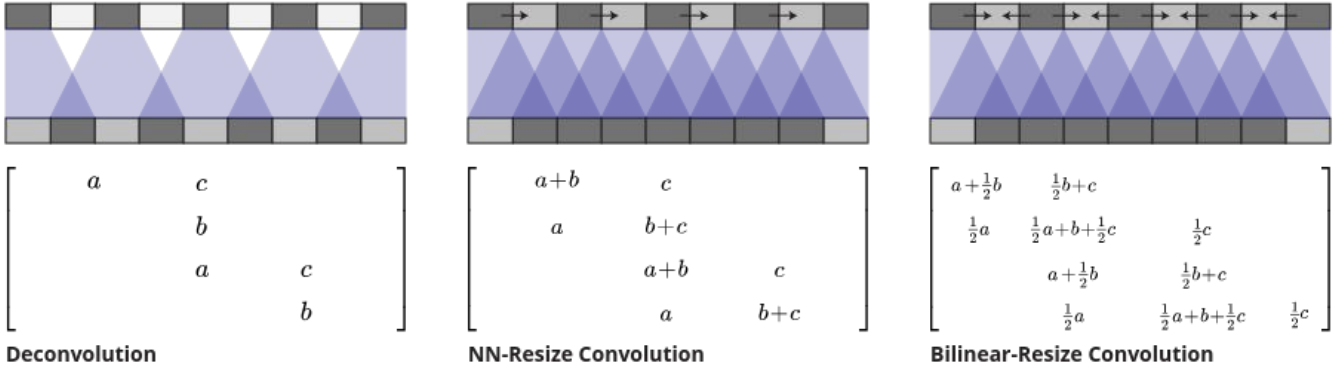
- **Tăng kích thước feature map đầu ra.**

Vấn đề checkerboard pattern cần lưu ý.

Checkerboard pattern

Xảy ra do việc sử dụng stride lớn trong TConV. Khi tính toán output nhiều vùng được cộng chồng lên nhau.

Solution:
- Thường dùng Kernel là số chẵn.
- Kết hợp thêm một lớp ConV ngày sau khi TConv giúp làm “mịn" hình ảnh và giảm thiểu sự không đồng đều.



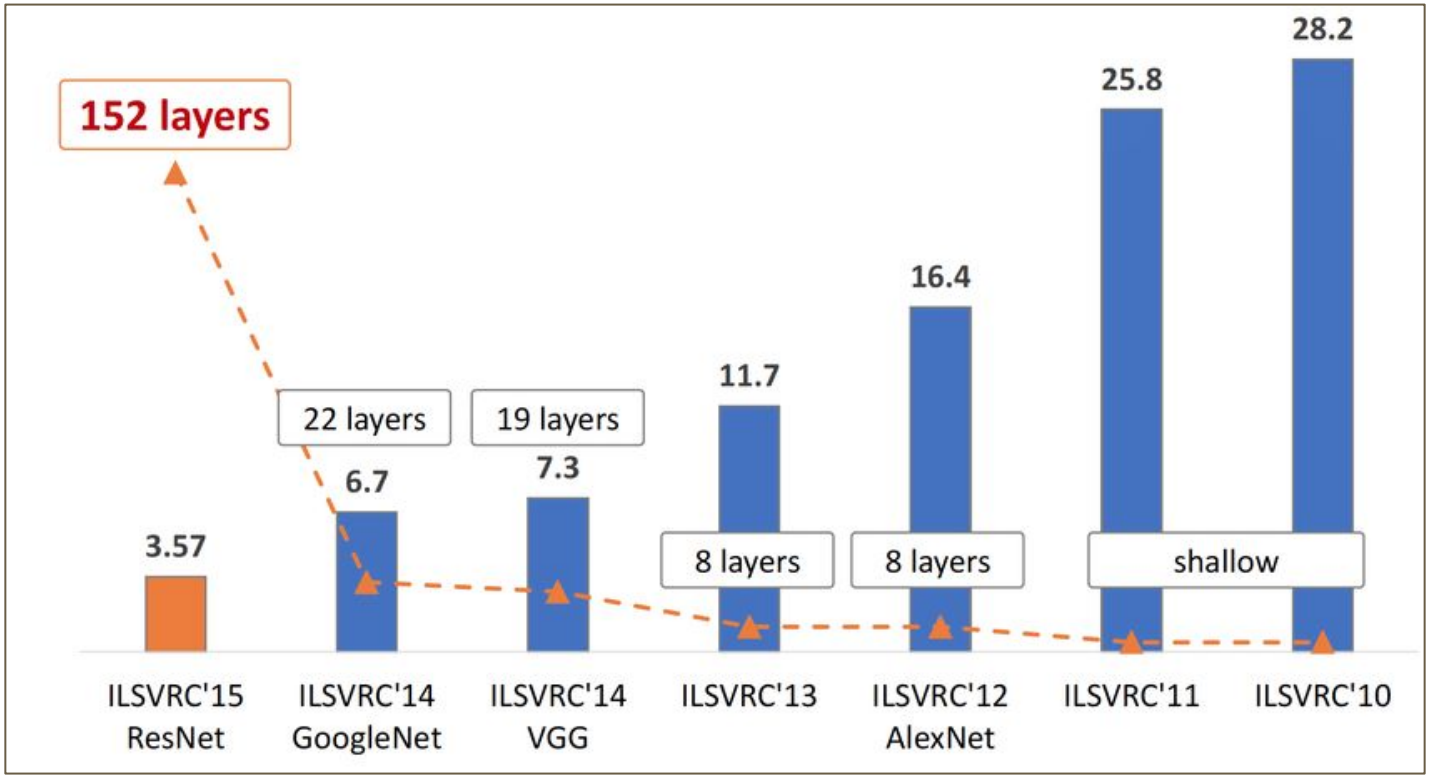
5. ImageNet Challenge

~14 million labeled images, 20k classes
Images gathered from Internet
Human labels via Amazon MTurk

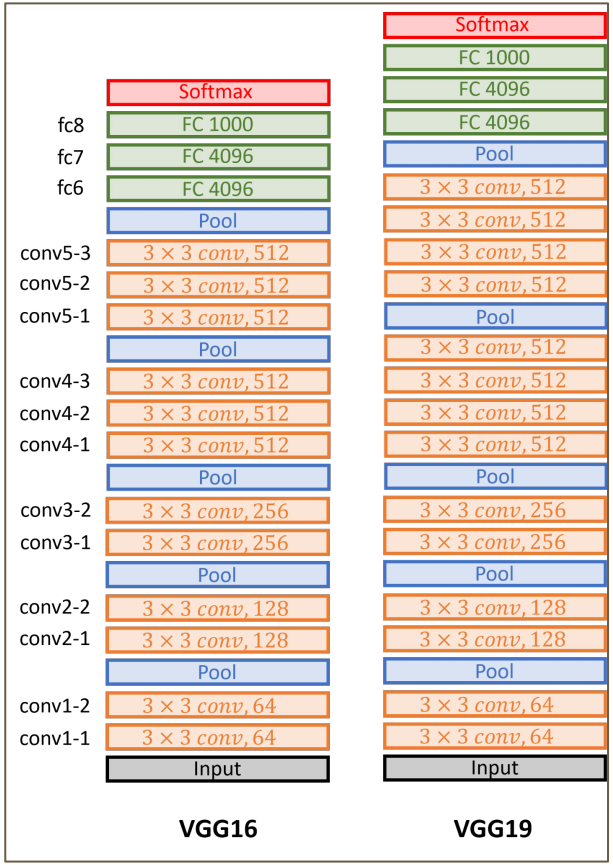
ImageNet Large-Scale Visual Recognition Challenge (ILSVRC): 1.2 million training images, 1000 classes

Link: www.image-net.org/challenges/LSVRC/

Winner in year 2015



VGG-19

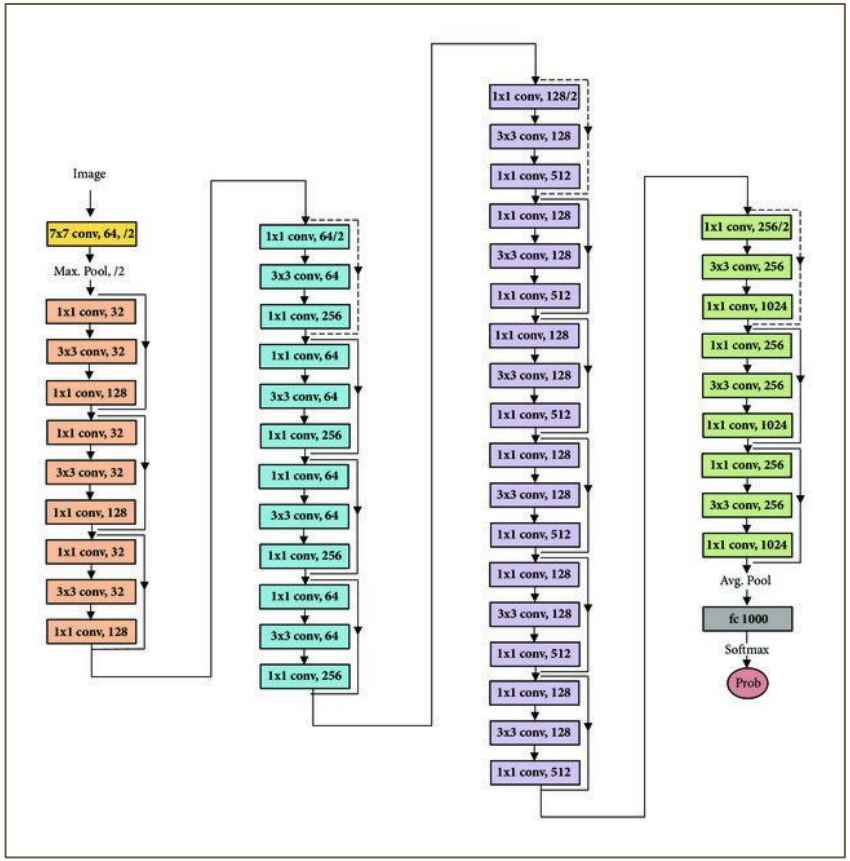


Model	Parameters (excl. classif. head[a])	ImageNet accuracy	104 flowers F1 score[b] (fine-tuning)	104 flowers F1 score (trained from scratch)
VGG19	20M	71%	88% precision: 89%, recall: 88%	N/A[c]
Previous best for comparison:				
AlexNet	3.7M	60%		39% precision: 44%, recall: 38%

- Dễ hiểu và triển khai

- **Số lượng tham số lớn** (Do lạm dụng nhiều Convolutional và Pooling) dẫn đến train và dự đoán lâu
- **Vẫn có thể bị vanishing gradient**
- **Dễ bị overfit với tập huấn luyện nhỏ** do số lượng tham số lớn.

ResNet-50



Model	Parameters (excl. classif. head[a])	ImageNet accuracy	104 flowers F1 score[b] (fine-tuning)	104 flowers F1 score (trained from scratch)
ResNet50	23M	75%	94% prec.: 95%, recall: 94%	73% prec.: 76%, recall: 72%
Previous best for comparison:				
InceptionV3	22M	78%	95% prec.: 95%, recall: 94%	
SqueezeNet, 24 layers	2.7M			76% prec.: 77%, recall: 75%

- **Mạng sâu và hiệu quả:** Nhờ sử dụng **Residual blocks** giúp không bị vanishing gradient.
- **Mạng sâu hơn ít tham số** (so với VGG): Nhờ việc sử dụng nhiều Convolutional 1x1.

- **Số lượng tham số lớn** (Do lạm dụng nhiều CNN và Pooling) dẫn đến train và dự đoán lâu
- **Dễ bị overfit với tập huấn luyện nhỏ** do số lượng tham số lớn.

7. CNN Applications

Applications of CNN


Computer Vision and related application

Natural Language Processing


Object Detection and Segmentation

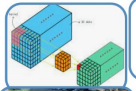

Image Classification


Speech Recognition


Video Processing


Low-resolution images


Limited Resource system


CNN for various dimensional data


Object Counting