AI VIET NAM
@aivietnam.edu.vn

AI VIET NAM – AI COURSE 2025

CNNs: Step-by-Step Examples

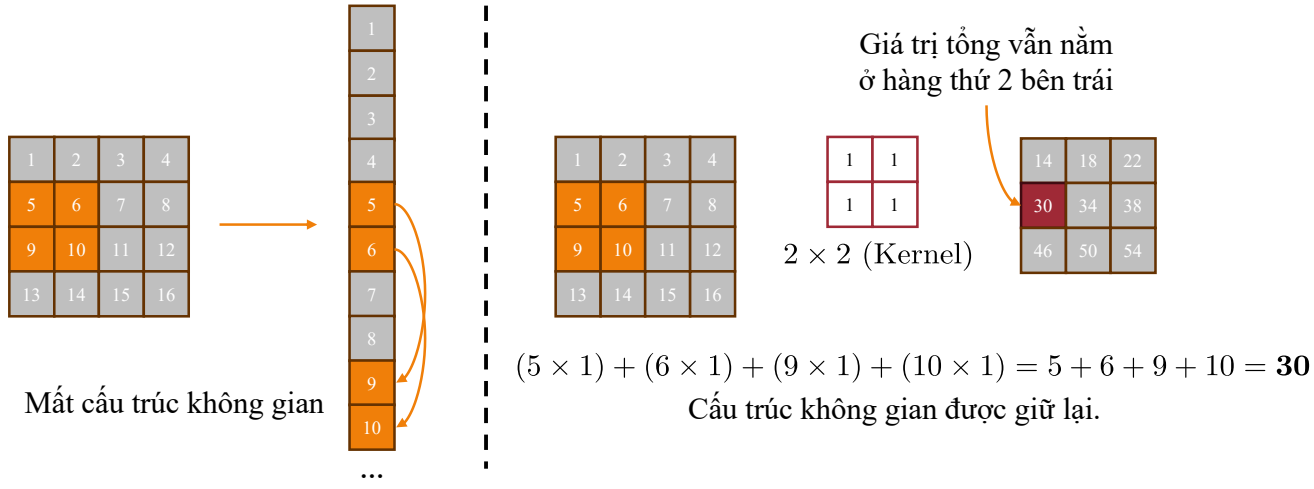
Nguyễn Phúc Thịnh và Đinh Quang Vinh

I. Giới thiệu

Trong lĩnh vực thị giác máy tính, việc xử lý hình ảnh đòi hỏi máy tính không chỉ nhận biết được giá trị của từng điểm ảnh (pixel) mà còn phải hiểu được mối quan hệ không gian giữa chúng. Trước khi sự ra đời của Convolutional Neural Networks (CNN), các mạng nơ-ron truyền thống (Fully Connected Networks - FC) thường gặp khó khăn lớn khi đối mặt với dữ liệu hình ảnh.

Vấn đề cốt lõi nằm ở việc mạng FC yêu cầu đầu vào phải được "duỗi phẳng" (flatten) thành một vector một chiều. Quá trình này vô tình phá vỡ cấu trúc không gian 2D/3D tự nhiên của ảnh, làm mất đi thông tin quan trọng về hình dạng và vị trí tương đối của các vật thể. Hơn nữa, với một bức ảnh màu có kích thước trung bình (ví dụ $200 \times 200 \times 3$), số lượng tham số cần huấn luyện trong mạng FC sẽ bùng nổ lên con số hàng triệu, dẫn đến chi phí tính toán khổng lồ và nguy cơ Overfitting rất cao.

Để giải quyết bài toán này, mạng CNN ra đời với tư duy mô phỏng cơ chế thị giác của con người, tập trung vào việc trích xuất các đặc trưng cục bộ (local features) như cạnh, góc, và mảng màu thông qua các bộ lọc (filters).

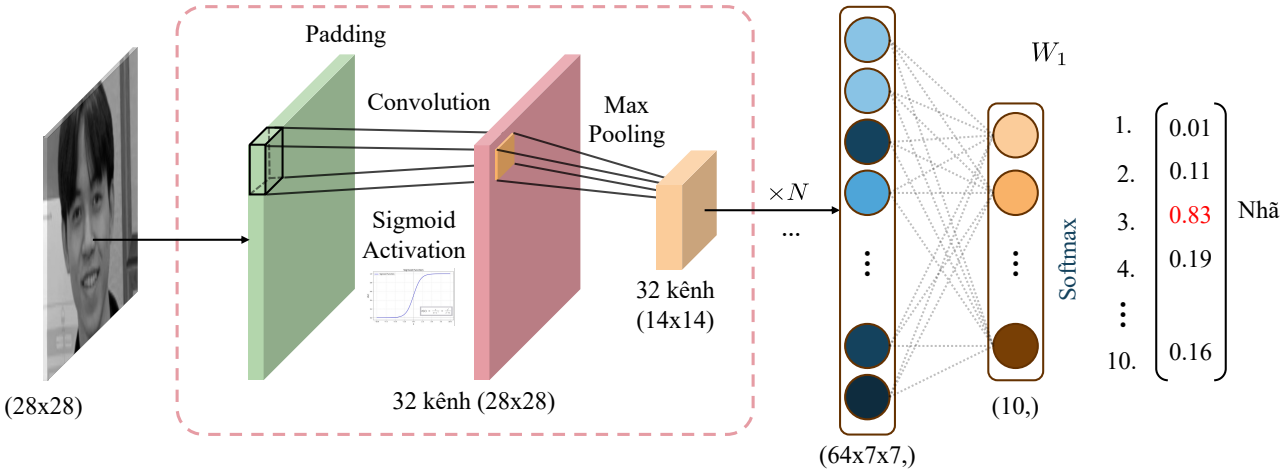


Hình 1: Sự khác biệt trong việc xử lý dữ liệu: FC Network phá vỡ cấu trúc không gian (trái) so với CNN bảo toàn cấu trúc ảnh (phải).

Trong bài đọc này, chúng ta sẽ không sử dụng các thư viện Deep Learning cấp cao (như PyTorch hay TensorFlow) ngay lập tức. Thay vào đó, chúng ta sẽ đi sâu vào "hộp đen" của CNN

bằng cách tính toán thủ công từng phép toán cốt lõi. Việc này giúp ta hiểu rõ bản chất toán học đằng sau cách máy tính "nhìn" một bức ảnh.

Chúng ta sẽ bắt đầu với khái niệm nền tảng là phép tích chập, tìm hiểu cách các tham số như Stride và Padding ảnh hưởng đến kích thước đầu ra. Tiếp theo, ta sẽ khám phá kỹ thuật Pooling để giảm chiều dữ liệu nhưng vẫn giữ lại đặc trưng quan trọng, và cuối cùng là kết nối tất cả thông qua lớp Flatten để đưa vào mô hình phân loại. Để đơn giản và các bạn có thể làm theo, bài đọc này sẽ sử dụng ảnh xám (số channel là 1), và sau đó sẽ mở rộng ra ảnh màu ở cuối bài.



Hình 2: Luồng xử lý dữ liệu qua các tầng của một mạng CNN cơ bản.

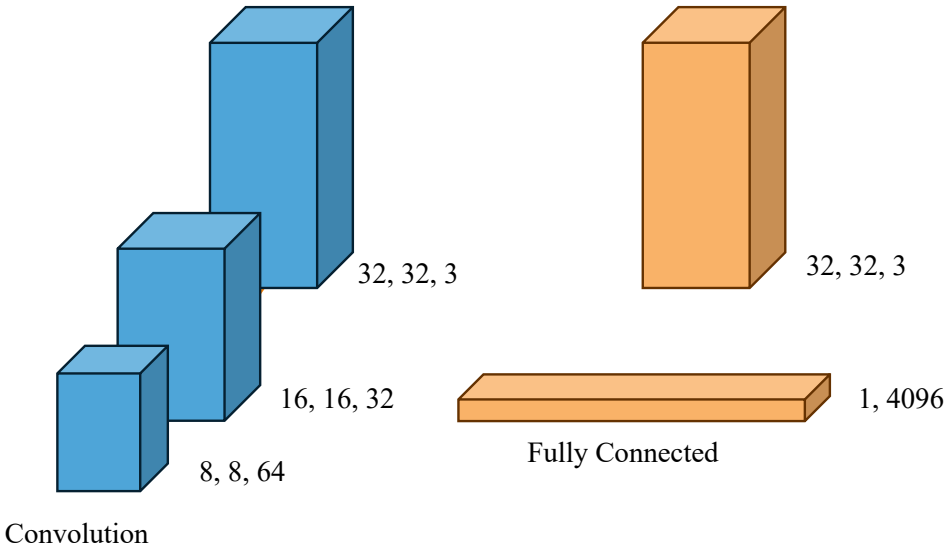
Mục lục

II. Tại sao cần mạng CNN cho hình ảnh?

Trước khi đi sâu vào các phép toán, ta cần hiểu lý do tại sao các mạng nơ-ron truyền thống FC lại gặp khó khăn khi xử lý dữ liệu hình ảnh, dẫn đến sự ra đời của CNN. Vấn đề lớn nhất của mạng FC là sự bùng nổ số lượng tham số và việc làm mất đi cấu trúc không gian của ảnh.

- Mất cấu trúc không gian (Spatial Structure): Để đưa một bức ảnh vào mạng FC, ta phải "duỗi phẳng" (flatten) bức ảnh đó thành một vector 1 chiều. Ví dụ, một bức ảnh 2D kích thước 28×28 sẽ trở thành vector 784 phần tử. Việc này làm mất đi mối quan hệ tương quan giữa các điểm ảnh lân cận (ví dụ: mắt thường nằm cạnh mũi).
- Số lượng tham số khổng lồ: Với một bức ảnh màu nhỏ kích thước $200 \times 200 \times 3$, ta có 120,000 điểm ảnh đầu vào. Nếu lớp ẩn tiếp theo có 1000 nơ-ron, số lượng trọng số W cần học sẽ là $120,000 \times 1000 = 120$ triệu tham số. Điều này dẫn đến chi phí tính toán cực lớn và dễ gây ra hiện tượng overfitting.

Mạng CNN giải quyết vấn đề này thông qua cơ chế "chia sẻ tham số" (parameter sharing) và kết nối cục bộ, giúp giữ nguyên cấu trúc không gian 3D của ảnh (chiều cao, chiều rộng, độ sâu).



Hình 3: So sánh cấu trúc dữ liệu giữa mạng CNN (giữ nguyên không gian 3D) và mạng FC (duỗi phẳng thành 1D).

III. Phép tích chập

Cốt lõi của mạng CNN là phép tích chập. Mặc dù tên gọi nghe có vẻ phức tạp, nhưng về bản chất toán học, đây là một phép tính đơn giản dựa trên việc trượt một ma trận nhỏ đè lên một ma trận lớn.

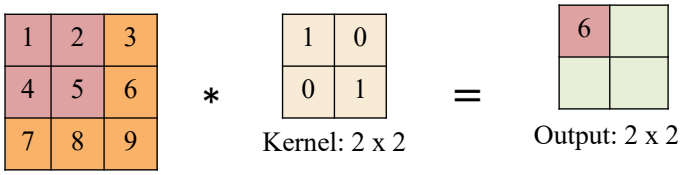
Trong xử lý ảnh, ta sử dụng một ma trận nhỏ gọi là Kernel (hoặc Filter) đóng vai trò như một cửa sổ trượt (sliding window) quét qua toàn bộ bức ảnh đầu vào. Tại mỗi vị trí mà Kernel ghé qua, ta thực hiện phép nhân từng phần tử giữa Kernel và vùng ảnh tương ứng, sau đó cộng tổng lại để tạo ra một điểm ảnh mới trên Feature Map.

Giả sử ta có:

- Input I: Ma trận 3×3
- Kernel K: Ma trận 2×2 được thiết kế để phát hiện một đặc trưng cụ thể.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Ta thực hiện tính toán cho vị trí đầu tiên (góc trái trên):



$$Output_{0,0} = (1 \times 1) + (2 \times 0) + (4 \times 0) + (5 \times 1)$$
$$= 1 + 0 + 0 + 5 = 6$$

Hình 4: Minh họa cách tính phép tích chập. Quá trình nhân và cộng này (dot product) lặp lại cho mỗi "bước nhảy" của Kernel khi trượt sang phải và xuống dưới.

Kết quả của phép tính này không chỉ là những con số vô tri. Giá trị của Kernel được thiết kế đặc biệt để **làm nổi bật** các đặc trưng quan trọng của ảnh như cạnh (edges), góc (corners), hoặc các vân bề mặt (textures). Những vùng ảnh khớp với mẫu của Kernel sẽ cho ra giá trị lớn trên Feature Map, và ngược lại.

Ví dụ Code PyTorch

```python
import torch
import torch.nn as nn

# Định nghĩa input và kernel như ví dụ trên
input_img = torch.tensor([[[[1., 2., 3.],
                            [4., 5., 6.],
                            [7., 8., 9.]]]])
conv = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=2, bias=False)

# Gán giá trị kernel thủ công để kiểm tra
conv.weight.data = torch.tensor([[[[1., 0.],
                                   [0., 1.]]]])

output = conv(input_img)
print(output)
# Kết quả mong đợi: [[ 6., 8.], [12., 14.]]
```

Không chỉ giới hạn ở hình ảnh 2D, phép tích chập là một công cụ đa năng trong Deep Learning. Ta có thể thực hiện tích chập trên dữ liệu 1D (ví dụ: tín hiệu âm thanh, chuỗi văn bản), 2D (hình ảnh) và thậm chí là 3D (video, ảnh y tế CT/MRI). Ngoài ra, tích chập thường được dùng để downsampling (giảm độ phân giải để cô đọng thông tin). Ngược lại, một biến thể đặc biệt của nó (Transposed Convolution) rất hữu ích cho việc upsampling (tăng độ phân giải) trong các bài toán sinh ảnh.

Lưu ý quan trọng

Mục tiêu cốt lõi của việc sử dụng Convolution trong Deep Learning không phải để trực tiếp dự đoán kết quả. Nhiệm vụ của các lớp Convolution là trích xuất đặc trưng - biến đổi các điểm ảnh thô thành các thông tin có ý nghĩa (như mắt, mũi, miệng). Các đặc trưng này sau đó mới được đưa vào các lớp mạng nơ-ron truyền thống (Feed-Forward Networks - FFNs) để thực hiện việc dự đoán cuối cùng.

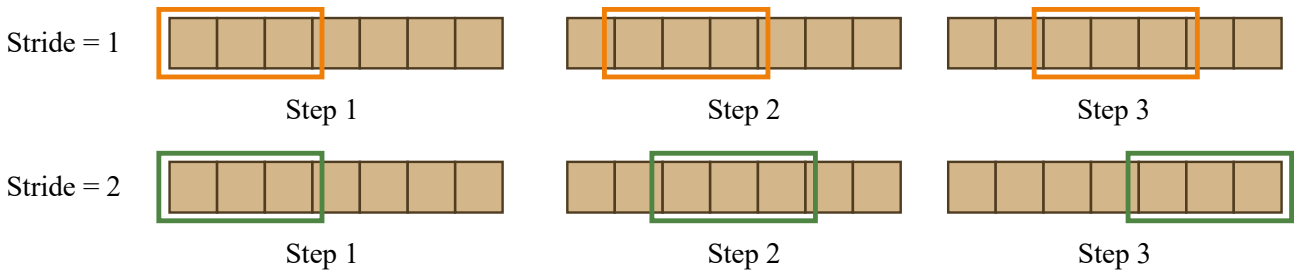
IV. Stride là gì?

Stride (bước trượt) là thông số quy định khoảng cách mà Kernel di chuyển mỗi lần trượt trên ảnh đầu vào.

- Stride = 1: Kernel trượt từng ô một (pixel-by-pixel). Đây là thiết lập tiêu chuẩn để giữ lại nhiều thông tin không gian nhất.
- Stride = 2: Kernel nhảy cóc 2 ô mỗi lần. Việc này làm giảm kích thước (chiều rộng và cao) của Feature Map đầu ra đi một nửa, giúp giảm chi phí tính toán.

Ví dụ: Với Input 5×5, Kernel 3×3, Padding 0:

- Nếu $S = 1$: $O = (5 - 3)/1 + 1 = 3 \rightarrow$ Output 3×3.
- Nếu $S = 2$: $O = (5 - 3)/2 + 1 = 2 \rightarrow$ Output 2×2.



Hình 5: Sự khác biệt về vùng trượt và kích thước đầu ra khi sử dụng Stride 1 so với Stride 2.

Thông thường, ta hay gặp Stride là một số nguyên duy nhất (ví dụ: $S = 1$ hoặc $S = 2$). Khi đó, Kernel sẽ trượt với khoảng cách bằng nhau trên cả hai chiều dọc (chiều cao) và ngang (chiều rộng). Tuy nhiên, trong các bài toán đặc thù, ta hoàn toàn có thể thiết lập Stride khác nhau cho mỗi trục.

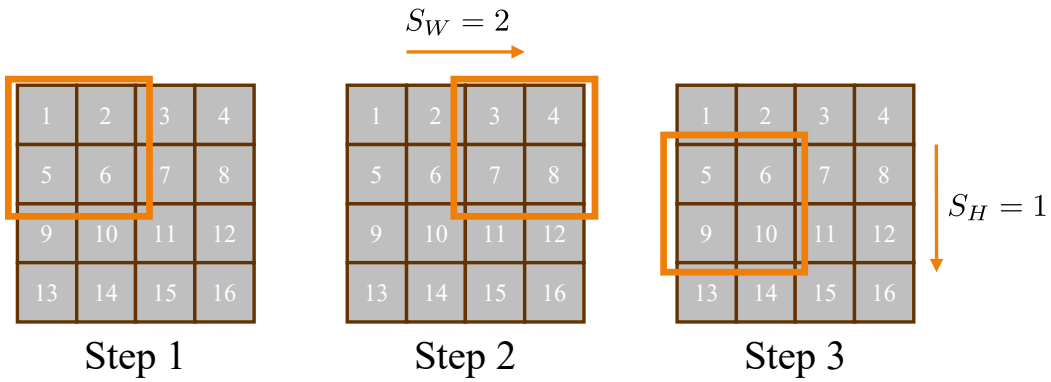
Ký hiệu Stride dưới dạng một cặp số (S_H, S_W), trong đó:

- S_H: Bước trượt theo trục dọc (Height - di chuyển xuống dưới).
- S_W: Bước trượt theo trục ngang (Width - di chuyển sang phải).

Công thức tính kích thước đầu ra tổng quát cho từng chiều sẽ là:

$$H_{out} = \left\lfloor \frac{H_{in} + 2P_H - K_H}{S_H} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2P_W - K_W}{S_W} + 1 \right\rfloor$$



Hình 6: Minh họa Stride = (1, 2). Kernel di chuyển từng ô một theo chiều dọc, nhưng nhảy cóc 2 ô theo chiều ngang.

Cấu hình Stride đa chiều trong PyTorch

```python
import torch
import torch.nn as nn

# Input: 1 ảnh, 1 kênh màu, kích thước 6x6
input_img = torch.randn(1, 1, 6, 6)

# Thiết lập Stride=(1, 2)
# - Trục dọc (Height): Trượt 1 ô
# - Trục ngang (Width): Trượt 2 ô
conv = nn.Conv2d(in_channels=1, out_channels=1,
                 kernel_size=3, stride=(1, 2))

output = conv(input_img)

# Kích thước đầu ra dự kiến:
# H_out = (6 - 3)/1 + 1 = 4
# W_out = (6 - 3)/2 + 1 = 2 (lấy phần nguyên của 1.5 + 1 = 2)
print("Kích thước đầu ra:", output.shape)
# Kết quả: torch.Size([1, 1, 4, 2])
```

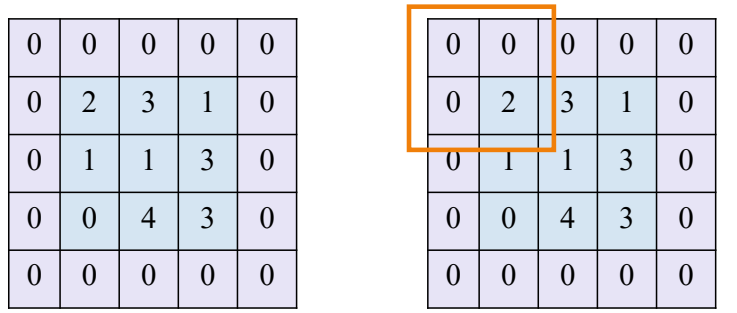
Mở rộng không gian 3D

Với dữ liệu video hoặc ảnh y tế (CT/MRI), ta sử dụng Convolution 3D. Khi đó Stride sẽ là bộ 3 số $(S_{Depth}, S_{Height}, S_{Width})$. Nguyên lý hoạt động hoàn toàn tương tự: Kernel sẽ "trượt" trong không gian 3 chiều theo các bước nhảy được quy định riêng cho từng chiều.

V. Padding

Khi thực hiện phép tích chập liên tục trên một bức ảnh mà không có biện pháp can thiệp, ta sẽ gặp phải hai vấn đề lớn. Thứ nhất là "sự thu hẹp kích thước", cứ qua mỗi lớp tích chập, kích thước Feature Map lại nhỏ dần đi, giới hạn độ sâu của mạng nơ-ron. Thứ hai là "mất mát thông tin các cạnh", các điểm ảnh ở góc và cạnh chỉ được Kernel quét qua ít lần hơn so với các điểm ảnh ở trung tâm, dẫn đến việc bỏ sót các đặc trưng quan trọng ở viền ảnh.

Để giải quyết vấn đề này, ta sử dụng kỹ thuật Padding - thêm các lớp giá trị (thường là số 0, gọi là Zero-padding) bao quanh bức ảnh đầu vào trước khi thực hiện tích chập.



Hình 7: Minh họa Zero-padding: Thêm một viền giá trị 0 (P=1) quanh ảnh Input 4x4 để giữ nguyên kích thước khi qua Kernel 3x3.

Dựa vào cách thiết lập Padding, ta thường chia làm hai loại chính trong các thư viện lập trình deep learning:

- Valid Padding (Không đệm): Đây là chế độ mặc định ($P = 0$). Ảnh đầu ra sẽ nhỏ hơn ảnh đầu vào. Chỉ những vùng mà Kernel phủ kín hoàn toàn mới được tính toán.
- Same Padding (Đệm giữ nguyên): Ta tính toán lượng Padding P cần thêm vào sao cho kích thước đầu ra bằng đúng kích thước đầu vào (với Stride $S = 1$). Điều này cực kỳ quan trọng trong các kiến trúc mạng sâu để duy trì cấu trúc không gian qua hàng chục lớp xử lý.

Công thức tổng quát tính kích thước đầu ra khi có Padding (P):

$$Output = \left\lfloor \frac{Input - Kernel + 2 \times Padding}{Stride} \right\rfloor + 1$$

Công thức này cho thấy kích thước đầu ra phụ thuộc vào kích thước đầu vào, kích thước kernel, lượng đệm và bước trượt.

VI. Pooling

Sau khi trích xuất đặc trưng bằng Convolution, ta thu được các Feature Map chứa thông tin chi tiết về vị trí của các đặc trưng (ví dụ: mắt ở toạ độ x, y). Tuy nhiên, trong nhiều bài toán nhận diện, ta chỉ cần biết "có mắt hay không" chứ không cần vị trí chính xác tuyệt đối. Quá nhiều chi tiết không gian sẽ làm tăng số lượng tính toán không cần thiết và dễ gây Overfitting.

Lớp Pooling ra đời nhằm mục đích giảm chiều dữ liệu, giúp mạng trở nên nhẹ hơn và bất biến với các dịch chuyển nhỏ (Translation Invariance). Khác với Convolution, lớp Pooling không có tham số trọng số nào để học, nó chỉ áp dụng một công thức cố định.

Có hai loại Pooling phổ biến nhất:

VI.1. Max Pooling

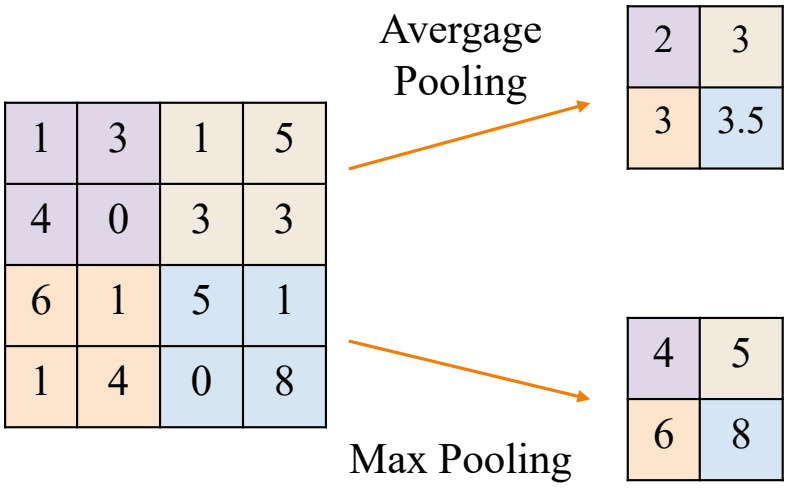
Max Pooling hoạt động bằng cách chia ảnh thành các vùng nhỏ và chỉ giữ lại **giá trị lớn nhất** trong vùng đó. Ý nghĩa của việc này là chỉ giữ lại đặc trưng nổi bật nhất (ví dụ: biên cạnh rõ nhất, màu sáng nhất) và loại bỏ các thông tin nhiễu xung quanh. Đây là phương pháp được sử dụng phổ biến nhất hiện nay.

VI.2. Average Pooling

Average Pooling tính **giá trị trung bình** của các phần tử trong vùng trượt. Phương pháp này có tác dụng làm mượt (smooth) hình ảnh và đặc trưng. Ngày nay, Average Pooling thường ít được dùng ở giữa mạng mà hay được dùng ở lớp cuối cùng (Global Average Pooling) để thay thế cho lớp Fully Connected.

Đặc điểm	Max Pooling	Average Pooling
Cơ chế	Lấy giá trị cực đại.	Tính trung bình cộng.
Tác dụng chính	Giữ lại đặc trưng nổi bật nhất.	Làm mượt đặc trưng, giữ lại thông tin nền.
Ứng dụng	Phổ biến trong hầu hết các mạng CNN hiện đại.	Thường dùng ở các tầng cuối hoặc mạng cổ điển (LeNet).

Bảng 1: So sánh Max Pooling và Average Pooling.



Hình 8: So sánh kết quả của Max Pooling và Average Pooling trên cùng một vùng dữ liệu.

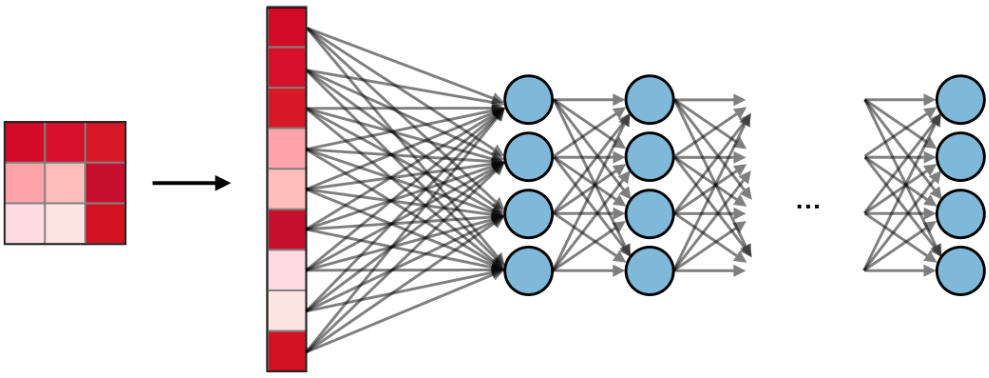
VII. Flatten

Sau khi dữ liệu đi qua một chuỗi các lớp Convolution và Pooling, ta thu được một khối tensor 3D chứa các đặc trưng cấp cao đã được cô đọng. Tuy nhiên, để thực hiện bài toán phân loại, ta cần đưa dữ liệu này vào một mạng nơ-ron truyền thống, nơi yêu cầu đầu vào phải là vector 1 chiều.

Lớp Flatten đóng vai trò là "cây cầu" nối giữa phần trích xuất đặc trưng và phần phân loại. Nhiệm vụ của nó đơn giản là duỗi thẳng toàn bộ khối tensor đa chiều thành một vector dài liên tục.

Ví dụ, nếu Feature Map cuối cùng có kích thước $5 \times 5 \times 64$ (Cao $\times$ Rộng $\times$ Số kênh):

$$\text{Flatten Output Size} = 5 \times 5 \times 64 = 1600 \text{ phần tử}$$



Hình 9: Minh họa quá trình Flatten: Biến đổi Tensor 3D thành Vector 1D để đưa vào lớp Fully Connected. Nguồn: link.

Thực hành Flatten trong PyTorch

```python
import torch
import torch.nn as nn

# Giả sử Feature Map cuối cùng có kích thước (Batch=1, Channel=64, H=4, W=4)
feature_map = torch.rand(1, 64, 4, 4)

flatten_layer = nn.Flatten()
output_vector = flatten_layer(feature_map)

print("Kích thước trước Flatten:", feature_map.shape)
# torch.Size([1, 64, 4, 4])

print("Kích thước sau Flatten:", output_vector.shape)
# torch.Size([1, 1024]) -> (64 * 4 * 4 = 1024)
```

VIII. Kênh màu trong CNN

Đến thời điểm này, chúng ta mới chỉ xem xét các phép tính trên "ảnh xám" (Grayscale), nơi mỗi điểm ảnh chỉ được biểu diễn bằng một ma trận 2 chiều ($Height \times Width$). Tuy nhiên, trong thế giới thực, hầu hết dữ liệu hình ảnh là "ảnh màu" (RGB), được cấu thành từ 3 kênh màu riêng biệt: Đỏ (Red), Lục (Green) và Lam (Blue).

Một câu hỏi thường gặp là: "Kernel sẽ hoạt động như thế nào trên 3 kênh này? Nó có trượt riêng lẻ trên từng kênh không?". Câu trả lời là: Kernel trong CNN luôn có "độ sâu" (depth) tương thích với đầu vào. Nếu ảnh Input có 3 kênh, thì mỗi Kernel cũng phải là một khối 3 chiều có kích thước $K \times K \times 3$.

Quá trình tích chập diễn ra như sau. Mỗi lớp của Kernel sẽ trượt trên kênh màu tương ứng của Input (Kênh Red nhân với lớp Red của Kernel, Green nhân với Green, v.v.). Các giá trị tích chập từ cả 3 kênh được cộng gộp lại với nhau (summed up). Cộng thêm một giá trị Bias (nếu có). Kết quả cuối cùng là một giá trị duy nhất trên Feature Map đầu ra.

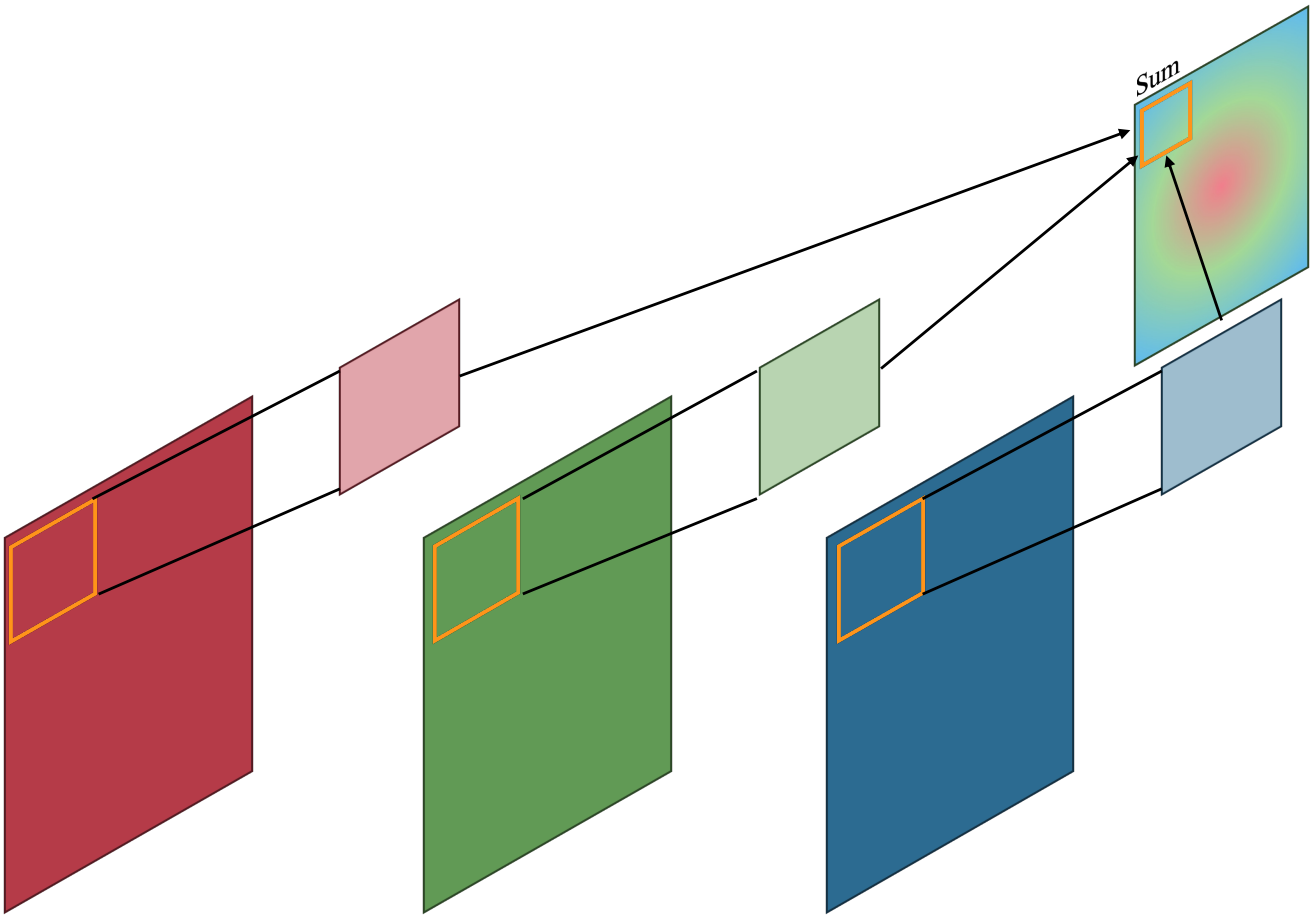
Như vậy, dù đầu vào là khối 3D ($H \times W \times 3$), nếu ta dùng 1 Filter thì đầu ra sẽ là ma trận 2D ($H' \times W' \times 1$). Để tạo ra đầu ra có nhiều kênh (ví dụ 64 kênh), ta cần sử dụng 64 Filter khác nhau.

Giá trị tại vị trí (i, j) của Feature Map đầu ra (O) được tính bằng tổng của tích chập trên tất cả các kênh c:

$$O(i,j) = \sum_{c=1}^{C_{in}} \left(\sum_{u=0}^{K_h-1} \sum_{v=0}^{K_w-1} I(c, i+u, j+v) \times K(c,u,v) \right) + b$$

Trong đó:

- c: chỉ số kênh (từ 1 đến 3 đối với ảnh RGB).
- u, v: chỉ số hàng và cột trong Kernel.



Hình 10: Minh họa phép tích chập trên ảnh RGB: Kernel 3 kênh trượt trên Input 3 kênh, tính tổng tất cả để ra 1 lớp Feature Map.

Hãy xét một vùng ảnh nhỏ kích thước 3×3 với 3 kênh màu (R, G, B) và một Kernel tương ứng.

- Input R $\cdot$ Kernel R $\rightarrow X_R$
- Input G $\cdot$ Kernel G $\rightarrow X_G$
- Input B $\cdot$ Kernel B $\rightarrow X_B$

Giá trị đầu ra tại vị trí đó sẽ là:

$$Output = X_R + X_G + X_B + Bias$$

Điều này cho thấy mạng CNN có khả năng học các mối quan hệ chéo kênh (cross-channel correlations), ví dụ như phát hiện một vật thể màu đỏ trên nền xanh.

Khai báo Conv2d cho ảnh màu trong PyTorch

```
import torch
import torch.nn as nn

```

```python
# Input: Batch=1, Channels=3 (RGB), Height=32, Width=32
input_rgb = torch.randn(1, 3, 32, 32)

# Khai báo lớp Convolution:
# - in_channels=3: Khớp với số kênh của ảnh đầu vào
# - out_channels=16: Số lượng Filter ta muốn dùng (sẽ tạo ra 16 Feature Maps)
# - kernel_size=3: Kích thước không gian (3x3)
conv_rgb = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3)

output = conv_rgb(input_rgb)

# Kích thước đầu ra: [1, 16, 30, 30]
# (30 = 32 - 3 + 1, do không có padding)
print("Output Shape:", output.shape)
```

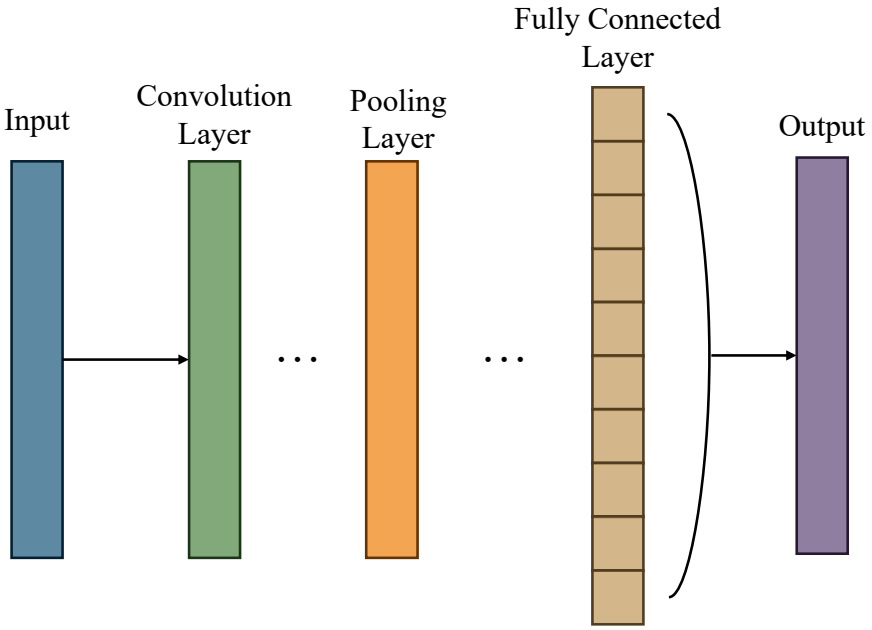
IX. Tính toán thủ công

Để củng cố toàn bộ kiến thức đã học, chúng ta sẽ thực hiện tính toán thủ công quá trình truyền tin (Forward Pass) của một mạng CNN đơn giản.

IX.1. Kiến trúc mạng

Giả sử chúng ta cần phân loại một ảnh nhị phân đơn giản. Kiến trúc mạng bao gồm các tầng sau:

1. **Input Layer:** Ảnh đầu vào kích thước 4×4.
2. **Convolution Layer:** Sử dụng 1 Kernel 3×3, Padding $P = 1$, Stride $S = 1$, Bias $b = 1$.
3. **Activation:** Hàm ReLU (loại bỏ giá trị âm).
4. **Pooling Layer:** Max Pooling kích thước 2×2, Stride $S = 2$.
5. **Flatten Layer:** Duỗi phẳng thành vector.
6. **Fully Connected Layer (Output):** Tính điểm số (Score) cuối cùng.



Hình 11: Sơ đồ kiến trúc mạng Mini-CNN dùng trong ví dụ tính toán.

IX.2. Bước 1: Khởi tạo giá trị

Ta có các ma trận dữ liệu giả định như sau:

Input Image (X):

$$X = \begin{bmatrix} 2 & 0 & 1 & 2 \\ 1 & 2 & 0 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

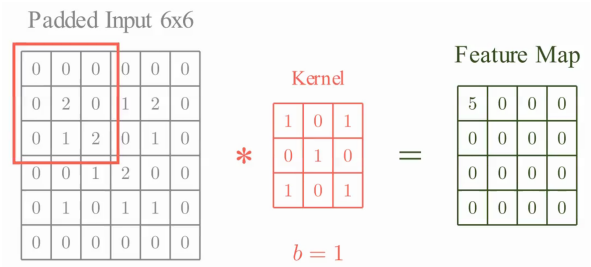
Kernel (K) và Bias (b):

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad b = 1$$

IX.3. Bước 2: Convolution (Padding & Stride)

Với Input 4×4, Kernel 3×3, ta sử dụng Padding $P = 1$ (thêm viền số 0) để giữ nguyên kích thước đầu ra là 4×4. Ảnh sau khi Padding (X_{pad}) sẽ có kích thước 6×6.

Ta áp dụng công thức: $Output = (X_{sub} \cdot K) + b$. Với Stride $S = 1$, Kernel sẽ trượt từng ô một. Hãy tính giá trị tại vị trí đầu tiên (góc trái trên) $O_{0,0}$.



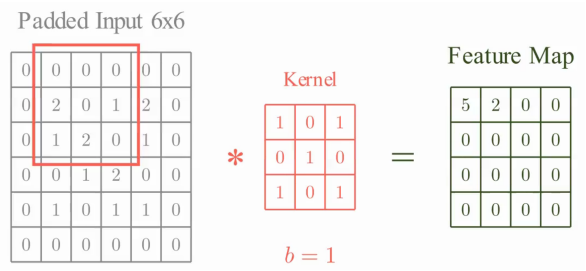
Hình 12: Vị trí (0,0): Kernel đặt tại góc trái trên cùng.

1. Vị trí hàng 0, cột 0 ($O_{0,0}$)
Thực hiện nhân từng phần tử với Kernel và cộng Bias:

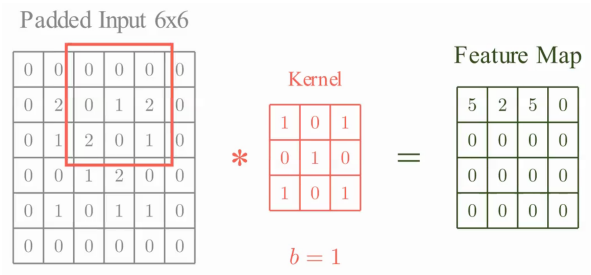
$$\begin{aligned} O_{0,0} &= (0 \times 1) + (0 \times 0) + (0 \times 1) \\ &\quad + (0 \times 0) + (2 \times 1) + (0 \times 0) \\ &\quad + (0 \times 1) + (1 \times 0) + (2 \times 1) + \mathbf{1} \\ &= 0 + 2 + 2 + 1 = \mathbf{5} \end{aligned}$$

2. Vị trí hàng 0, cột 1 ($O_{0,1}$)
Kernel trượt sang phải 1 ô:

$$\begin{aligned} O_{0,1} &= (0 \times 1) + (0 \times 0) + (0 \times 1) \\ &\quad + (2 \times 0) + (0 \times 1) + (1 \times 0) \\ &\quad + (1 \times 1) + (2 \times 0) + (0 \times 1) + \mathbf{1} \\ &= 0 + 0 + 1 + 1 = \mathbf{2} \end{aligned}$$



Hình 13: Vị trí (0,1): Kernel trượt sang phải theo Stride=1.



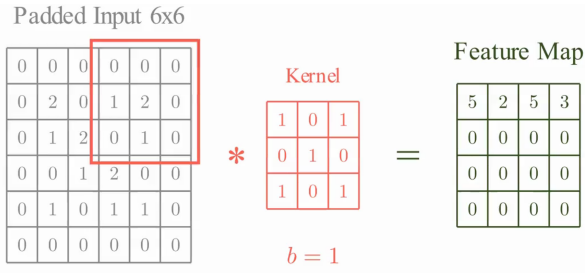
Hình 14: Vị trí (0,2): Tiếp tục trượt sang phải.

3. Vị trí hàng 0, cột 2 ($O_{0,2}$)
Tiếp tục nhân và cộng:

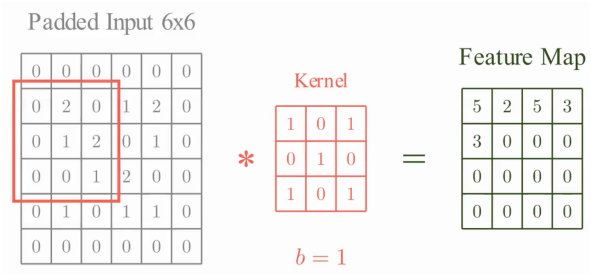
$$\begin{aligned} O_{0,2} &= (0\times 1)+(0\times 0)+(0\times 1) \\ &\quad +(0\times 0)+(1\times 1)+(2\times 0) \\ &\quad +(2\times 1)+(0\times 0)+(1\times 1)+\mathbf{1} \\ &= 0+1+3+1=\mathbf{5} \end{aligned}$$

4. Vị trí hàng 0, cột 3 ($O_{0,3}$)
Vị trí cuối cùng của hàng đầu tiên:

$$\begin{aligned} O_{0,3} &= (0\times 1)+(0\times 0)+(0\times 1) \\ &\quad +(1\times 0)+(2\times 1)+(0\times 0) \\ &\quad +(0\times 1)+(1\times 0)+(0\times 1)+\mathbf{1} \\ &= 0+2+0+1=\mathbf{3} \end{aligned}$$



Hình 15: Vị trí (0,3): Chạm biên phải của ảnh Input.



Hình 16: Vị trí (1,0): Kernel xuống hàng và quay về đầu trái.

5. Vị trí hàng 1, cột 0 ($O_{1,0}$)
Kernel trượt xuống dưới 1 ô:

$$\begin{aligned} O_{1,0} &= (0\times 1)+(2\times 0)+(0\times 1) \\ &\quad +(0\times 0)+(1\times 1)+(2\times 0) \\ &\quad +(0\times 1)+(0\times 0)+(1\times 1)+\mathbf{1} \\ &= 0+1+1+1=\mathbf{3} \end{aligned}$$

Lặp lại quy trình này cho toàn bộ ma trận, ta thu được Feature Map (trước ReLU):

$$\text{Feature Map} = \begin{bmatrix} 5 & 2 & 5 & 4 \\ 3 & 7 & 4 & 5 \\ 3 & 4 & 7 & 3 \\ 3 & 2 & 4 & 4 \end{bmatrix}$$

IX.4. Bước 3: Activation (ReLU)

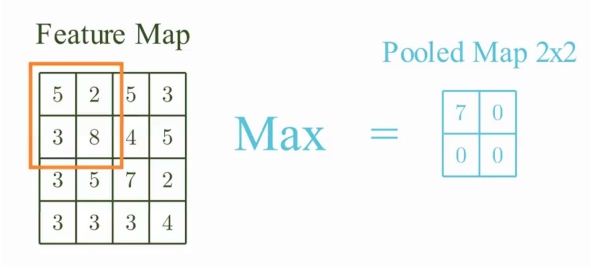
Hàm kích hoạt ReLU ($f(x) = \max(0, x)$) sẽ chuyển đổi các giá trị âm thành 0. Trong ví dụ này, tất cả giá trị đều dương, nên Feature Map giữ nguyên.

$$\text{ReLU Output} = \begin{bmatrix} 5 & 2 & 5 & 4 \\ 3 & 7 & 4 & 5 \\ 3 & 4 & 7 & 3 \\ 3 & 2 & 4 & 4 \end{bmatrix}$$

IX.5. Bước 4: Pooling (Max Pooling)

Sau khi qua hàm kích hoạt ReLU, ta thu được Feature Map. Tiếp theo, ta áp dụng Max Pooling với kích thước 2×2 và Stride $S = 2$.

Điều này có nghĩa là ta sẽ chia Feature Map thành các ô lưới 2×2 không chồng lấn lên nhau (non-overlapping) và chỉ giữ lại giá trị lớn nhất trong mỗi ô.



Hình 17: Vùng 1: Góc trái trên.

1. Vùng (0,0) - Góc trái trên

Xét vùng dữ liệu 2×2:

$$\text{Patch} = \begin{bmatrix} 5 & 2 \\ 3 & 7 \end{bmatrix}$$

Lấy giá trị lớn nhất:

$$\max(5, 2, 3, 7) = \mathbf{7}$$

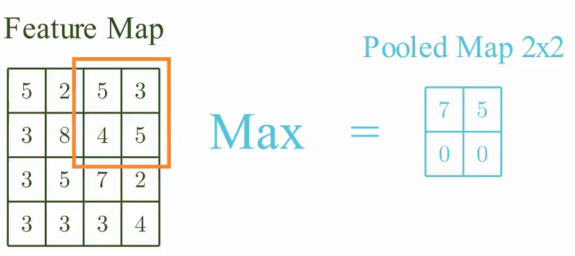
2. Vùng (0,1) - Góc phải trên

Do Stride $S = 2$, cửa sổ trượt sang phải 2 ô:

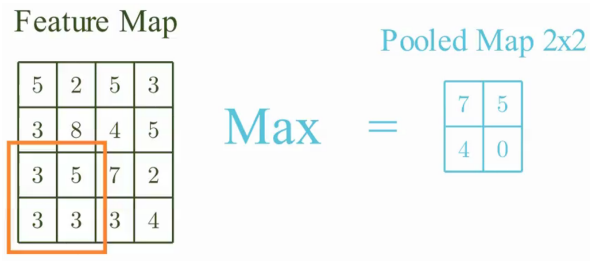
$$\text{Patch} = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

Lấy giá trị lớn nhất:

$$\max(5, 4, 4, 5) = \mathbf{5}$$



Hình 18: Vùng 2: Góc phải trên.



Hình 19: Vùng 3: Góc trái dưới.

3. Vùng (1,0) - Góc trái dưới

Cửa sổ xuống dòng 2 ô, quay về biên trái:

$$\text{Patch} = \begin{bmatrix} 3 & 4 \\ 3 & 2 \end{bmatrix}$$

Lấy giá trị lớn nhất:

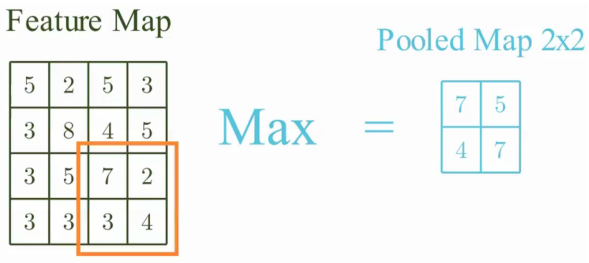
$$\max(3, 4, 3, 2) = \mathbf{4}$$

4. Vùng (1,1) - Góc phải dưới
Vị trí cuối cùng:

$$\text{Patch} = \begin{bmatrix} 7 & 3 \\ 4 & 4 \end{bmatrix}$$

Lấy giá trị lớn nhất:

$$\max(7, 3, 4, 4) = \mathbf{7}$$



Hình 20: Vùng 4: Góc phải dưới.

Kết quả cuối cùng sau khi Pooling là một ma trận 2×2:

$$\text{Pooled Map} = \begin{bmatrix} 7 & 5 \\ 4 & 7 \end{bmatrix}$$

IX.6. Bước 5: Flatten

Lớp Flatten sẽ duỗi ma trận 2×2 thành vector 1 chiều.

$$\text{Vector} = [7, 5, 4, 7]$$

IX.7. Bước 6: Output (Fully Connected)

Để ra kết quả cuối cùng, giả sử lớp Fully Connected có bộ trọng số W_{fc} và bias b_{fc}.

$$W_{fc} = [0.5, -1, 0, 1], \quad b_{fc} = 0.5$$

Tính toán điểm số (Score):

$$\begin{aligned} \text{Score} &= (7 \times 0.5) + (5 \times -1) + (4 \times 0) + (7 \times 1) + 0.5 \\ &= 3.5 - 5 + 0 + 7 + 0.5 \\ &= \mathbf{6.0} \end{aligned}$$

Vậy kết quả đầu ra của mạng với bức ảnh trên là 6.0.

Tổng kết bài học

Qua ví dụ này, ta thấy rõ vai trò của từng thành phần:

- **Conv:** Phát hiện đặc trưng cục bộ (tăng giá trị tại các vùng khớp với Kernel).
- **Stride/Pooling:** Giảm kích thước dữ liệu, cô đọng thông tin.
- **Flatten/FC:** Tổng hợp thông tin để đưa ra quyết định cuối cùng.

X. Tài liệu tham khảo
Phụ lục

1. **Code:** Các file code được đề cập trong bài có thể được tải tại đây.
2. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại đây. Tất cả câu hỏi sẽ được trả lời trong vòng 3-4 tiếng.

Hình 21: Hình ảnh group facebook AIO Q&A.