Understand and Build CNN from the Grounth Up and Intuition

Nhóm AIO_TimeSeries

Ngày 18 tháng 11 năm 2025

Bài viết này giải mã Convolutional Neural Networks (CNN) từ góc độ tư duy bộ lọc (filter perspective) thay vì chỉ liệt kê công thức toán học. mình sẽ đi từ hạn chế của mạng nơ-ron truyền thống đến cách CNN "nhìn" thế giới, cơ chế lan truyền ngược phức tạp và các kỹ thuật nâng cao như 1x1 Convolution.

Phần 1: Tại sao Neural Network truyền thống (MLP) "bó tay" với hình ảnh?

Phần 2: Tư duy Bộ lọc (Filter Perspective) & Cơ chế cơ bản

Phần 3: Kiến trúc CNN - Xếp tầng các khối xử lý

Phần 4: CNN Backpropagation & Sự thật về phép xoay Kernel

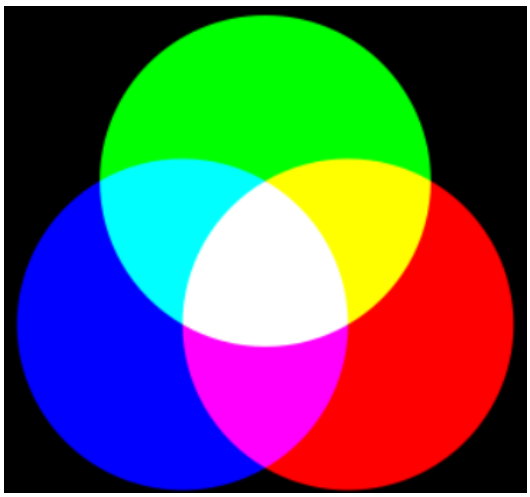
Phần 5: CNN Nâng cao - 1x1 Convolution & Phân biệt chiều không gian

Phần 1: Tại sao Neural Network truyền thống (MLP) "bó tay" với hình ảnh?

Trước khi nói về giải pháp, hãy nhìn vào vấn đề. Tại sao mình không thể cứ thế ném một bức ảnh vào mạng nơ-ron đa tầng (MLP) thông thường? Để hiểu điều này, ta cần "mổ xẻ" cách máy tính lưu trữ một bức ảnh.

1.1 Bản chất của màu sắc (RGB Color)

Trong thị giác máy tính, màu sắc không phải là khái niệm trừu tượng mà là các con số cụ thể được định nghĩa bởi hệ màu **RGB** (Red - Green - Blue). Mỗi kênh màu này đại diện cho một sắc độ riêng biệt với giá trị chạy từ 0 đến 255 (tương ứng với 256 mức độ khác nhau). Bằng cách phối hợp cường độ sáng của 3 kênh này lại, máy tính có thể tái tạo hàng triệu màu sắc khác nhau mà mắt người nhìn thấy được.



1.2 Biểu diễn ảnh dưới dạng Ma trận (Matrix Representation)

Một bức ảnh kỹ thuật số thực chất là một lưới các điểm ảnh (pixels). Ví dụ, một bức ảnh kích thước 800×600 sẽ được cấu thành từ 800 hàng và 600 cột.

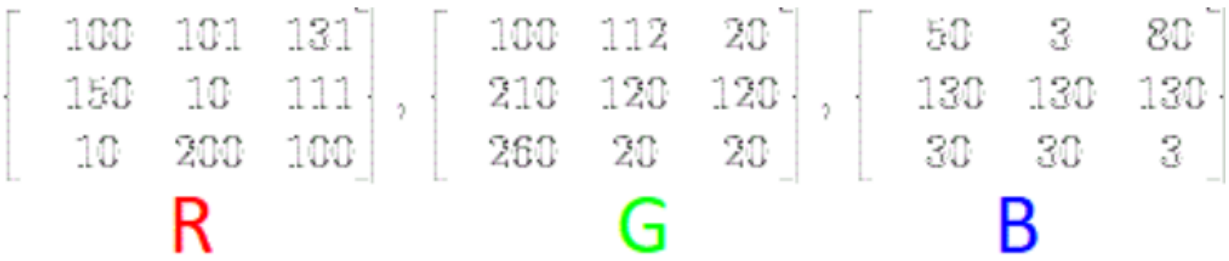
$$\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,800} \\ w_{2,1} & w_{2,2} & \dots & w_{2,800} \\ \dots & \dots & \dots & \dots \\ w_{600,1} & w_{600,2} & \dots & w_{600,800} \end{bmatrix}$$

(a) Ma trận tổng quát $W_{i,j}$

$$\begin{bmatrix} (100,100,50) & (101,112,3) & (131,20,80) \\ (150,210,130) & (10,120,130) & (111,120,130) \\ (10,260,30) & (200,20,30) & (100,20,3) \end{bmatrix}$$

(b) Giá trị pixel là bộ 3 số (R,G,B)

Tại mỗi vị trí tọa độ (i, j), giá trị pixel w_{ij} không tồn tại dưới dạng một số đơn lẻ mà là một bộ ba giá trị (r_{ij}, g_{ij}, b_{ij}), ví dụ như $(0, 233, 256)$ để tạo ra màu xanh ngọc. Để tối ưu hóa cho việc lưu trữ và xử lý tính toán, máy tính thường không gộp chung mà tách riêng các giá trị này thành 3 ma trận độc lập tương ứng với 3 kênh màu Đỏ, Xanh lá và Xanh dương.



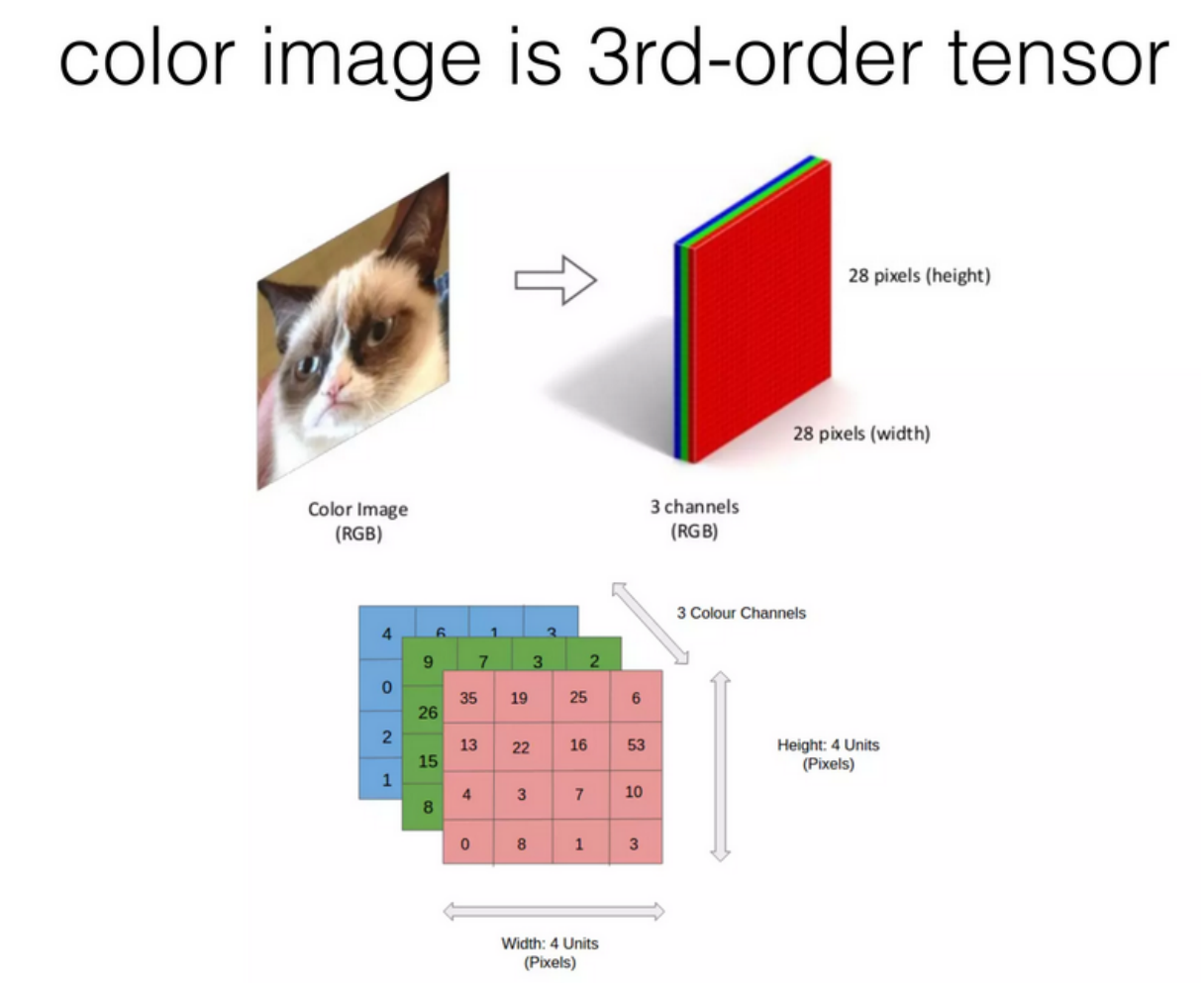
(c) Tách một ảnh màu thành 3 ma trận R, G, B riêng biệt

$$\begin{bmatrix} (r_{1,1}, g_{1,1}, b_{1,1}) & (r_{1,2}, g_{1,2}, b_{1,2}) & \dots & (r_{1,800}, g_{1,800}, b_{1,800}) \\ (r_{2,1}, g_{2,1}, b_{2,1}) & (r_{2,2}, g_{2,2}, b_{2,2}) & \dots & (r_{2,800}, g_{2,800}, b_{2,800}) \\ \dots & \dots & \dots & \dots \\ (r_{600,1}, g_{600,1}, b_{600,1}) & (r_{600,2}, g_{600,2}, b_{600,2}) & \dots & (r_{600,800}, g_{600,800}, b_{600,800}) \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} r_{1,1} & r_{1,2} & \dots & r_{1,800} \\ r_{2,1} & r_{2,2} & \dots & r_{2,800} \\ \dots & \dots & \dots & \dots \\ r_{600,1} & r_{600,2} & \dots & r_{600,800} \end{bmatrix}, \begin{bmatrix} g_{1,1} & g_{1,2} & \dots & g_{1,800} \\ g_{2,1} & g_{2,2} & \dots & g_{2,800} \\ \dots & \dots & \dots & \dots \\ g_{600,1} & g_{600,2} & \dots & g_{600,800} \end{bmatrix}, \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,800} \\ b_{2,1} & b_{2,2} & \dots & b_{2,800} \\ \dots & \dots & \dots & \dots \\ b_{600,1} & b_{600,2} & \dots & b_{600,800} \end{bmatrix},$$

1.3 Từ Ma trận đến Tensor

Khái niệm **Tensor** xuất hiện để tổng quát hóa các cấu trúc dữ liệu này theo số chiều. Trong khi Vector là Tensor 1D và Ma trận là Tensor 2D (như ảnh xám chỉ cần một ma trận duy nhất để biểu diễn độ sáng từ đen sang trắng), thì ảnh màu lại phức tạp hơn. Vì ảnh màu được tạo thành từ việc chồng 3 ma trận R, G, B lên nhau (collapse on top of each other), nó được định nghĩa là một **Tensor 3D** với kích thước $Height \times Width \times Depth$ (trong đó Depth = 3).



Việc hiểu đúng chiều sâu (Depth) của Tensor là mấu chốt để hiểu cách CNN vận hành sau này.

1.4 Sự bùng nổ tham số (The Parameter Explosion)

Chính cấu trúc Tensor 3D này là nguyên nhân khiến mạng MLP truyền thống "đầu hàng". Chỉ thử làm một phép tính với bức ảnh rất nhỏ kích thước 64×64: tổng số giá trị đầu vào sẽ là $64 \times 64 \times 3 = 12,288$. Nếu ta kết nối đầu vào này vào một lớp ẩn chỉ gồm 1,000 nơ-ron theo cách kết nối toàn bộ (fully connected) của MLP, số lượng trọng số cần học sẽ lên tới hơn **12 triệu** tham số $(12,288 \times 1,000)$. Khối lượng tính toán khổng lồ này không chỉ làm chậm hệ thống mà còn dẫn đến hiện tượng Overfitting nghiêm trọng, buộc mình phải tìm đến giải pháp thông minh hơn là CNN để xử lý cấu trúc Tensor một cách hiệu quả.

Phần 2: Tư duy Bộ lọc (Filter Perspective) & Cơ chế cơ bản

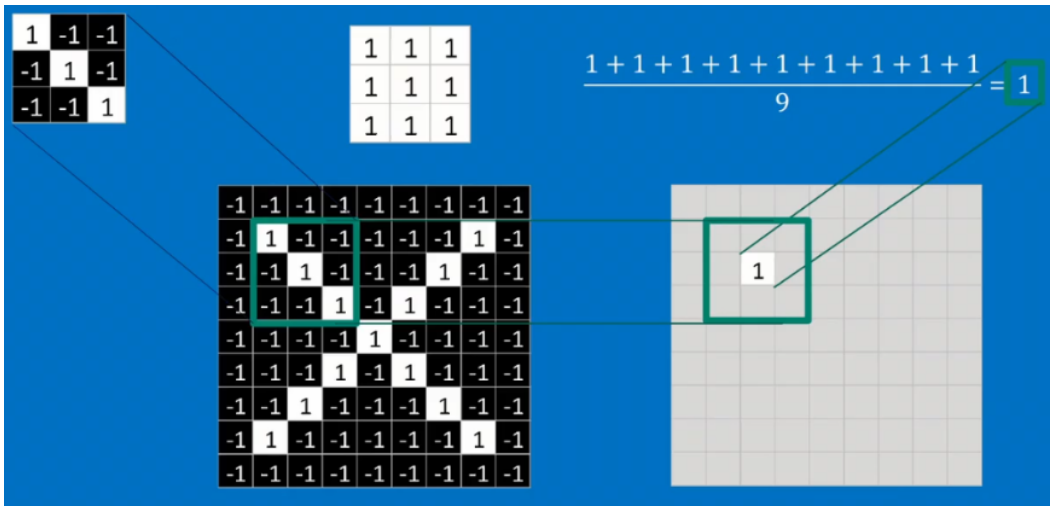
Đây là phần trọng tâm để hiểu CNN từ gốc rễ. Thay vì tư duy theo kiểu "kết nối tất cả mọi thứ" như mạng MLP truyền thống, CNN tiếp cận hình ảnh theo cách con người quan sát: tìm kiếm các đặc trưng (features).

2.1 Kernel/Filter là gì? - Chiếc "đèn pin" soi mẫu

Trong thế giới của CNN, để máy tính hiểu được một bức ảnh, nó cần biết được những đặc trưng nào làm cho bức ảnh đó trở nên độc nhất. Thay vì phân tích từng điểm ảnh riêng lẻ, chúng ta sử dụng **Kernel** (hay Filter) - một ma trận nhỏ (thường là 3×3 hoặc 5×5). Kernel hoạt động như một "cửa sổ trượt", di chuyển quét qua toàn bộ bức ảnh để tìm kiếm sự tồn tại của các mẫu cụ thể, ví dụ như một đường cong, một cạnh dọc, hay một đường chéo. Để hiểu rõ, mình đi vào 1 ví dụ luon nhé.

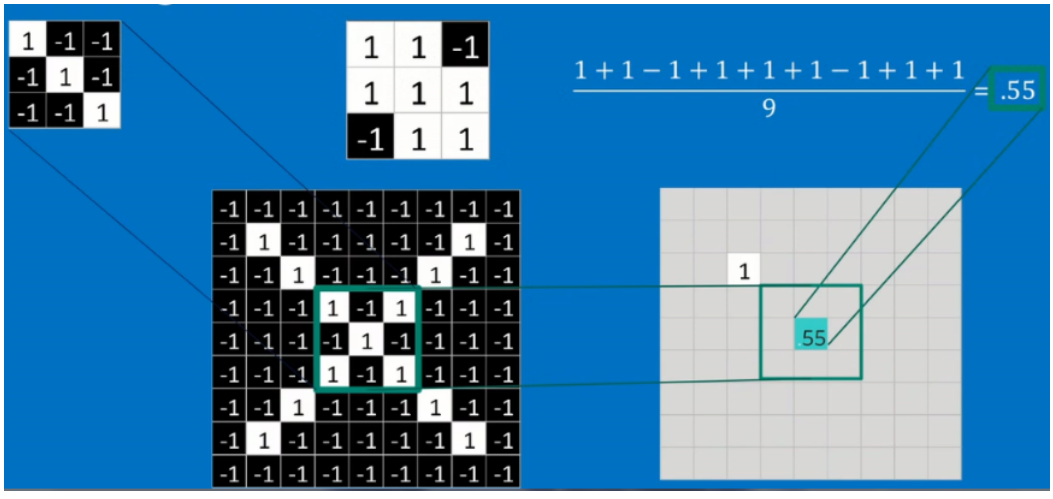
2.2 Cơ chế khớp mẫu (Feature Matching) - Bí mật của phép tích chập

Hãy hình dung chúng ta muốn dạy máy tính nhận diện chữ "X". Chữ "X" được cấu tạo bởi hai đường chéo bắt chéo nhau. Một Kernel được thiết kế để tìm "đường chéo trái" sẽ có các giá trị dương (ví dụ: 1) nằm trên đường chéo đó và các giá trị âm (ví dụ: -1) ở những vị trí còn lại. (Note: ma trận trắng đại diện cho kết quả sau khi áp dụng kernel)



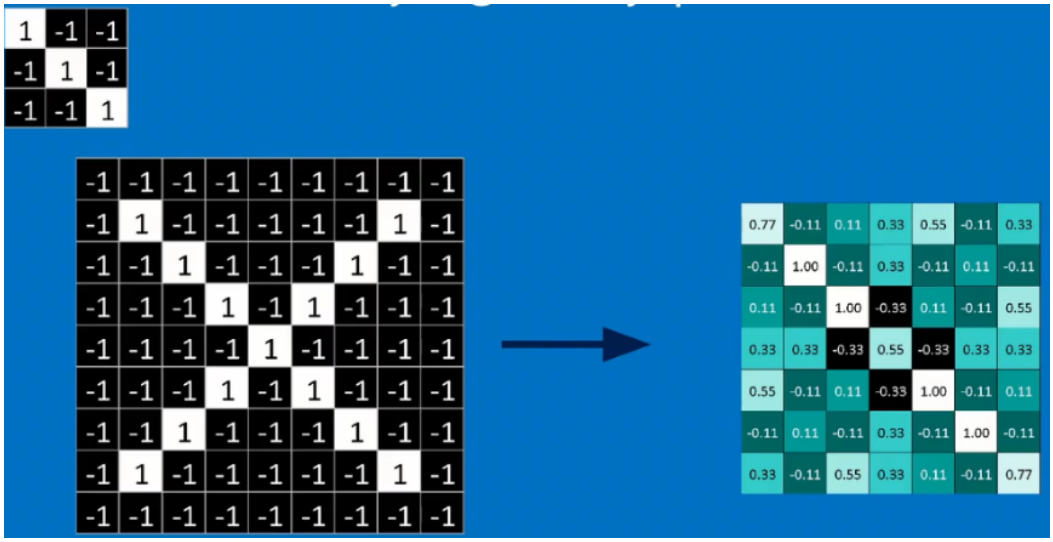
Dây là phần trọng tâm để hiểu CNN từ gốc rễ. Thay vì tư duy theo kiểu "kết nối tất cả mọi thứ" như mạng MLP truyền thống, CNN tiếp cận hình ảnh theo cách con người quan sát: tìm kiếm các đặc trưng (features).

Quá trình "Filtering" (lọc) diễn ra như sau:



1. **Line up**: Đặt Kernel chồng lên một vùng ảnh cùng kích thước.
2. **Multiply**: Nhân từng giá trị pixel của ảnh với giá trị tương ứng trong Kernel (Element-wise multiplication).
3. **Add**: Cộng tổng tất cả các kết quả lại và chia trung bình.

Kết quả:



- Nếu vùng ảnh bên dưới thực sự có hình đường chéo (khớp với Kernel), phép nhân sẽ tạo ra các số dương lớn → Kết quả tổng rất cao (ví dụ: 1.0 hoặc 100%). Ta nói pattern đó đã được "kích hoạt" (activated).
- Nếu vùng ảnh không khớp (ví dụ: Kernel đường chéo nhưng đặt lên vùng ảnh đường thẳng đứng), các phép nhân dương và âm sẽ triệt tiêu lẫn nhau → Kết quả tổng xấp xỉ 0 hoặc âm.
- Ví dụ tương tự đối với ảnh 3D.

Stride (Bước nhảy - ô vuông kernel di chuyển bao nhiêu bước mỗi lượt): Đây là khoảng cách mà Kernel di chuyển sau mỗi lần tính toán. Nếu $Stride = 1$, Kernel nhích từng pixel một, giữ độ chi tiết cao nhất. Nhưng nếu ta tăng $Stride > 1$ (ví dụ: $Stride = 2$), Kernel sẽ "nhảy cóc" qua các pixel. Điều này có tác dụng giảm kích thước dữ liệu đầu ra (Downsampling) ngay lập tức mà không cần lớp Pooling, giống như việc ta lướt nhanh qua một văn bản để nắm ý chính thay vì đọc từng chữ.

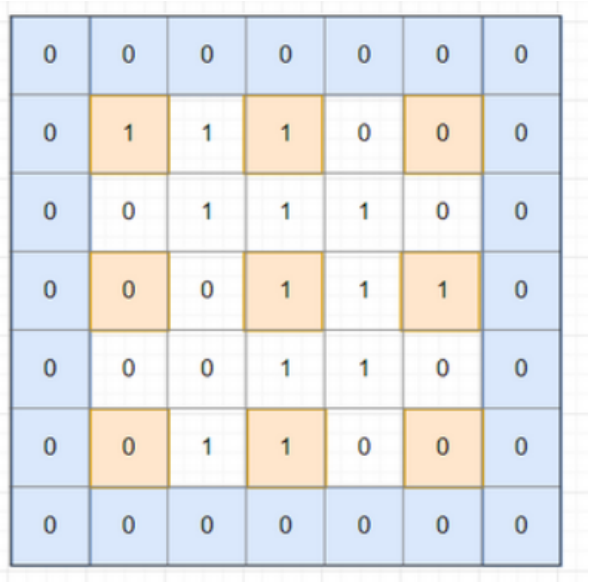
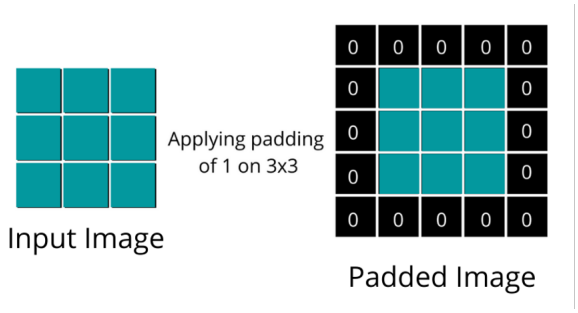
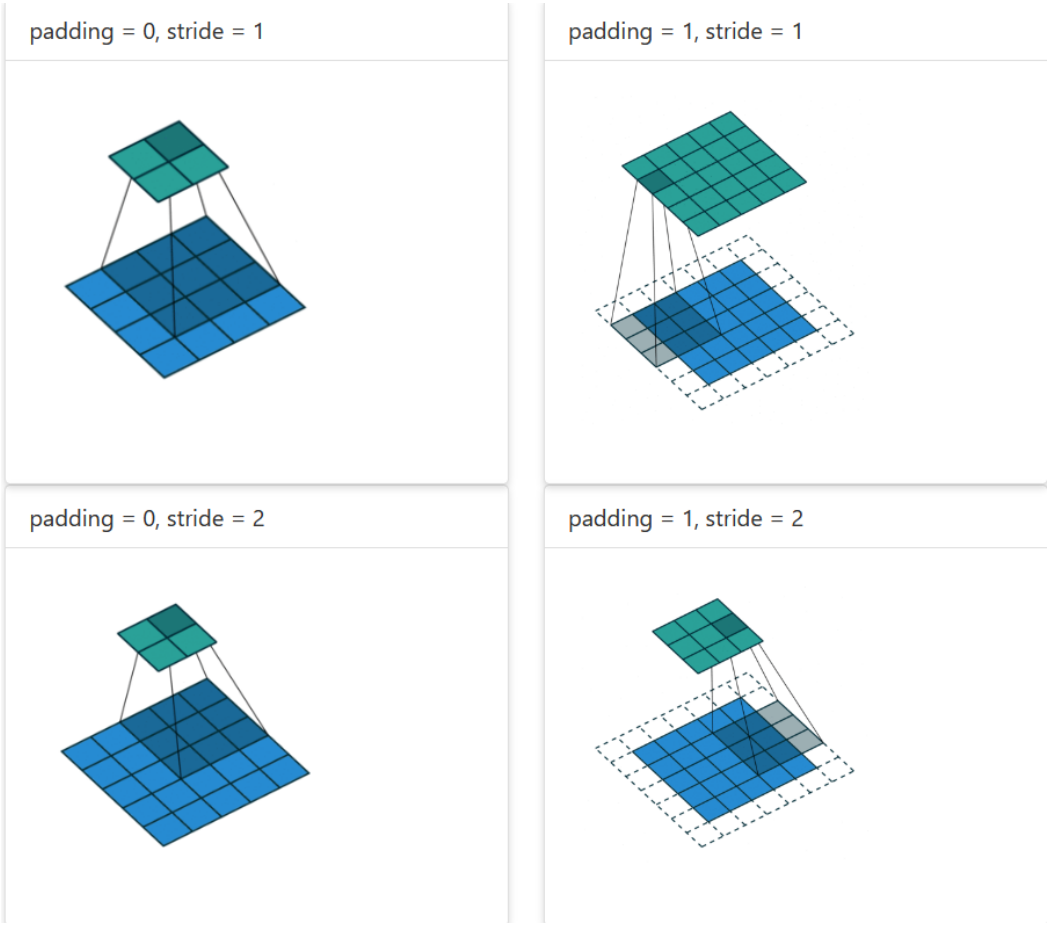


Figure 2: Với phần in đậm là tâm của kernel, với stride = 2, kernel sẽ đi 2 bước tính từ tâm. Nếu kernel là 2x2 thì lấy pixel trên cùng bên trái làm tâm

Padding (Lề - Pixel được bọc thêm bên): Khi Kernel trượt ở rìa bức ảnh, nó thường bị thiếu hụt dữ liệu (không đủ kích thước 3×3 để nhân). Điều này dẫn đến hai vấn đề: ảnh bị thu nhỏ lại sau mỗi lớp tích chập và thông tin ở rìa ảnh bị mất mát. Giải pháp là **Zero-Padding**: thêm một viền các số 0 bao quanh ảnh gốc. Lớp "đệm" này cho phép Kernel đặt tâm ngay tại pixel ngoài cùng, giúp giữ nguyên kích thước không gian ($Height \times Width$) của Feature Map đầu ra.



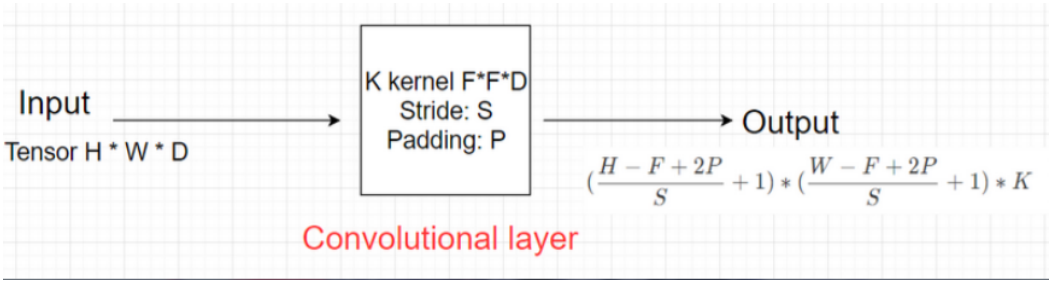
Minh họa Padding và Stride: Note: kích thước ảnh hay feature map (ảnh sau khi áp dụng convolution) phải lớn tương ứng để áp dụng stride.



Công thức kích thước đầu ra: Để thiết kế kiến trúc mạng chính xác, ta cần tính toán được kích thước của ma trận sau khi qua lớp Conv:

$$Output = \frac{Input - Filter + 2 \times Padding}{Stride} + 1$$

Trong đó: *Input* là kích thước ảnh đầu vào, *Filter* là kích thước Kernel, *Padding* là số lớp viền thêm vào, và *Stride* là bước nhảy.

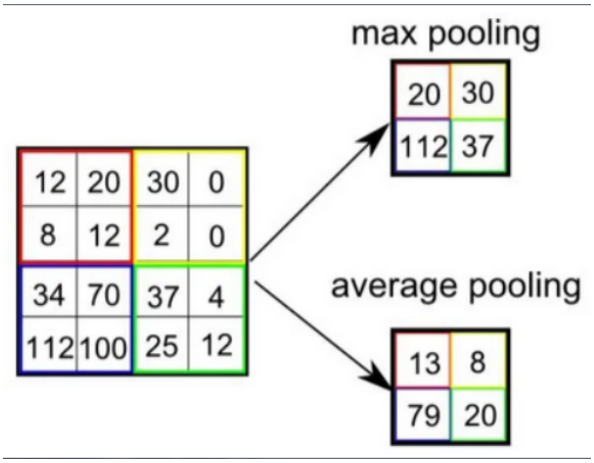


2.3 Các phép toán kiểm soát Bộ lọc (Mechanics)

Việc trượt Kernel qua ảnh không phải lúc nào cũng tuỳ ý, nó được kiểm soát bởi các tham số toán học nghiêm ngặt để định hình dữ liệu đầu ra.

Pooling (Gộp đặc trưng):

- **Max Pooling:** Chỉ giữ lại giá trị lớn nhất trong vùng (đặc trưng nổi bật nhất), giúp nén ảnh và giảm tải tính toán nhưng vẫn giữ được thông tin cốt lõi.
- **Average Pooling:** Lấy giá trị trung bình của toàn bộ các pixel trong vùng, nhằm tổng hợp thông tin một cách "mềm", phản ánh mức độ hiện diện trung bình của đặc trưng thay vì chỉ tập trung vào điểm mạnh nhất.



Phần 3: Kiến trúc CNN - Xếp tầng các khối xử lý

Sau khi hiểu về bộ lọc, ta đi vào cách các lớp (Layers) kết hợp với nhau để tạo thành mạng nơ-ron.

- **Quy trình chuẩn:** Input → Conv → ReLU → Pooling → FC → Output.
- **ReLU (Rectified Linear Unit):**
  - Đóng vai trò chuẩn hóa (Normalization) và loại bỏ các giá trị âm (các đặc trưng không quan trọng/không khớp mẫu).
  - Giúp mô hình học được tính phi tuyến.
- **Receptive Field (Vùng cảm nhận):** Các lớp sâu hơn "nhìn thấy" vùng ảnh rộng hơn như thế nào.

Phần 4: CNN Backpropagation & Sự thật về phép xoay Kernel

Phần này dành cho người đọc muốn hiểu sâu về toán học (Deep Dive).

- **Cơ chế học:** Mọi giá trị trong CNN đều là một "phiếu bầu" (vote), và Backprop giúp điều chỉnh sức nặng của các lá phiếu này.
- **Góc khuất của toán học - Xoay 180 độ:**
  - Giải thích tại sao trong quá trình Backpropagation, để tính đạo hàm chính xác, ta phải xoay Kernel 180 độ trước khi thực hiện phép tích chập ngược.
  - Liên hệ với khái niệm Cross-Correlation trong thực tế so với Convolution trong toán học thuần túy.

Phần 5: CNN Nâng cao - 1x1 Convolution & Phân biệt chiều không gian

Mở rộng kiến thức sang các khái niệm hiện đại và sửa chữa các hiểu lầm phổ biến.

- **Phân biệt 2D vs 3D CNN:**
  - Làm rõ hiểu lầm: Ảnh màu RGB (3 kênh) vẫn dùng 2D CNN (xử lý không gian Height-Width).
  - 3D CNN thực sự dùng cho dữ liệu video (thêm chiều thời gian/Temporal).
- **Phép màu của 1×1 Convolution:**
  - Nó hoạt động như một "Single Neuron" áp dụng lên chiều sâu (channels) của từng pixel.
  - Ứng dụng: Giảm số chiều dữ liệu (Dimensionality Reduction) và trộn thông tin giữa các kênh màu (Channel Pooling) mà không thay đổi kích thước không gian.
- **Triển khai OOP:** Code minh họa kiến trúc CNN bằng Python (Pytorch/Tensorflow) theo hướng đối tượng.