

Convolution và pooling

☑ Trắc nghiệm 76 ☒ Thực hành 2 ⬇ PDF bài

Đưa ảnh vào MLP phải trải mọi điểm ảnh thành một vector dài. Cách đó có hai vấn đề: số trọng số tăng rất nhanh khi ảnh lớn, và một nét ở góc trái với cùng nét ấy ở giữa ảnh được học như hai thứ khác nhau. Convolution dùng một bộ lọc nhỏ, ví dụ 3×3 , trượt qua mọi vị trí của ảnh với cùng một bộ trọng số. Bài này học convolution, padding, stride và pooling trên ví dụ dò nét dọc, nét ngang tính được bằng tay.

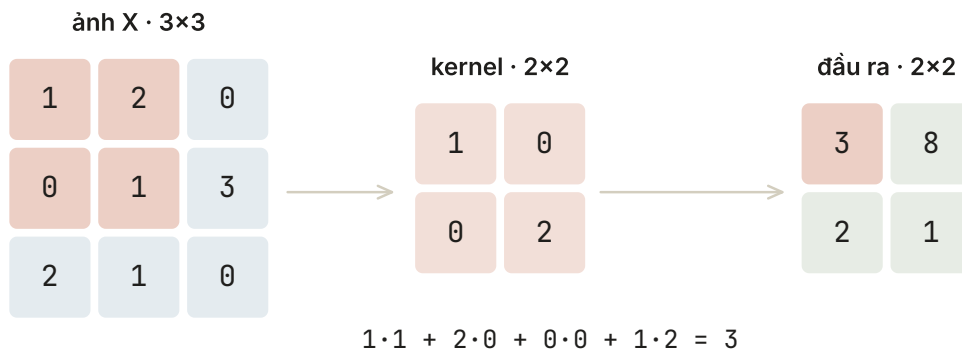
MỤC TIÊU

- Tính convolution bằng tay và so với PyTorch.
- Tính kích thước đầu ra theo kernel, padding, stride, dilation.
- Phân biệt convolution và pooling.
- Tính receptive field của nhiều lớp.

Một bộ dò đi qua nhiều vị trí

Ảnh xám là bảng số $H \times W$. Mỗi ô của bảng gọi là một điểm ảnh (pixel), mang một số từ 0 (đen) tới 255 (trắng). Ảnh xám có một kênh; ảnh màu có ba bảng như vậy, gọi là ba kênh RGB.

Các điểm cạnh nhau có quan hệ: cạnh của viên phấn thường kéo dài qua nhiều ô lân cận. Vì vậy ta đọc một cửa sổ nhỏ trước khi đọc cả ảnh.



Nhân từng ô của cửa sổ với kernel rồi cộng: ô đầu ra đầu tiên bằng 3.

Kernel (bộ lọc) là bảng trọng số K . Tại mỗi vị trí, nhân từng ô trong cửa sổ với ô tương ứng của K , rồi cộng. Với một kênh, stride 1, chưa có padding và bias:

$$Y_{i,j} = \sum_{u=0}^{k_h-1} \sum_{v=0}^{k_w-1} X_{i+u,j+v} K_{u,v}.$$

i, j chỉ vị trí trên bảng kết quả; u, v chỉ vị trí bên trong kernel. Tổng ở đây là tích vô hướng của hai bảng khi trải thành vector. Nếu đã học linear regression, ta đang dùng một mô hình tuyến tính nhỏ trên từng cửa sổ, và mọi cửa sổ dùng **chung trọng số**.

Trong toán học, convolution lật kernel 180° trước khi nhân; phép nhân thẳng không lật gọi là cross-correlation. Conv2d của PyTorch làm cross-correlation, nhưng ngành học sâu vẫn gọi nó là convolution.

Khi kernel được học, khác biệt này không quan trọng: mạng sẽ học luôn bản đã lật. Nó chỉ quan trọng khi đề cho sẵn kernel và hỏi kết quả; khi đó tính theo cách không lật như PyTorch.

Dùng chung trọng số còn quyết định số tham số. Nếu mỗi vị trí đầu ra có kernel riêng, một ảnh xám cho đầu ra 28×28 cần 784 bộ trọng số; dùng chung thì chỉ một bộ, và bộ ấy học từ mọi vị trí cùng lúc. So với nối thẳng mọi pixel vào lớp kết nối đầy đủ, khác biệt rất lớn: ảnh $224 \times 224 \times 3$ vào 1000 neuron đã cần khoảng 150 triệu trọng số.

HỎI NHANH

Hàng ảnh $[0, 0, 5, 5]$ có một cạnh chuyển từ tối sang sáng. Kernel $[-1, 1]$, stride 1, không padding. Conv2d của PyTorch cho ra kết quả nào?

A $[0, 5, 0]$

B $[0, -5, 0]$

C $[0, 0, 5]$

D $[0, 5, 0, 0]$

HỎI NHANH

Ảnh xám 64×64 , kernel 3×3 , không padding, stride 1, đầu ra 62×62 . Nếu mỗi vị trí đầu ra có một kernel riêng thay vì dùng chung, lớp này cần bao nhiêu trọng số, bỏ qua bias?

A 9

B 3844

C 34 596

D 36 864

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q1.4(a)

Lấy ảnh và kernel sau. Chưa cần ảnh thật: bảng số nhỏ giúp nhìn mọi phép nhân.

Python

```
import numpy as np

image = np.array([[1, 2, 0],
                  [0, 1, 3],
                  [2, 1, 0]], dtype=float)
```

```
kernel = np.array([[1, 0],
                   [0, 2]], dtype=float)

def correlate_valid(image, kernel):
    height, width = image.shape
    kh, kw = kernel.shape
    result = np.zeros((height - kh + 1, width - kw + 1))
    for row in range(result.shape[0]):
        for col in range(result.shape[1]):
            window = image[row:row + kh, col:col + kw]
            result[row, col] = np.sum(window * kernel)
    return result

feature_map = correlate_valid(image, kernel)
print(feature_map)
```

Ô trên trái là $1 \cdot 1 + 2 \cdot 0 + 0 \cdot 0 + 1 \cdot 2 = 3$. Ô trên phải là $2 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 3 \cdot 2 = 8$. Hai ô hàng dưới lần lượt là 2 và 1. Bảng đầu ra nhỏ hơn vì kernel phải nằm trọn trong ảnh.

Kernel này được đặt tay cho dễ tính, nhưng đã cho thấy cách một bộ dò hoạt động: nó chỉ có trọng số dương trên đường chéo chính, nên cửa sổ có số lớn trên đường chéo ấy, $\begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix}$, cho giá trị lớn nhất là 8. Trong CNN, giá trị kernel không đặt tay mà được học từ dữ liệu. Vài kernel cổ điển vẫn đáng nhớ:

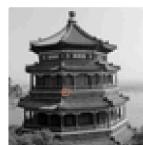
- **Sobel** $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ phản ứng mạnh ở cạnh dọc, nơi độ sáng đổi đột ngột theo chiều ngang. Tìm những chỗ như vậy gọi là phát hiện cạnh.
- **Làm mờ 3×3** : 9 ô đều bằng $1/9$, tổng bằng 1 để giữ nguyên độ sáng trung bình.



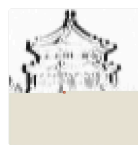
cửa sổ ở hàng 3, cột 6



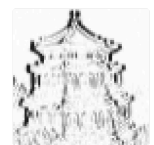
cửa sổ ở hàng 20, cột 29



cửa sổ ở hàng 43, cột 29



xong: 70×70 ô, mỗi ô dùng cùng 9 trọng số

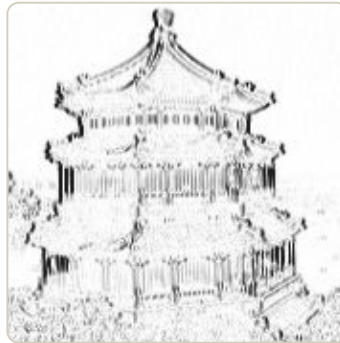


Kernel nét dọc trượt qua ảnh tháp; mỗi vị trí cho một ô của bảng đầu ra.

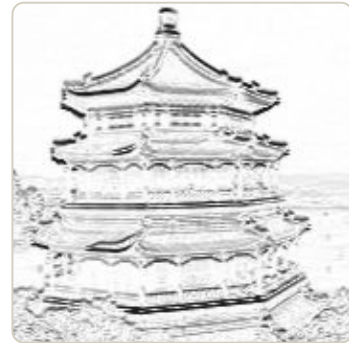
ảnh xám



kernel nét dọc



kernel nét ngang



cùng 9 trọng số trượt qua mọi vị trí

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Kernel nét dọc làm sáng cạnh đứng của tháp; kernel ngang làm sáng mái và lan can.

HỎI NHANH

Ta muốn một kernel 2×2 cho số dương với cửa sổ nét chéo $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, nhưng cho 0 với mọi vùng nền đều màu $\begin{bmatrix} c & c \\ c & c \end{bmatrix}$. Chọn kernel nào?

A $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

B $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

C $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

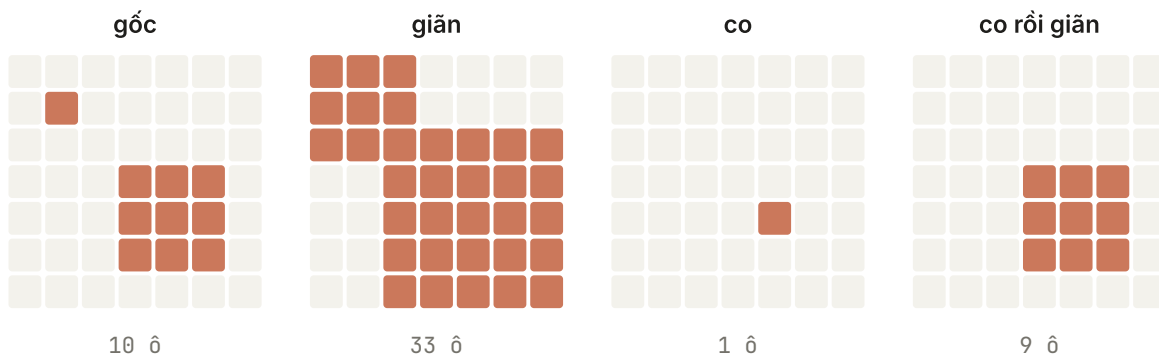
D $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

Không phải bộ lọc cổ điển nào cũng là tổng có trọng số. Trên ảnh nhị phân, nền là 0 và vật là 1, phép giãn và phép co thay nhân-cộng bằng max và min trên cửa sổ 3×3 . Giãn đặt ô ra bằng 1 nếu cửa sổ có ít nhất một ô 1, nên nét dày lên và lấp được khe nhỏ. Co chỉ đặt ô ra bằng 1 khi cả cửa sổ đều là 1, nên nét mỏng đi và chấm nhiễu nhỏ biến mất. Co rồi giãn (opening) xóa chấm nhiễu mà vật lớn gần như giữ nguyên; giãn rồi co (closing) lấp khe. Phép giãn (morphological dilation) này khác dilation của convolution ở mục sau.

```
def morph(img, op):
    padded = np.pad(img, 1)
    out = np.zeros_like(img)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            out[i, j] = op(padded[i:i + 3, j:j + 3])
    return out

mask = np.zeros((7, 7), dtype=int)
mask[1, 1] = 1          # một chấm nhiễu
mask[3:6, 3:6] = 1      # một vật 3 x 3
dilated, eroded = morph(mask, np.max), morph(mask, np.min)
opened = morph(eroded, np.max)
print("Số ô bằng 1: gốc", mask.sum(), " giãn", dilated.sum(),
      " co", eroded.sum(), " co rồi giãn", opened.sum())
print("Chấm nhiễu sau co rồi giãn:", opened[1, 1])
```

Giãn biến chấm nhiễu thành 9 ô và vật 3×3 thành 5×5 ; hai vùng chạm nhau ở một ô nên tổng là $9 + 25 - 1 = 33$. Co xóa chấm nhiễu và chỉ để lại ô giữa của vật. Giãn lại từ đó cho đúng vật 3×3 ban đầu, còn chấm nhiễu không quay lại.



Giãn làm vật dày lên, co làm vật mỏng đi; co rồi giãn xóa chấm nhiễu mà giữ vật.

Padding, stride và dilation thay đổi điều gì?

Padding thêm vành ô quanh ảnh, thường bằng 0. Nó cho kernel đọc cả vùng sát biên và có thể giữ kích thước. Giá trị 0 ở biên là một giả định về bên ngoài ảnh; với ảnh nhỏ, giả

định này có ảnh hưởng đáng kể.

Stride là số ô dịch mỗi bước. Stride 2 đọc thưa hơn, giảm chi phí nhưng có thể bỏ sót nét mảnh.

Dilation tạo khoảng trống giữa các ô kernel: kernel 3 ô với dilation 2 bao phủ 5 vị trí, dù vẫn chỉ có 3 trọng số theo chiều đó. Kích thước bao phủ hiệu dụng là $d(k - 1) + 1$.

Ta đếm số vị trí hợp lệ theo một chiều. Chiều dài sau padding là $H + 2p$; cửa sổ đầu chiếm $d(k - 1) + 1$ ô, phần còn lại cho phép dịch từng s ô:

$$H_{\text{out}} = \left\lfloor \frac{H + 2p - d(k - 1) - 1}{s} \right\rfloor + 1.$$

Chiều rộng tính tương tự. Công thức giả định mỗi đầu thêm đúng p ô. Với stride 1, muốn giữ kích thước thì cần $2p = d(k - 1)$; padding="same" của PyTorch tự chọn padding như vậy, và khi $d(k - 1)$ lẻ (kernel chẵn) phần dư được thêm ở cuối.

Ví dụ $H = 7, k = 3, p = 1, s = 2, d = 1$ cho $\lfloor 6/2 \rfloor + 1 = 4$. Các vị trí bắt đầu là 0, 2, 4, 6 trên ảnh đã thêm vành, không phải $7/2$ làm tròn tùy ý.

Vài trường hợp hay gặp: ảnh 32×32 , kernel 5, không padding ra 28×28 ; giữ 64×64 với kernel 3 cần $p = 1$; kernel 3 với dilation 2 trên 32 ô ra 28.

Python

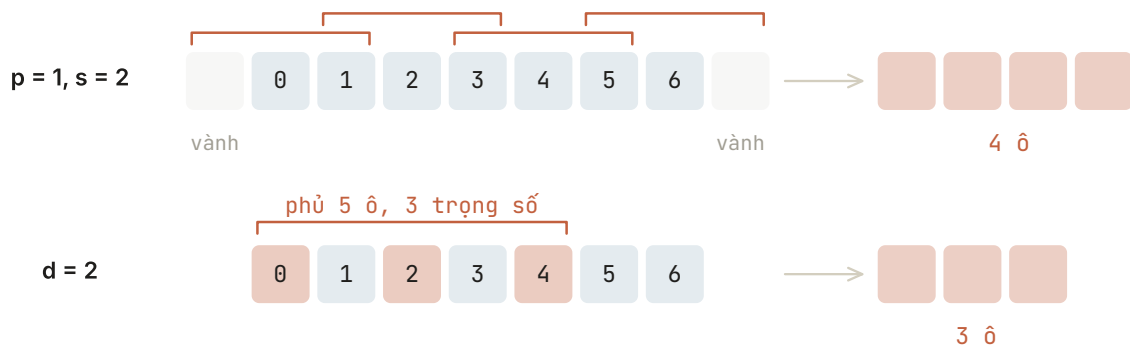
```
def output_size(size, kernel_size, padding=0, stride=1, dilation=1):
    covered = dilation * (kernel_size - 1) + 1
    return (size + 2 * padding - covered) // stride + 1

print(output_size(7, 3, padding=1, stride=2))
print(output_size(7, 3, dilation=2))
```

Dòng thứ hai có dilation 2, nên kernel phủ 5 ô, còn 3 vị trí hợp lệ trên chiều dài 7. Tăng dilation không tăng số tham số của kernel nhưng đổi những điểm ảnh được đọc.

Xếp chồng nhiều lớp cùng dilation 2 thì có những pixel không bao giờ được đọc, để lại lưới lỗ; đổi dilation giữa các lớp hoặc ghép nhiều dilation song song, như ASPP trong

DeepLab, vừa lấp lỗ vừa nhìn ngữ cảnh ở nhiều cỡ. Tăng stride từ 1 lên 2 giữ nguyên số tham số nhưng đầu ra còn khoảng một phần tư số ô.



Padding thêm vành 0, stride dịch nhiều ô, dilation đọc cách ô.

HỎI NHANH

Ảnh dài 8 ô, kernel 3, padding 0, stride 2, dilation 1. Đầu ra dài bao nhiêu?

A 2

B 3

C 4

D 6

HỎI NHANH

Conv có kernel 3, dilation 2, stride 1. Padding đối xứng bằng bao nhiêu thì giữ nguyên chiều dài đầu vào?

A 1

B 2

C 3

D 4

Từ một bảng sang nhiều kênh

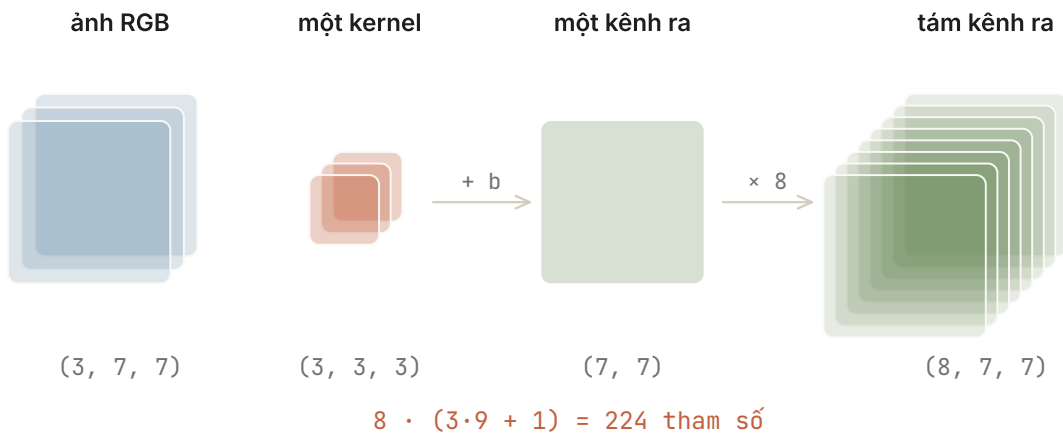
Một ảnh RGB có ba giá trị tại mỗi tọa độ. Một kernel cho **một kênh đầu ra** phải đọc cả ba kênh đầu vào, cộng kết quả rồi cộng một bias. Muốn tám kiểu bộ dò thì dùng tám kernel độc lập.

Đại lượng	Shape theo PyTorch
Batch ảnh	(N, C_{in}, H, W)
Trọng số convolution thông thường	$(C_{out}, C_{in}, k_h, k_w)$
Bias	$(C_{out},)$
Feature map	$(N, C_{out}, H_{out}, W_{out})$

N là số ảnh, không phải số kênh. Với $groups=1$, số tham số có bias bằng $C_{out}(C_{in}k_hk_w + 1)$. Một lớp $RGB \rightarrow 8$ kênh, kernel 3×3 có $8(3 \cdot 9 + 1) = 224$ tham số, dù ảnh lớn hay nhỏ. Chi phí tính vẫn tăng theo số vị trí đầu ra; ít tham số không có nghĩa là chạy miễn phí.

Tương tự, `nn.Conv2d(3, 16, 3)` có $16(27 + 1) = 448$ tham số, `nn.Conv2d(3, 64, 3)` có $64 \cdot 28 = 1792$, và `nn.Conv2d(3, 16, kernel_size=5).weight` có shape $(16, 3, 5, 5)$.

Kernel 1×1 không nhìn hàng xóm mà chỉ trộn kênh tại từng vị trí: khối $28 \times 28 \times 256$ qua 64 bộ lọc 1×1 thành $28 \times 28 \times 64$. Tích chập theo chiều sâu (depthwise, groups bằng số kênh) làm ngược lại: mỗi kênh một kernel riêng, trộn không gian trong từng kênh mà không trộn giữa các kênh; ghép depthwise với 1×1 là khối quen thuộc của MobileNet, ConvNeXt.



Mỗi kernel đọc cả ba kênh RGB; tám kernel cho tám kênh ra, 224 tham số.

Đối chiếu bảng tính tay với thư viện trước, rồi xem shape của một lớp nhiều kênh:

Python

```
import torch
from torch import nn

torch.set_num_threads(1)
x = torch.tensor(image, dtype=torch.float32)[None, None]
layer = nn.Conv2d(1, 1, kernel_size=2, bias=False)
with torch.no_grad():
    layer.weight.copy_(torch.tensor(kernel, dtype=torch.float32)[None, None])
    y = layer(x)
print(tuple(y.shape))
print(np.allclose(y[0, 0].numpy(), feature_map))

rgb_layer = nn.Conv2d(3, 8, 3, padding=1)
print(sum(parameter.numel() for parameter in rgb_layer.parameters()))
print(tuple(rgb_layer(torch.zeros(2, 3, 7, 7)).shape))
```

True xác nhận ta và thư viện dùng cùng quy ước. Shape cuối giữ 7×7 vì kernel 3, padding 1, stride 1. Hai ảnh cùng chạy qua tám bộ lọc, nên đầu ra có hai ảnh, tám kênh.

HỎI NHANH

`nn.Conv2d(16, 32, kernel_size=(3, 5))` có `weight.shape` là gì?

A (16, 32, 3, 5)

B (32, 3, 5)

C (32, 16, 3, 5)

D (32, 16, 5, 3)

HỎI NHANH

Lớp `nn.Conv2d(3, 8, 3, padding=1)` chạy trên ảnh 7×7 , rồi chạy trên ảnh 70×70 . Số tham số và số phép nhân thay đổi thế nào?

A Tham số vẫn là 224, phép nhân tăng 100 lần

B Tham số vẫn là 224, phép nhân cũng giữ nguyên

C Tham số tăng 100 lần, phép nhân tăng 100 lần

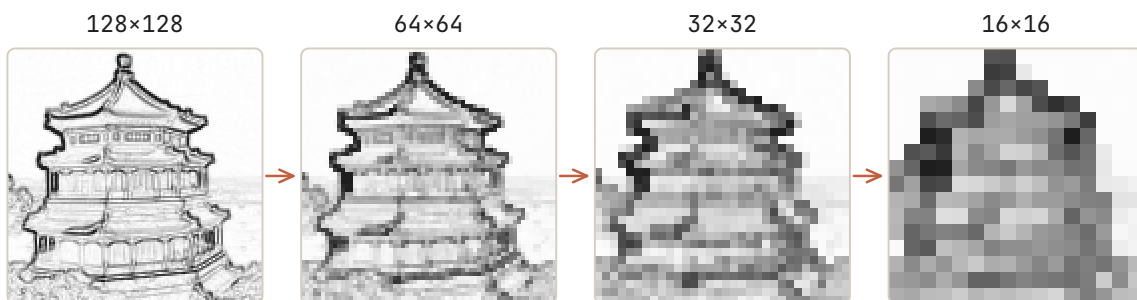
D Tham số tăng 10 lần, phép nhân tăng 10 lần

Pooling giữ điều gì, mất điều gì?

Sau khi biết một vùng có nét, đôi khi ta không cần vị trí chính xác từng ô. Pooling gộp mỗi cửa sổ thành một số, và không có trọng số cần học:

- **Max pooling** lấy giá trị lớn nhất trong cửa sổ; **average pooling** lấy trung bình. Trên feature map $\begin{bmatrix} 3 & 8 \\ 2 & 1 \end{bmatrix}$, max pooling 2×2 ra 8, average pooling ra $14/4 = 3.5$.
- **Global pooling**: cửa sổ phủ cả feature map, mỗi kênh chỉ còn một số. Global max trả lời “mẫu có xuất hiện ở đâu đó không”; global average đo mẫu phủ nhiều hay ít.

MaxPool2d(2) trên 7×7 ra 3×3 vì hàng, cột lẻ cuối bị bỏ; ceil_mode=True giữ cửa sổ thiếu nên ra 4×4 . Thay pooling bằng tích chập stride 2 cũng thu nhỏ map, nhưng cách gộp được học thay vì cố định.



mỗi lần max pool 2×2 giữ một số trong bốn: nét còn, vị trí chính xác mất dần

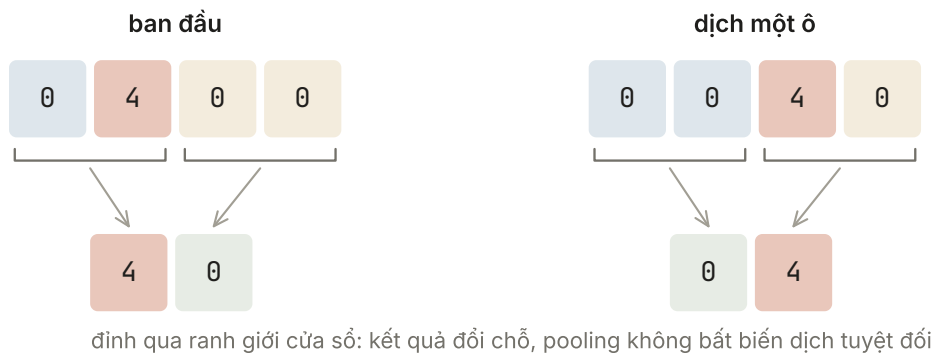
Ba lần max pool 2×2 : từ 128×128 xuống 16×16 , nét còn nhưng chi tiết mất dần.

Python

```
print(nn.MaxPool2d(2)(y).flatten().tolist())
print(nn.AvgPool2d(2)(y).flatten().tolist())
left = torch.tensor([[[[0., 4., 0., 0.]]]])
right = torch.tensor([[[[0., 0., 4., 0.]]]])
pool = nn.MaxPool2d((1, 2))
print(pool(left).flatten().tolist())
print(pool(right).flatten().tolist())

window = y.detach().clone().requires_grad_(True)
nn.MaxPool2d(2)(window).sum().backward()
print(window.grad[0, 0].tolist())
```

Dòng ba và bốn cho thấy dịch một ô có thể đưa đỉnh sang cửa sổ khác. Pooling có thể giảm nhảy với dịch nhỏ **bên trong cùng cửa sổ**, nhưng không tạo bất biến dịch tuyệt đối. Với segmentation, làm mất vị trí như vậy còn phải được bù bằng decoder hoặc nhánh giữ chi tiết.



Định dịch một ô qua ranh giới cửa sổ thì đầu ra max pool đổi từ (4, 0) thành (0, 4).

Dòng cuối là gradient khi lan truyền ngược qua max pooling. Chỉ ô đạt max (ô chứa 8) nhận gradient 1, ba ô kia nhận 0: trong bước học này, chỉ ô thắng truyền tín hiệu về các lớp trước. Average pooling thì chia đều, mỗi ô nhận $1/4$. Khi các cửa sổ chồng nhau, một ô có thể thuộc nhiều cửa sổ và gradient của các cửa sổ ấy được cộng lại.

HỎI NHANH

Max pooling trên một cửa sổ 2×2 chỉ trả về số 8. Từ đó có khôi phục duy nhất được bốn ô ban đầu không?

- A Có, mỗi ô bằng $8/4 = 2$
- B Có, nếu biết thêm stride của pooling
- C Có, nếu biết thêm average pooling của cửa sổ
- D Không, vì nhiều cửa sổ khác nhau có cùng giá trị lớn nhất

HỎI NHANH

MaxPool1d(kernel_size=2, stride=1) trên [1, 9, 3] cho ra [9, 9]. Lấy tổng đầu ra rồi gọi backward(). Gradient về đầu vào là gì?

A [0.5, 1, 0.5]

B [0, 9, 0]

C [0, 1, 0]

D [0, 2, 0]

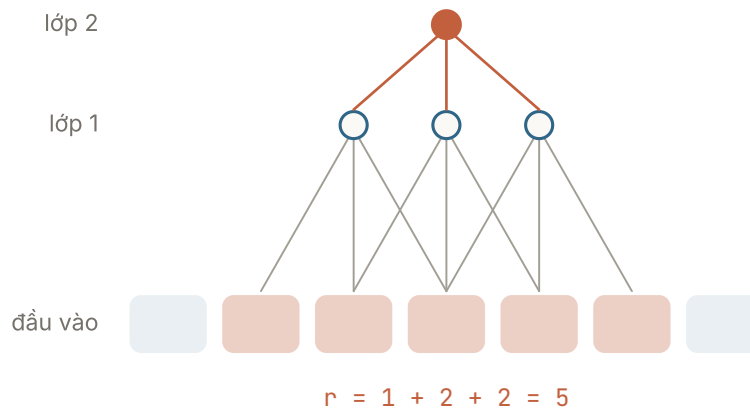
Receptive field: một ô đã nhìn xa đến đâu?

Vùng tiếp nhận (receptive field) là vùng đầu vào có thể ảnh hưởng đến một ô ở lớp đang xét. Hai kernel 3×3 stride 1 cho vùng 5×5 , vì ba ô cạnh nhau ở lớp thứ nhất có các vùng nhìn chồng lấn. Đây là vùng **có thể** ảnh hưởng theo cấu trúc; trọng số hoặc ReLU có thể làm ảnh hưởng thực tế nhỏ hơn.

Để tính chuỗi lớp, giữ hai số: r là chiều dài vùng nhìn và j là khoảng cách trên ảnh gốc giữa hai ô đầu ra kề nhau. Bắt đầu $r = 1, j = 1$. Với lớp kernel k , dilation d , stride s , cập nhật:

$$r_{\text{mới}} = r_{\text{cũ}} + (k - 1)dj_{\text{cũ}}, \quad j_{\text{mới}} = s j_{\text{cũ}}.$$

Một conv 3, pool 2 stride 2, rồi conv 3 cho $r : 1 \rightarrow 3 \rightarrow 4 \rightarrow 8, j : 1 \rightarrow 1 \rightarrow 2 \rightarrow 2$. Conv cuối đọc ba ô cách nhau hai ô trên ảnh gốc, nên thêm 4 vào vùng nhìn. Công thức đo độ rộng; muốn biết vị trí chính xác ở biên còn phải theo dõi padding. Ba lớp 3×3 stride 1 cho 7×7 ; nếu lớp đầu có stride 2 thì j thành 2 và ba lớp cho 11×11 . Kernel 1×1 không làm vùng nhìn rộng thêm.



Hai lớp kernel 3: mỗi ô ở lớp 2 nhìn được 5 ô đầu vào.

Xếp chồng kernel nhỏ là cách thường dùng để mở vùng nhìn. Hai lớp 3×3 nhìn 5×5 như một lớp 5×5 , nhưng mỗi cặp kênh vào-ra chỉ cần $2 \cdot 9 = 18$ trọng số thay vì 25, và có thêm một activation ở giữa. Vì thế các họ mạng như VGG, ResNet chủ yếu dùng kernel 3×3 .

Vùng nhìn lớn dần cũng giải thích vì sao feature map khác nhau theo độ sâu. Lớp đầu chỉ nhìn vài pixel nên phản ứng với cạnh, nét, chỗ đổi màu; feature map của nó còn giống ảnh gốc đã qua bộ lọc. Sau vài lần stride hoặc pooling, mỗi ô nhìn một vùng rộng, map nhỏ và thô đi, phản ứng với tổ hợp lớn hơn như góc, vân, bộ phận của vật. Cấu trúc cho phép thứ bậc ấy xuất hiện; chính huấn luyện mới tạo ra nó.

HỎI NHANH

Một conv kernel 3 stride 2, tiếp theo là một conv kernel 3 stride 1, cả hai có dilation 1. Một ô ở lớp thứ hai nhìn bao nhiêu ô đầu vào theo mỗi chiều?

A 5

B 6

C 7

D 9

Giả định và ứng dụng

Convolution giả định các mẫu cục bộ hữu ích và cùng mẫu có thể xuất hiện ở nhiều vị trí. Điều này hợp với cạnh, góc, vân; chưa đủ để tự hiểu quan hệ “quyển sách bên trái cửa sổ”. Ta cần nhiều lớp để ghép chi tiết thành cấu trúc, hoặc cơ chế khác để trao đổi thông tin xa như attention ở bài 5.4.

Với stride 1, dịch ảnh đi một ô thì feature map dịch theo đúng một ô, các giá trị giữ nguyên trừ vùng sát biên. Tính chất này gọi là đồng biến với phép dịch (translation equivariance).

Bất biến, tức đầu ra không đổi khi ảnh dịch, chỉ xuất hiện khi về sau có phép gộp vị trí như global pooling; pooling cục bộ chỉ giảm nhạy với dịch nhỏ và vỡ khi đỉnh qua ranh giới cửa sổ, như cặp $[4.0, 0.0]$ và $[0.0, 4.0]$ ở trên. CNN cũng không tự bất biến với xoay, đổi tỉ lệ hay đổi ánh sáng.



Dịch ảnh 20 px thì feature map dịch theo đúng 20 px.

Giả định “mẫu có thể ở bất kỳ đâu” yếu đi khi ảnh đã được căn chỉnh, ví dụ ảnh thẻ mà mắt luôn ở cùng một chỗ. Ngược lại, khi đề cần tọa độ tuyệt đối như vị trí một nguồn hay một keypoint, global pooling xóa đúng thông tin cần tìm; head phải đọc feature map còn chiều không gian, hoặc input được thêm hai kênh chứa tọa độ x, y của từng ô.

Quay lại nét phần: bộ dò cục bộ giúp tái sử dụng tri thức khi nét dịch chỗ. Ta vẫn phải kiểm ảnh của nhiều học sinh, nhiều camera, nhiều độ sáng. Nếu dữ liệu train chỉ có nét ở giữa ảnh, bộ lọc dùng chung chưa bảo đảm classifier cuối không học thói quen “nét luôn ở giữa”.

Chia sẻ trọng số là một giả định: bớt tham số nên giảm variance, đổi lại thêm bias khi giả định sai. Lớp kết nối đầy đủ cùng số đơn vị vào, ra biểu diễn được mọi thứ tích chập làm được, chỉ cần nhiều dữ liệu hơn hẳn để học; khi ảnh đã căn chỉnh sẵn, vật luôn cùng cỡ và cùng chỗ, lợi thế của tích chập nhỏ đi.

HỎI NHANH

Một bộ dò nét đã học tốt trên những ảnh chỉ có nét dựng đứng, ở đủ mọi vị trí. Ảnh test bị xoay 90° nên nét thành nằm ngang. Nhiều khả năng chuyện gì xảy ra?

- A Nhận kém, vì kernel nét dọc phản ứng yếu với nét ngang
- B Nhận kém, vì xoay làm đổi shape (N, C, H, W)
- C Vẫn nhận tốt, vì dùng chung kernel ở mọi vị trí
- D Vẫn nhận tốt, nếu thêm padding cho ảnh đã xoay

Khi đi thi

- Đưa một batch giả `torch.zeros(2, C, H, W)` qua model và in shape từng khối trước khi nạp dữ liệu thật.
- Dữ liệu dạng (H, W, C) thì permute về (N, C, H, W), rồi so giá trị một pixel trước và sau.
- Để cần tọa độ hay mask thì giữ chiều không gian, dùng global pooling.

Tóm tắt

- Convolution đọc cửa sổ cục bộ bằng trọng số dùng chung: số tham số không phụ thuộc kích thước ảnh, còn chi phí tính thì có.
- Kernel cổ điển như Sobel, làm mờ được đặt tay; giãn và co lấy max, min trên cửa sổ để làm dày, làm mỏng vật trên ảnh nhị phân.
- Padding, stride và dilation quyết định vị trí đọc và kích thước đầu ra: $\lfloor (H + 2p - d(k - 1) - 1)/s \rfloor + 1$.
- Pooling gộp cửa sổ mà không học trọng số; max pooling chỉ gửi gradient về ô thắng.
- Receptive field tăng qua nhiều lớp; xếp chồng 3×3 mở vùng nhìn với ít trọng số.
- Convolution đồng biến với phép dịch, nhưng không tự bất biến với xoay hay đổi tỉ lệ.

Kiểm tra cuối bài

CÂU 1

Ảnh 28×28 đi qua 5 lớp conv 3×3 stride 1 nối tiếp nhau. Kích thước cuối cùng khi không có padding, và khi mọi lớp có padding=1, lần lượt là bao nhiêu?

A 23×23 và 28×28

B 18×18 và 28×28

C 13×13 và 28×28

D 18×18 và 38×38

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 202

CÂU 2

Ảnh 6×6 , kernel 3×3 , không padding, stride 1 cho đầu ra 4×4 . Giữ nguyên kernel, cấu hình nào cho đầu ra đúng 3×3 ?

A Stride 1, padding 1

B Stride 2, padding 0

C Stride 2, padding 1

D Stride 3, padding 0

PHÒNG THEO CMU 10-601 · Exam 3 Practice · Q1.2(a)

CÂU 3

Trên ảnh 32×32 , một kênh vào, một kênh ra, stride 1, bỏ qua bias, so hai lớp: conv kernel 3, dilation 2, padding 2, và conv kernel 5, padding 2. Nhận xét nào đúng?

A Kernel giãn cho ra 34×34 , cả hai đều có 25 trọng số

B Kernel giãn cho ra 34×34 , một bên 9, một bên 25 trọng số

C Cả hai cùng ra 32×32 , đều có 25 trọng số

D Cả hai cùng ra 32×32 , một bên 9, một bên 25 trọng số

CÂU 4

Model nhận batch (8, 3, 32, 32):

```
nn.Sequential(  
  nn.Conv2d(3, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),  
  nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),  
  nn.Flatten(), nn.Linear(IN, 10))
```

IN phải bằng bao nhiêu?

A $32 \cdot 32 \cdot 32 = 32\,768$

B $32 \cdot 16 \cdot 16 = 8192$

C $32 \cdot 8 \cdot 8 = 2048$

D $32 \cdot 4 \cdot 4 = 512$

PHÒNG THEO NOAI 2025 Singapore · Phần 1 · Câu 11

CÂU 5

Một conv 1D có một bộ lọc, và z_j lớn nghĩa là có mẫu cần tìm ở vị trí j . Mạng A cho ra $\sigma(\max_j z_j)$, mạng B cho ra $\sigma(\min_j z_j)$. Đầu ra cao nghĩa là “mẫu có ở mọi vị trí” ở mạng nào, và nghĩa là “mẫu có ở ít nhất một vị trí” ở mạng nào?

A A rồi B

B B rồi A

C A ở cả hai

D B ở cả hai

PHÒNG THEO MIT 6.036 · Final, Spring 2022 · Q8

CÂU 6

Hai lớp conv 3×3 , stride 1, không padding. Xét ô $(0, 0)$ của lớp thứ hai, với chỉ số tính từ 0. Nhận xét nào đúng?

- A Đổi pixel đầu vào $(4, 4)$ có thể làm ô này thay đổi
- B Chỉ các pixel trong khối 3×3 đầu tiên ảnh hưởng tới ô này
- C Đổi pixel đầu vào $(5, 5)$ cũng có thể làm ô này thay đổi
- D Mọi pixel đều ảnh hưởng, vì hai lớp nối tiếp nhau

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 214

CÂU 7

Một đội thay một conv 7×7 , 64 kênh vào, 64 kênh ra, bằng ba conv 3×3 từ 64 sang 64 nối tiếp, stride 1, có ReLU ở giữa. So với lớp cũ, bỏ qua bias, điều nào đúng?

- A Vùng nhìn chỉ còn 3×3 , và ít trọng số hơn
- B Vùng nhìn thành 9×9 , và ít trọng số hơn
- C Vùng nhìn vẫn là 7×7 , và ít trọng số hơn
- D Vùng nhìn vẫn là 7×7 , nhưng nhiều trọng số hơn

PHÒNG THEO NOAI 2026 Singapore · Vòng sơ loại · Câu 215

CÂU 8

Đưa một ảnh qua một CNN pretrained, lấy feature map ở năm lớp có độ sâu khác nhau, rồi xáo thứ tự. Dấu hiệu nào giúp xếp lại theo độ sâu đáng tin nhất?

- A Map sáng hơn, tức giá trị lớn hơn, là lớp sâu hơn
- B Map lớn, còn thấy được đường viền của vật, là lớp nông
- C Map càng sắc nét, càng nhiều chi tiết thì càng là lớp sâu
- D Không xếp được, vì map không giữ gì của ảnh

PHÒNG THEO [USA-NA-AIO 2026 · Vòng 1 · Problem 6](#)

CÂU 9

Mọi ảnh xe đều chụp ngang, cùng cỡ, đầu xe luôn quay sang phải, nên bánh xe, cửa kính, đèn luôn ở gần cùng một chỗ. Bỏ qua chi phí tính toán, dùng chung một bộ kernel cho mọi vị trí có còn là lựa chọn rõ ràng nhất không?

- A Có, vì không dùng chung thì không học được nét nào
- B Có, vì dùng chung trọng số tạo ra tính bất biến với phép xoay
- C Không, vì pooling đã bất biến với phép dịch tuyệt đối
- D Không hẳn, vì mỗi vùng có thể có bộ lọc riêng của nó

PHÒNG THEO [CMU 10-601 · Exam 3 Practice · Q1.4\(b\)](#)

CÂU 10

Mỗi mẫu là một tensor $(64, 64, 2)$: vector lực (F_x, F_y) đo trên lưới 64×64 . Phải dự đoán tọa độ nguồn (x_s, y_s) cùng 100 giá trị cường độ. Một đội dùng các conv stride 1, rồi global average pooling, rồi Linear ra 102 số, và lỗi tọa độ rất cao. Thay đổi nào nhằm đúng nguyên nhân?

- A Đổi global average pooling thành global max pooling
- B Tăng gấp đôi số kênh của lớp conv cuối
- C Thêm padding để conv đọc được cả vùng biên
- D Bỏ global pooling, flatten feature map để còn giữ vị trí

PHÒNG THEO USA-NA-AIO 2026 · Vòng 2 · Ngày 1 · Problem 2

Đọc thêm

- [Dive into Deep Learning — Convolutions for Images](#): phân biệt cross-correlation với convolution, và xem một kernel phát hiện cạnh được học lại từ dữ liệu.
- [PyTorch — Conv2d](#): đối chiếu shape, groups, padding="same" và công thức kích thước khi cấu trúc khác ví dụ.
- [Dumoulin và Visin — A guide to convolution arithmetic for deep learning](#): hình vẽ từng trường hợp padding, stride, dilation và kích thước đầu ra tương ứng.
- [Dive into Deep Learning — Pooling](#): xem các phép tổng hợp và vai trò giảm kích thước, rồi tự thử một ảnh có nét mảnh.