

Computer Vision

Deep Convolutional Neural Networks

NGUYỄN HẢI TRIỀU¹

¹ Bộ môn Kỹ thuật phần mềm,
Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

1 Ôn tập: biểu diễn ảnh màu, video

2 Convolutional Neural Networks

Nhắc lại

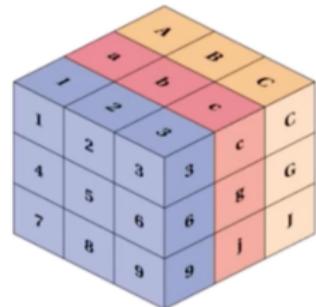
- Dữ liệu biểu diễn 1 chiều → Gọi là Vector → Thông thường biểu diễn dưới dạng cột → Kích thước N
- Dữ liệu biểu diễn 2 chiều → Gọi là Matrix → Kích thước $M \times N$
- Dữ liệu biểu diễn **hơn** 2 chiều → Gọi là Tensor → Nếu là K chiều thì kích thước $M \times N \times K$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

Vector

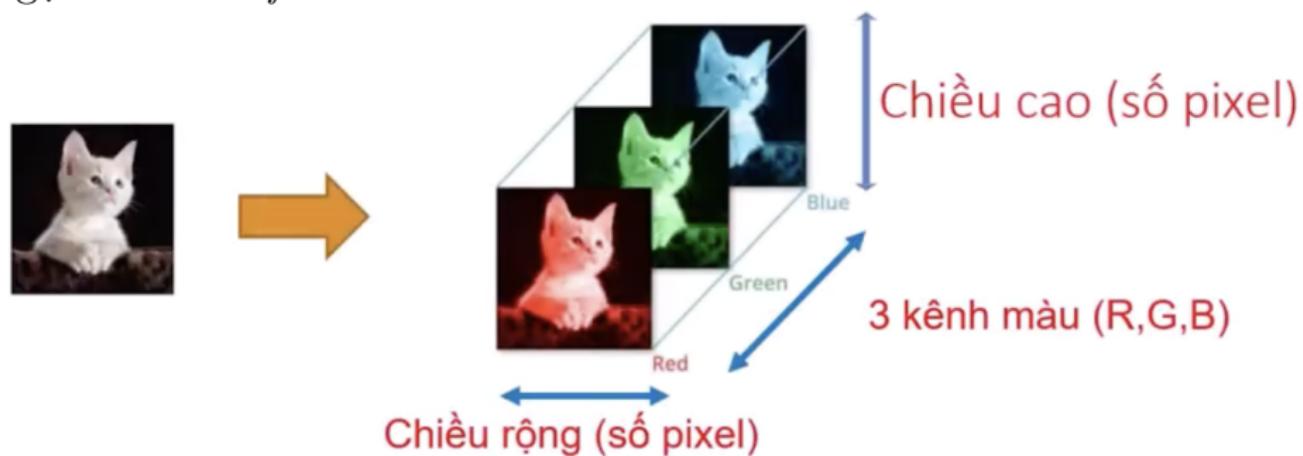
Matrix



Tensor 3D

Biểu diễn ảnh màu

Ảnh màu được biểu diễn bằng **Tensor 3 chiều**. Chúng ta có thể gọi là *stack of matrices*.



Biểu diễn video

Video được biểu diễn bằng **Tensor 4 chiều**



- Dữ liệu ảnh xám là dữ liệu Tensor 2 chiều
- Dữ liệu ảnh màu là dữ liệu Tensor 3 chiều
- Dữ liệu video là dữ liệu Tensor 4 chiều

- 1 Ôn tập: biểu diễn ảnh màu, video
- 2 Convolutional Neural Networks

Làm việc với dữ liệu ảnh/video

Thay vì trích xuất đặc trưng của ảnh bằng tay sang dạng bảng, trong chương này, chúng ta sẽ tìm hiểu các trích xuất đặc trưng của ảnh/video tự động bằng cách sử dụng **Deep Convolutional Neural**.

“Manual” feature engineering



Sepal length	Sepal width	Petal length	Petal width
5.1	3.5	1.4	0.3
4.9	3	1.4	0.2
5.9	3	5.1	1.8
...

Hình 1: Thay vì sử dụng các tiếp cận truyền thống của ML, chúng ta có thể sử dụng ảnh như là đầu vào trực tiếp cho mạng.

Convolutional neural network—**CNN**

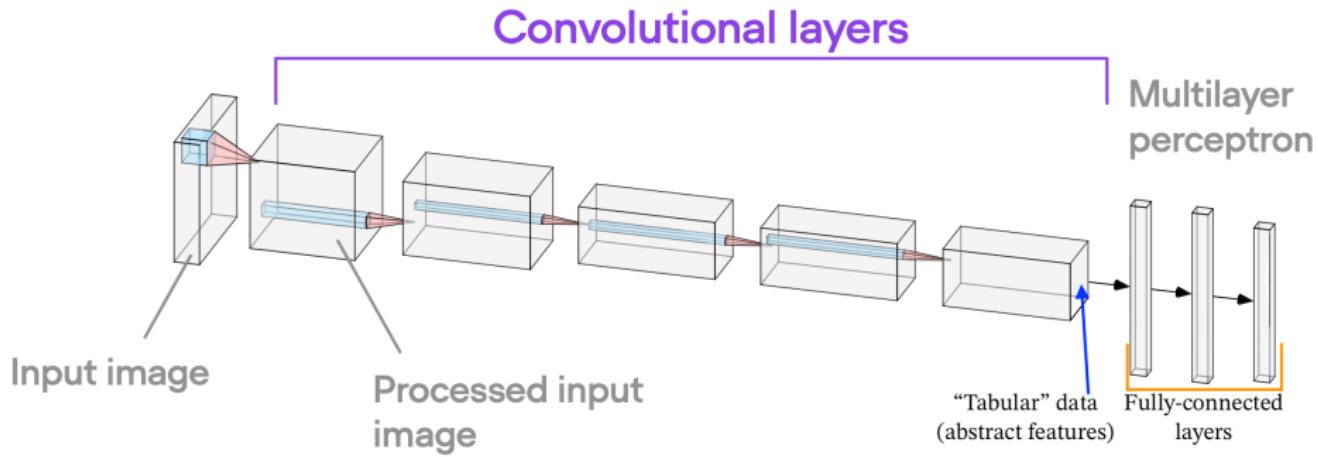
Mạng neural tích chập là gì?

Lớp tích chập là công cụ (feature extractors) dùng để trích xuất đặc trưng của ảnh. Convolutional neural networks form the backbone for many tasks such as

- image classification
- image generation
- object detection
- object segmentation

Cụ thể chúng ta sẽ tìm hiểu ứng dụng của Convolutional neural networks trong bài toán phân loại ảnh.

A typical convolutional neural network architecture



Hình 2: Ví dụ một kiến trúc CNN tiêu biểu cho bài toán phân loại ảnh sử dụng mạng Perceptron. **Mỗi khối trong convolutional layers tương ứng với 1 lớp tích chập.** Lớp fully connected layer cuối cùng là lớp phân loại (xem như multilayer perceptron).

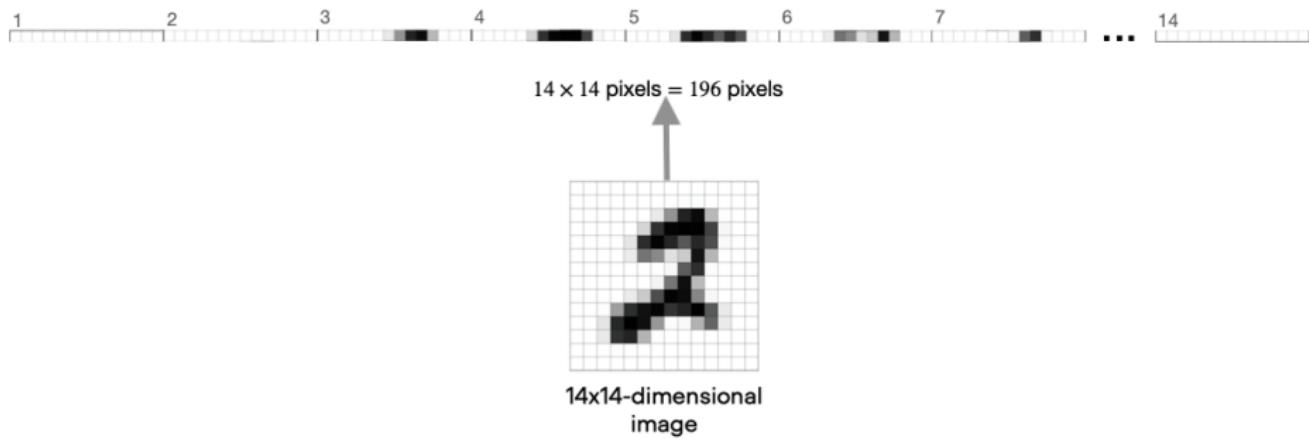
Convolutional neural network—**CNN**

Câu hỏi đặt ra:

- ① Tại sao chúng ta cần sử dụng CNN?
- ② Chỉ cần sử dụng Multilayer Perceptron cho dữ liệu ảnh là đủ?
- ③ CNN có những ưu điểm gì so với Perceptron?

Để trả lời những câu hỏi trên, chúng ta xem xét ví dụ trên dữ bộ dữ liệu ảnh MNIST.

Convolutional neural network—**CNN**



Hình 3: Giả sử một bức ảnh kích thước 14×14 (kích thước thực 28×28) trong tập MNIST được thu lại thành 196-dimensional row vector (bằng cách nối các hàng) để tạo ra một *single training example* áp dụng cho thuật toán *MLP*.

Convolutional neural network—CNN



Hình 4: Giả sử ta có 3 training examples như trên, ta thu được 3 vector đặc trưng theo vị trí của các pixel với các giá trị từ 0-255.

Convolutional neural network—CNN



```

class PyTorchMLP(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super().__init__()

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(num_features, 50),
            torch.nn.ReLU(),
            # 2nd hidden layer
            torch.nn.Linear(50, 25),
            torch.nn.ReLU(),
            # output layer
            torch.nn.Linear(25, num_classes),
        )

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        logits = self.all_layers(x)
        return logits

train_acc = compute_accuracy(model, train_loader)
val_acc = compute_accuracy(model, val_loader)
test_acc = compute_accuracy(model, test_loader)

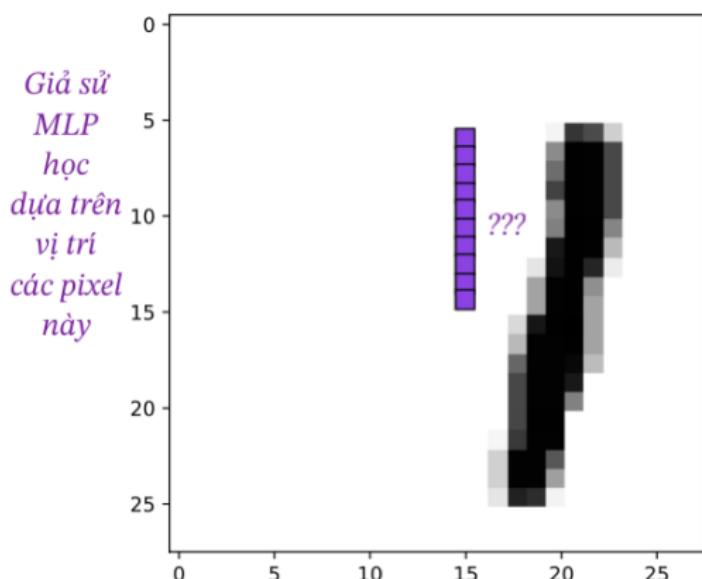
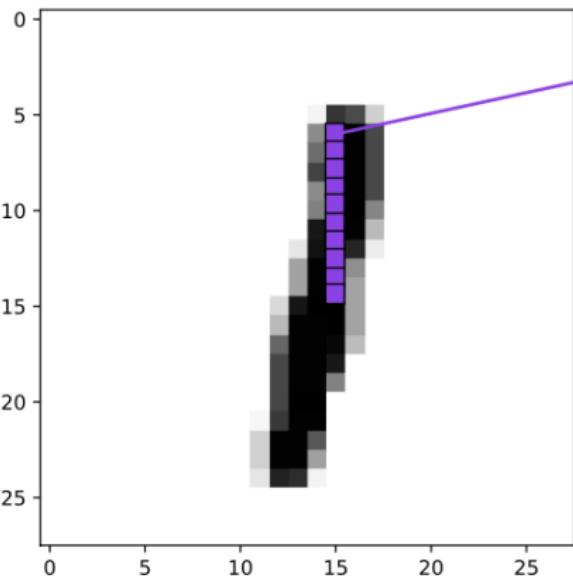
print(f"Train Acc {train_acc*100:.2f}%")
print(f"Val Acc {val_acc*100:.2f}%")
print(f"Test Acc {test_acc*100:.2f}%")

```

Reshapes the
28x28 images to
784-dimensional
vectors

Hình 5: Trong các chương trước, chúng ta đã thấy rằng MLP dễ dàng đạt được $Val\ acc = 96\%$. Tuy nhiên câu hỏi đặt ra: **nhếu vị trí đặc trưng của các con số nằm ở vị trí khác thì sao?**

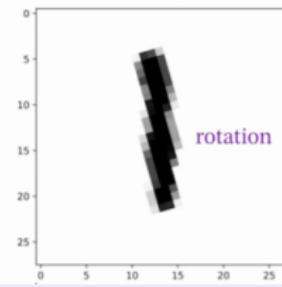
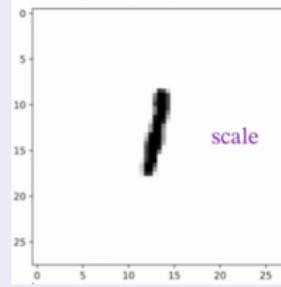
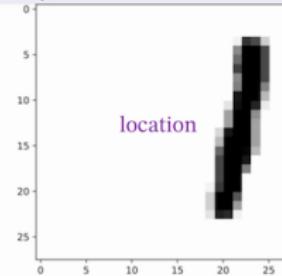
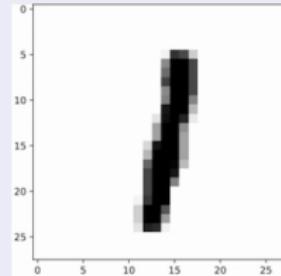
Convolutional neural network—**CNN**



Hình 6: Rõ ràng rằng MLP không thể học được các *đặc trưng không gian* của ảnh khi ta chỉ dời nhẹ vị trí đối tượng sang vị trí khác.

Convolutional neural network—CNN

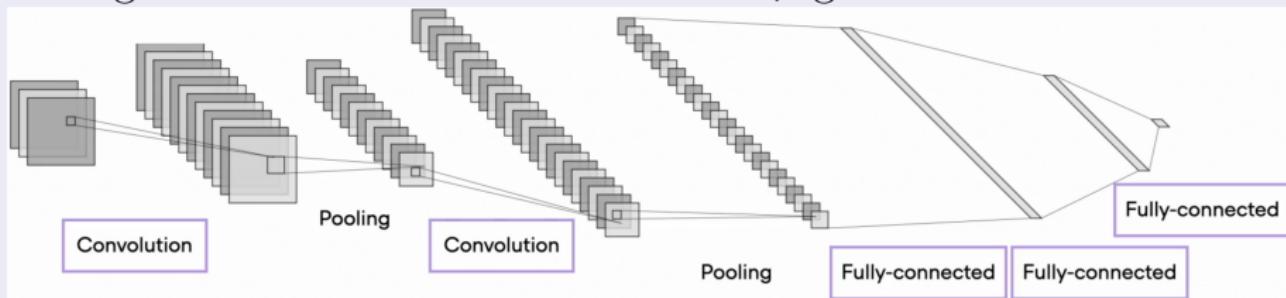
MLP coi các đặc trưng là độc lập với nhau mà không quan tâm đến mối quan hệ không gian giữa các đặc trưng khác. Điều này là không hợp lý với dữ liệu ảnh. Đó là lý do mà chúng ta sử dụng CNN (*take locality into account*).



The Convolutional Network Architecture

Giới thiệu tổng quan về kiến trúc CNN

Chúng ta có thể vẽ chi tiết kiến trúc mạng CNN như sau:



Trong đó, chỉ các lớp convolution, fully-connected là **Trainable layers** (= layers with weight parameters) có thể cập nhật trong quá trình huấn luyện. Chúng ta sẽ đi vào chi tiết các lớp để hiểu rõ hơn trong phần sau.

The Convolutional Network Architecture

Giới thiệu tổng quan về kiến trúc CNN

Thiết kế mạng CNN trong PyTorch gồm *conv_layers* (mới) và *fc_layers* (giống như MLP).

```
import torch.nn as nn

class PyTorchCNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        self.conv_layers = torch.nn.Sequential(
            nn.Conv2d(...),
            nn.MaxPool2d(...),
            nn.Conv2d(...),
            nn.MaxPool2d(...),
        )

        self.fc_layers = nn.Sequential(
            nn.Linear(24 * 16 * 16, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, num_classes),
        )

    def forward(self, x):
        features = self.conv_layers(x)
        features = torch.flatten(features, start_dim=1)
        logits = self.fc_layers(features)
        return logits
```

The Convolutional Network Architecture

Giới thiệu tổng quan về kiến trúc CNN

Thiết kế mạng CNN trong PyTorch gồm 2 phần *conv_layers* (mới) và *fc_layers* (giống như MLP). Kết nối 2 phần là phương thức *forward()*.

```
import torch.nn as nn

class PyTorchCNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        self.conv_layers = torch.nn.Sequential(
            nn.Conv2d(...),
            nn.MaxPool2d(...),
            nn.Conv2d(...),
            nn.MaxPool2d(...),
        )

        self.fc_layers = nn.Sequential(
            nn.Linear(24 * 16 * 16, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, num_classes),
        )

    def forward(self, x):
        features = self.conv_layers(x)
        features = torch.flatten(features, start_dim=1)
        logits = self.fc_layers(features)
        return logits
```

The Convolutional Network Architecture

```

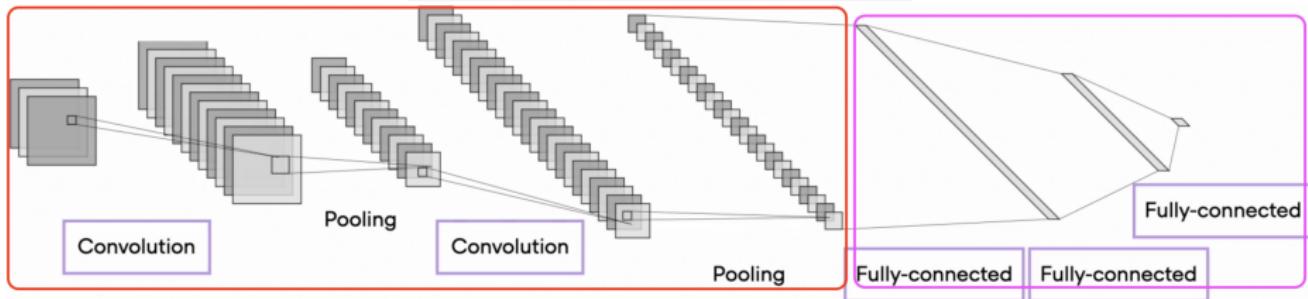
import torch.nn as nn

class PyTorchCNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
            Convolution part
        self.conv_layers = nn.Sequential(
            nn.Conv2d(...),
            nn.MaxPool2d(...),
            nn.Conv2d(...),
            nn.MaxPool2d(...),
        )

            Classifier part
        self.fc_layers = nn.Sequential(
            nn.Linear(24 * 16 * 16, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, num_classes),
        )

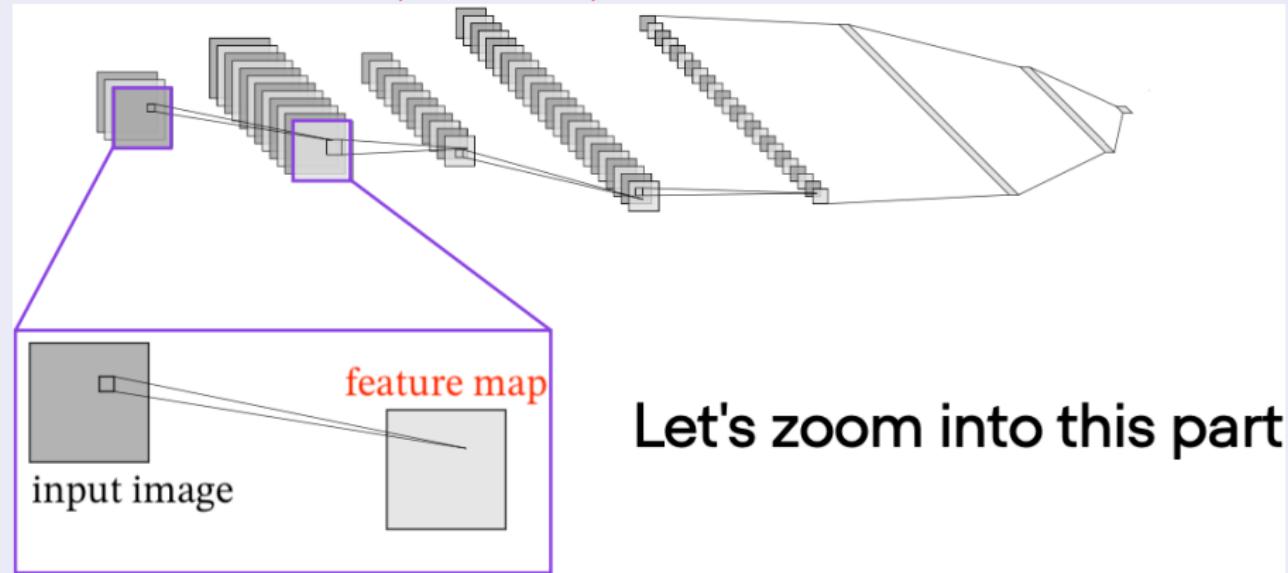
    def forward(self, x):
        features = self.conv_layers(x)
        features = torch.flatten(features, start_dim=1)
        logits = self.fc_layers(features)
        return logits

```



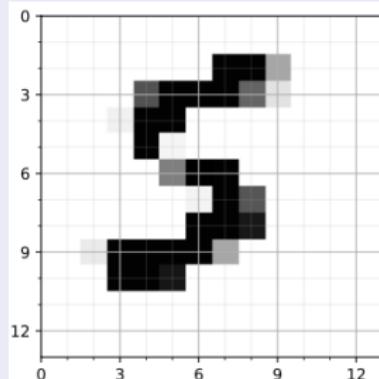
Convolutional Layers

Khi áp dụng lớp convolution vào ảnh đầu vào, chúng ta **thu được các feature maps (đặc trưng) của ảnh**. Vậy *feature maps* là gì?



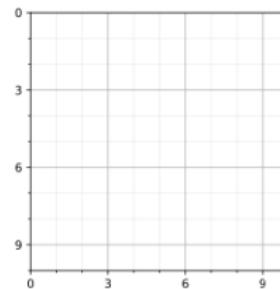
Convolutional Layers

Dể tìm hiểu cách hoạt động của lớp convolution, chúng ta xem xét một ví dụ đơn giản với ảnh là *digit 5* và *kernel/feature detector* 3×3 .



Input (image)

3x3 feature detector (kernel, filter)

A 3x3 grid of squares, representing a 3x3 kernel or feature detector. The squares are arranged in a 3x3 pattern, forming a single unit.

Feature map

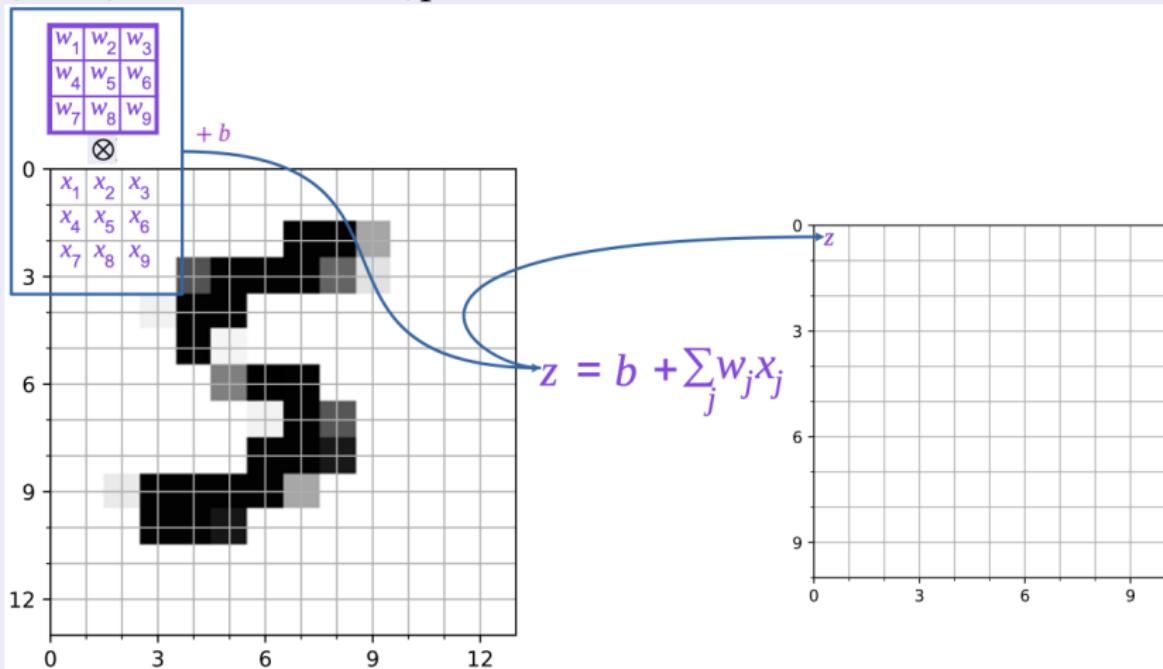
Convolutional Layers

Cách hoạt động của lớp convolution

Convolutional Layers

Cách hoạt động của lớp convolution

Thực hiện nhân tích chập:

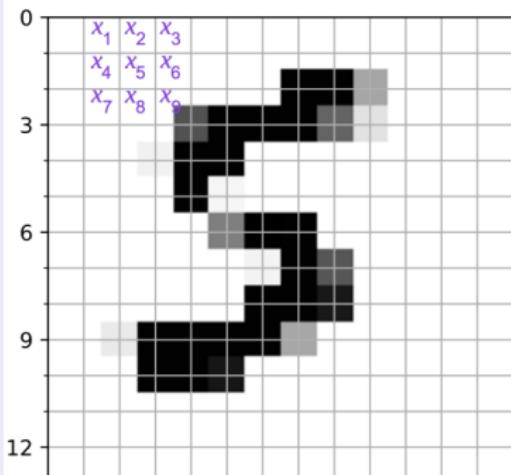


Convolutional Layers

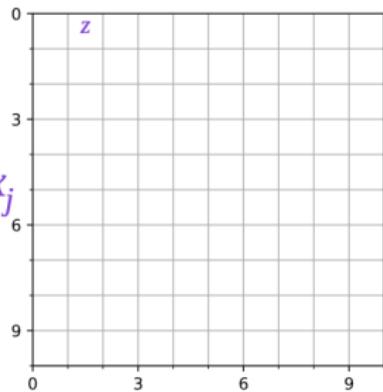
Cách hoạt động của lớp convolution

Lưu ý *weight sharing*: các trọng số w_i không đổi khi trượt qua toàn bộ ảnh đầu vào.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9



$$z = b + \sum_j w_j x_j$$



Convolutional Layers

Cách hoạt động của lớp convolution

Weight sharing có tác dụng:

- giúp giảm số lượng tham số cần học trong mạng CNN so với MLP. Ví dụ trường hợp kernel 3×3 trên ảnh màu 3 kênh RGB kích thước 32×32 và đầu ra sử dụng 64 bộ lọc, $\text{Tổng số tham số} = [(\text{kernel size} \times \text{in_channels}) + \text{bias}] \times \text{out_channels} = [(3 \times 3 \times 3) + 1] \times 64 = 1792$. So với MLP, số lượng tham số cần học là:
 $32 \times 32 \times 3 \times \text{số lượng neural} = 3072 \times 64 = 196608 >> 1792$.
- Một bộ lọc hoạt động tốt ở một vùng có thể cũng hoạt động tốt ở vùng khác. Ta có thể dùng nhiều bộ lọc để trích xuất các đặc trưng khác nhau, mỗi bộ lọc học một kiểu đặc trưng nhất định.

Convolutional Layers

Khởi tạo Convolutional Layer trong PyTorch

Để tạo ra một lớp convolutional layer trong PyTorch, chúng ta sử dụng `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)`. Trong đó:

- `in_channels`: số lượng kênh đầu vào
- `out_channels`: số lượng kênh đầu ra tương ứng với số lượng bộ lọc cần học.
- `kernel_size`: kích thước kernel
- `stride`: bước trượt
- `padding`: đệm

Convolutional Layers

Khởi tạo Convolutional Layer trong PyTorch

Ví dụ tạo một lớp tích chập với 1 `in_channels` và 1 `out_channels`, kích thước bộ lọc bằng 3×3 trong PyTorch

```
layer = torch.nn.Conv2d(1, 1, kernel_size=3)
```

```
layer.weight
```

Parameter containing:

```
tensor([[[[ 0.0930,  0.1602,  0.1177],  
        [-0.0802, -0.0701, -0.2747],  
        [ 0.1806,  0.2647,  0.2281]]]], requires_grad=True)
```

```
layer.bias
```

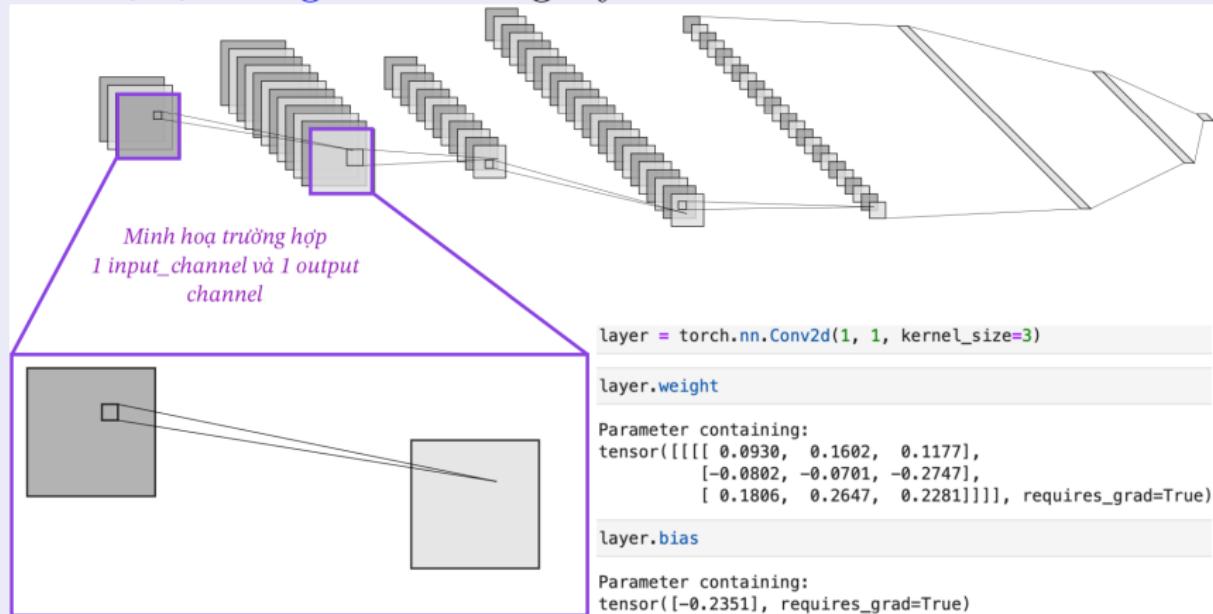
Parameter containing:

```
tensor([-0.2351], requires_grad=True)
```

Convolutional Layers

Khởi tạo Convolutional Layer trong PyTorch

Minh họa kết quả với 1 `in_channels` và 1 `out_channels`, kích thước bộ lọc bằng 3×3 trong PyTorch



Convolutional Layers

Sử dụng convolution với nhiều channels

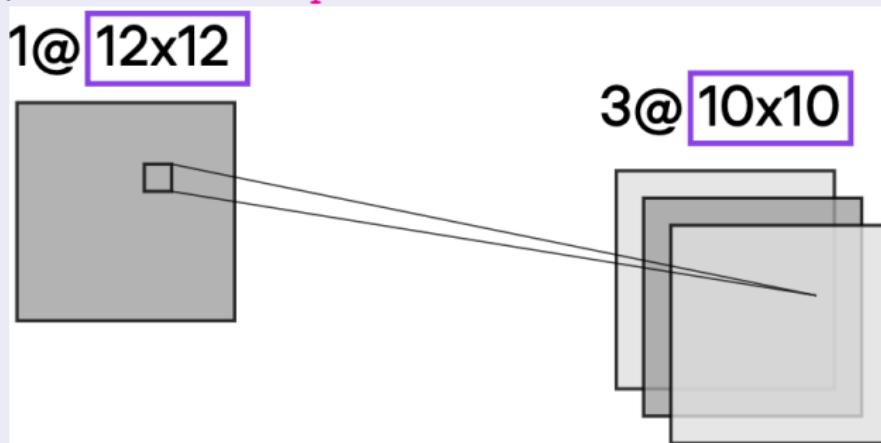
Chúng ta cùng tìm hiểu cách hoạt động của lớp convolution với nhiều **in/out_channels** qua các ví dụ:

- 1 input channel, 3 output channels
- 3 input channels, 1 output channel
- multiple input AND output channels

Convolutional Layers

1 input channel, 3 output channels

Hình minh họa một ảnh đầu vào kích thước 12×12 có 1 channel và thu được “**feature map**” kích thước 10×10 với 3 channels.



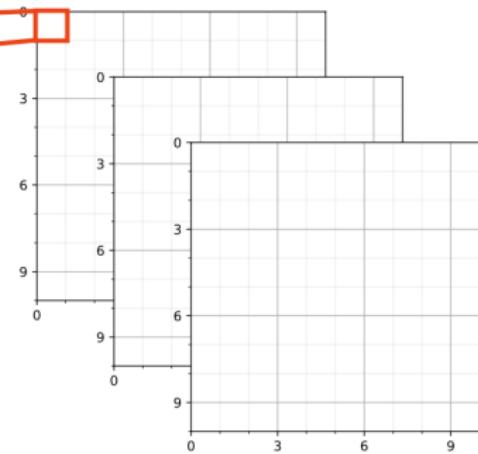
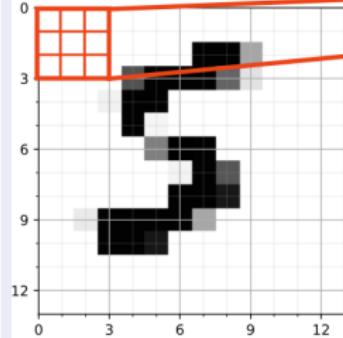
Kí hiệu @ để chỉ số lượng channel.

Convolutional Layers

1 input channel, 3 output channels

Cách thức hoạt động: sử dụng 3 kernels khác nhau để tạo ra 3 feature maps.

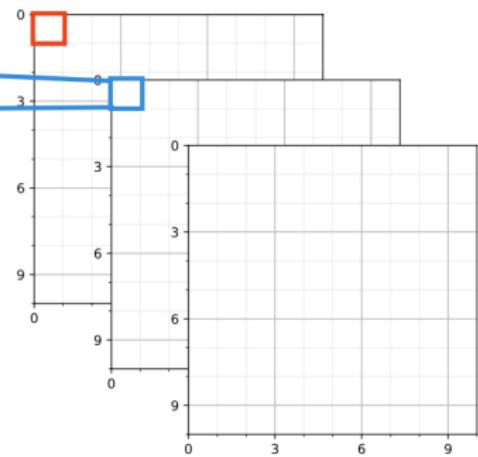
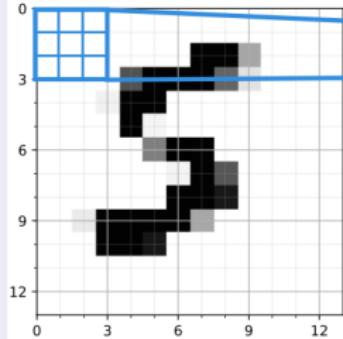
$$z = b^{(1)} + \sum_j w_j^{(1)} x_j$$



Convolutional Layers

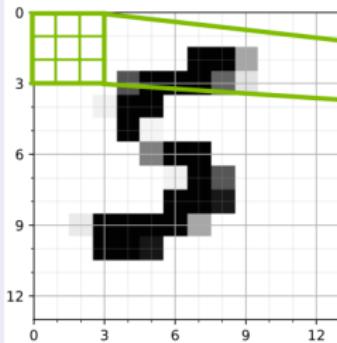
1 input channel, 3 output channels

$$z = b^{(2)} + \sum_j w_j^{(2)} x_j$$

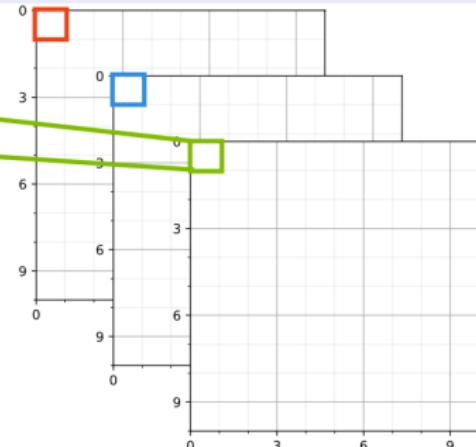


Convolutional Layers

1 input channel, 3 output channels



$$z = b^{(3)} + \sum_j w_j^{(3)} x_j$$



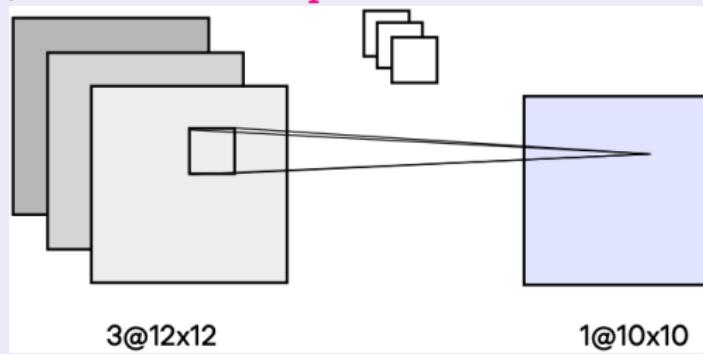
Multiple kernels are used to create “*multiple feature maps*”



Convolutional Layers

3 input channels, 1 output channel

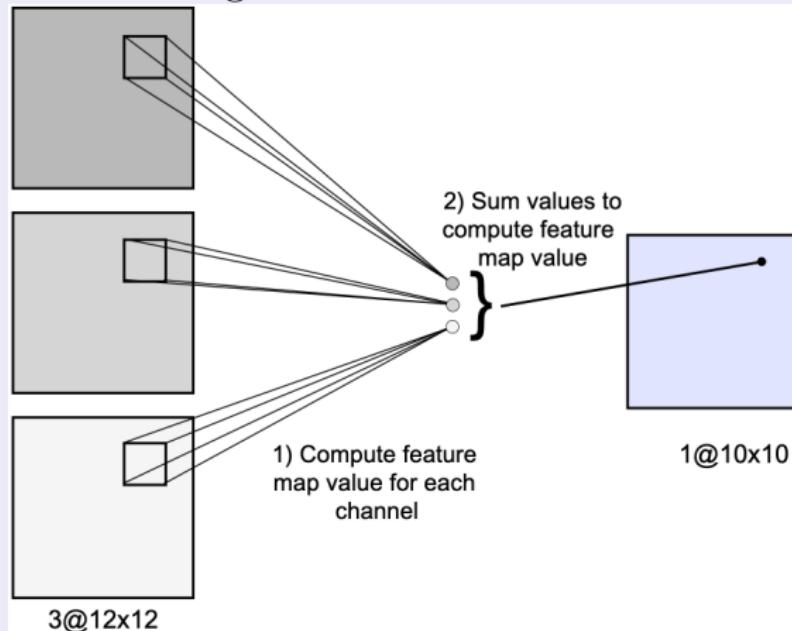
Hình minh họa một ảnh đầu vào kích thước 12×12 có 3 channels và thu được một “**feature map**” kích thước 10×10 với 1 channel.



Convolutional Layers

3 input channels, 1 output channel

Cách thức hoạt động: sử dụng 1 kernel, chúng ta cần thực hiện tích chập kernel với từng channel đầu vào và sau đó cộng lại.

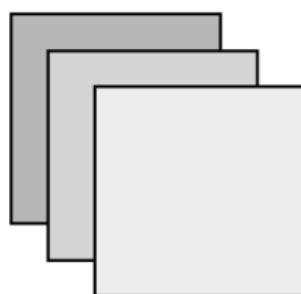


Convolutional Layers

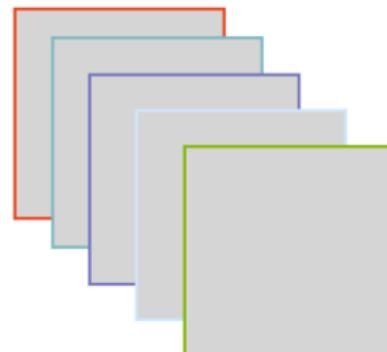
Multiple input AND output channels

Hình minh họa một ảnh đầu vào kích thước 64×64 có 3 channels và thu được “**feature map**” kích thước 64×64 với 5 channel.

3 input channels 5 output channels



3@64x64



5@64x64

5 kernels with
3 channels each

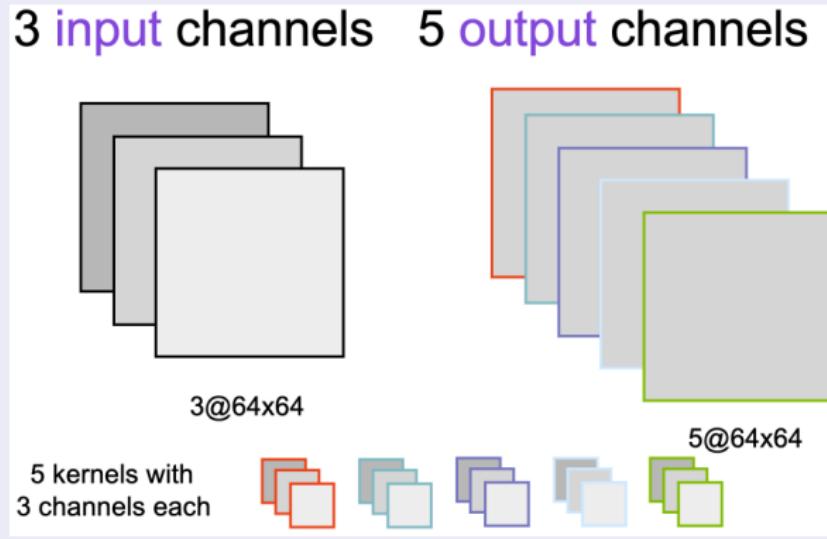


Convolutional Layers

Multiple input AND output channels

Cách thức hoạt động: sử dụng 5 kernels khác nhau để tạo ra 5 feature maps. Mỗi kernel sẽ có 3 channels đầu vào và 1 channel đầu ra.

3 **input channels** 5 **output channels**



Convolutional Layers

Multiple input AND output channels

Kiểm tra bằng PyTorch với 3 channels đầu vào và 5 channels đầu ra: `Conv2d(in_channels=3, out_channels=5, kernel_size=2, stride=1, padding=1)`. Ta thu được trọng số có kích thước $5 \times 3 \times 2 \times 2$.

```
import torch

layer = torch.nn.Conv2d(in_channels=3, out_channels=5, kernel_size=2)
layer.weight.shape

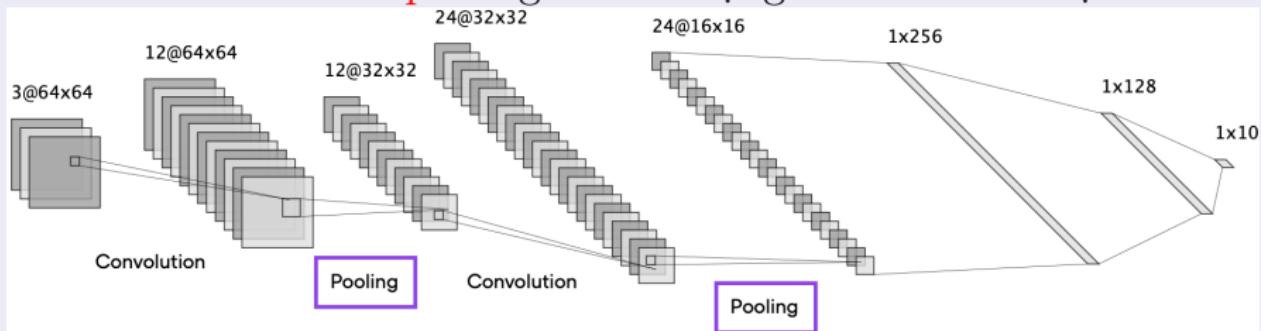
torch.Size([5, 3, 2, 2])
```

5 kernels with
3 channels each



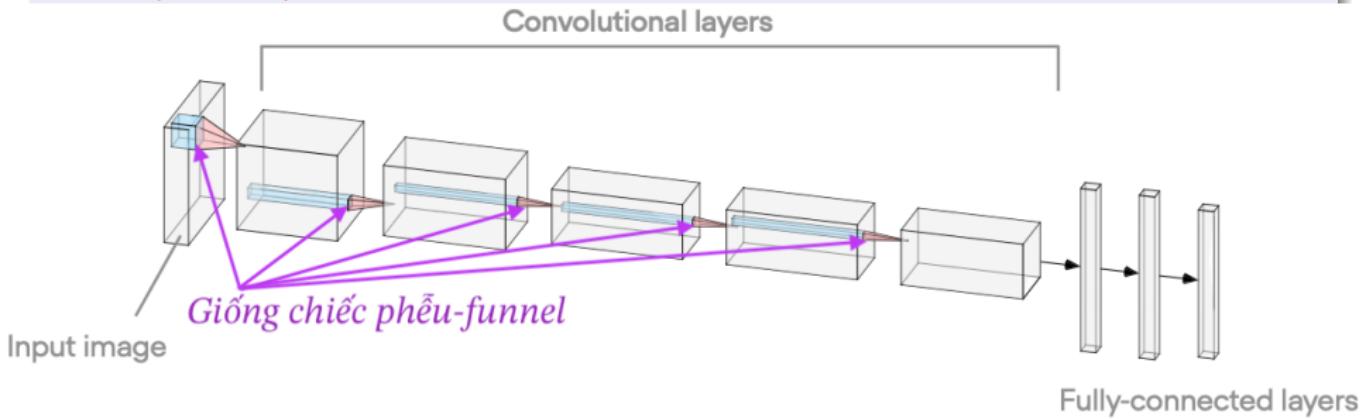
Pooling Layers

Kiến trúc tổng quan của một mạng CNN bao gồm các lớp convolutional layers và pooling layers. Lớp pooling giúp giảm kích thước của feature maps và giảm số lượng tham số cần học.



Pooling Layers

Để đơn giản, ta có thể xem *kiến trúc CNN giống như một cái phễu (funnel)*.

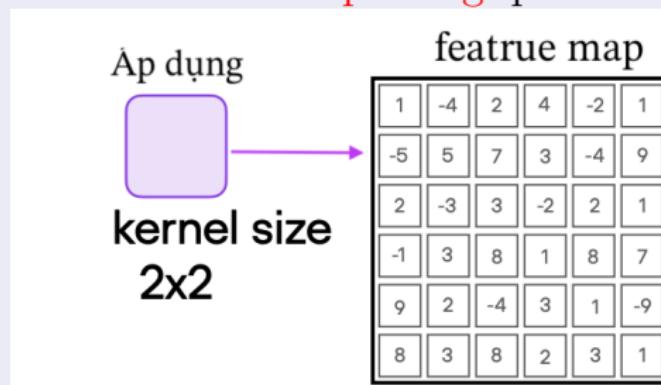


Cụ thể, chúng ta:

- ① **tăng số lượng channels** qua các *lớp convolutional layers*
- ② **giảm kích thước của feature maps** qua các *lớp pooling layers*

Pooling Layers

Có 2 loại pooling layers phổ biến là **Max pooling** và **Average pooling**. Chúng ta tìm hiểu **Max pooling** qua ví dụ sau:



Pooling Layers

Chúng ta tìm hiểu **Max pooling** qua ví dụ sau:

Pooling Layers

Chúng ta tìm hiểu **Average pooling** qua ví dụ sau:



Lưu ý

Các lớp Pooling **không** có tham số cần học trong quá trình huấn luyện mạng, nghĩa là *không có các trọng số cần cập nhật trong quá trình lan truyền ngược*.

Pooling Layers

Lưu ý

Ngoài ra, lớp Pooling còn giúp tăng tính bất biến cục bộ (local invariance) của mạng. Điều này có nghĩa rằng mạng CNN có thể nhận diện một đối tượng ngay cả khi đối tượng đó có sự thay đổi nhỏ về vị trí, góc quay hoặc tỷ lệ.

Tuy nhiên, lớp Pooling có thể làm mất một vài đặc trưng quan trọng của feature map \rightarrow Vì vậy một số kiến trúc hiện đại không sử dụng lớp Pooling mà sử dụng các phương pháp khác như *strided convolution*.

- Khi stride (bước di chuyển của cửa sổ pooling) nhỏ hơn kích thước vùng pooling, thông tin bị mất sẽ ít hơn.

Pooling Layers

Ví dụ *sử dụng và không sử dụng* lớp Max pooling trong PyTorch:

```

1 import torch
2 import torch.nn as nn
3
4 layer_with_pooling = nn.Sequential(
5     nn.Conv2d(1, 16, 3, 1, 1),
6     nn.MaxPool2d(2, 2),
7     nn.Conv2d(16, 32, 3, 1, 1),
8     nn.MaxPool2d(2, 2)
9 )
10 example_input = torch.randn(5, 1, 28, 28)
11 example_output =
12     layer_with_pooling(example_input)
13 print(example_output.size())

```

✓ 0.0s

Sử dụng
Pooling

Python

```

1 import torch
2 import torch.nn as nn
3
4 layer_with_pooling = nn.Sequential(
5     nn.Conv2d(1, 16, 3, 1, 1),
6     # nn.MaxPool2d(2, 2),
7     nn.Conv2d(16, 32, 3, 1, 1),
8     # nn.MaxPool2d(2, 2)
9 )
10 example_input = torch.randn(5, 1, 28, 28)
11 example_output =
12     layer_with_pooling(example_input)
13 print(example_output.size())

```

[126] ✓ 0.0s

... torch.Size([5, 32, 28, 28])

Nhận thấy rằng, khi sử dụng Pooling, kích thước feature map giảm đi đáng kể giúp tiết kiệm chi phí huấn luyện.

Pooling Layers

Chúng ta có thể thay thế *Max pooling* bằng *stride convolution* để giảm kích thước feature map:

```

1 import torch
2 import torch.nn as nn
3
4 layer_with_pooling = nn.Sequential(
5     nn.Conv2d(1, 16, 3, 1, 1),
6     nn.MaxPool2d(2, 2),
7     nn.Conv2d(16, 32, 3, 1, 1),
8     nn.MaxPool2d(2, 2)
9 )
10 example_input = torch.randn(5, 1, 28, 28)
11 example_output =
12     layer_with_pooling(example_input)
13 print(example_output.size())

```

✓ 0.0s Python [131] ✓ 0.0s

`torch.Size([5, 32, 7, 7])`

Sử dụng
Pooling

```

1 import torch
2 import torch.nn as nn
3
4 layer_with_pooling = nn.Sequential(
5     nn.Conv2d(1, 16, 3, 2, 1),
6     # nn.MaxPool2d(2, 2),
7     nn.Conv2d(16, 32, 3, 2, 1),
8     # nn.MaxPool2d(2, 2)
9 )
10 example_input = torch.randn(5, 1, 28, 28)
11 example_output =
12     layer_with_pooling(example_input)
13 print(example_output.size())

```

Không sử
dụng Pooling với Stride = 2

✓ 0.0s Python [131] ✓ 0.0s

`...> torch.Size([5, 32, 7, 7])`

Nhận thấy rằng, kích thước feature map thu được là nhau.

Pooling Layers I

Chúng ta cũng có thể đọc một ảnh màu (“trex.png”), sử dụng CNN để thu được các feature map như code minh họa bên dưới:

```
1 import torch
2 import torch.nn as nn
3 from PIL import Image
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from torchvision import transforms
7
8 layer_with_pooling = nn.Sequential(
9     nn.Conv2d(3, 16, 3, 1, 1), # 3 là số kênh ảnh màu của ảnh đầu vào
10    nn.MaxPool2d(2, 2),
11    nn.Conv2d(16, 32, 3, 1, 1),
12    nn.MaxPool2d(2, 2)
13 )
14 # chuyển ảnh màu thành tensor sử dụng transforms từ torchvision
15 transform = transforms.Compose([
```

Pooling Layers II

```
16     transforms.Resize((64, 64)),  
17     transforms.ToTensor()  
18 ])  
19 # Đọc ảnh bằng thư viện PIL  
20 img = Image.open("trex.png")  
21 img = transform(img)  
22  
23 # plt.imshow(np.transpose(img.detach().numpy(), (1, 2, 0)))  
24 example_input = torch.stack([img],dim=0)  
25 example_output = layer_with_pooling(example_input)  
26 print("Kích thước feature map khi qua CNN:",example_output.size())  
27 # hiển thị ảnh đầu vào đã transform  
28 plt.figure(figsize=(2, 2))  
29 plt.title('Input: Example 0')  
30 plt.imshow(np.transpose(img.detach().numpy(),(1,2,0)))  
31  
32 # hiển thị feature map của dữ liệu mẫu đầu tiên  
33 plt.figure(figsize=(10, 7))  
34 plt.title('Output: Example 0')  
35 plt.axis('off')
```

Pooling Layers III

```
36
37     for i, _ in enumerate(example_output):
38         for j, _ in enumerate(example_output[i]):
39             # vẽ example_output[i][j] trên lưới có 8 cột và 4 hàng (do 8*4 = 32 =
40             # số feature map cuối cùng)
41             plt.subplot(4, 8, j+1)
42             plt.imshow(example_output[i,j].detach().numpy(), cmap='gray')
43             plt.axis('off')
44             plt.title(f'channel {j+1}')
45         break
46     plt.show()
```

Pooling Layers IV



Hình 7: Kết quả feature map của ảnh màu “trex.png” sau khi qua CNN.

Pooling Layers V

Chi tiết code xem ở file “*Chuong_6.0_pooling_setup.pdf*”

Controlling The Output Size With Padding

Padding thường được sử dụng trong mạng CNN nhằm mục đích:

- giữ nguyên kích thước input/output của ảnh.
- giảm mất thông tin ở biên của ảnh khi thực hiện tích chập.

Padding thường sẽ thêm các giá trị 0 xung quanh biên của ảnh đầu vào trước khi qua lớp tích chập.

1	-4	2	4	-2
-5	5	7	3	-4
2	-3	3	-2	2
-1	3	8	1	8
9	2	-4	3	1

image with no padding

0	0	0	0	0	0	0
0	1	-4	2	4	-2	0
0	-5	5	7	3	-4	0
0	2	-3	3	-2	2	0
0	-1	3	8	1	8	0
0	9	2	-4	3	1	0
0	0	0	0	0	0	0

padding = 1

Padding

Padding

Feature map size

Để tính kích thước của feature map sau khi qua lớp convolutional hoặc pooling, ta sử dụng công thức sau:

$$O = \frac{W - K + 2P}{S} + 1.$$

Trong đó: W is the **input width**, K is the **kernel width**, P is the **padding**, S is the **stride**, and O is the **output width**. Tính toán tương tự cho output height.

Feature map size

Ví dụ so sánh sử dụng padding 1: $O = \frac{100-3+2\times1}{1} + 1 = 100$ và không sử dụng padding 0: $O = \frac{100-3+2\times0}{1} + 1 = 98$

```

1 import torch
2 import torch.nn as nn
3
4 layer = nn.Conv2d(1, 1, 3, 1, 0)
5 example = torch.randn(1, 100, 100)
6 Output = layer(example)
7 print(Output.shape)      Padding 0

```

✓ 0.0s

`torch.Size([1, 98, 98])`

Python

```

1 import torch
2 import torch.nn as nn
3
4 layer = nn.Conv2d(1, 1, 3, 1, 1)
5 example = torch.randn(1, 100, 100)
6 Output = layer(example)
7 print(Output.shape)      Padding 1

```

[5] ✓ 0.0s

`... torch.Size([1, 100, 100])`

Nhận thấy rằng, nếu sử dụng padding, kích thước output được giữ nguyên so với input.

Feature map size

Trường hợp kích thước **output O** tính từ công thức không phải là **số nguyên**, ta thực hiện làm tròn xuống. Ví dụ padding θ , kernel size = 5 và stride = 2: $O = \lfloor \frac{100-5+2\times0}{2} + 1 \rfloor = 48$

```
1 import torch
2 import torch.nn as nn
3
4 layer = nn.Conv2d(1, 1, kernel_size=5, stride=2, padding=0)
5 example = torch.randn(1, 100, 100)
6 Output = layer(example)
7 print(Output.shape)
8 # Output: torch.Size([1, 48, 48])
```

Feature map size

Nếu chỉ đơn thuần muốn giữ lại đúng kích thước input mà **không cần quan tâm về công thức tính toán kích thước output**, ta có thể sử dụng padding “*same*” trong PyTorch.

```
1 import torch
2 import torch.nn as nn
3
4 layer = nn.Conv2d(1, 1, kernel_size=5, stride=1, padding="same")
5 example = torch.randn(1, 100, 100)
6 Output = layer(example)
7 print(Output.shape)
8 # Output: torch.Size([1, 100, 100])
```

Tài liệu tham khảo

-  Rafael C. Gonzalez, Richard E. Woods
Digital Image Processing (2018), Fourth Edition, Global Edition, Pearson.
-  Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (2022). Published by Packt Publishing Ltd, ISBN 978-1-80181-931-2.
-  LightningAI
LightningAI: PyTorch Lightning (2024) .