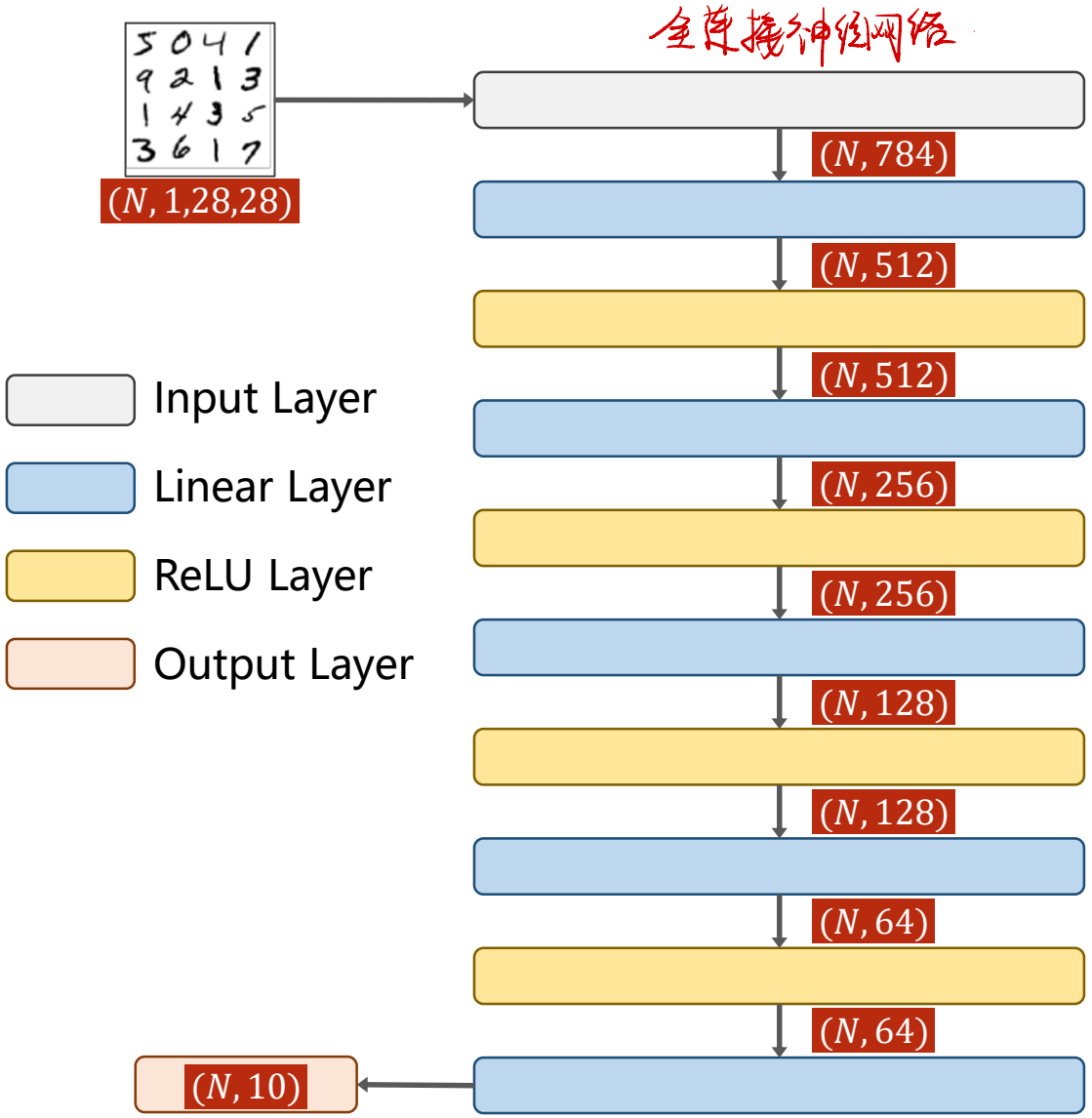
神经网络 { 全连接神经网络 / 卷积神经网络 / 循环神经网络

PyTorch Tutorial

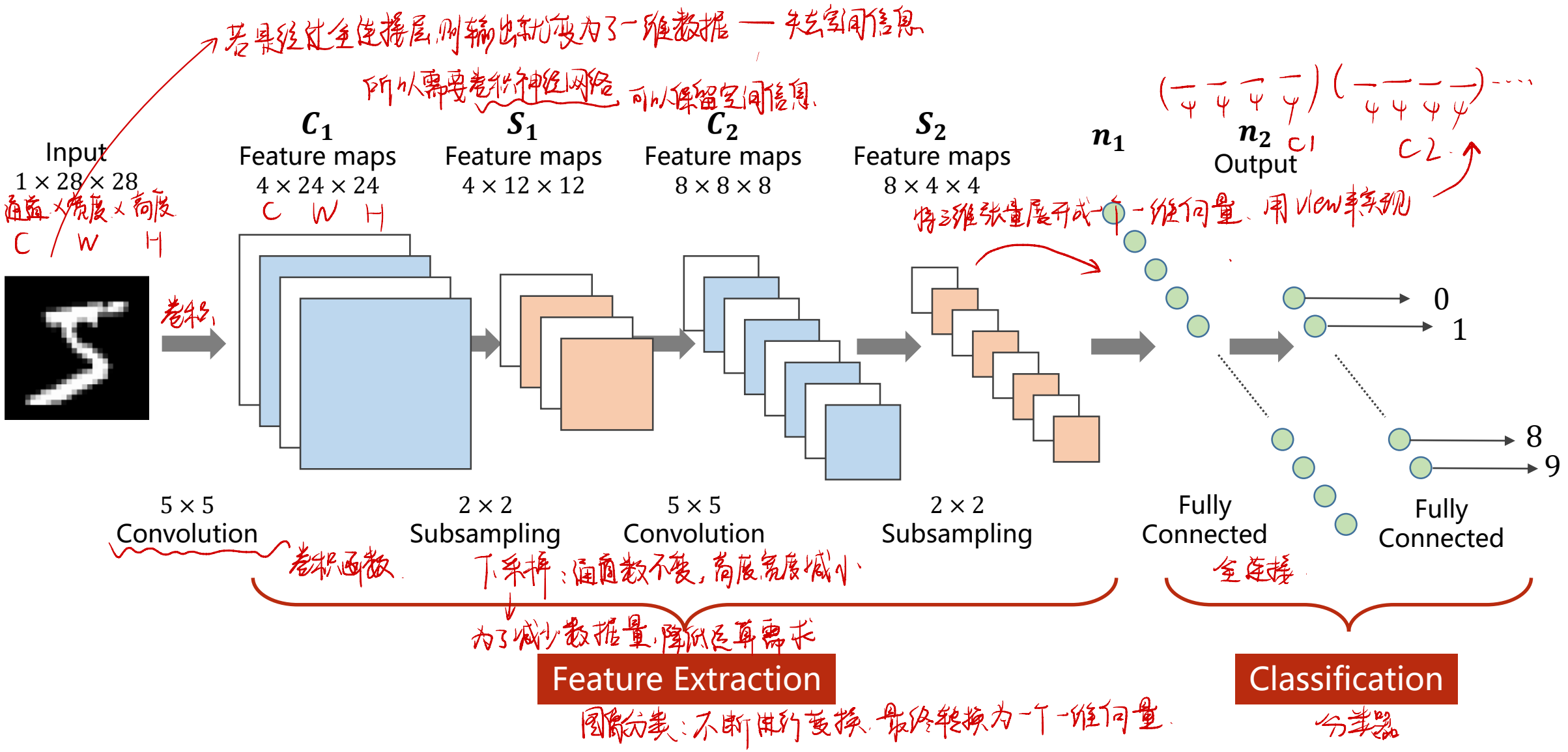
10. Basic CNN

Revision: Fully Connected Neural Network

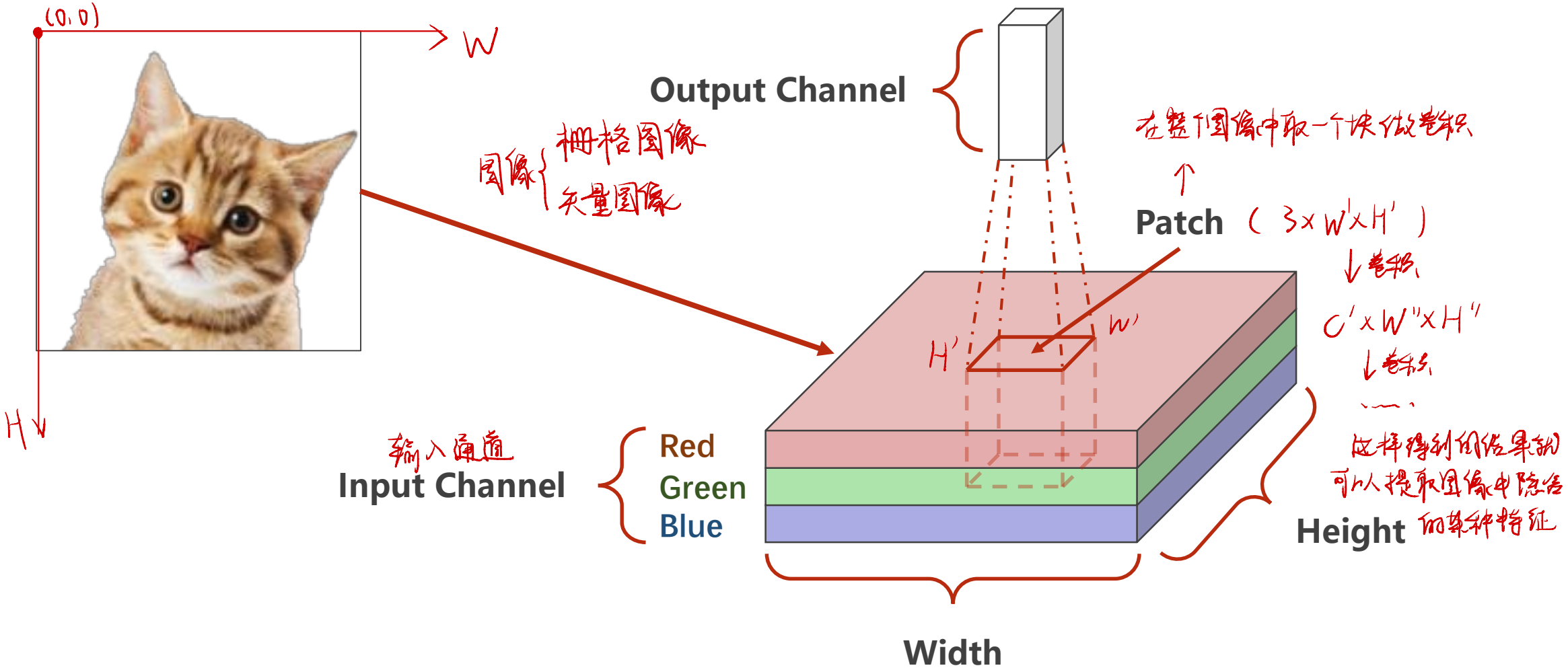



```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.l1 = torch.nn.Linear(784, 512)
        self.l2 = torch.nn.Linear(512, 256)
        self.l3 = torch.nn.Linear(256, 128)
        self.l4 = torch.nn.Linear(128, 64)
        self.l5 = torch.nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        return self.l5(x)

model = Net()
```

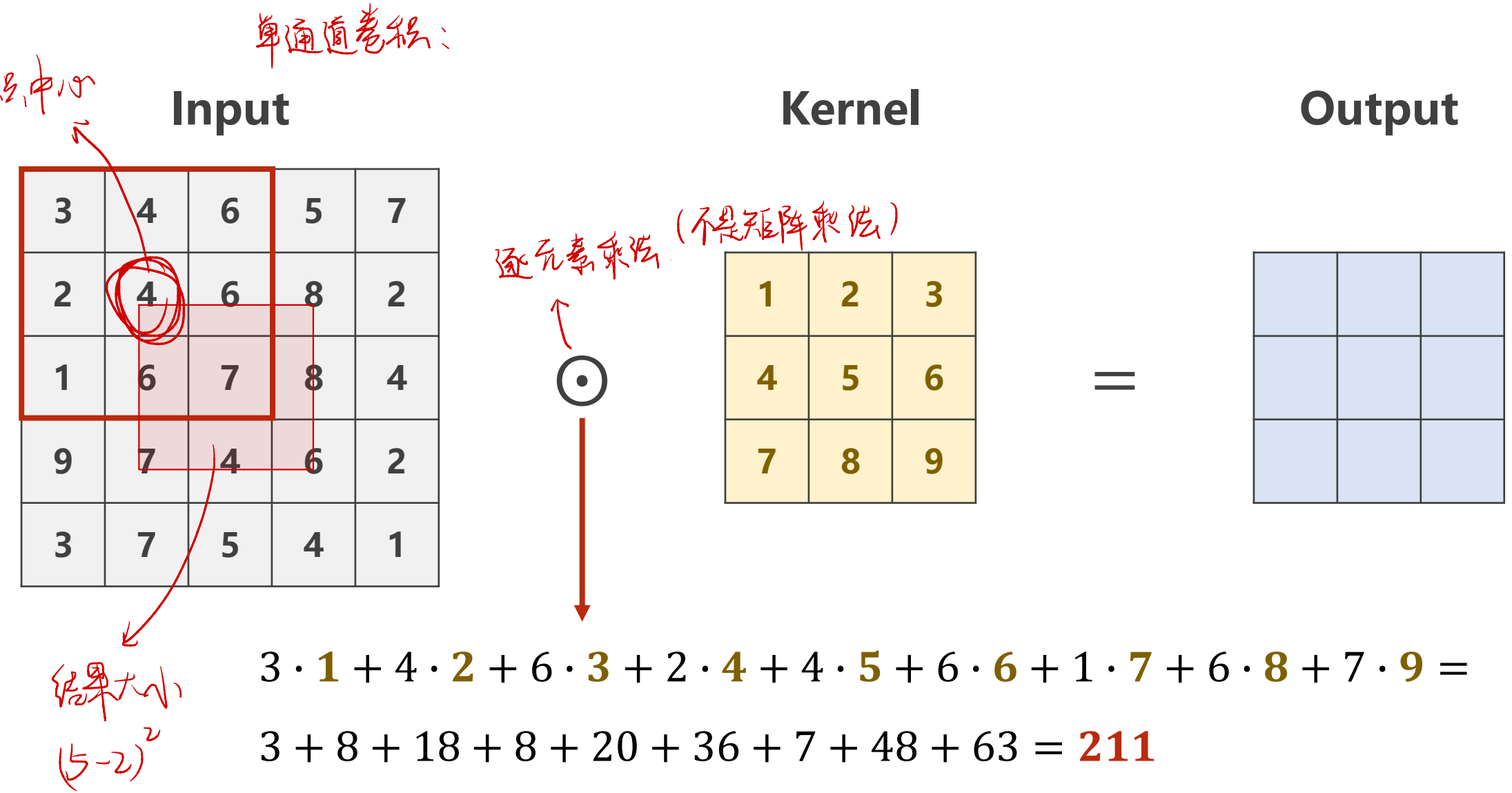
Convolutional Neural Network



Convolution



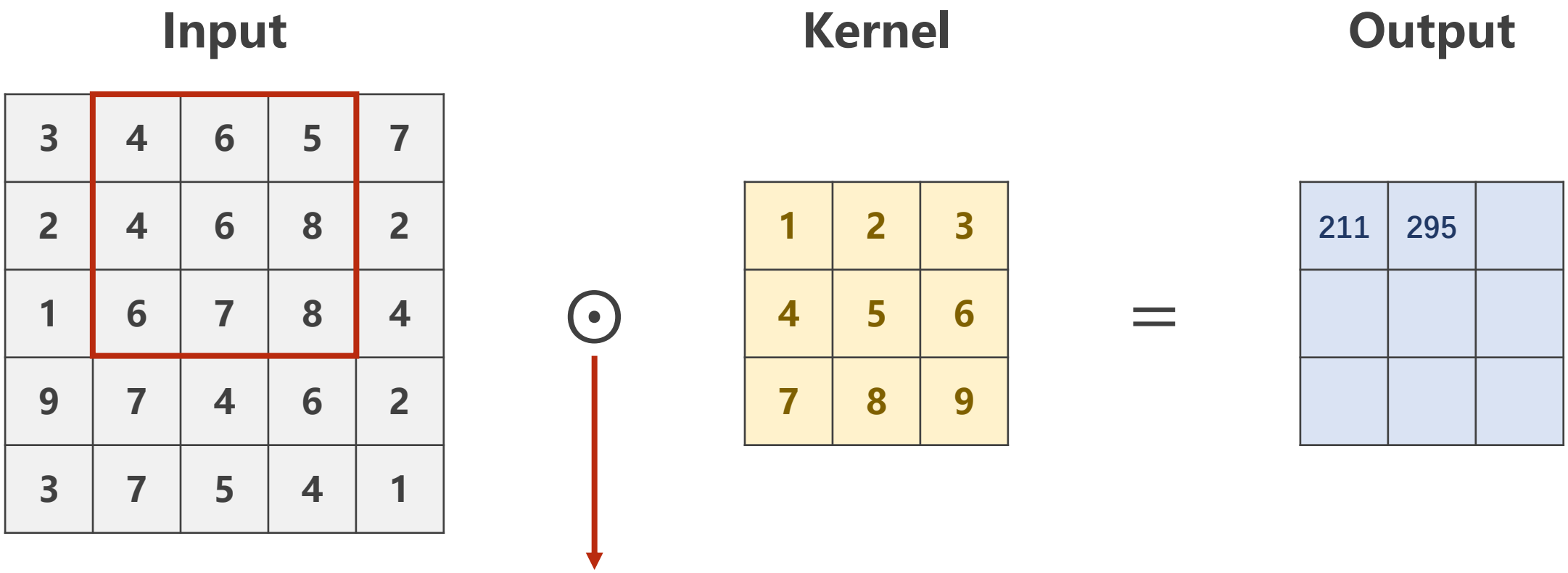
Convolution – Single Input Channel



$$3 \cdot 1 + 4 \cdot 2 + 6 \cdot 3 + 2 \cdot 4 + 4 \cdot 5 + 6 \cdot 6 + 1 \cdot 7 + 6 \cdot 8 + 7 \cdot 9 =$$

$$3 + 8 + 18 + 8 + 20 + 36 + 7 + 48 + 63 = 211$$

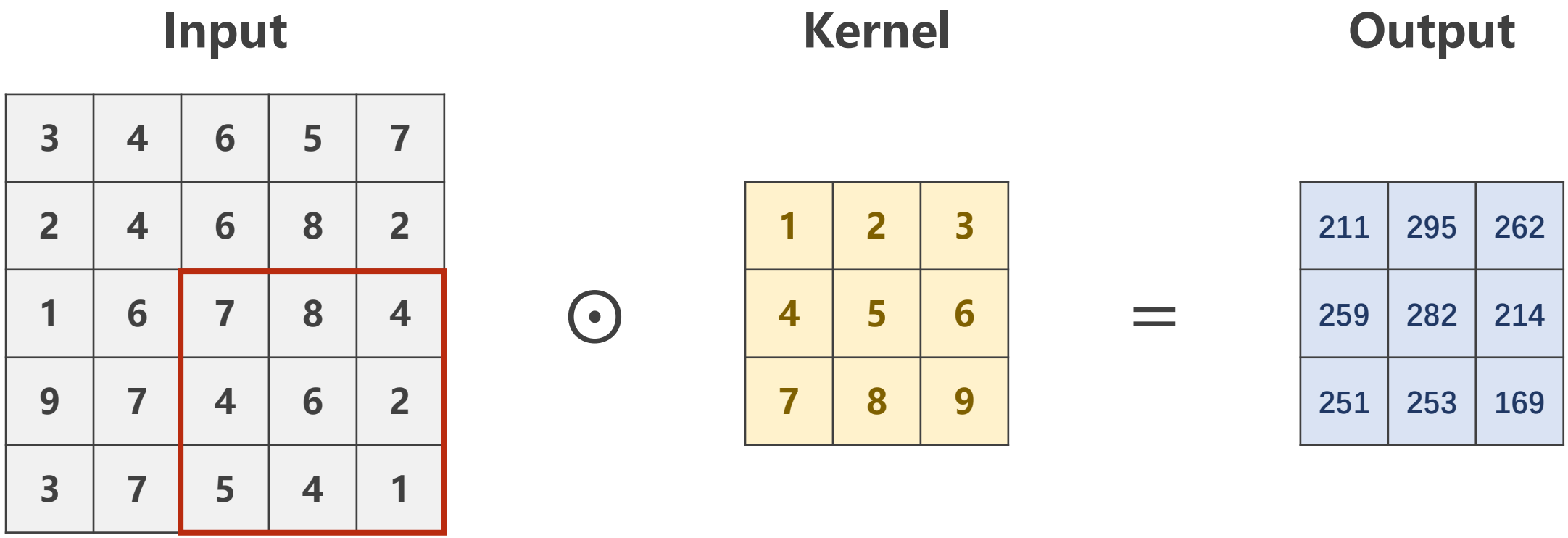
Convolution – Single Input Channel



$$4 \cdot 1 + 6 \cdot 2 + 5 \cdot 3 + 4 \cdot 4 + 6 \cdot 5 + 8 \cdot 6 + 6 \cdot 7 + 7 \cdot 8 + 8 \cdot 9 =$$

$$4 + 12 + 15 + 16 + 30 + 48 + 42 + 56 + 72 = 295$$

Convolution – Single Input Channel



Convolution – 3 Input Channels

每个通道卷积：

每一个通道都有单独的卷积核

3	4	6	5	7
2	4	6	8	2
1	6	7	8	4
9	7	4	6	2
3	7	5	4	1

1	2	3
4	5	6
7	8	9

$\odot$ $=$

211	295	262
259	282	214
251	253	169

3	4	6	5	7
2	4	6	8	2
1	6	7	8	4
9	7	4	6	2
3	7	5	4	1

9	8	7
6	5	4
3	2	1

$\odot$ $=$

179	245	268
201	278	256
239	287	241

3	4	6	5	7
2	4	6	8	2
1	6	7	8	4
9	7	4	6	2
3	7	5	4	1

1	4	7
2	5	8
3	6	9

$\odot$ $=$

235	297	248
253	294	204
255	259	169

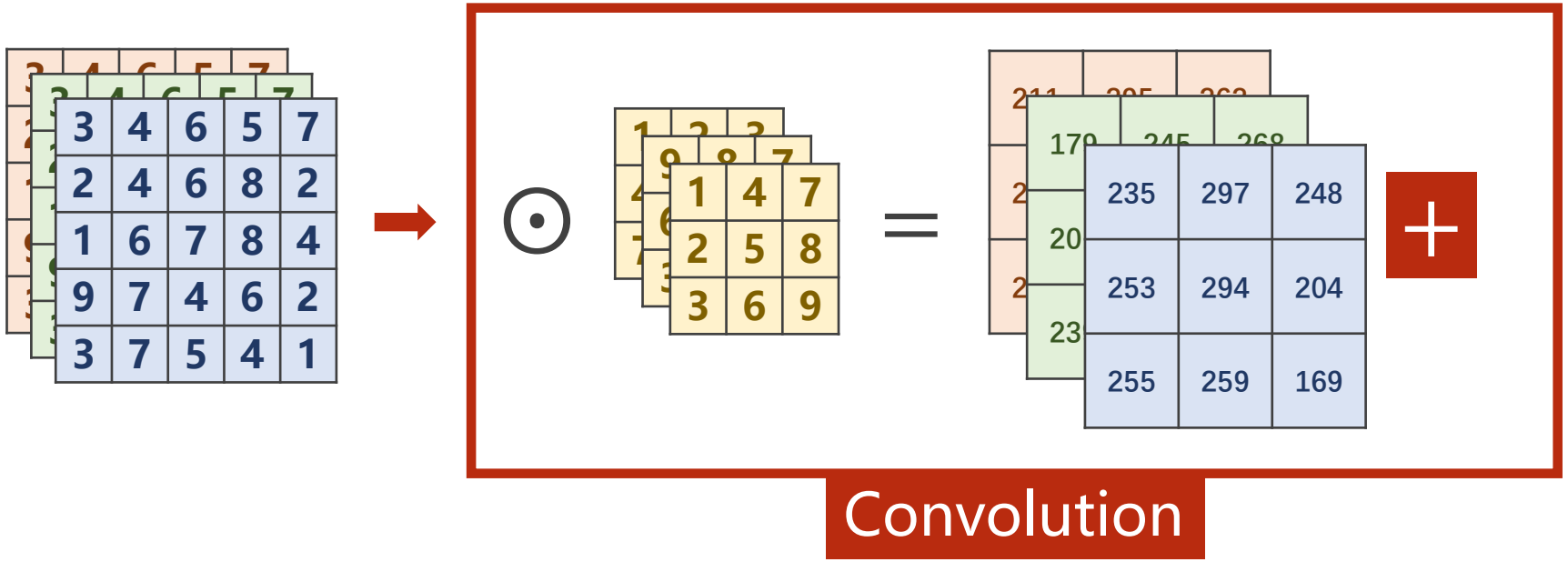
最后将各通道作矩阵加法

$+$ $=$

Convolution

625	837	778
713	854	674
745	799	579

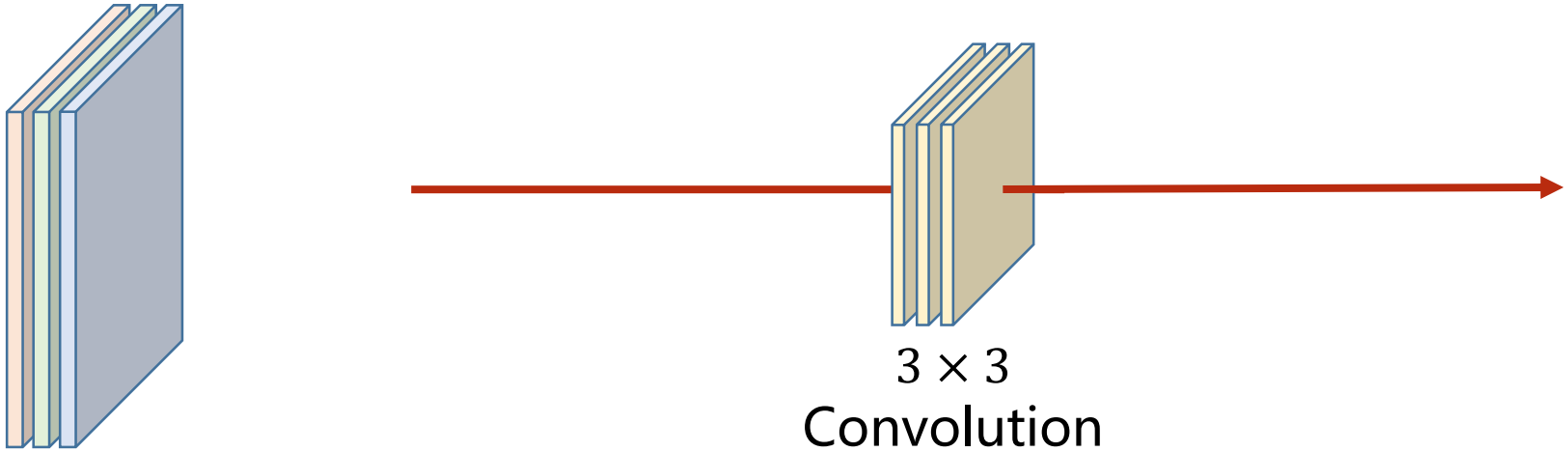
Convolution – 3 Input Channels



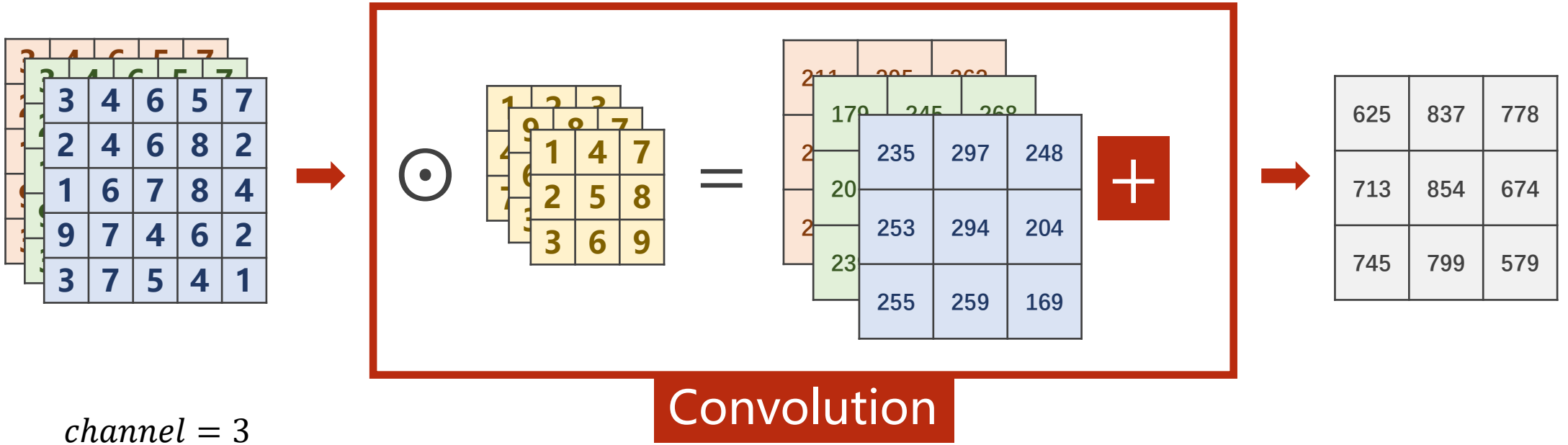
在这里输出通道数为1

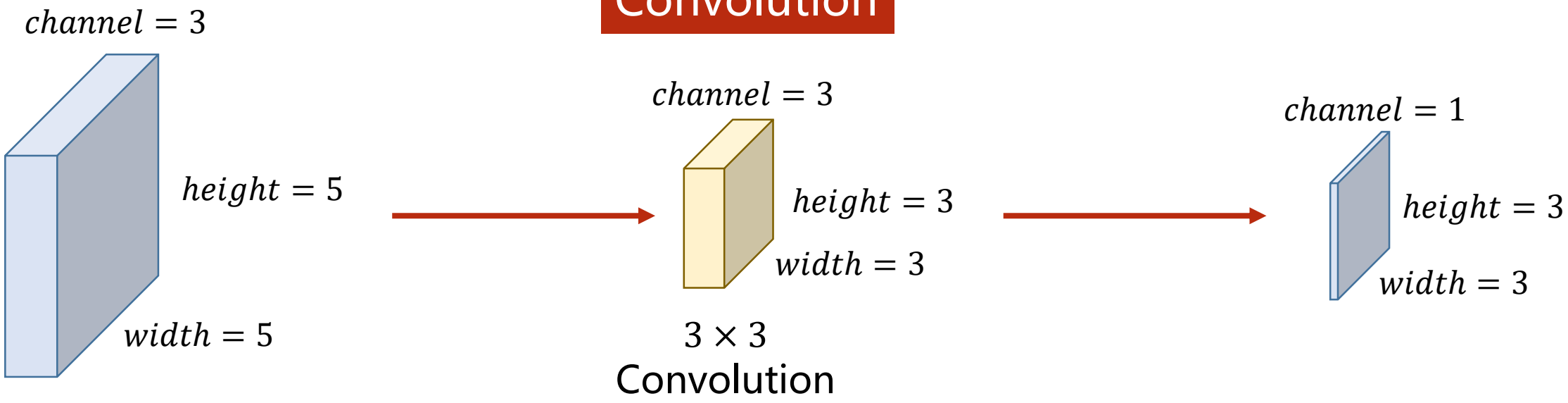
↓

也可以为很多



Convolution – 3 Input Channels

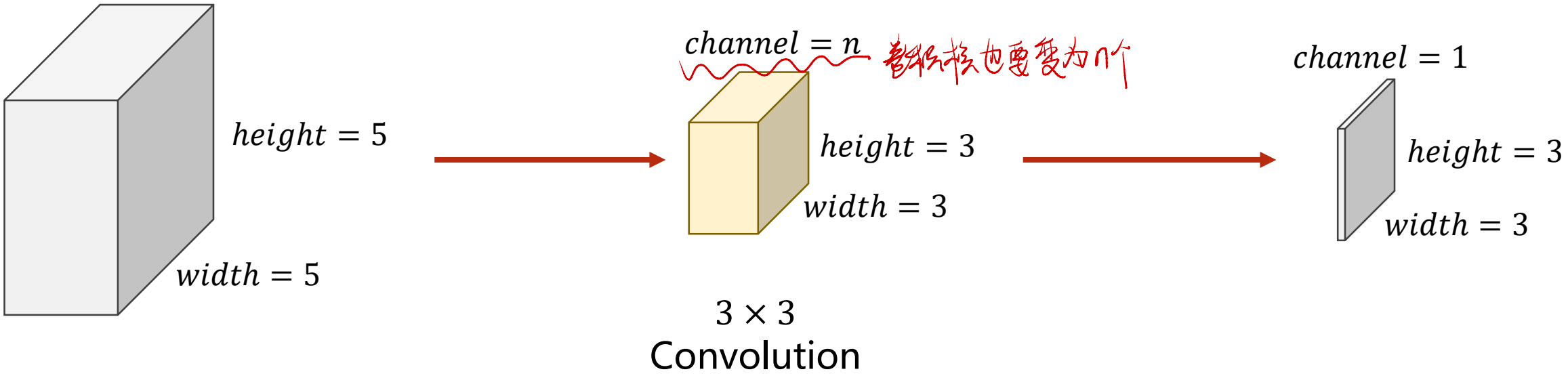




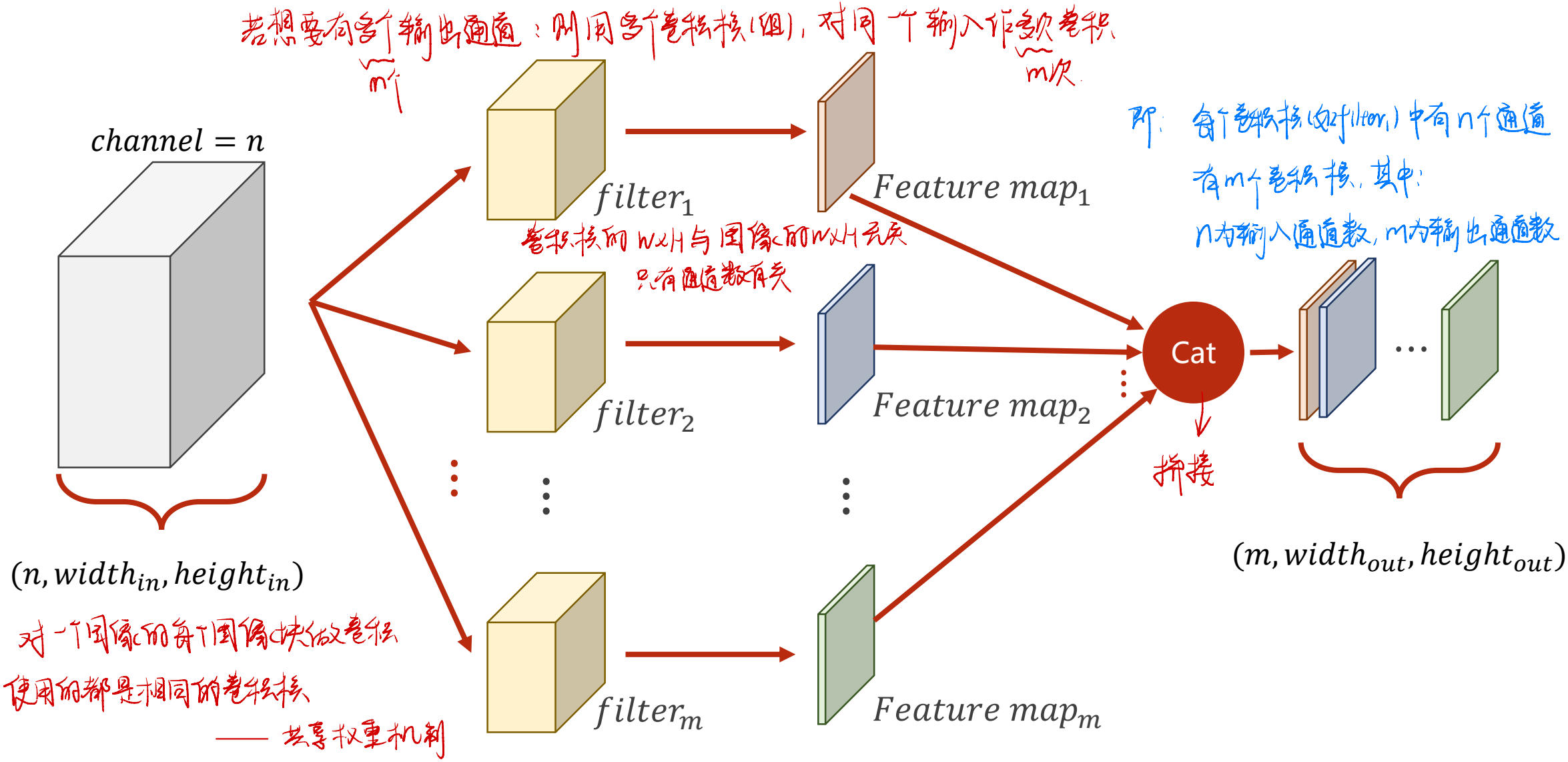
Convolution – N Input Channels

若输入通道数为n

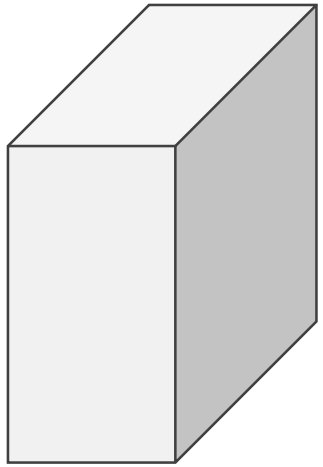
The shape of kernel tensor is $(n, 3, 3)$



Convolution – N Input Channels and M Output Channels



Convolutional Layer



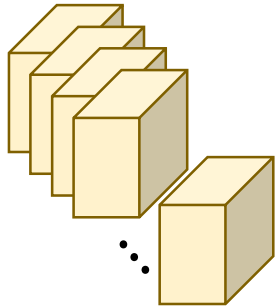
$n \times width_{in}, \times height_{in}$

This layer needs m filters, which size are:

$$n \times kernel\_size_{width} \times kernel\_size_{height}$$

$m \times width_{out}, \times height_{out}$

4-Dimension Tensor



$m \times n \times kernel\_size_{width} \times kernel\_size_{height}$

卷积核个数

每个卷积核通道数

一个四维张量。

Convolutional Layer

计算过程：

```
import torch                                    n   m
in_channels, out_channels= 5, 10
width, height = 100, 100 //图像大小
kernel_size = 3 //卷积核大小
batch_size = 1 // pytorch中所有输入必须是小批量输入
                                所以要把一个图像分为多个Batch
input = torch.randn(batch_size,
                    in_channels,
                    width,
                    height)

conv_layer = torch.nn.Conv2d(in_channels,//n
                             out_channels, // m
                             kernel_size=kernel_size)// 3x3

output = conv_layer(input)

print(input.shape)
print(output.shape)
print(conv_layer.weight.shape)
```

随机数 → 先随机一个权重，后面训练(?)

卷积层

生成卷积层的模块

可以是正方形

也可以长方形，一般正方形，常见大小为奇数

只有一个值表示边正方形nxn，长方形输入元组

Convolutional Layer

```
import torch
in_channels, out_channels= 5, 10
width, height = 100, 100
kernel_size = 3
batch_size = 1

input = torch.randn(batch_size,
                    in_channels,
                    width,
                    height)

conv_layer = torch.nn.Conv2d(in_channels,
                             out_channels,
                             kernel_size=kernel_size)

output = conv_layer(input)

print(input.shape)
print(output.shape)
print(conv_layer.weight.shape)
```

```
torch.Size([1, 5, 100, 100])
torch.Size([1, 10, 98, 98])
torch.Size([10, 5, 3, 3])
```

注意：卷积层并不在乎图像or batch的大小，它只是从头到尾遍历一遍

输入通道 输出通道

Convolutional Layer – padding=1

卷积层的常见参数

① padding：为使输入大小和输出大小一致，我们可以扩大图像（常见方法是在周围添0）

添加圈数与卷积核大小有关，若卷积核大小 n×n

则圈数为 $\frac{n-1}{2}$ 取整

Input

0	0	0	0	0	0	0
0	3	4	6	5	7	0
0	2	4	6	8	2	0
0	1	6	7	8	4	0
0	9	7	4	6	2	0
0	3	7	5	4	1	0
0	0	0	0	0	0	0

⊙

Kernel

1	2	3
4	5	6
7	8	9

=

Output

91				
	211	295	262	
	259	282	214	
	251	253	169	

Convolutional Layer – padding=1

```
import torch

input = [3,4,6,5,7,
         2,4,6,8,2,
         1,6,7,8,4,
         9,7,4,6,2,
         3,7,5,4,1]
input = torch.Tensor(input).view(1, 1, 5, 5)

conv_layer = torch.nn.Conv2d(1, 1, kernel_size=3, padding=1, bias=False)

kernel = torch.Tensor([1,2,3,4,5,6,7,8,9]).view(1, 1, 3, 3)
conv_layer.weight.data = kernel.data

output = conv_layer(input)
print(output)
```

Convolutional Layer – stride=2

卷积层参数：

② 步长，stride，即每次卷积核移动的距离

Input

3	4	6	5	7
2	4	6	8	2
1	6	7	8	4
9	7	4	6	2
3	7	5	4	1

$\odot$

Kernel

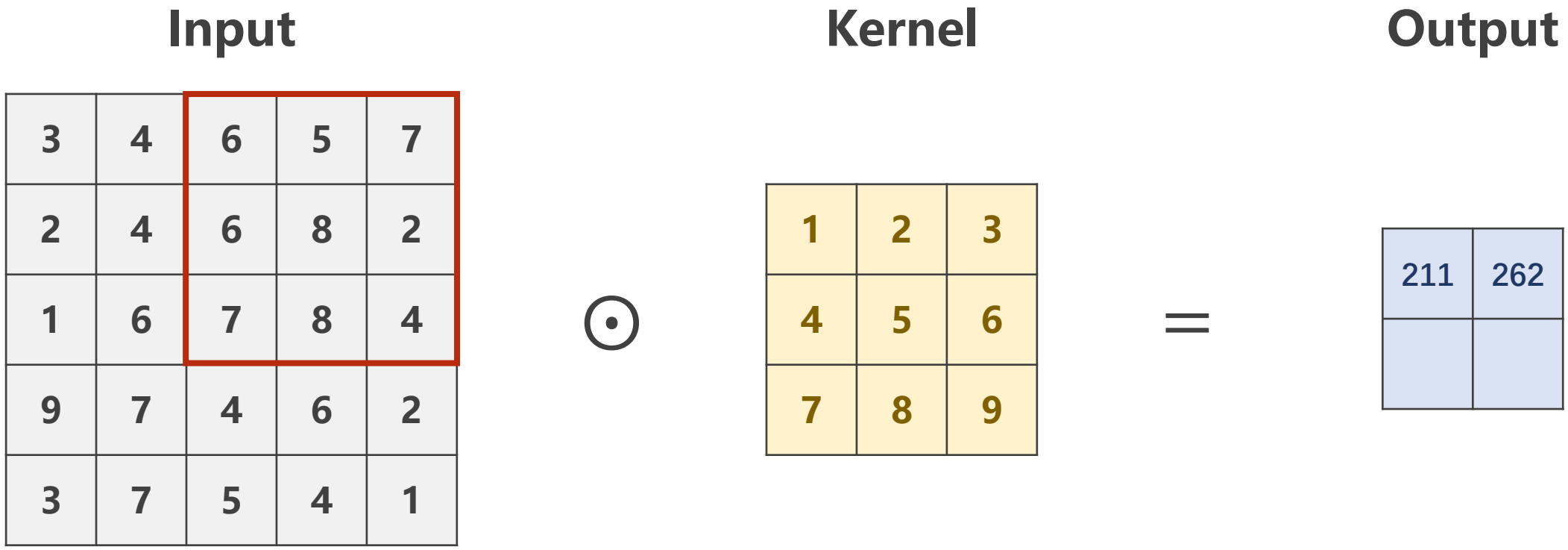
1	2	3
4	5	6
7	8	9

$=$

Output

211	

Convolutional Layer – stride=2



Convolutional Layer – stride=2

Input

3	4	6	5	7
2	4	6	8	2
1	6	7	8	4
9	7	4	6	2
3	7	5	4	1

$\odot$

Kernel

1	2	3
4	5	6
7	8	9

$=$

Output

211	262
251	

Convolutional Layer – stride=2

```
import torch

input = [3,4,6,5,7,
         2,4,6,8,2,
         1,6,7,8,4,
         9,7,4,6,2,
         3,7,5,4,1]
input = torch.Tensor(input).view(1, 1, 5, 5)

conv_layer = torch.nn.Conv2d(1, 1, kernel_size=3, stride=2, bias=False)

kernel = torch.Tensor([1,2,3,4,5,6,7,8,9]).view(1, 1, 3, 3)
conv_layer.weight.data = kernel.data

output = conv_layer(input)
print(output)
```

Max Pooling Layer

下采样：最大池化层

找4个值中的最大值

3	4	6	5
2	4	6	8
1	6	7	8
9	7	4	6

默认从 stride = 2 (步长)

2 × 2
MaxPooling

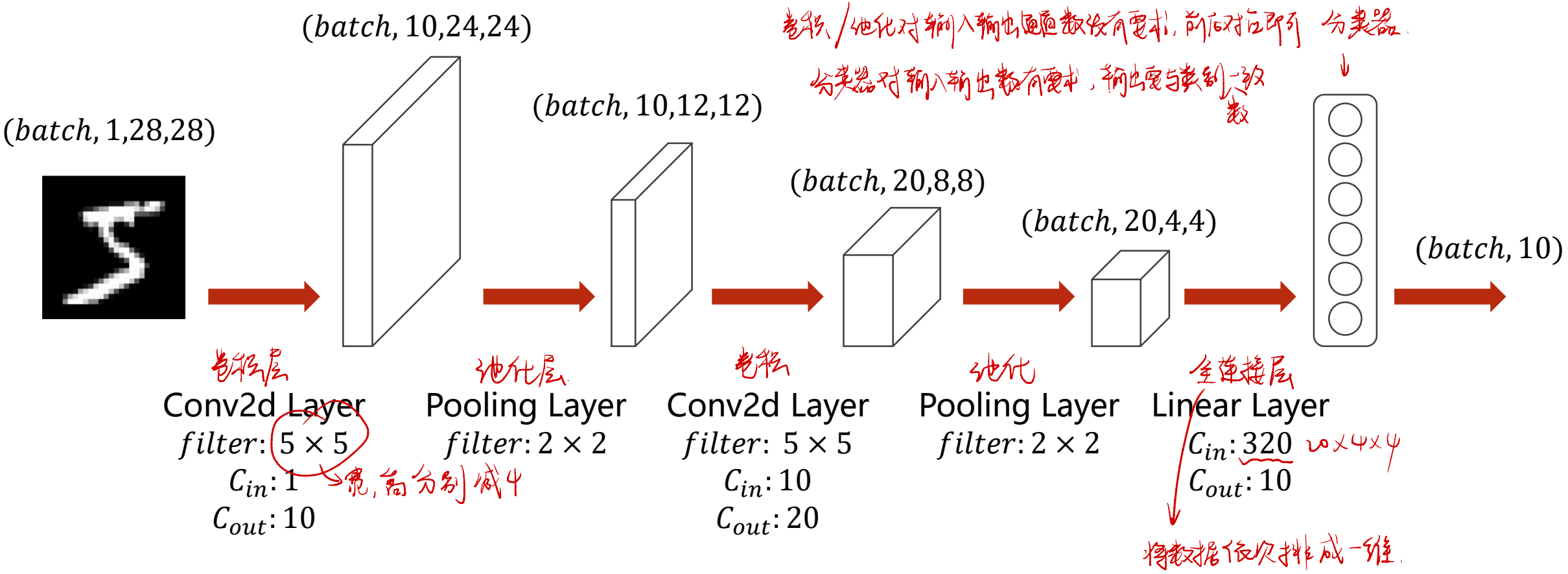
4	8
9	8

通道数量不变，图像大小变为原来1/4（宽、高均减半）

Max Pooling Layer

```
import torch

input = [3,4,6,5,
         2,4,6,8,
         1,6,7,8,
         9,7,4,6,
        ]
input = torch.Tensor(input).view(1, 1, 4, 4)

maxpooling_layer = torch.nn.MaxPool2d(kernel_size=2)

output = maxpooling_layer(input)
print(output)
```

A Simple Convolutional Neural Network



Revision: Fully Connected Neural Network

$(batch, 1, 28, 28)$

将全连接网络改为卷积神经网络

$(batch, 1, 28, 28)$

$C_{in} = 1, C_{out} = 10, kernel = 5$

```
self.conv1 = torch.nn.Conv2d(1, 10, kernel_size=5)
```

第一个卷积层

$(batch, 10, 24, 24)$

$(batch, 10, 24, 24)$

$kernel = 2 \times 2$

```
self.pooling = torch.nn.MaxPool2d(2)
```

池化层

做一个实例即可，因为每有权重，不会改变值，都是一样的

$(batch, 10, 12, 12)$

$C_{in} = 10, C_{out} = 20, kernel = 5$

```
self.conv2 = torch.nn.Conv2d(10, 20, kernel_size=5)
```

第二个

$(batch, 20, 8, 8)$

$(batch, 20, 8, 8)$

$kernel = 2 \times 2$

```
self.pooling = torch.nn.MaxPool2d(2)
```

$(batch, 20, 4, 4) \rightarrow (batch, 320)$

在这里面有一个view，将 $20 \times 2 \times 2$ 变为320

$f_{in} = 320, f_{out} = 10$

```
self.fc = torch.nn.Linear(320, 10)
```

$(batch, 10)$

- Input Layer
- Conv2d Layer
- ReLU Layer
- Pooling Layer
- Linear Layer
- Output Layer

Revision: Fully Connected Neural Network

$(batch, 1,28,28)$

$(batch, 1,28,28)$

$C_{in} = 1, C_{out} = 10, kernel = 5$

$(batch, 10,24,24)$

$(batch, 10,24,24)$

$kernel = 2 \times 2$

$(batch, 10,12,12)$

$C_{in} = 10, C_{out} = 20, kernel = 5$

$(batch, 20,8,8)$

$(batch, 20,8,8)$

$kernel = 2 \times 2$

$(batch, 20,4,4) \rightarrow (batch, 320)$

$f_{in} = 320, f_{out} = 10$

$(batch, 10)$

- Input Layer
- Conv2d Layer
- ReLU Layer
- Pooling Layer
- Linear Layer
- Output Layer

```python
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = torch.nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = torch.nn.Conv2d(10, 20, kernel_size=5)
        self.pooling = torch.nn.MaxPool2d(2)
        self.fc = torch.nn.Linear(320, 10) # 全连接层

    def forward(self, x):
        # Flatten data from (n, 1, 28, 28) to (n, 784)
        batch_size = x.size(0) # 先求batch大小，固定写法
        x = F.relu(self.pooling(self.conv1(x)))
        x = F.relu(self.pooling(self.conv2(x)))
        x = x.view(batch_size, -1) # flatten // view
        x = self.fc(x)
        return x

model = Net()
```

// 全连接一不激活，因为要用交叉熵的那个方法

先卷积，再池化，再ReLU

非线性化

How to use GPU – 1. Move Model to GPU

如何使用GPU

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = torch.nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = torch.nn.Conv2d(10, 20, kernel_size=5)
        self.pooling = torch.nn.MaxPool2d(2)
        self.fc = torch.nn.Linear(320, 10)

    def forward(self, x):
        # Flatten data from (n, 1, 28, 28) to (n, 784)
        batch_size = x.size(0)
        x = F.relu(self.pooling(self.conv1(x)))
        x = F.relu(self.pooling(self.conv2(x)))
        x = x.view(batch
        x = self.fc(x)
        return x

model = Net()
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Define device as the first visible cuda device if we have CUDA available.

第一块显卡

How to use GPU – 1. Move Model to GPU

构造模型时

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = torch.nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = torch.nn.Conv2d(10, 20, kernel_size=5)
        self.pooling = torch.nn.MaxPool2d(2)
        self.fc = torch.nn.Linear(320, 10)

    def forward(self, x):
        # Flatten data from (n, 1, 28, 28) to (n, 784)
        batch_size = x.size(0)
        x = F.relu(self.pooling(self.conv1(x)))
        x = F.relu(self.pooling(self.conv2(x)))
        x = x.view(batch_size, -1) # flatten
        x = self.fc(x)
        return x

model = Net()
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device) // 将所有内容放入显卡
```

Convert parameters and buffers of all modules to CUDA Tensor.

How to use GPU – 2. Move Tensors to GPU

训练时

```
def train(epoch):
    running_loss = 0.0                                  在这里使用了 mini-batch
    for batch_idx, data in enumerate(train_loader, 0):
        inputs, target = data
        inputs, target = inputs.to(device), target.to(device)  ← 添加这一行，将数据放入GPU
        optimizer.zero_grad()

        # forward + backward + update
        outputs = model(inputs)
        loss = criterion(outputs, target)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if batch_idx % 300 == 299:
            print('[%d, %5d] loss: %.3f' % (epoch + 1, batch_idx + 1, running_loss / 2000))
            running_loss = 0.0
```

Send the inputs and targets at every step to the GPU.

How to use GPU – 2. Move Tensors to GPU

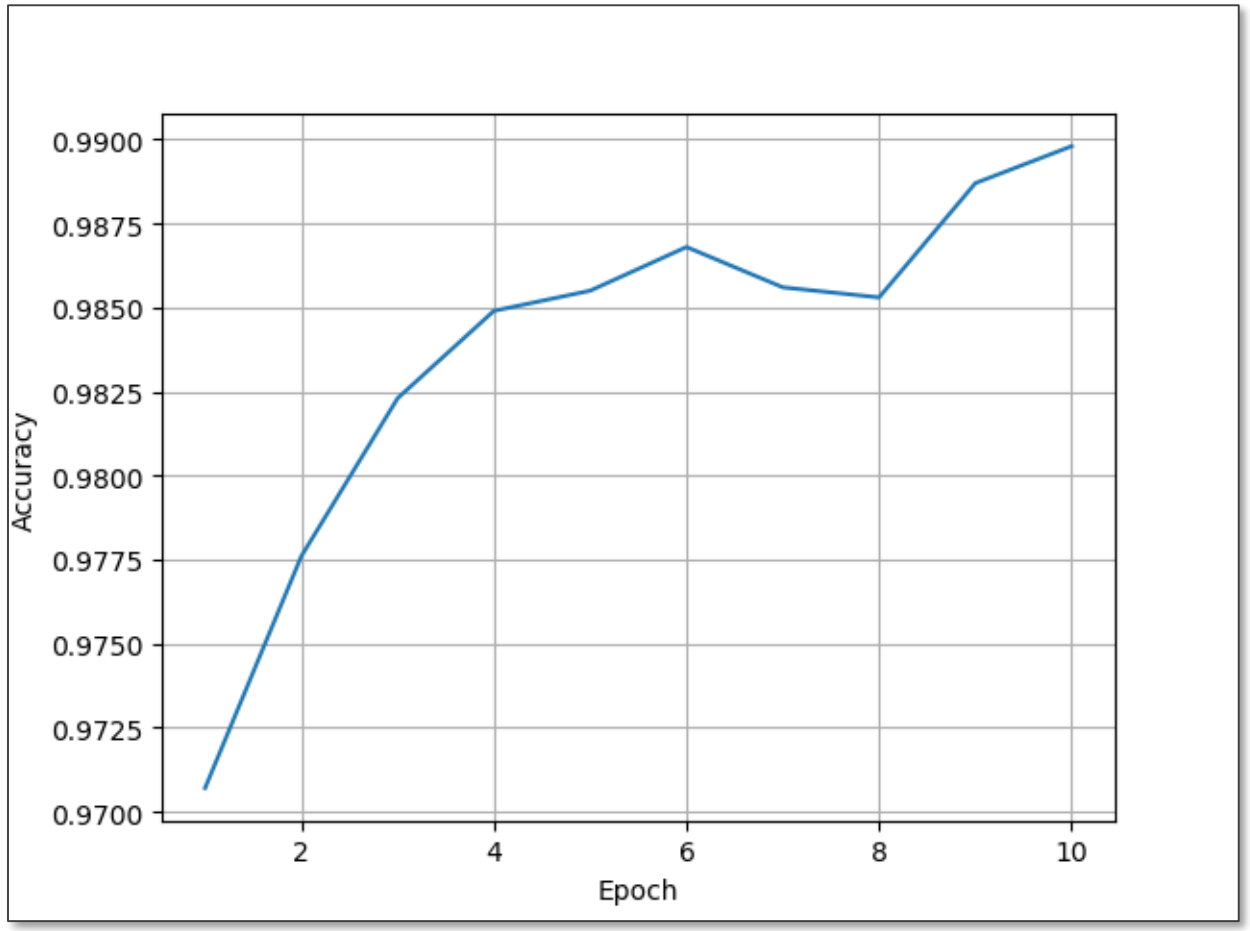
测试时

```
def test():
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            inputs, target = data
            inputs, target = inputs.to(device), target.to(device) ← 添加这一行
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, dim=1)
            total += target.size(0)
            correct += (predicted == target).sum().item()
    print('Accuracy on test set: %d %% [%d/%d]' % (100 * correct / total, correct, total))
```

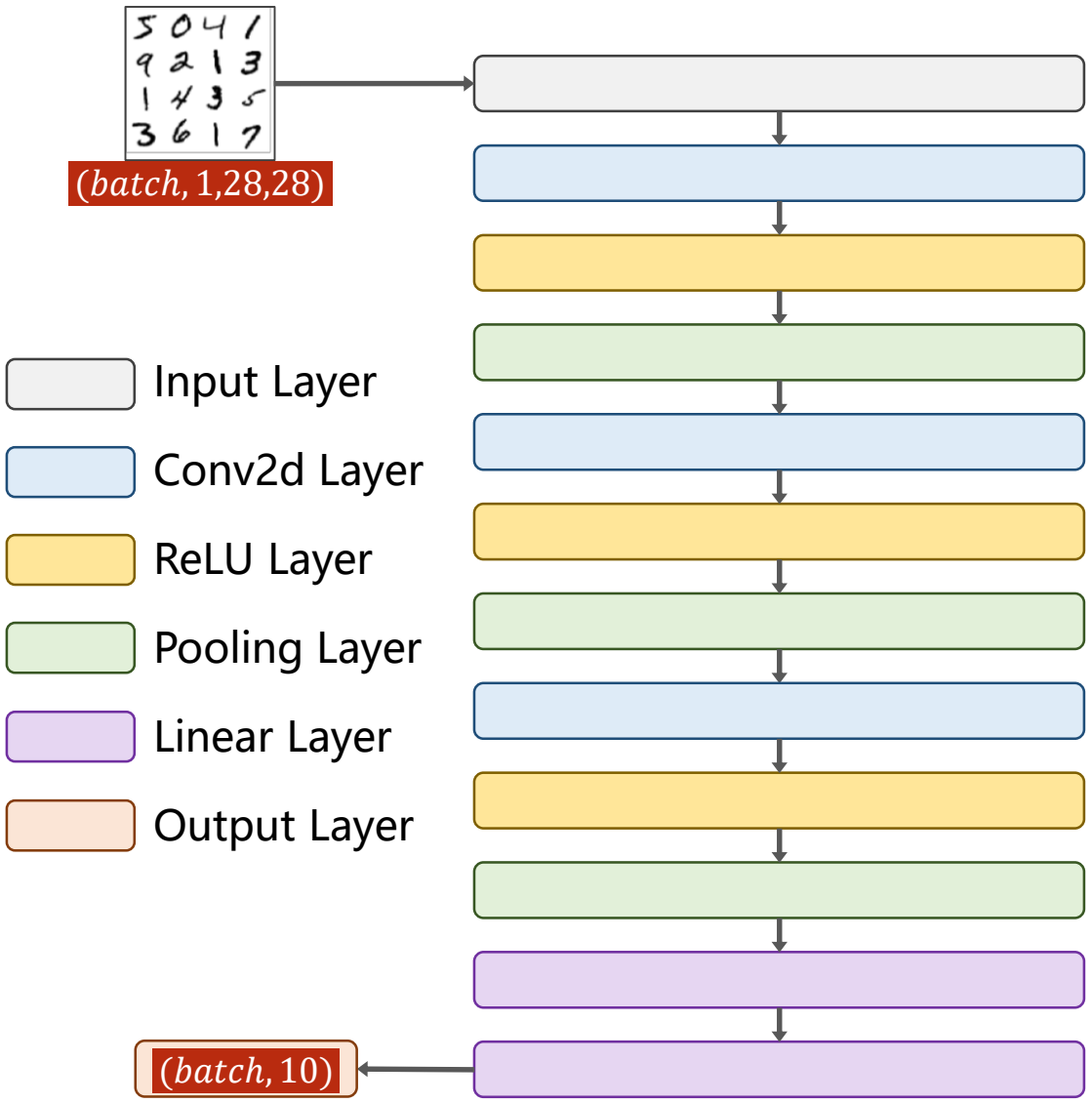
Send the inputs and targets at every step to the GPU.

Results

Accuracy on test set: 6 % [637/10000]
[1, 300] loss: 0.098
[1, 600] loss: 0.035
[1, 900] loss: 0.025
Accuracy on test set: 96 % [9605/10000]
[2, 300] loss: 0.021
[2, 600] loss: 0.017
[2, 900] loss: 0.015
Accuracy on test set: 97 % [9709/10000]
……
[9, 300] loss: 0.006
[9, 600] loss: 0.006
[9, 900] loss: 0.007
Accuracy on test set: 98 % [9857/10000]
[10, 300] loss: 0.006
[10, 600] loss: 0.006
[10, 900] loss: 0.006
Accuracy on test set: 98 % [9869/10000]



Exercise 10-1



- Try a more complex CNN:
  - Conv2d Layer *3
  - ReLU Layer * 3
  - MaxPooling Layer * 3
  - Linear Layer * 3
- Try different configuration of this CNN:
  - Compare their performance.

PyTorch Tutorial

10. Basic CNN