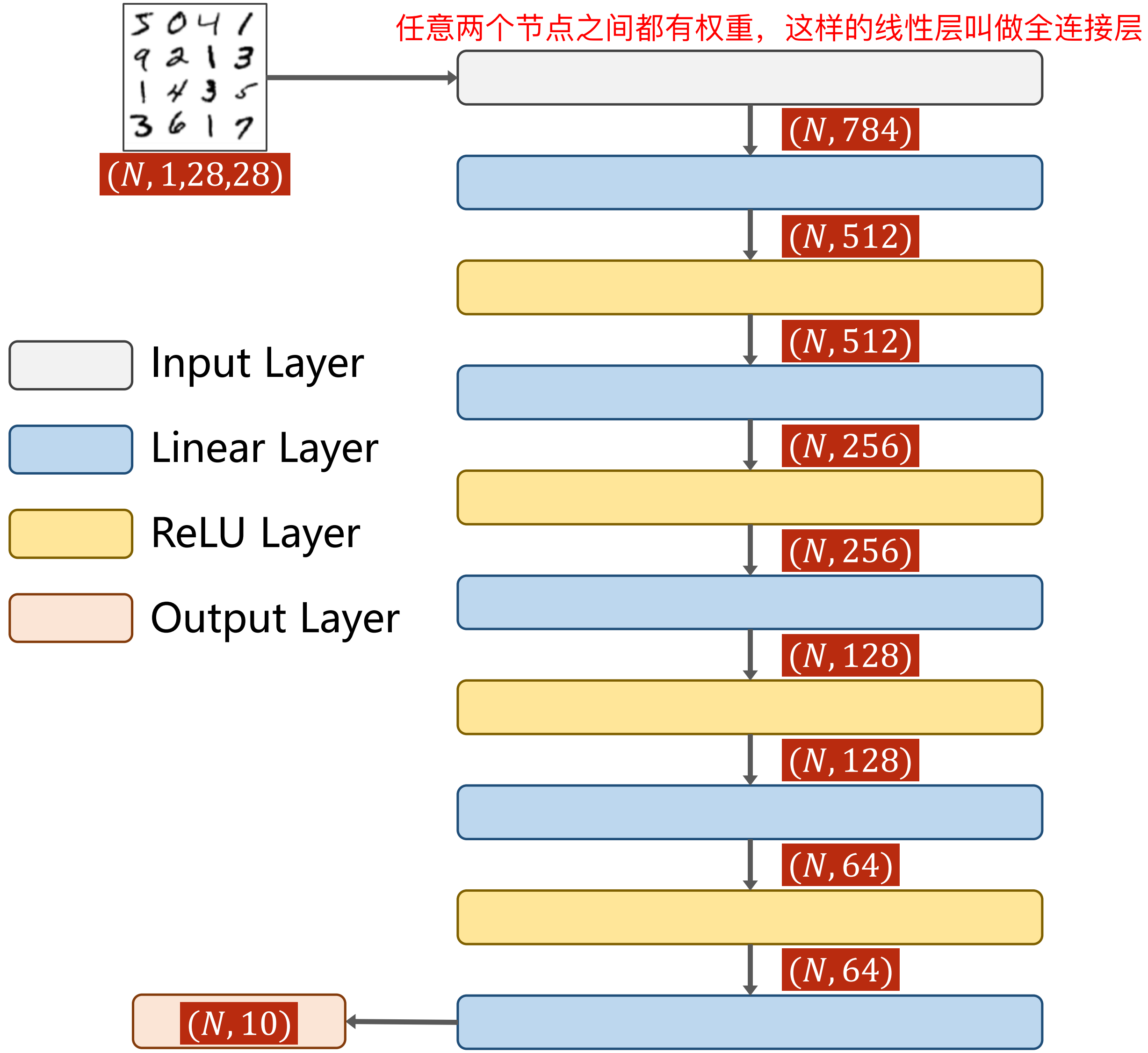
PyTorch Tutorial

10. Basic CNN

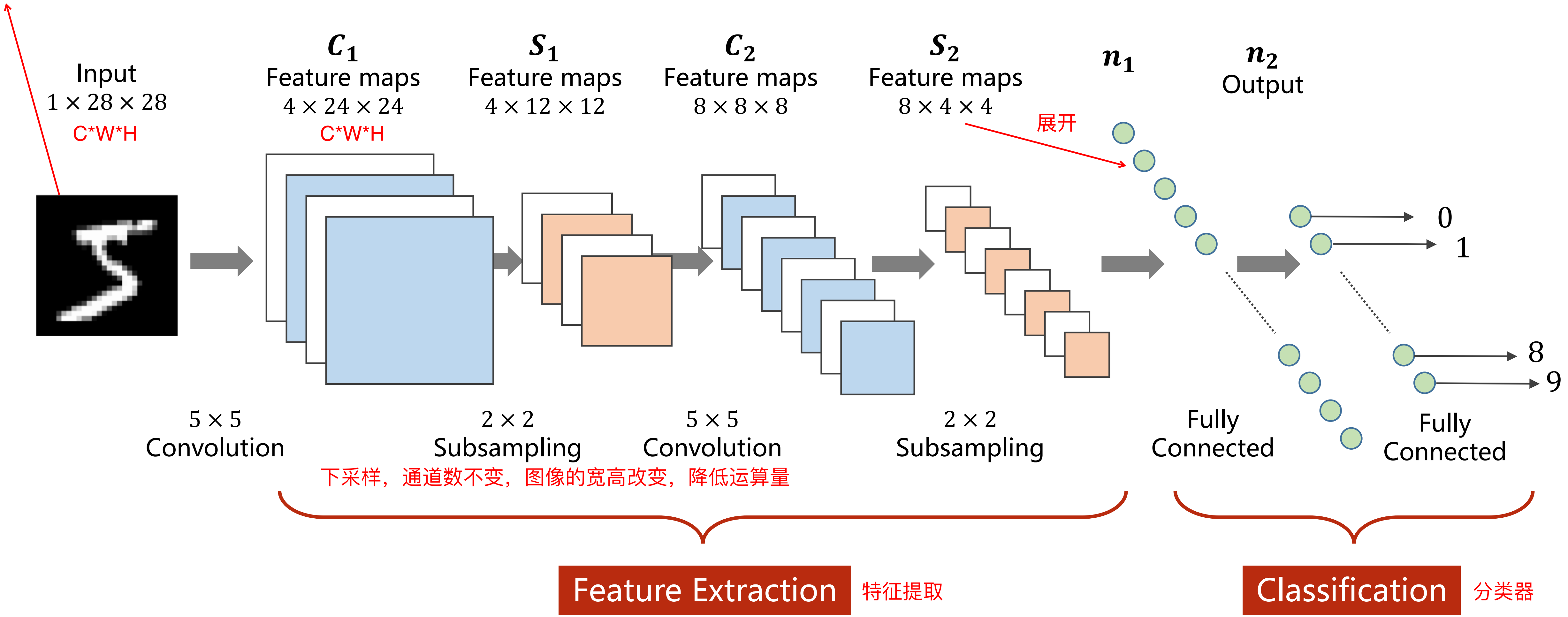
Revision: Fully Connected Neural Network




```python
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()
```

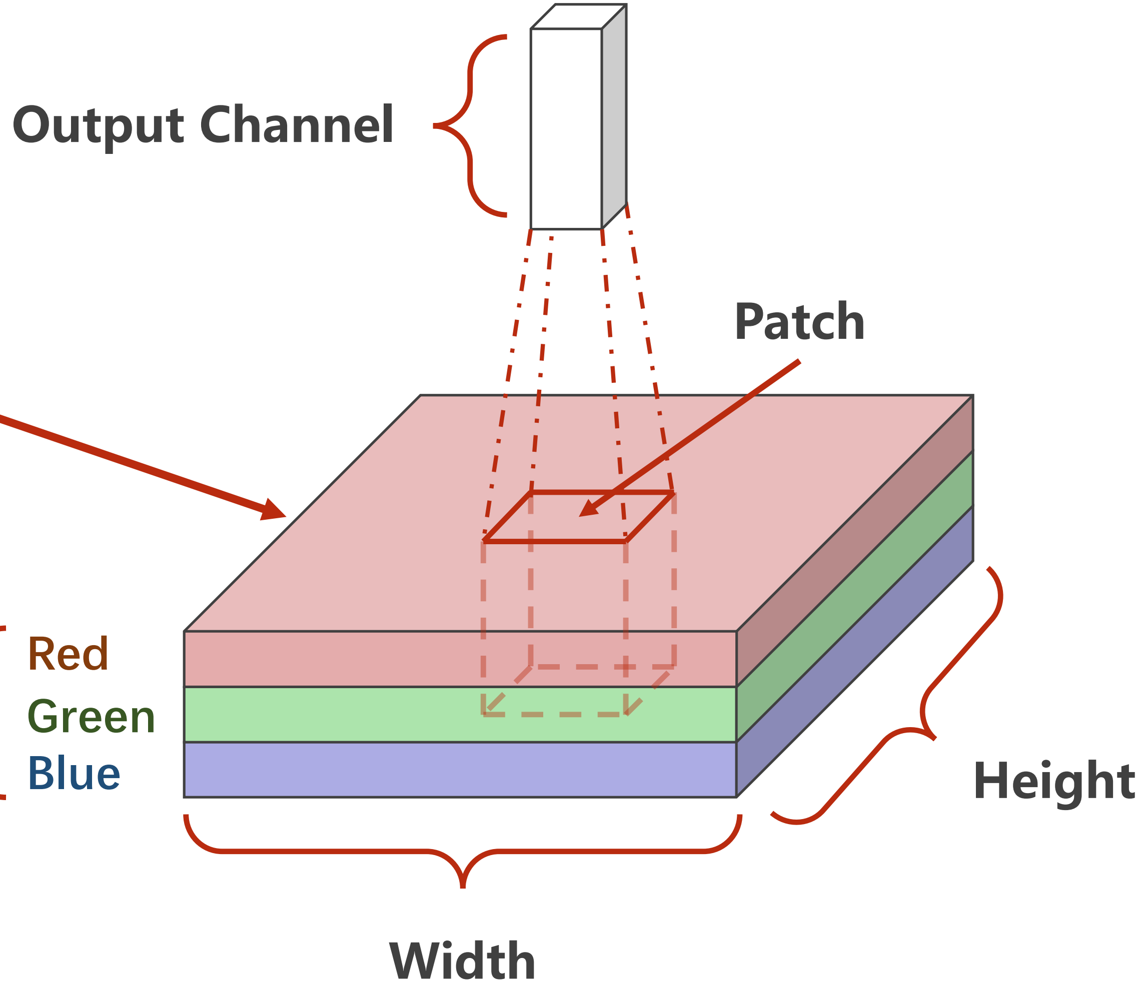
Convolutional Neural Network

之前用全连接做预测时，将图像所有像素直接变为一维向量，这个过程会丧失部分像素之间的位置关联。

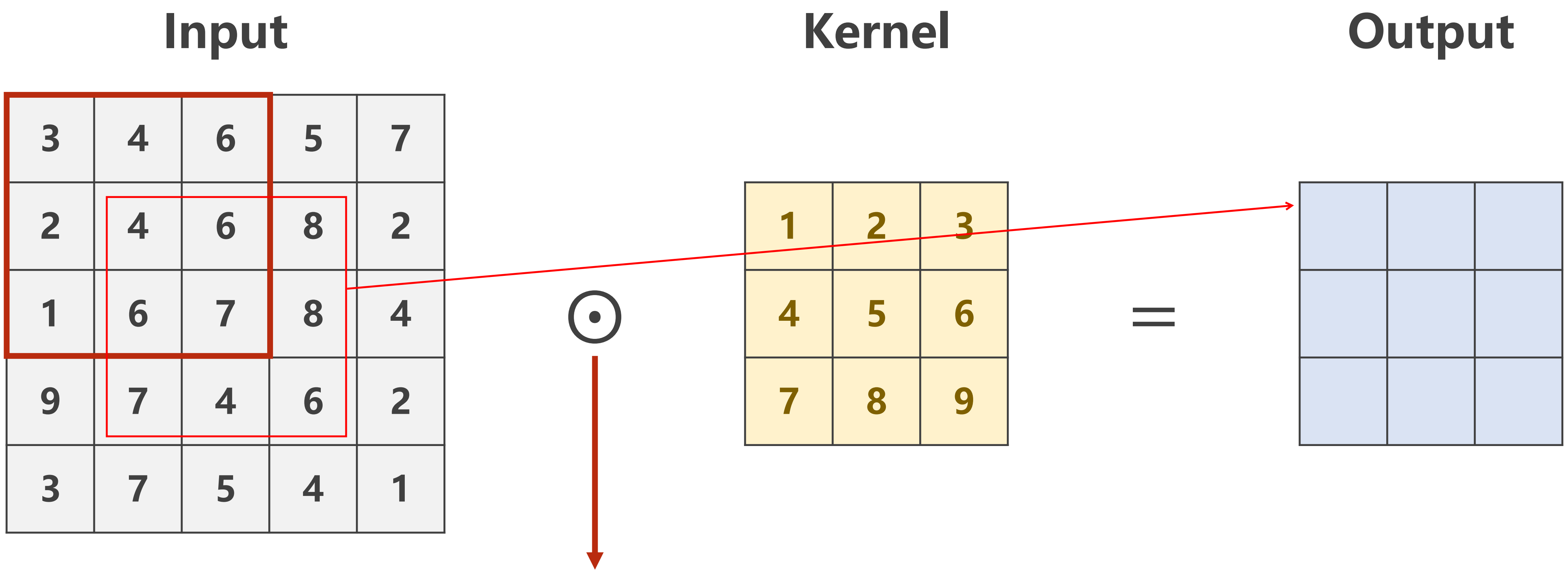


不断进行大小和维度的变化，直到成为一个1维的向量

Convolution



Convolution – Single Input Channel



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

$$3 + 8 + 18 + 8 + 20 + 36 + 7 + 48 + 63 = \mathbf{211}$$

Convolution – Single Input Channel

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		

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

$$3 + 8 + 18 + 8 + 20 + 36 + 7 + 48 + 63 = \mathbf{211}$$

Convolution – Single Input Channel

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		

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

$$4 + 12 + 15 + 16 + 30 + 48 + 42 + 56 + 72 = \mathbf{295}$$

Convolution – Single Input Channel

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	295	

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

$$4 + 12 + 15 + 16 + 30 + 48 + 42 + 56 + 72 = \mathbf{295}$$

Convolution – Single Input Channel

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	295	262

Convolution – Single Input Channel

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	295	262
259		

Convolution – Single Input Channel

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	295	262
259	282	

Convolution – Single Input Channel

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	295	262
259	282	214

Convolution – Single Input Channel

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	295	262
259	282	214
251		

Convolution – Single Input Channel

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	295	262
259	282	214
251	253	

Convolution – Single Input Channel

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	295	262
259	282	214
251	253	169

Convolution – 3 Input Channels

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

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

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

Convolution – 3 Input Channels

$$\begin{bmatrix}3&4&6&5&7\\2&4&6&8&2\\1&6&7&8&4\\9&7&4&6&2\\3&7&5&4&1\end{bmatrix} \odot \begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} = \begin{bmatrix}211&295&262\\259&282&214\\251&253&169\end{bmatrix}$$

$$\begin{bmatrix}3&4&6&5&7\\2&4&6&8&2\\1&6&7&8&4\\9&7&4&6&2\\3&7&5&4&1\end{bmatrix}$$

$$\begin{bmatrix}3&4&6&5&7\\2&4&6&8&2\\1&6&7&8&4\\9&7&4&6&2\\3&7&5&4&1\end{bmatrix}$$

Convolution – 3 Input Channels

$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 211 & 295 & 262 \\ 259 & 282 & 214 \\ 251 & 253 & 169 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix} \odot \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 179 & 245 & 268 \\ 201 & 278 & 256 \\ 239 & 287 & 241 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix}$$

Convolution – 3 Input Channels

$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 211 & 295 & 262 \\ 259 & 282 & 214 \\ 251 & 253 & 169 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix} \odot \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 179 & 245 & 268 \\ 201 & 278 & 256 \\ 239 & 287 & 241 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} = \begin{bmatrix} 235 & 297 & 248 \\ 253 & 294 & 204 \\ 255 & 259 & 169 \end{bmatrix}$$

Convolution – 3 Input Channels

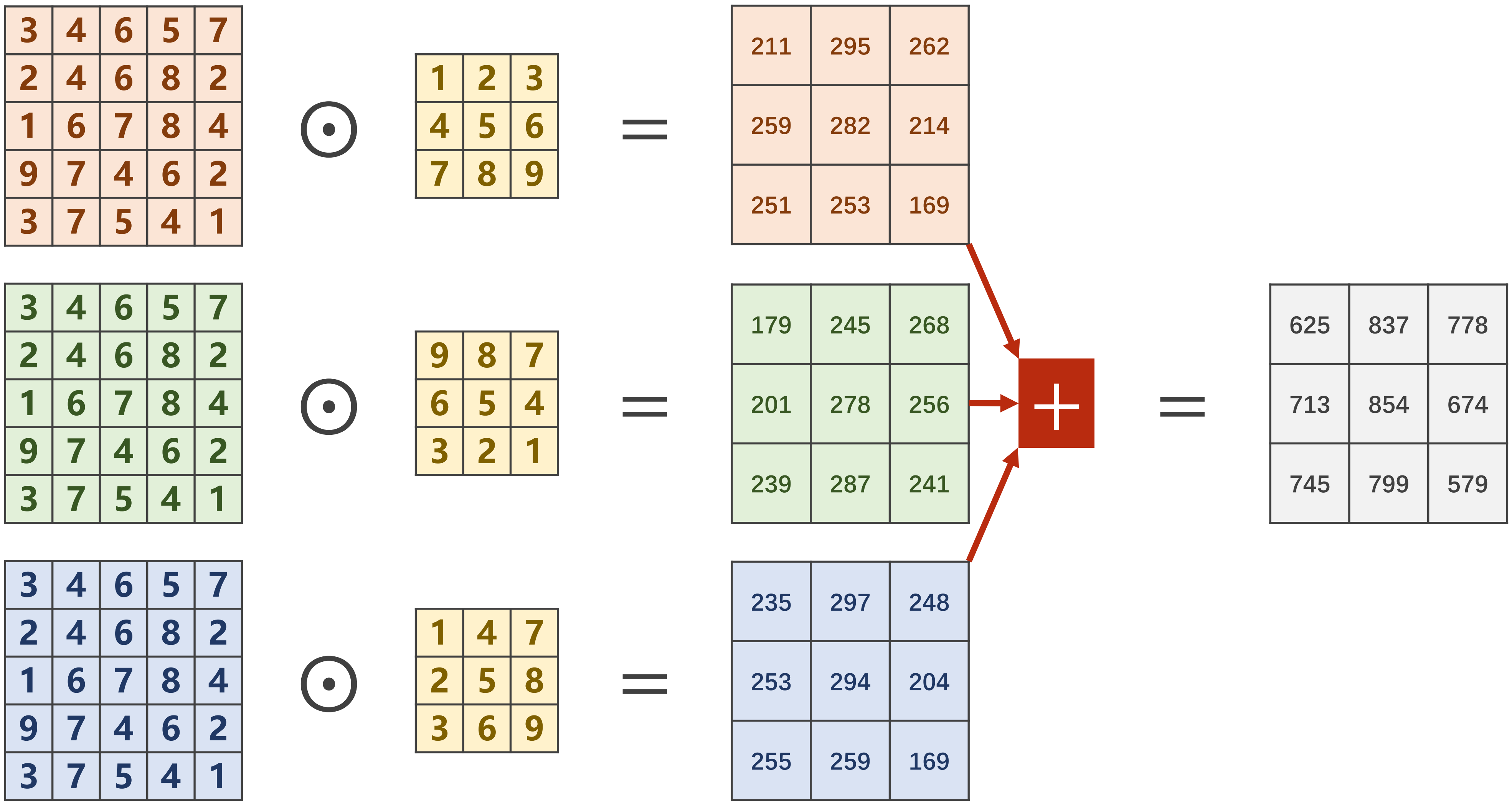
$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 211 & 295 & 262 \\ 259 & 282 & 214 \\ 251 & 253 & 169 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix} \odot \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 179 & 245 & 268 \\ 201 & 278 & 256 \\ 239 & 287 & 241 \end{bmatrix}$$

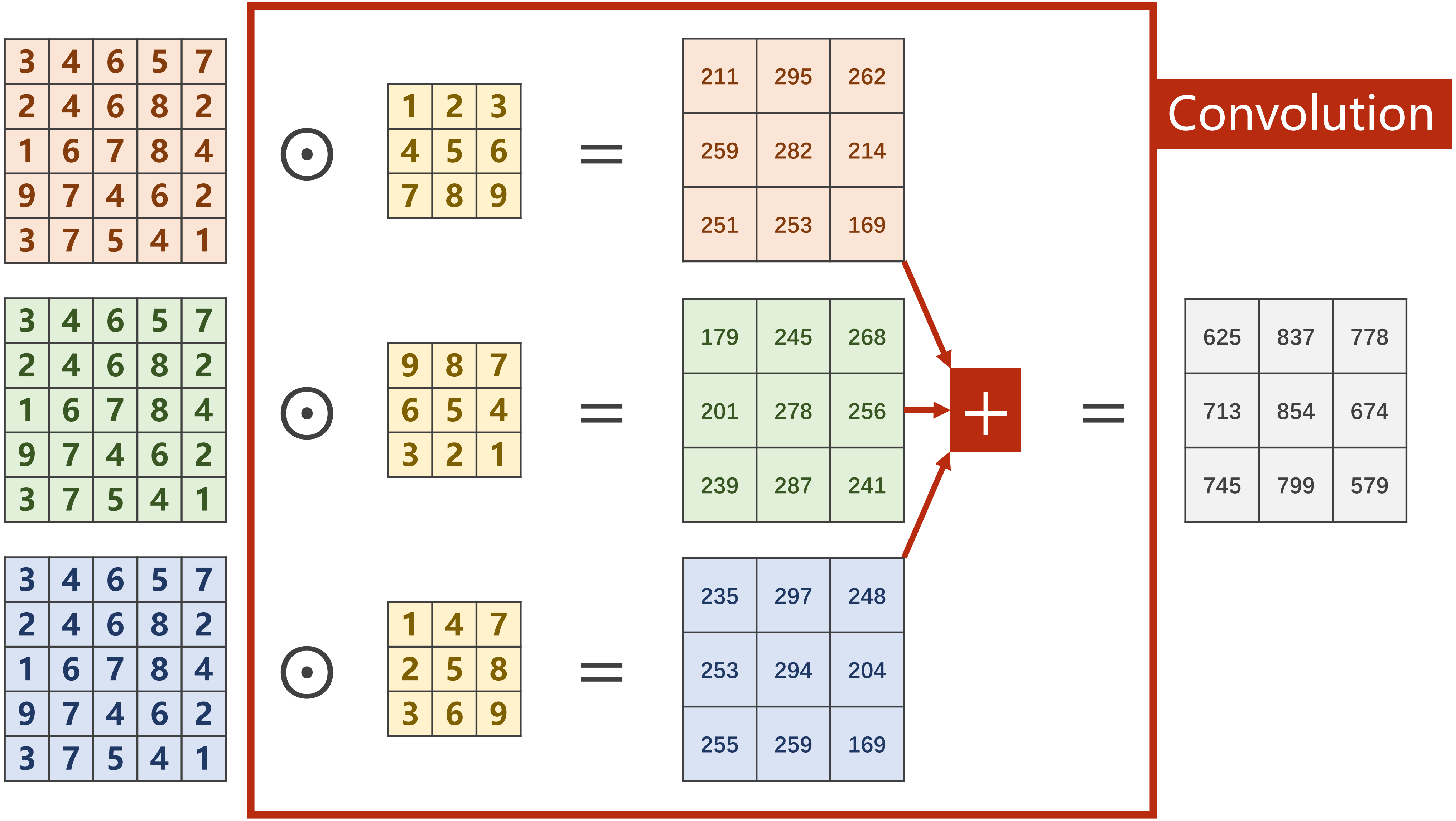
$$\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} = \begin{bmatrix} 235 & 297 & 248 \\ 253 & 294 & 204 \\ 255 & 259 & 169 \end{bmatrix}$$

$$+$$

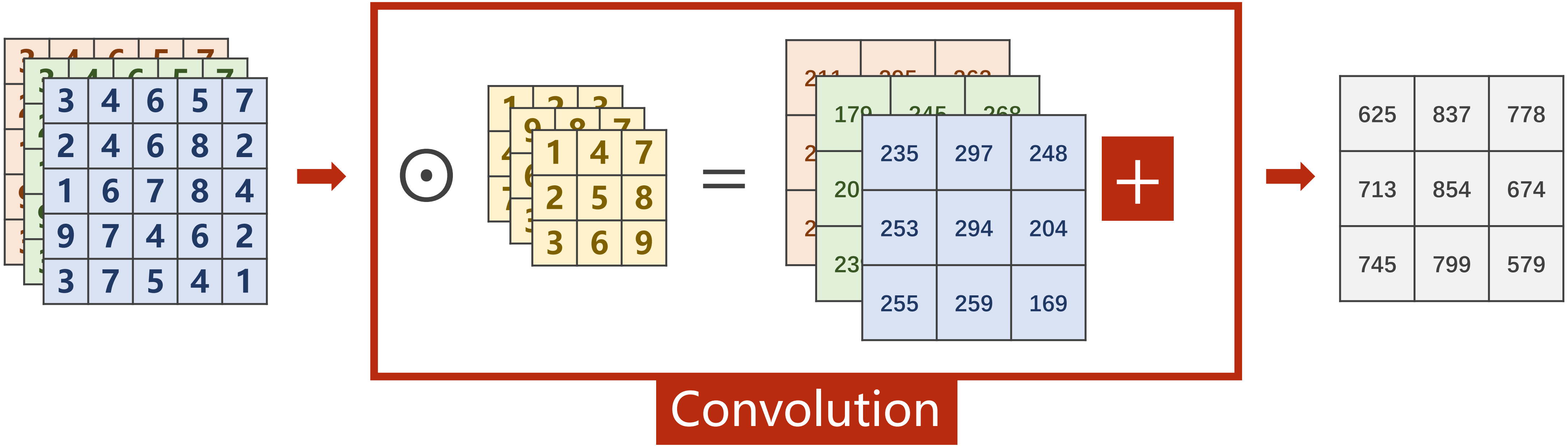
Convolution – 3 Input Channels

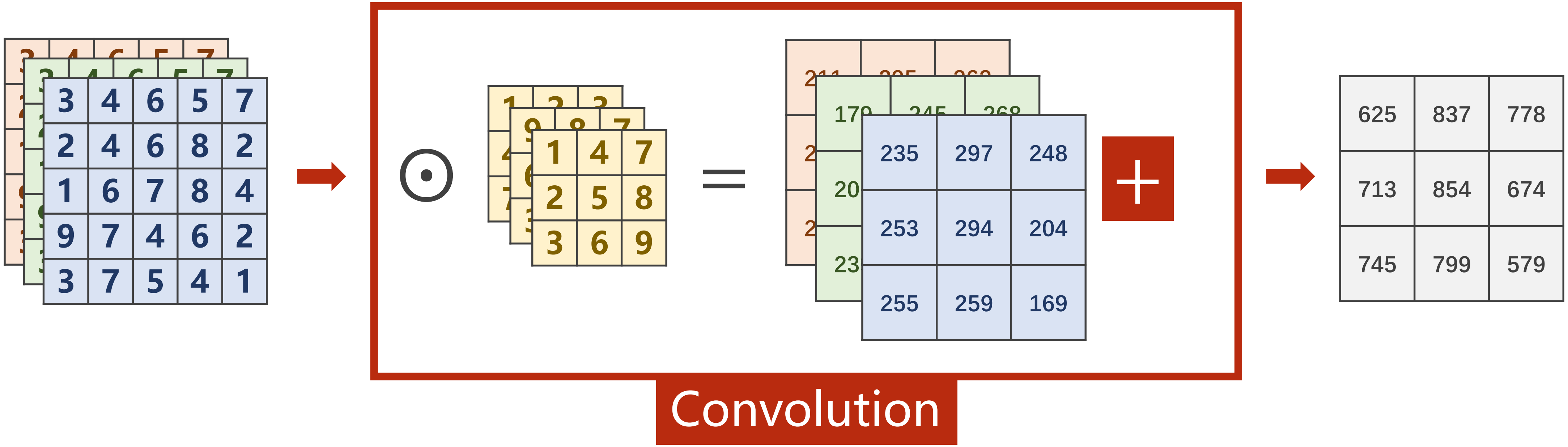


Convolution – 3 Input Channels

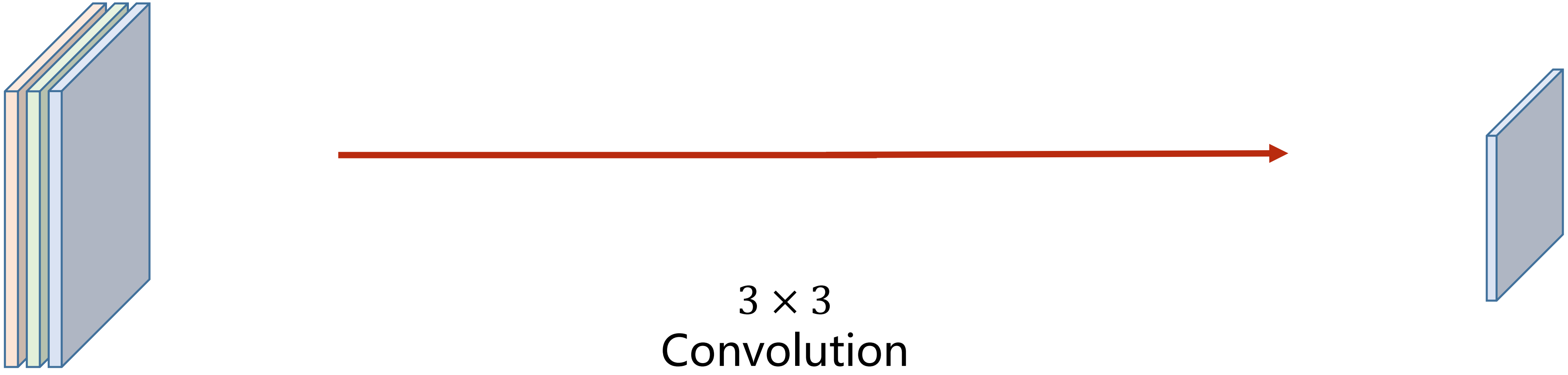


Convolution – 3 Input Channels



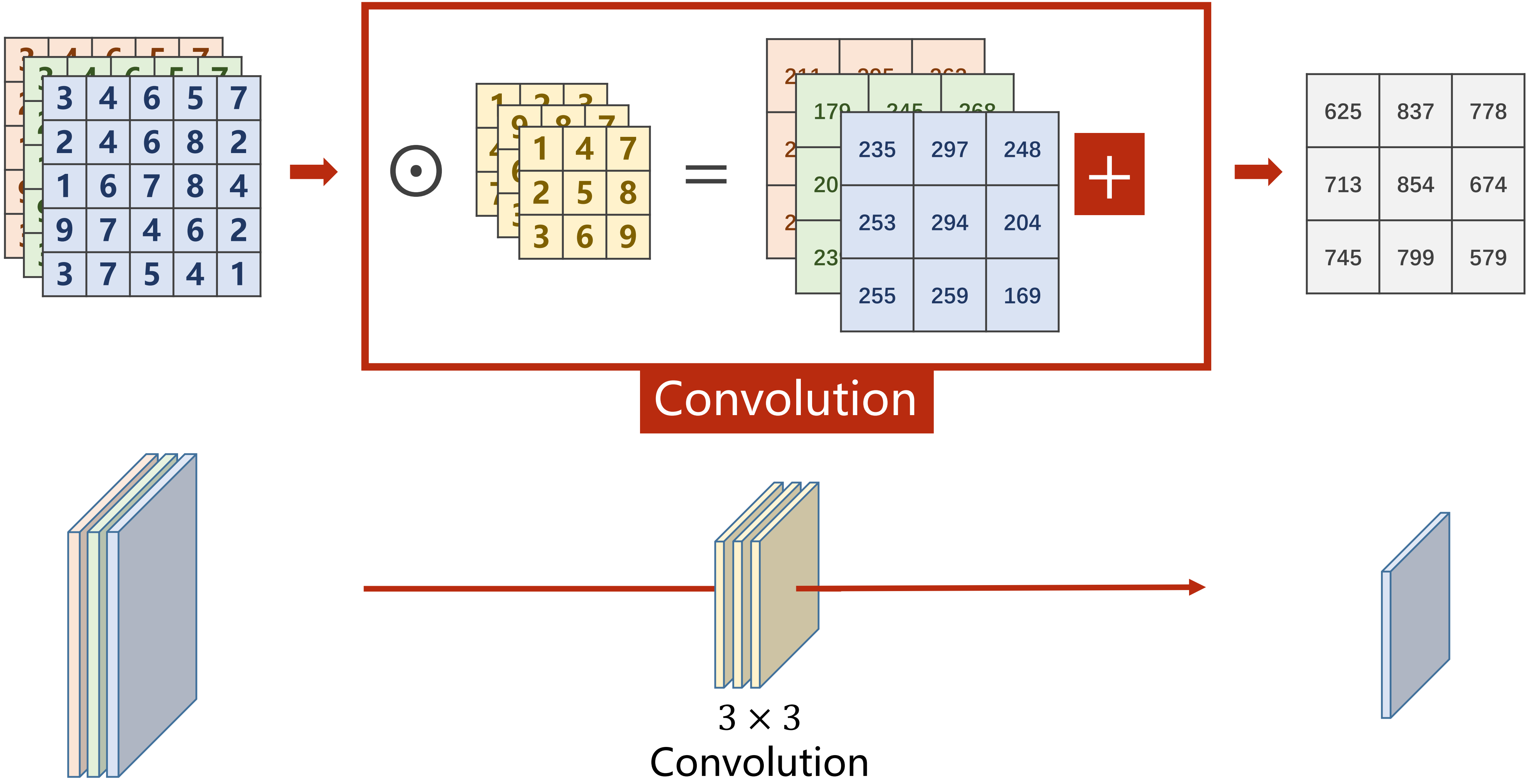


3 4 6 5 7
2 4 6 8 2
1 6 7 8 4
9 7 4 6 2
3 7 5 4 1
⊙
1 4 7
2 5 8
3 6 9
=
235 297 248
253 294 204
255 259 169
+
625 837 778
713 854 674
745 799 579
Convolution

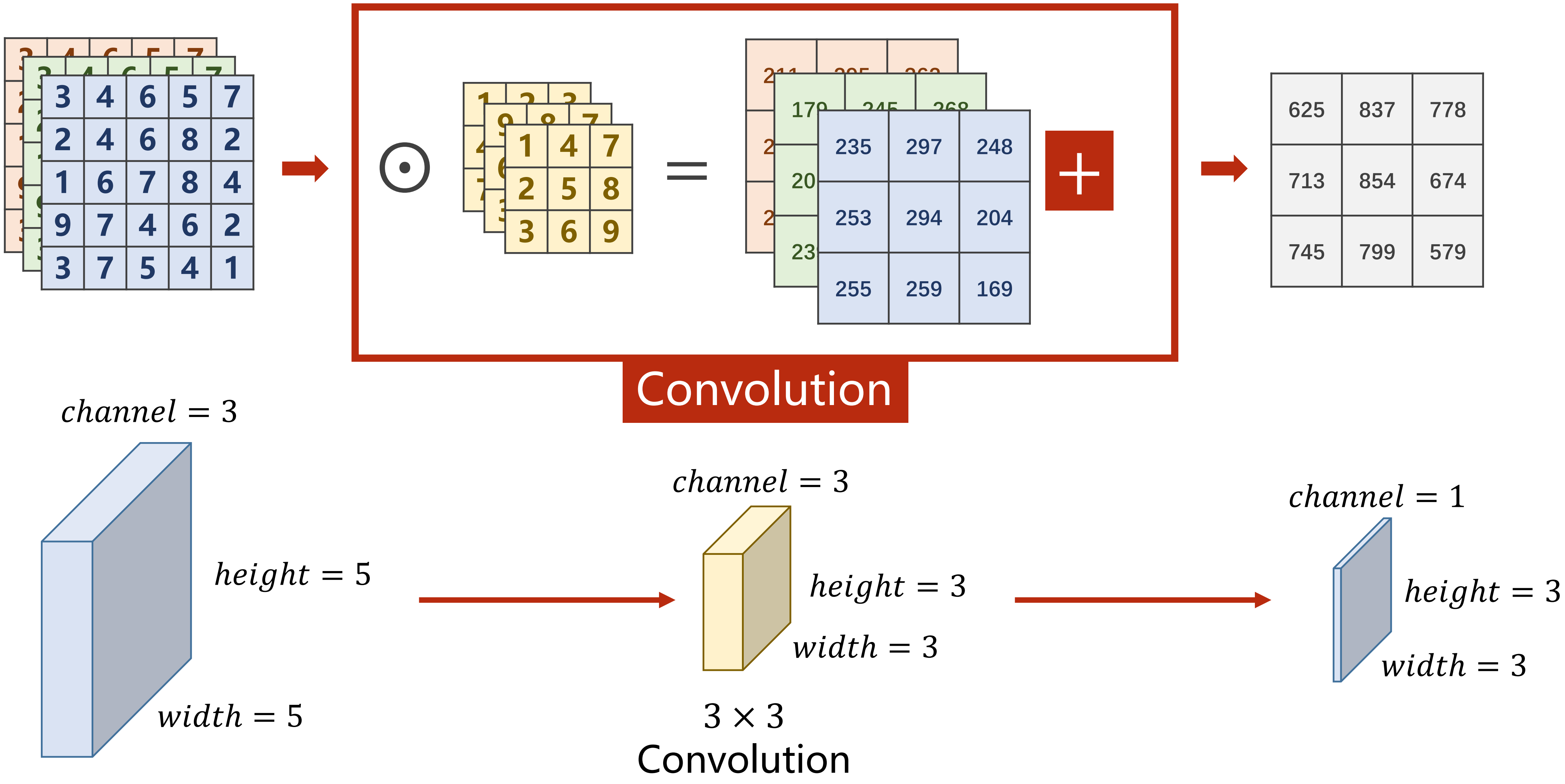


3 × 3
Convolution

Convolution – 3 Input Channels

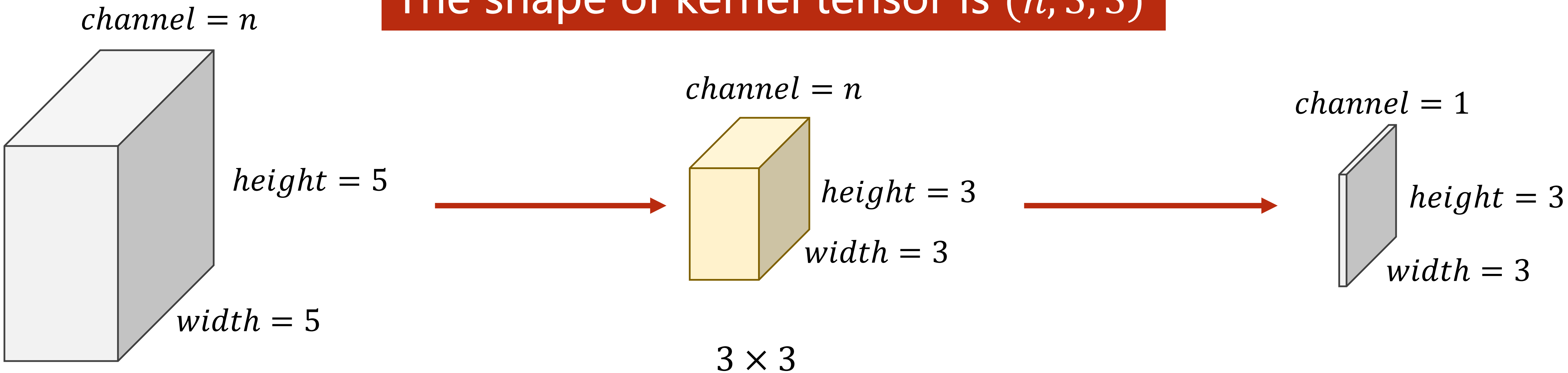


Convolution – 3 Input Channels



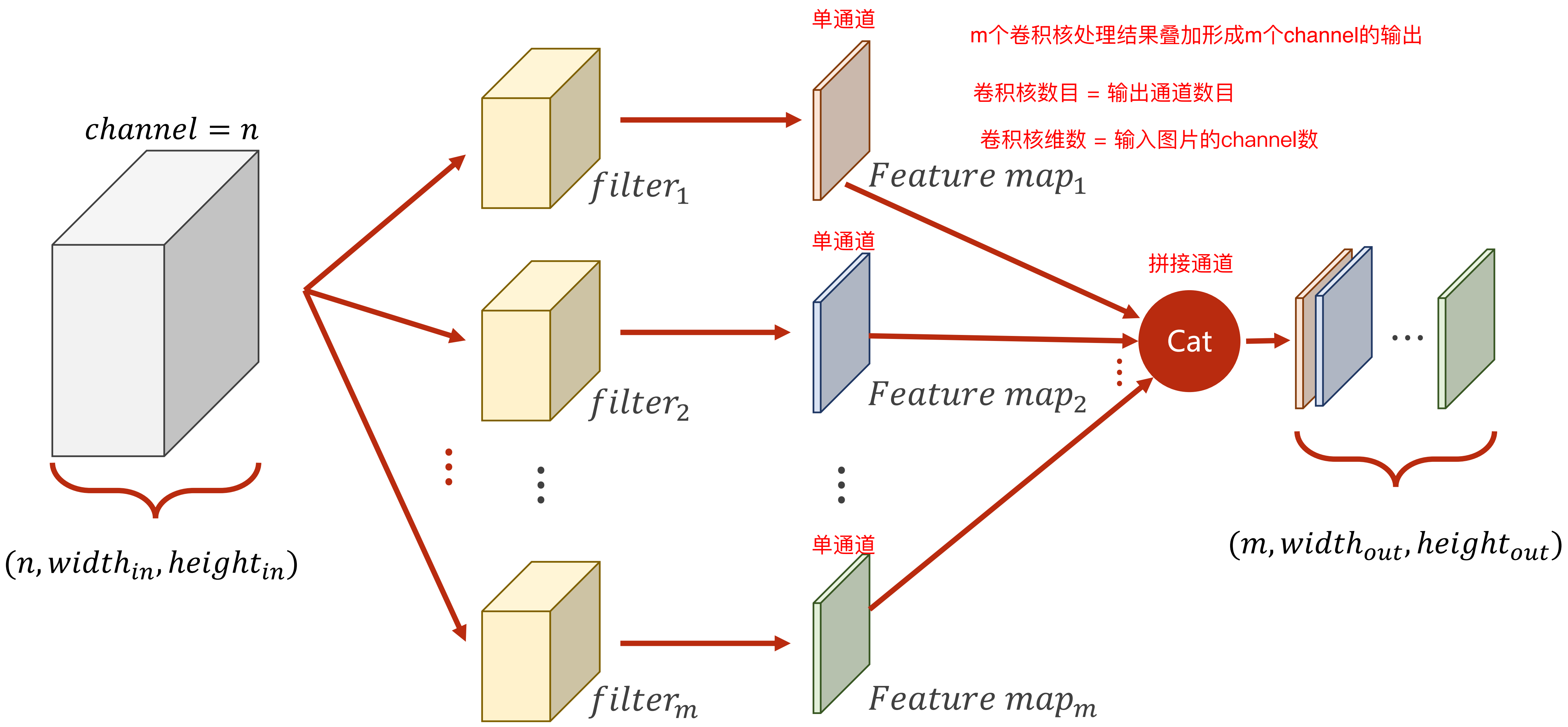
Convolution – N Input Channels

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



3×3
Convolution

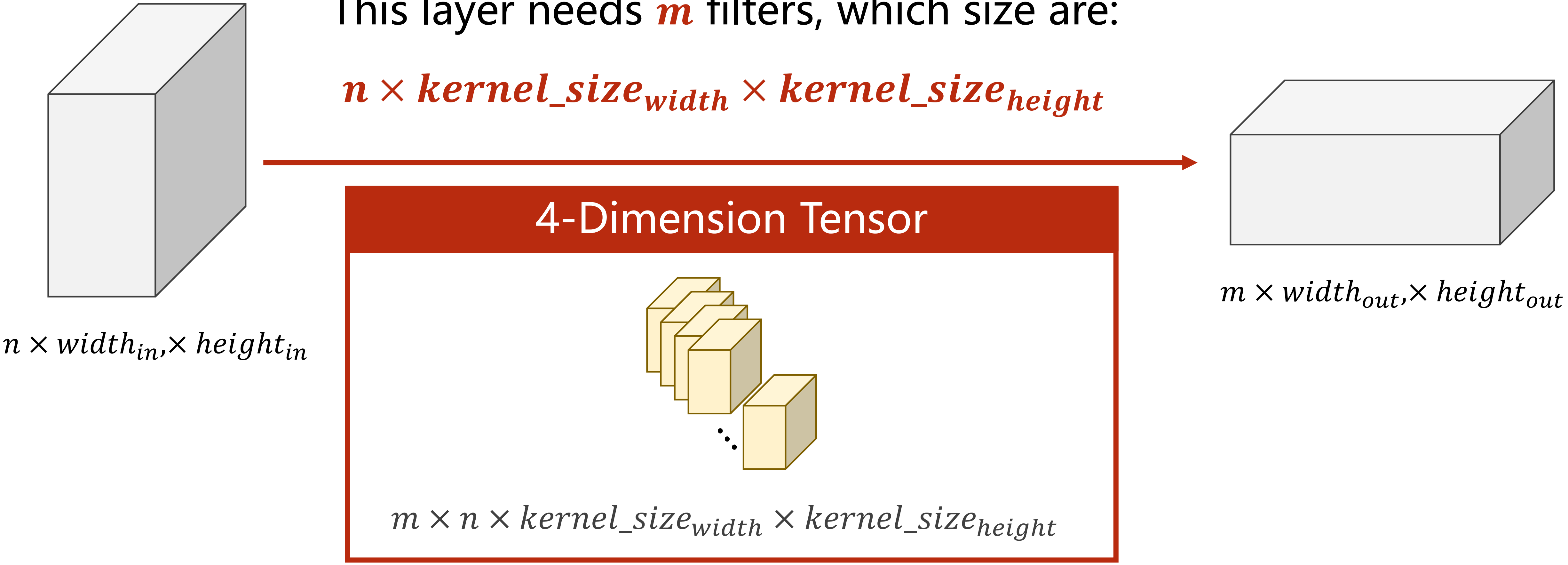
Convolution – N Input Channels and M Output Channels



Convolutional Layer

This layer needs m filters, which size are:

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



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)
```

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)
```

Convolutional Layer

```python
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)
```

Convolutional Layer

```python
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)
```

Convolutional Layer

```python
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)
```

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])
```

Convolutional Layer – padding

边缘填充，扩展input大小，使output满足大小要求。
对于3*3的卷积核，如果需要output与input形态大小一样，则需要填充一圈。5*5则需要填充两圈

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	295	262
259	282	214
251	253	169

Convolutional Layer – padding=1

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	295	262	
	259	282	214	
	251	253	169	

Convolutional Layer – padding=1

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

$\odot$

Kernel

1	2	3
4	5	6
7	8	9

$=$

Output

	211	295	262	
	259	282	214	
	251	253	169	

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

$\odot$

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

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

$\odot$

Kernel

1	2	3
4	5	6
7	8	9

$=$

Output

91	168	224	215	127
114	211	295	262	149
192	259	282	214	122
194	251	253	169	86
96	112	110	68	31

Convolutional Layer – padding=1

```python
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

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

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	

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	

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

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

Convolutional Layer – stride=2

Input **Kernel** **Output**

$$
\begin{bmatrix} 3 & 4 & 6 & 5 & 7 \\ 2 & 4 & 6 & 8 & 2 \\ 1 & 6 & 7 & 8 & 4 \\ 9 & 7 & 4 & 6 & 2 \\ 3 & 7 & 5 & 4 & 1 \end{bmatrix}
\odot
\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}
=
\begin{bmatrix} 211 & 262 \\ 251 & \end{bmatrix}
$$

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

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	169

Convolutional Layer – stride=2

```python
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

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

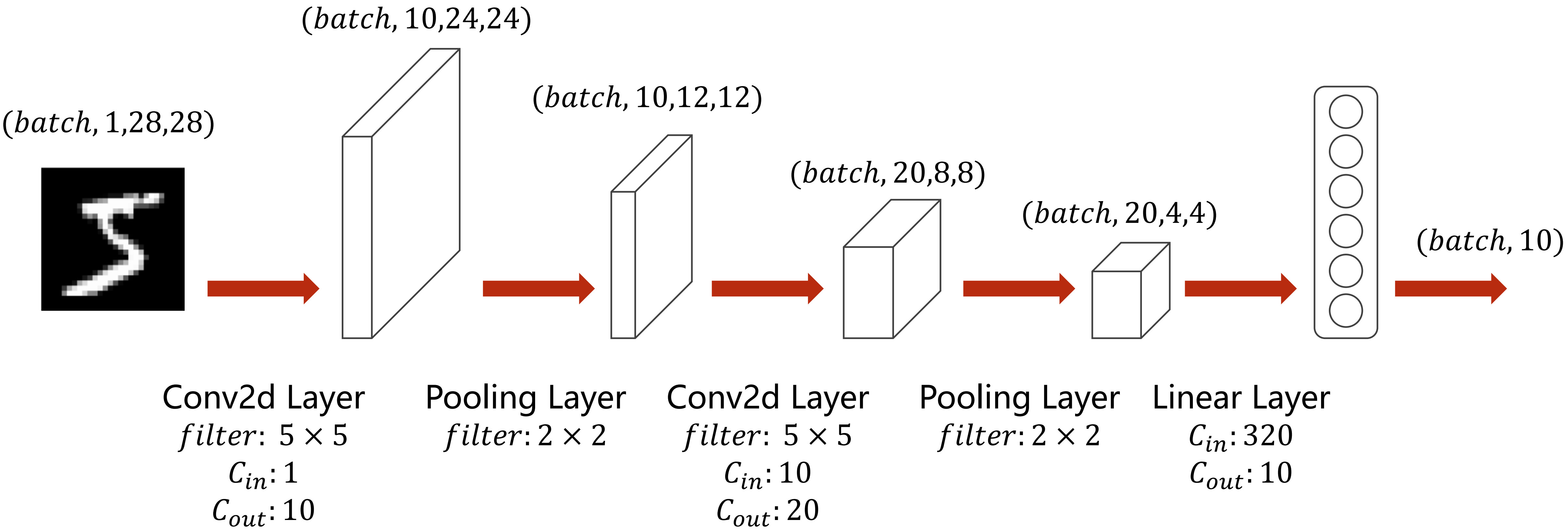
2×2
MaxPooling

4	8
9	8

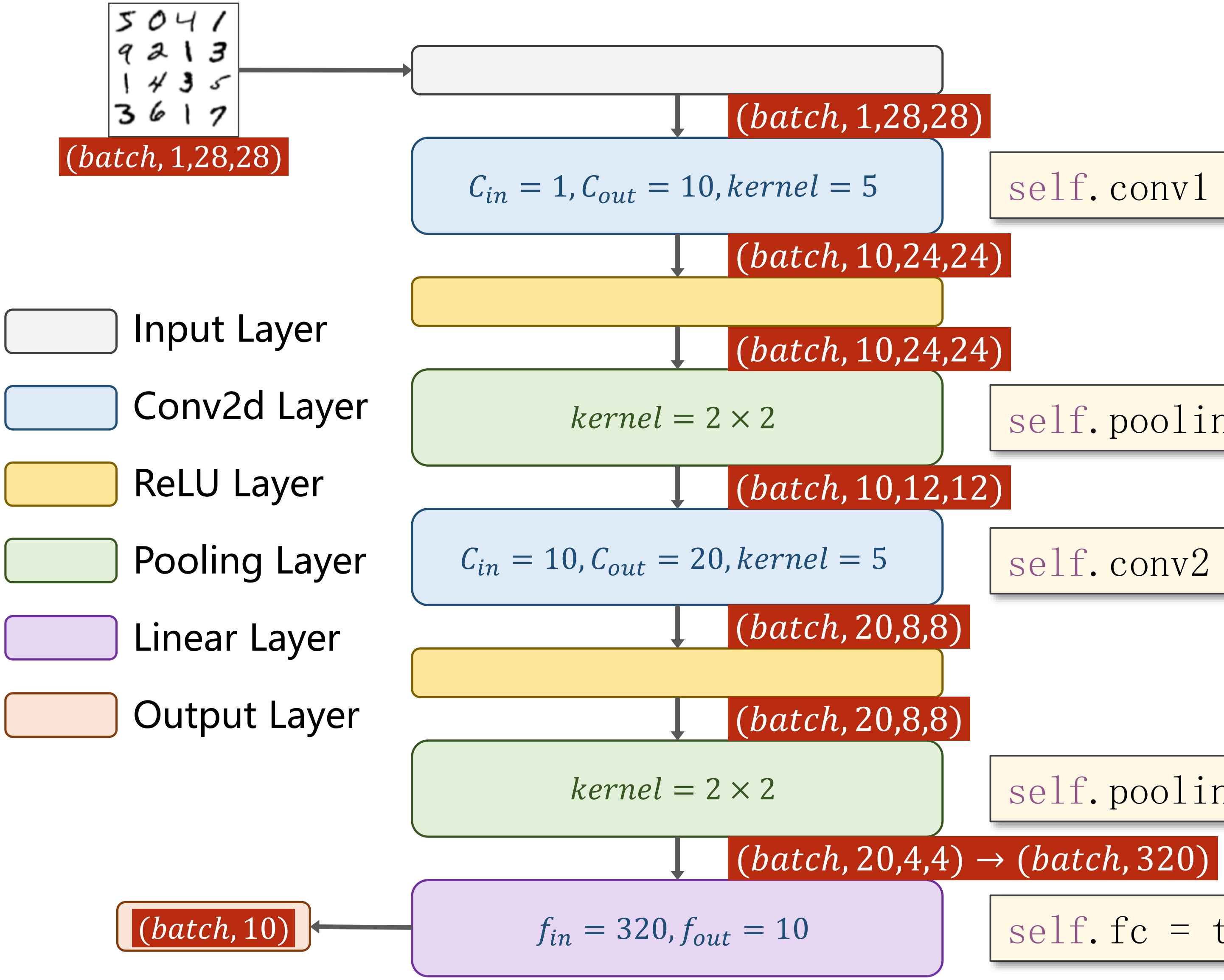
Max Pooling Layer

```python
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




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

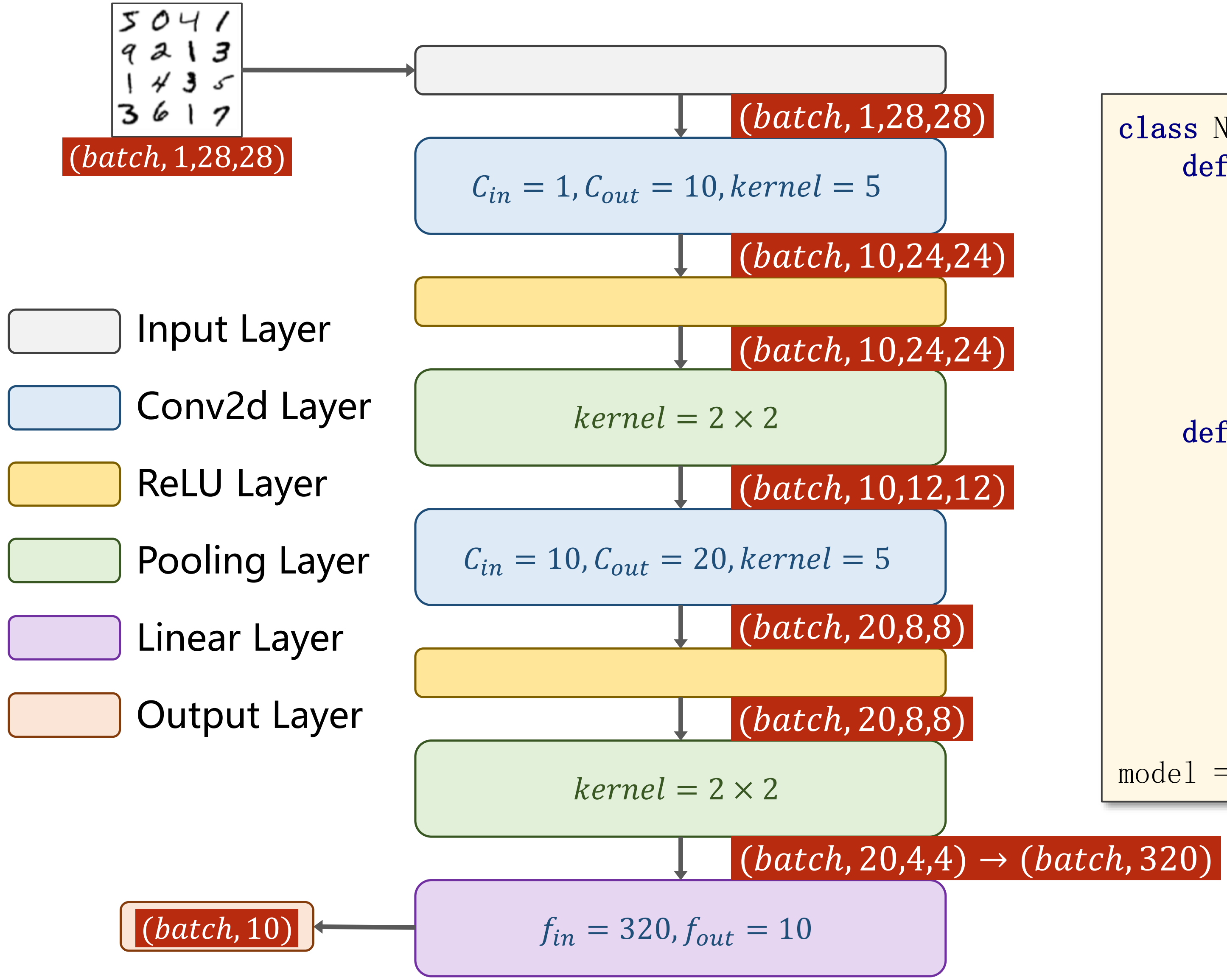
```python
self.pooling = torch.nn.MaxPool2d(2)
```

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

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

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

Revision: Fully Connected Neural Network




```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)

    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()
```

How to use GPU – 1. Move Model to GPU

```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)

    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()
```

How to use GPU – 1. Move Model to GPU

```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)

    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.

How to use GPU – 1. Move Model to GPU

```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)

    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)
```

Convert parameters and buffers of all modules to CUDA Tensor.

How to use GPU – 2. Move Tensors to GPU

```
def train(epoch):
    running_loss = 0.0
    for batch_idx, data in enumerate(train_loader, 0):
        inputs, target = data
        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
```

How to use GPU – 2. Move Tensors to GPU

```python
def train(epoch):
    running_loss = 0.0
    for batch_idx, data in enumerate(train_loader, 0):
        inputs, target = data
        inputs, target = inputs.to(device), target.to(device)
        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

```
def test():
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            inputs, target = data
            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))
```

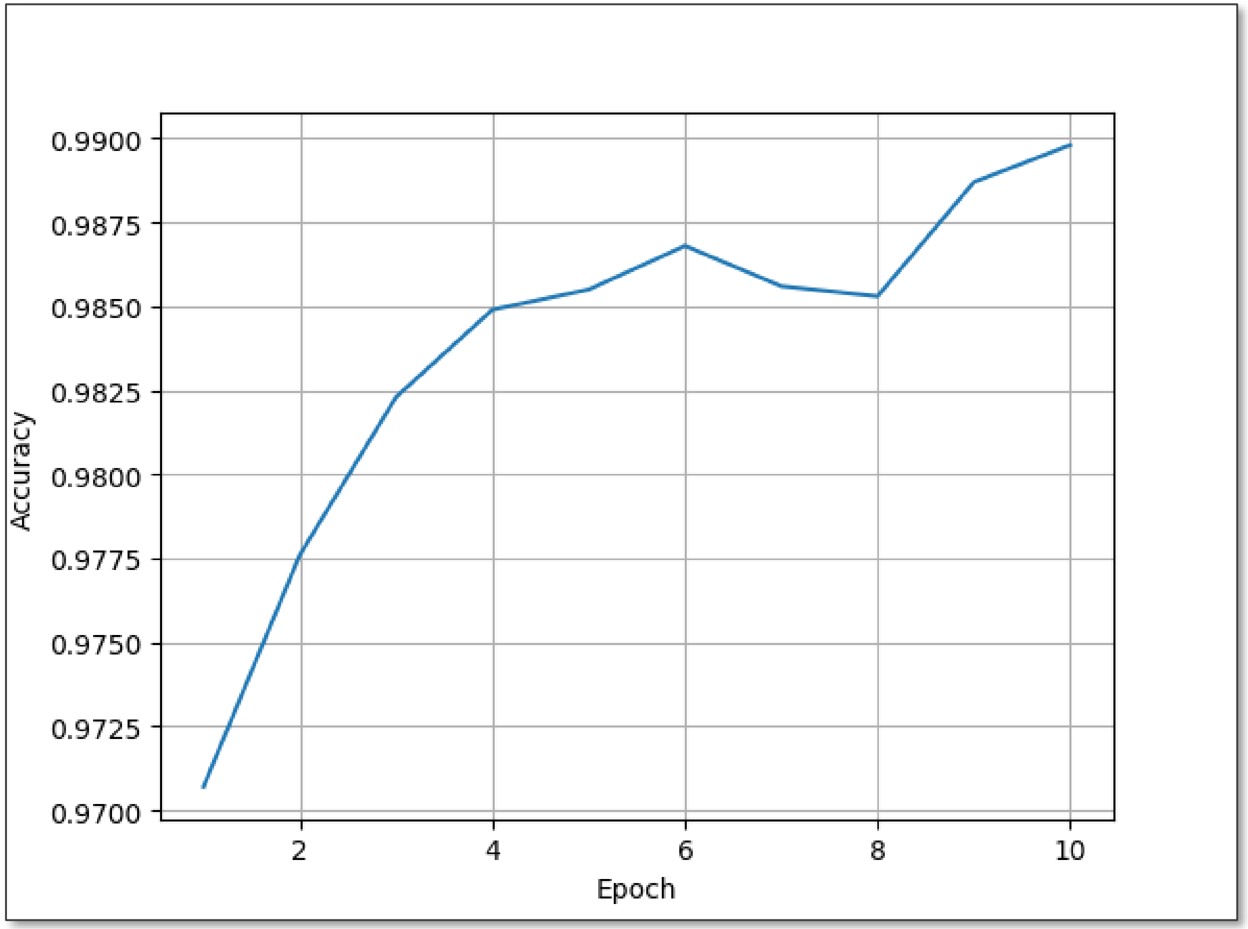
How to use GPU – 2. Move Tensors to GPU

```
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)
            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))
```

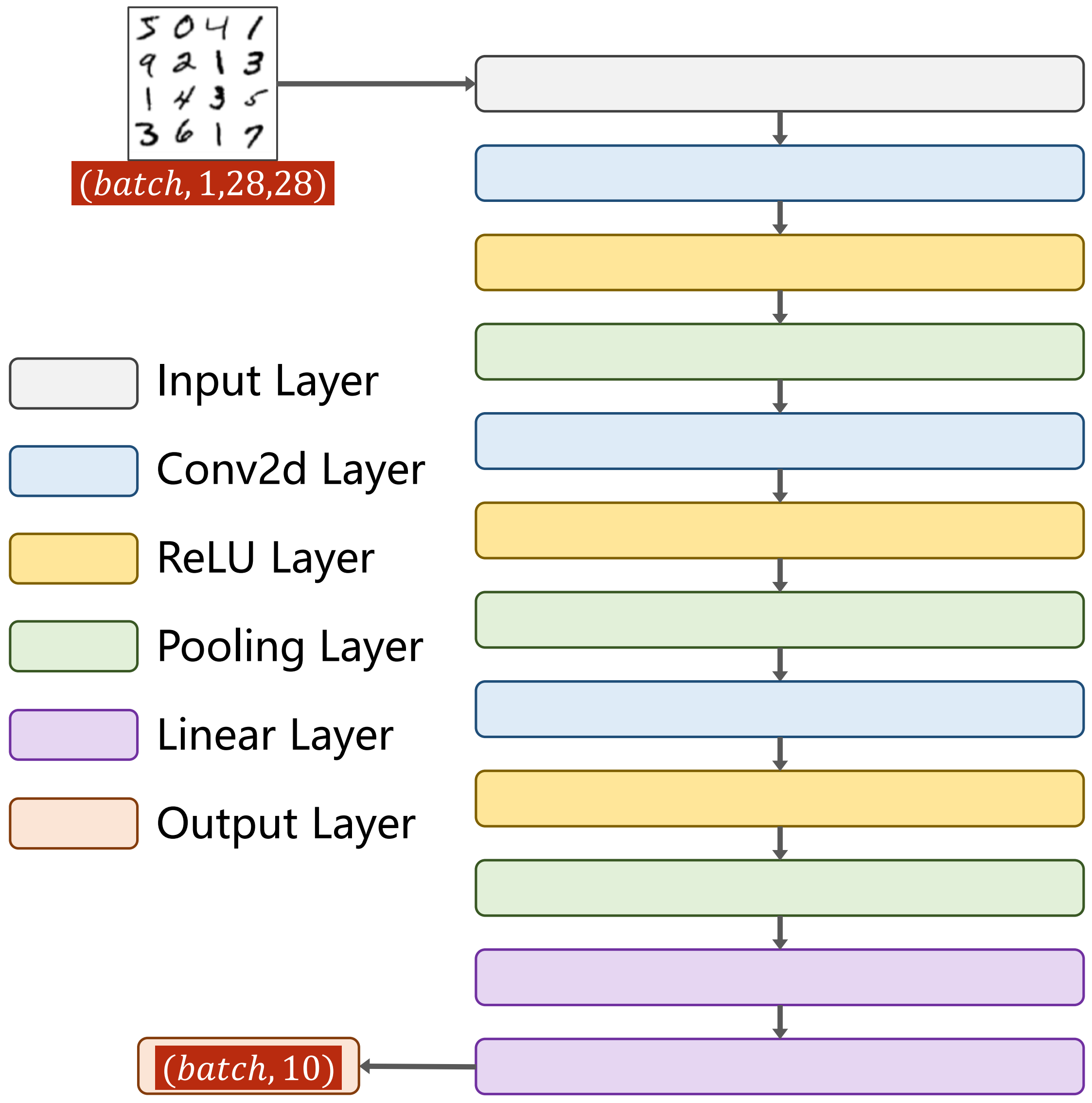
Send the inputs and targets at every step to the GPU.

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