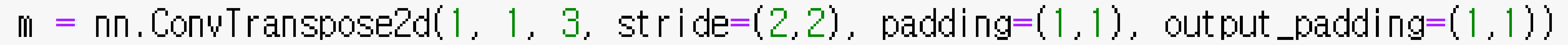
FSRCNN 분석

염지현

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



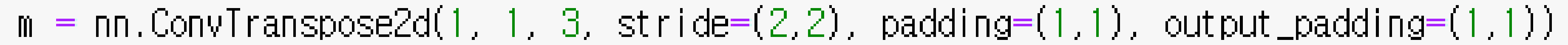
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

$$output\ size = (input\ size\ - 1) * stride\ - 2\ * padding + dilation\ * (kernel\ size\ - 1) + output\ padding + 1$$

$$output\ size = (3\ - 1) * 2\ - 2\ * 1 + 1\ * (3 - 1) + 1 + 1 = 6$$

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



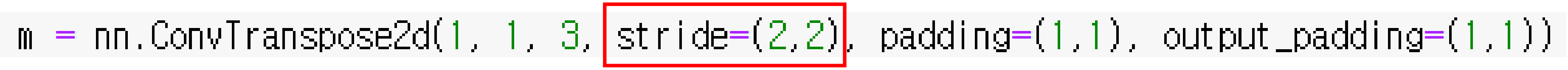
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

1	1.1	1.2
1.3	1.4	1.5
1.6	1.7	1.8

input

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

1	1.1	1.2
1.3	1.4	1.5
1.6	1.7	1.8

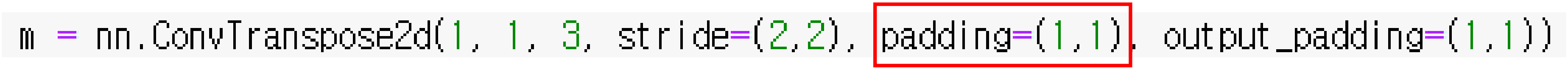
input

1	0	1.1	0	1.2
0	0	0	0	0
1.3	0	1.4	0	1.5
0	0	0	0	0
1.6	0	1.7	0	1.8

z = stride - 1
각 행과 열 사이에
z만큼 0을 추가

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

1	1.1	1.2
1.3	1.4	1.5
1.6	1.7	1.8

input

1	0	1.1	0	1.2
0	0	0	0	0
1.3	0	1.4	0	1.5
0	0	0	0	0
1.6	0	1.7	0	1.8

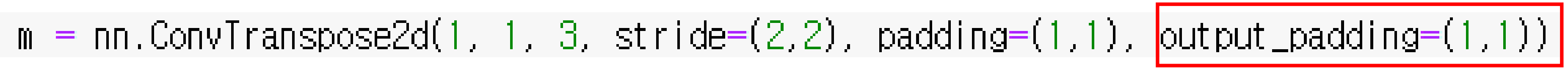
z = stride - 1
각 행과 열 사이에
z만큼 0을 추가

0	0	0	0	0	0	0
0	1	0	1.1	0	1.2	0
0	0	0	0	0	0	0
0	1.3	0	1.4	0	1.5	0
0	0	0	0	0	0	0
0	1.6	0	1.7	0	1.8	0
0	0	0	0	0	0	0

P' = k - p - 1
p;'만큼 padding 추가

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

1	1.1	1.2
1.3	1.4	1.5
1.6	1.7	1.8

input

1	0	1.1	0	1.2
0	0	0	0	0
1.3	0	1.4	0	1.5
0	0	0	0	0
1.6	0	1.7	0	1.8

z = stride - 1
각 행과 열 사이에
z만큼 0을 추가

0	0	0	0	0	0	0
0	1	0	1.1	0	1.2	0
0	0	0	0	0	0	0
0	1.3	0	1.4	0	1.5	0
0	0	0	0	0	0	0
0	1.6	0	1.7	0	1.8	0
0	0	0	0	0	0	0

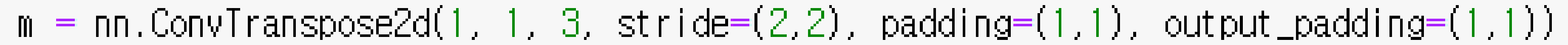
P’ = k - p - 1
p;’만큼 padding 추가

0	0	0	0	0	0	0	0
0	1	0	1.1	0	1.2	0	0
0	0	0	0	0	0	0	0
0	1.3	0	1.4	0	1.5	0	0
0	0	0	0	0	0	0	0
0	1.6	0	1.7	0	1.8	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

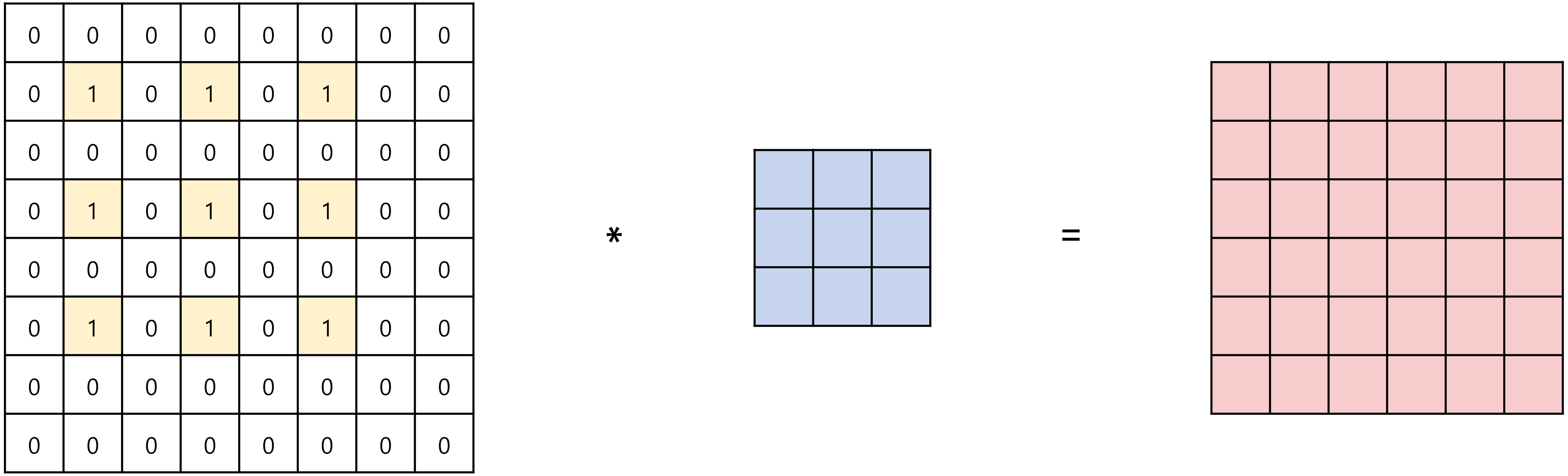
Output_padding 만큼
아래쪽, 오른쪽에 0으로 패딩 추가

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



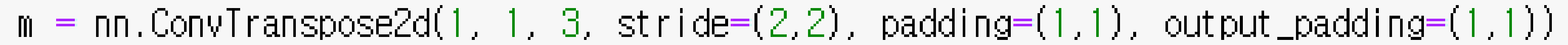
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



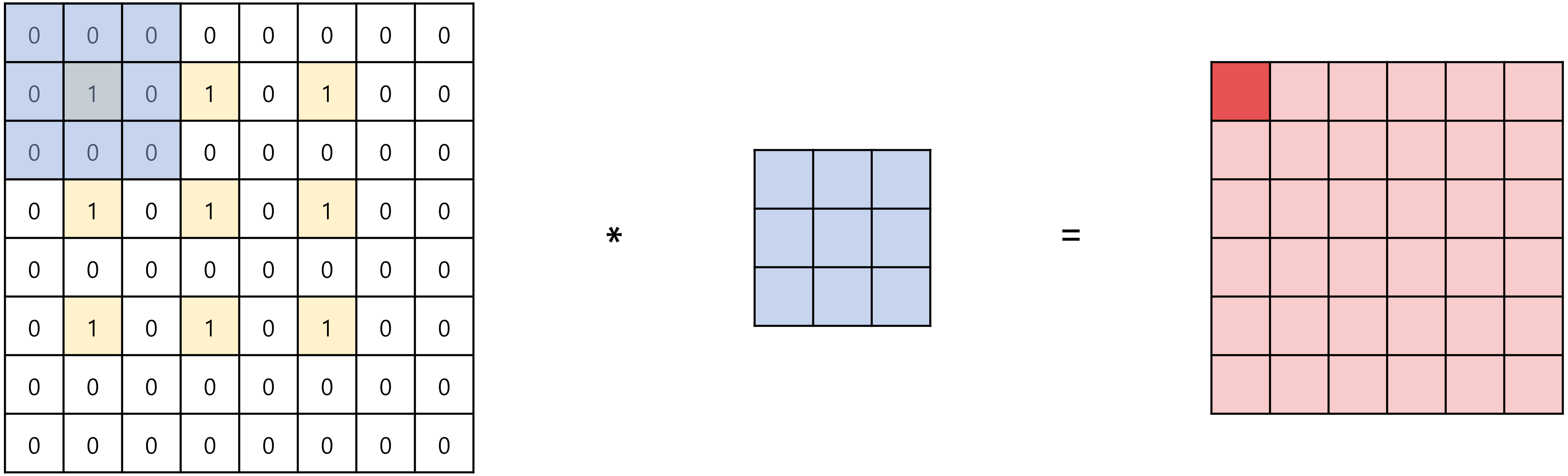
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

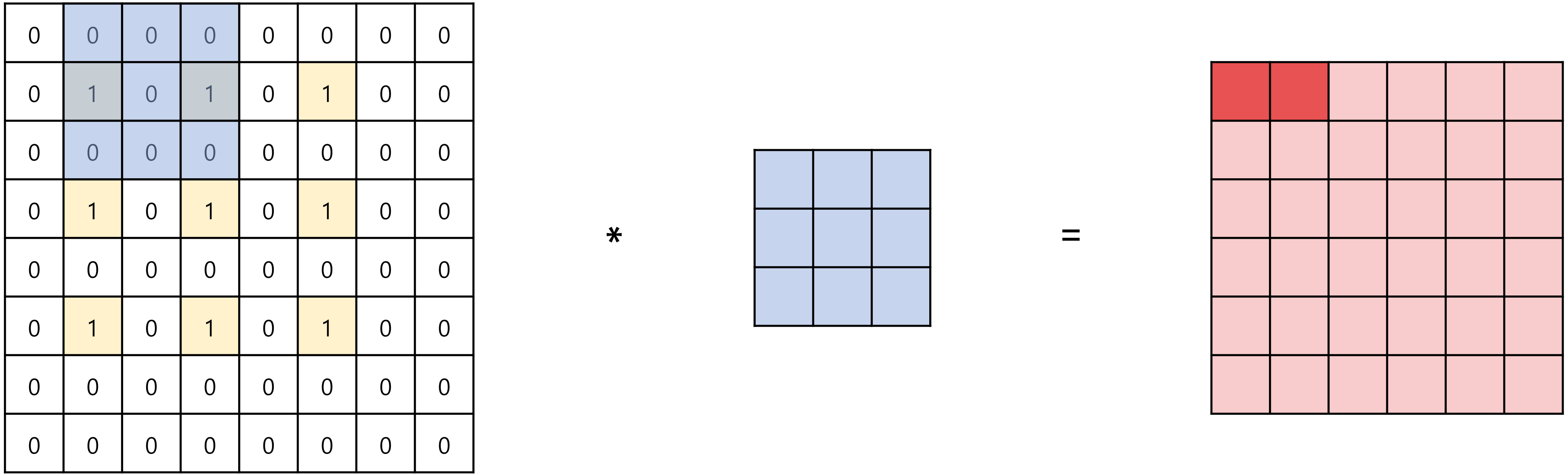


0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]

```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

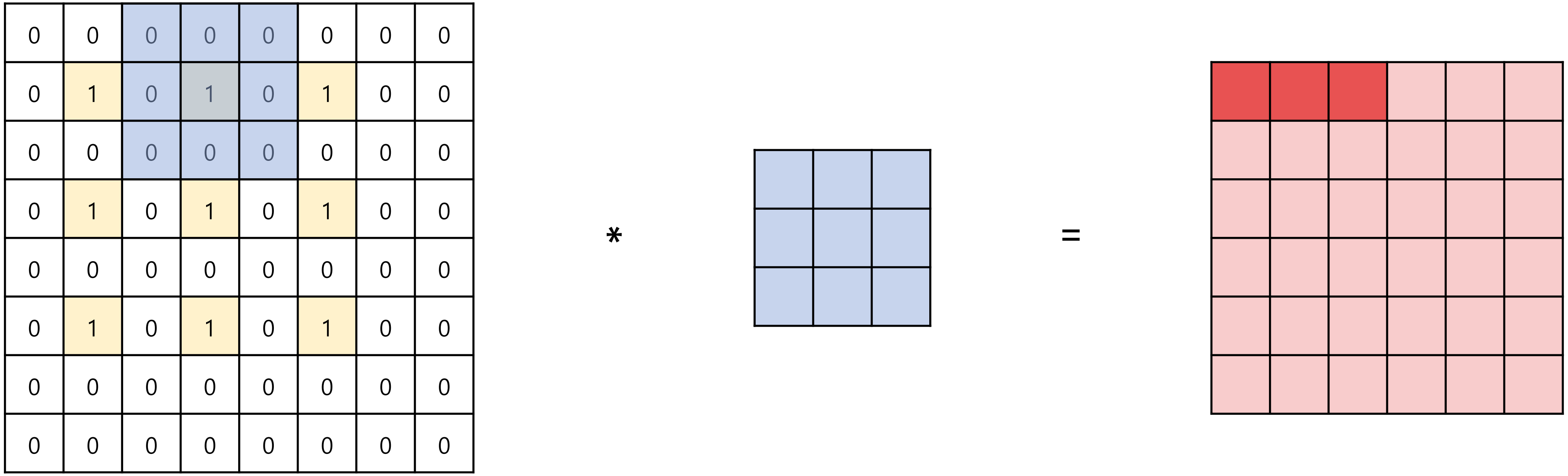


0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]

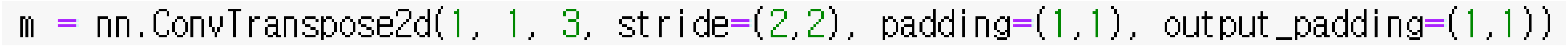
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



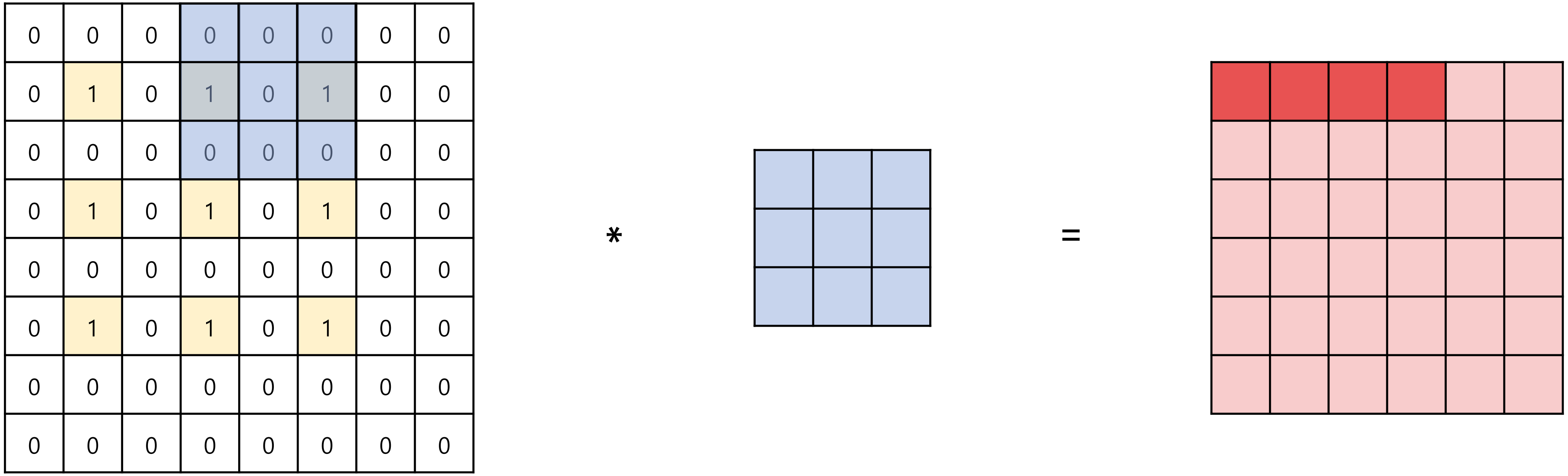
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



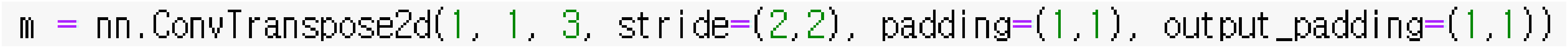
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



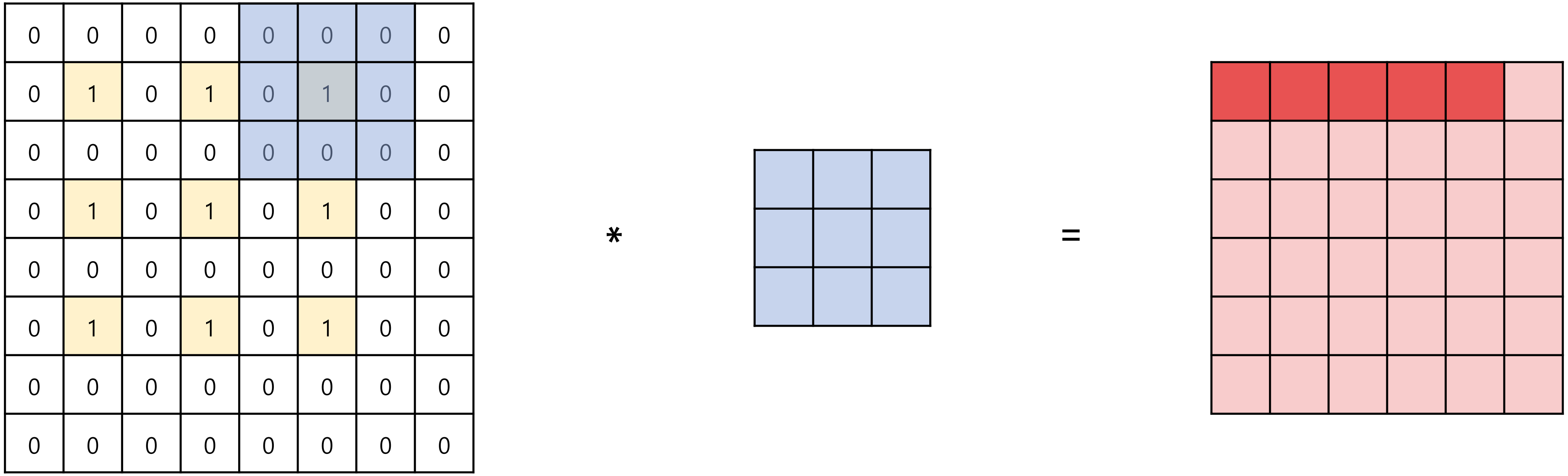
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



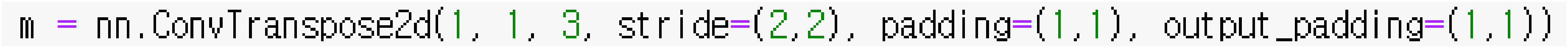
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



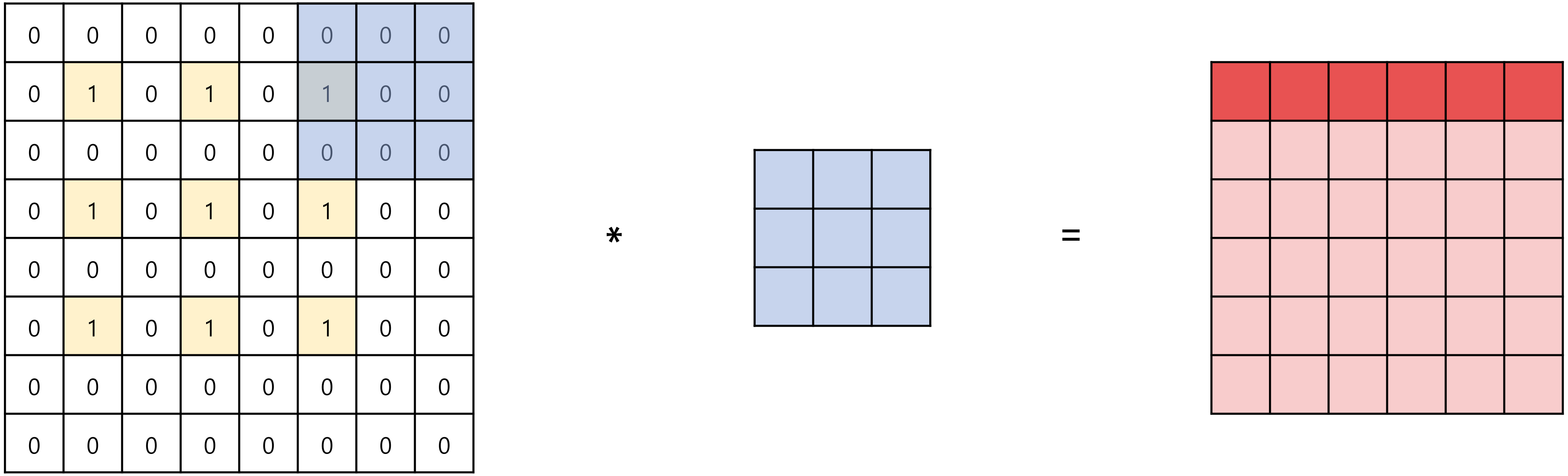
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



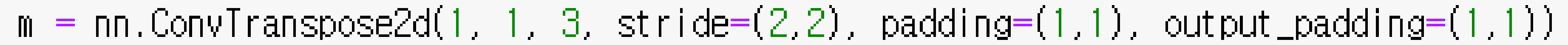
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



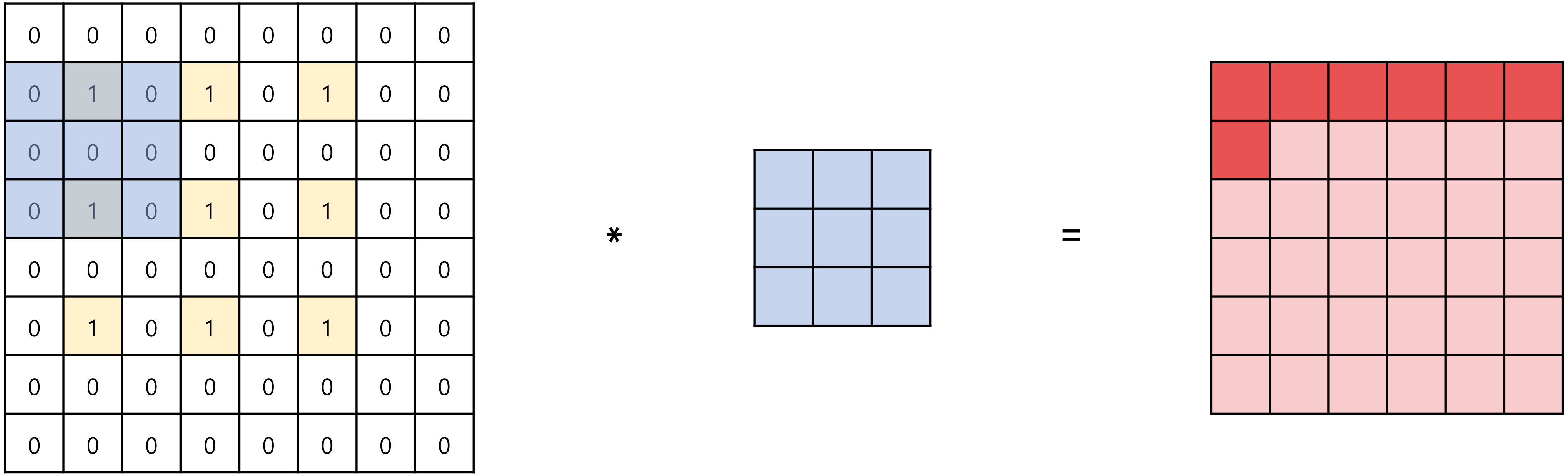
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

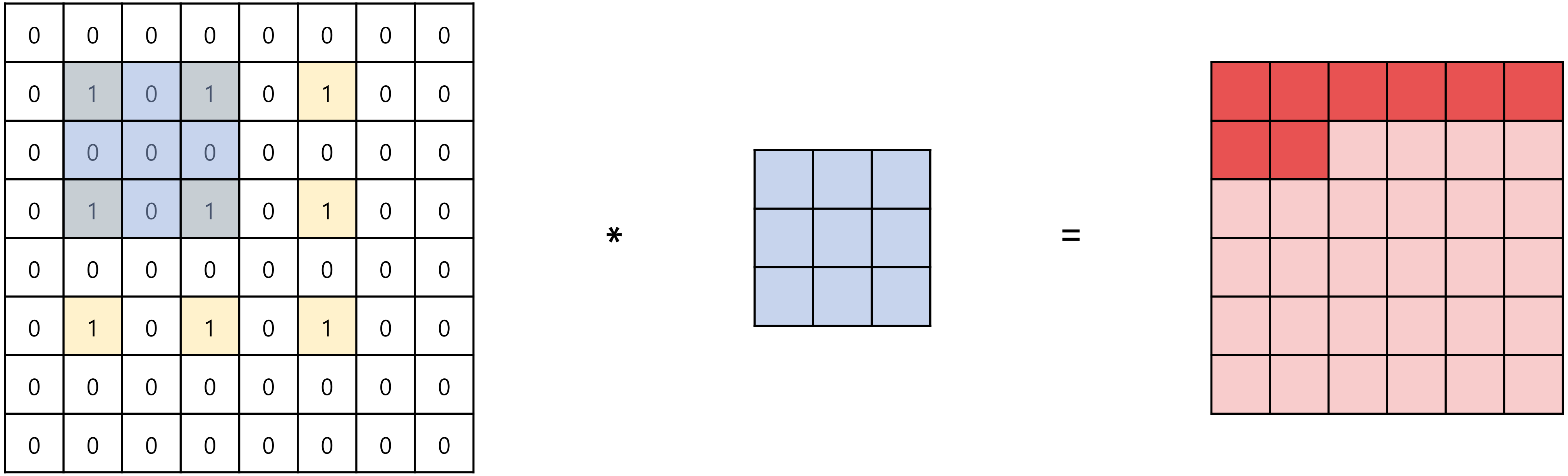


0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]

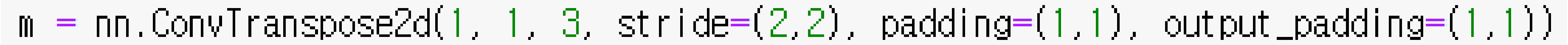
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



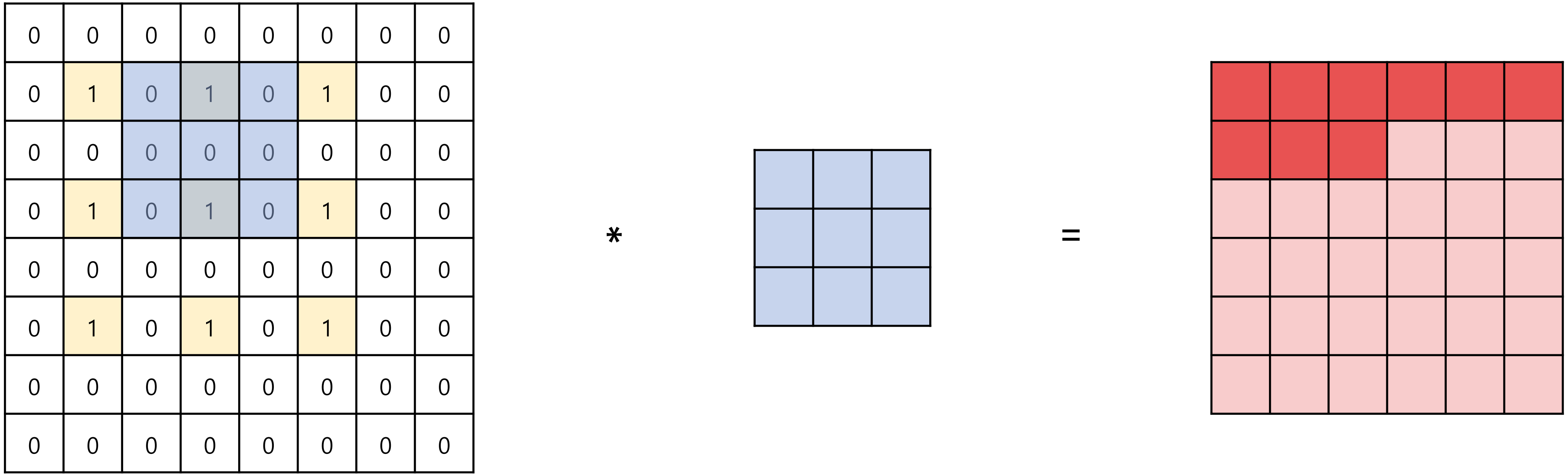
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

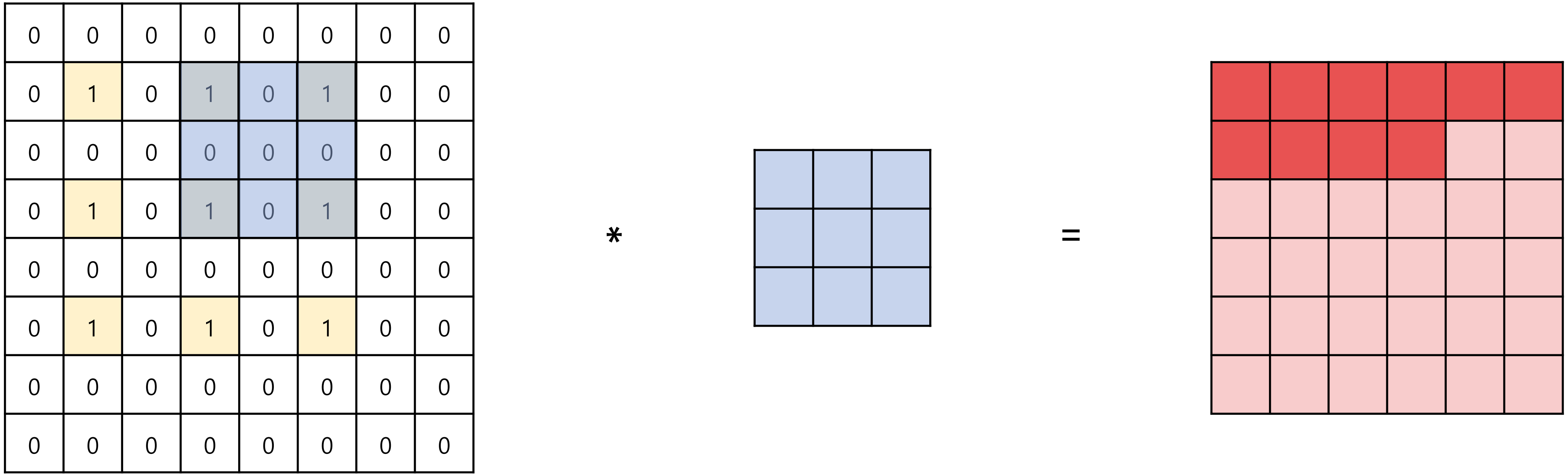


0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]

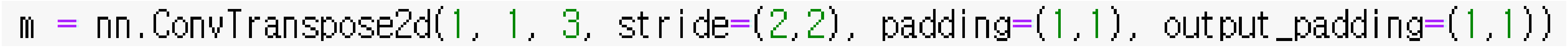
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



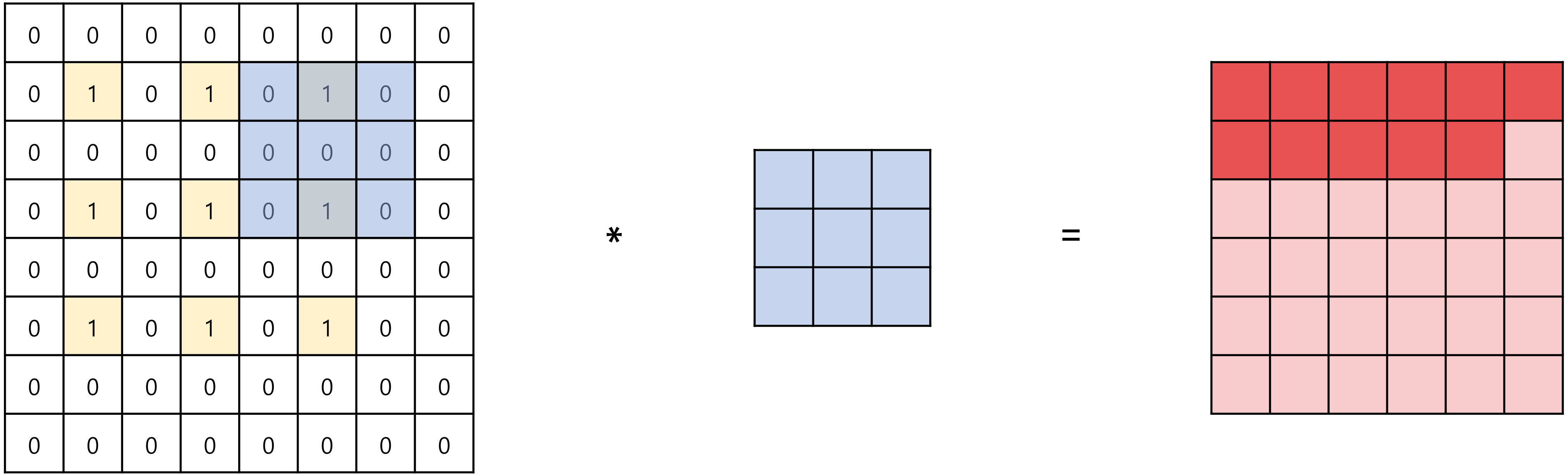
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

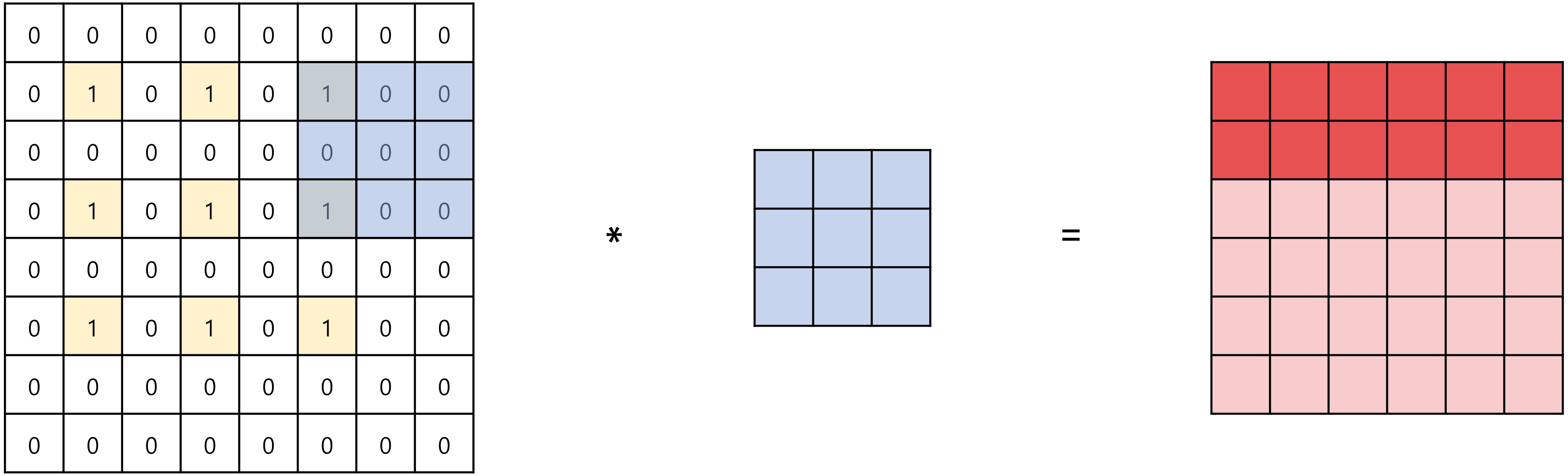


0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]

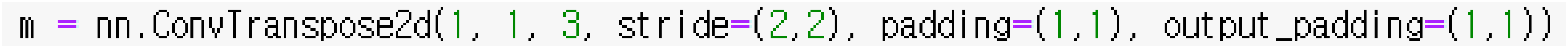
```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



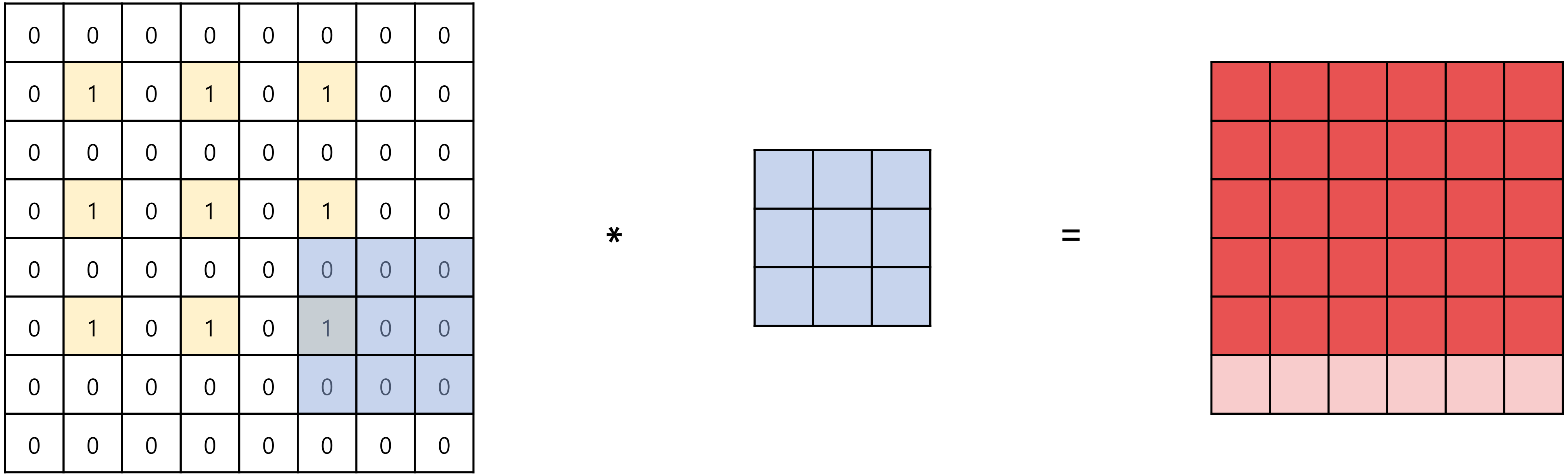
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]



```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```

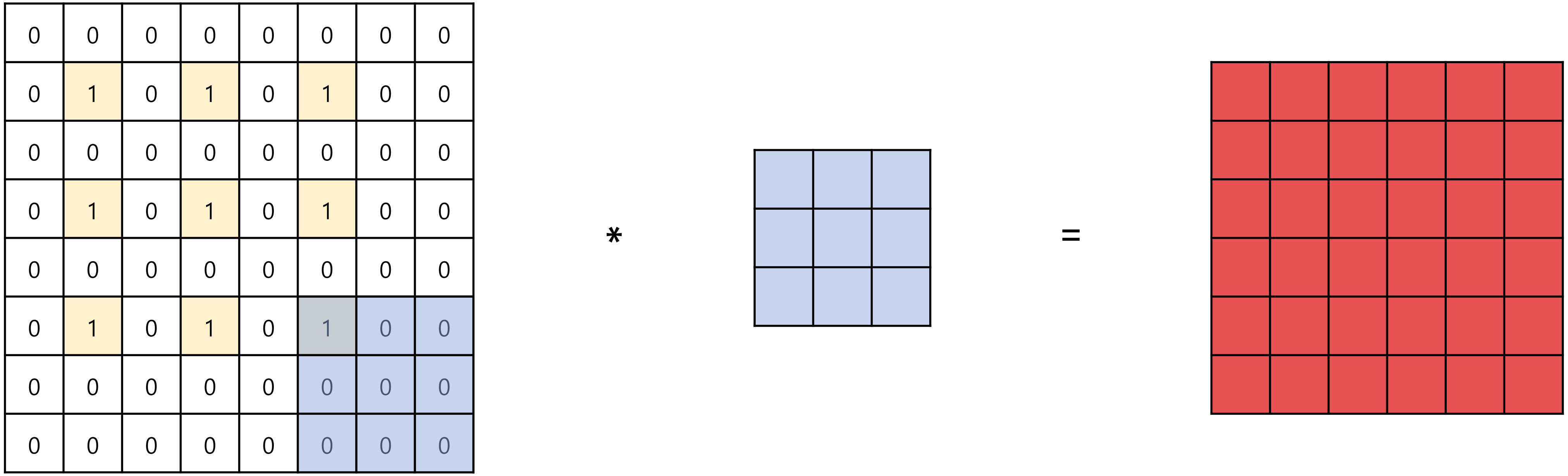


0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. ConvTranspose2d 연산 방법

* ConvTranspose2d 연산 예
Input : [1, 1, 3, 3] → Output: [1, 1, 6, 6]

```
m = nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1))
```



0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

2. 실제 nn.ConvTranspose2d 연산 결과 비교

* nn.ConvTranspose2d(1, 1, 9, stride=(2,2), padding=(4,4), output_padding=(1,1), bias = False)

Jihyun's reuslt

0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

*

1	2	3
0	0	0
0	0	0

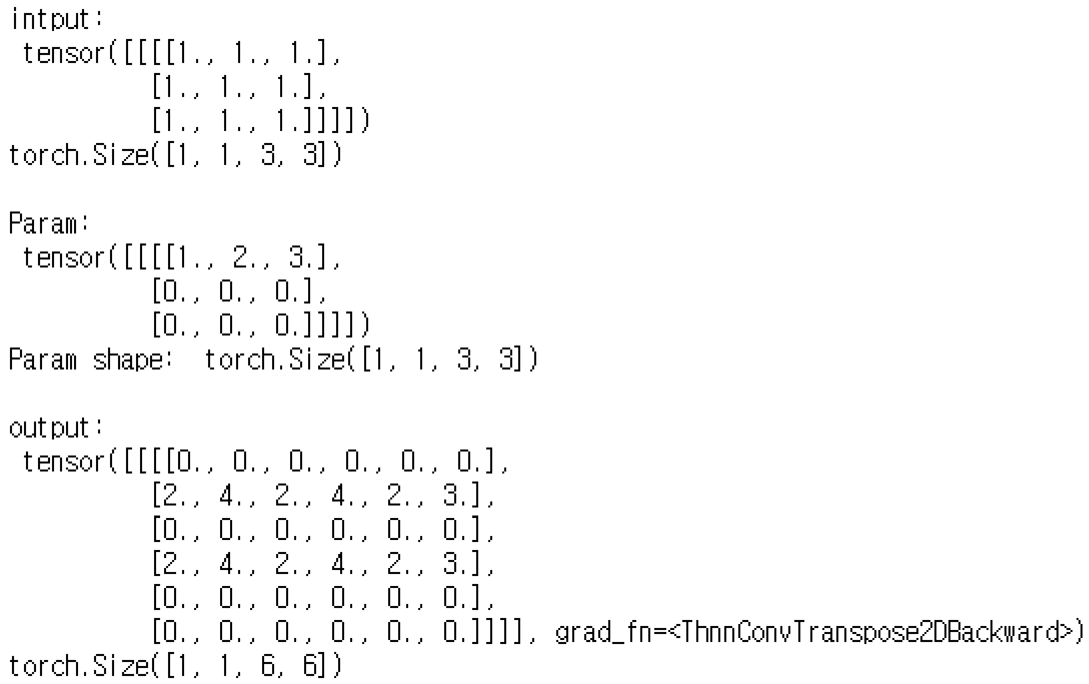
0	0	0
0	0	0
3	2	1

=

0	0	0	0	0	0
2	4	2	4	2	1
0	0	0	0	0	0
2	4	2	4	2	1
0	0	0	0	0	0
2	4	2	4	2	1

0	0	0	0	0	0
2	4	2	4	2	3
0	0	0	0	0	0
2	4	2	4	2	3
0	0	0	0	0	0
0	0	0	0	0	0

Python result

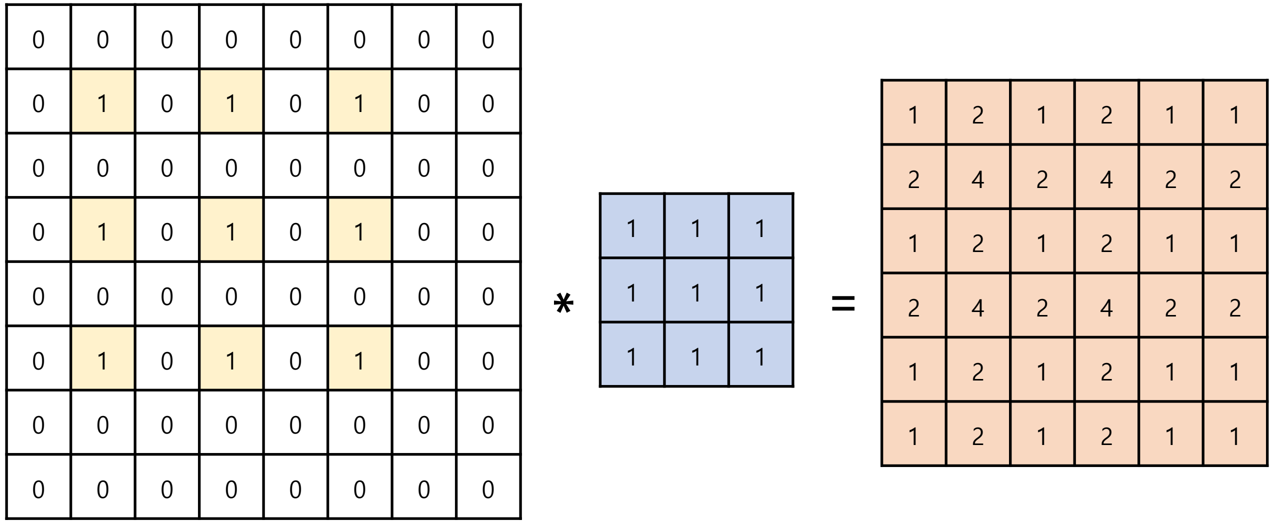


```
intput:
 tensor([[[[1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.]]]])
torch.Size([1, 1, 3, 3])

Param:
 tensor([[[[1., 2., 3.],
          [0., 0., 0.],
          [0., 0., 0.]]]])
Param shape:  torch.Size([1, 1, 3, 3])

output:
 tensor([[[[0., 0., 0., 0., 0., 0.],
          [2., 4., 2., 4., 2., 3.],
          [0., 0., 0., 0., 0., 0.],
          [2., 4., 2., 4., 2., 3.],
          [0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0.]]]], grad_fn=<ThnnConvTranspose2DBackward>)
torch.Size([1, 1, 6, 6])
```

2. 실제 nn.ConvTranspose2d 연산 결과 비교

* nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1), bias = False)

Jihyun’s reuslt



0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

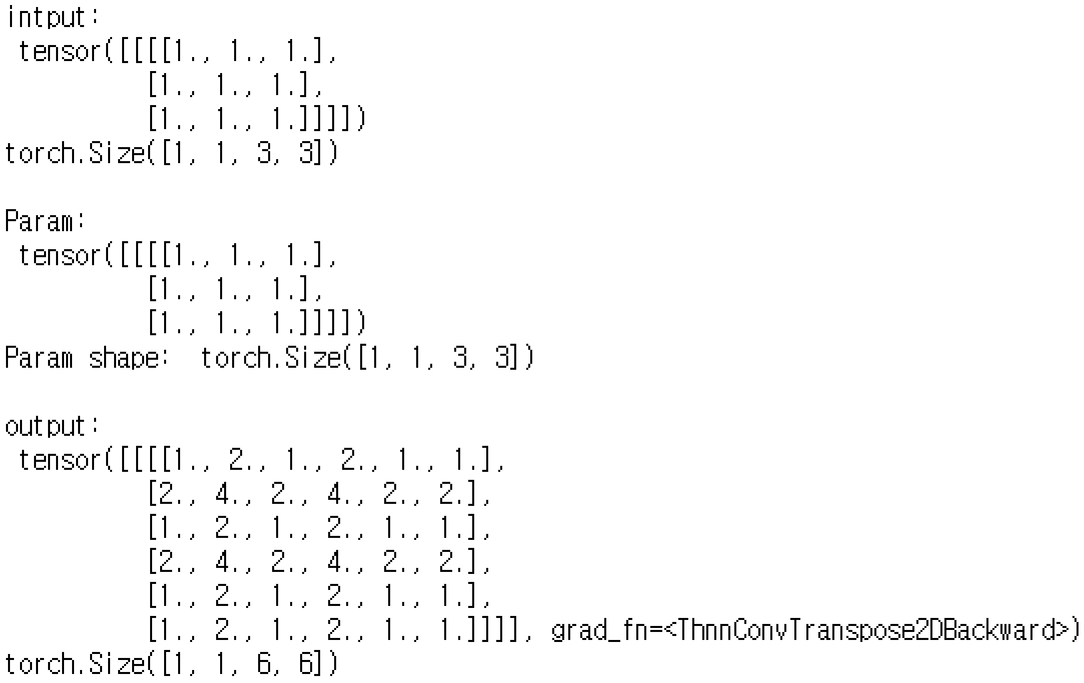
*

1	1	1
1	1	1
1	1	1

=

1	2	1	2	1	1
2	4	2	4	2	2
1	2	1	2	1	1
2	4	2	4	2	2
1	2	1	2	1	1
1	2	1	2	1	1

Python result

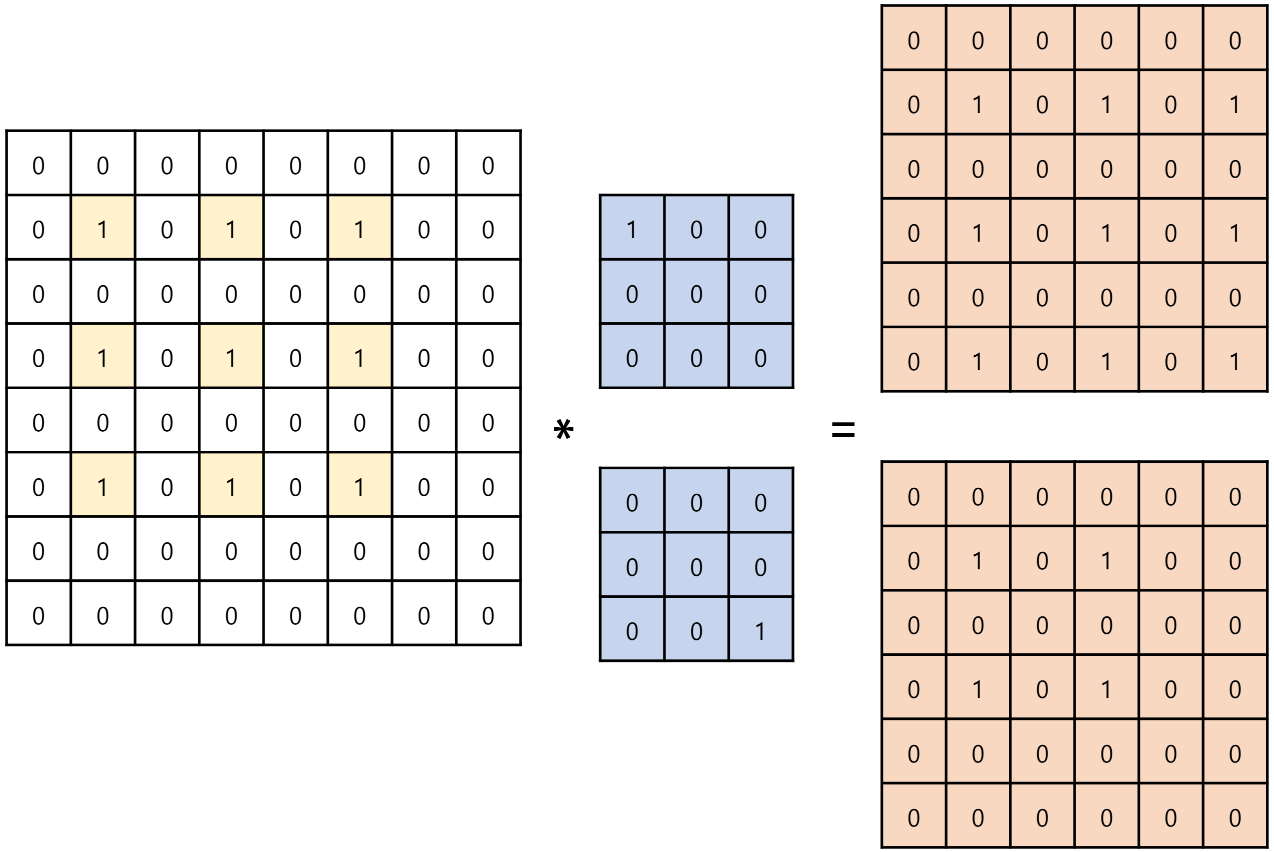


```
intput:
 tensor([[[[1., 1., 1.],
           [1., 1., 1.],
           [1., 1., 1.]]]])
torch.Size([1, 1, 3, 3])

Param:
 tensor([[[[1., 1., 1.],
           [1., 1., 1.],
           [1., 1., 1.]]]])
Param shape:  torch.Size([1, 1, 3, 3])

output:
 tensor([[[[1., 2., 1., 2., 1., 1.],
           [2., 4., 2., 4., 2., 2.],
           [1., 2., 1., 2., 1., 1.],
           [2., 4., 2., 4., 2., 2.],
           [1., 2., 1., 2., 1., 1.],
           [1., 2., 1., 2., 1., 1.]]]], grad_fn=<ThnnConvTranspose2DBackward>)
torch.Size([1, 1, 6, 6])
```

2. 실제 nn.ConvTranspose2d 연산 결과 비교

* nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1), bias = False)

Jihyun's reuslt



0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

*

1	0	0
0	0	0
0	0	0

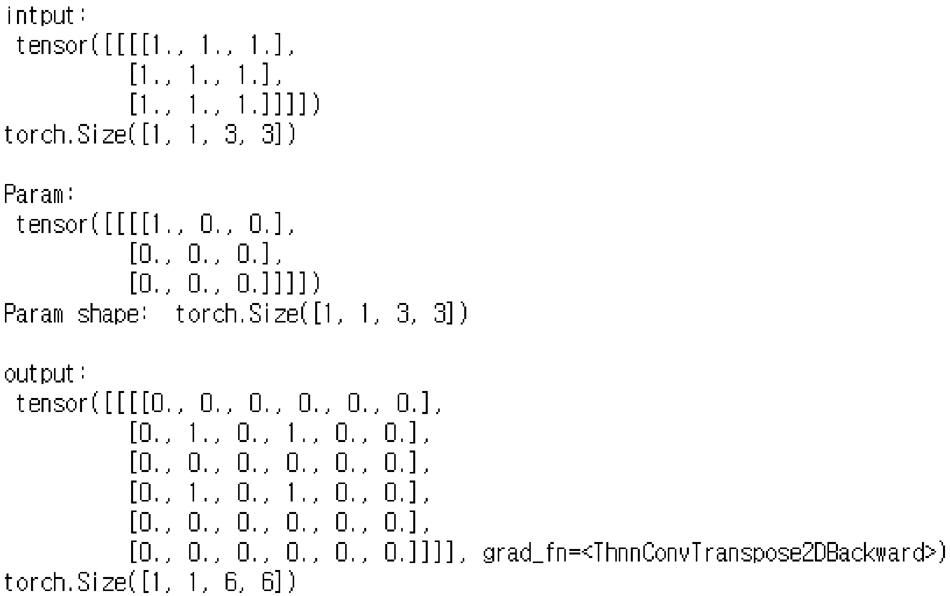
0	0	0
0	0	0
0	0	1

=

0	0	0	0	0	0
0	1	0	1	0	1
0	0	0	0	0	0
0	1	0	1	0	1
0	0	0	0	0	0
0	1	0	1	0	1

0	0	0	0	0	0
0	1	0	1	0	0
0	0	0	0	0	0
0	1	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Python result

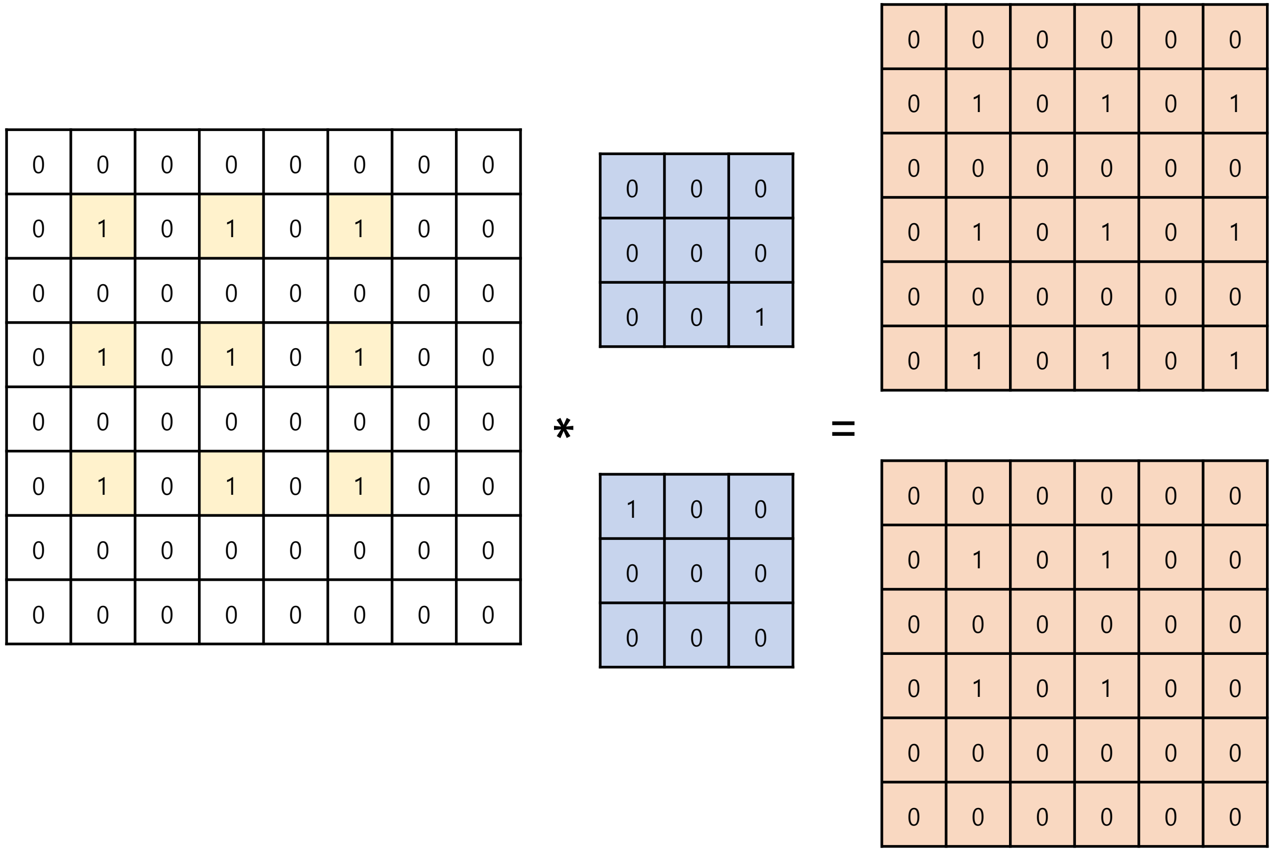


```
intput:
 tensor([[[[1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.]]]])
torch.Size([1, 1, 3, 3])

Param:
 tensor([[[[1., 0., 0.],
          [0., 0., 0.],
          [0., 0., 0.]]]])
Param shape:  torch.Size([1, 1, 3, 3])

output:
 tensor([[[[0., 0., 0., 0., 0., 0.],
          [0., 1., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0.],
          [0., 1., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0.]]]], grad_fn=<ThnnConvTranspose2DBackward>)
torch.Size([1, 1, 6, 6])
```

2. 실제 nn.ConvTranspose2d 연산 결과 비교

* nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1), bias = False)

Jihyun's reuslt



0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

*

0	0	0
0	0	0
0	0	1

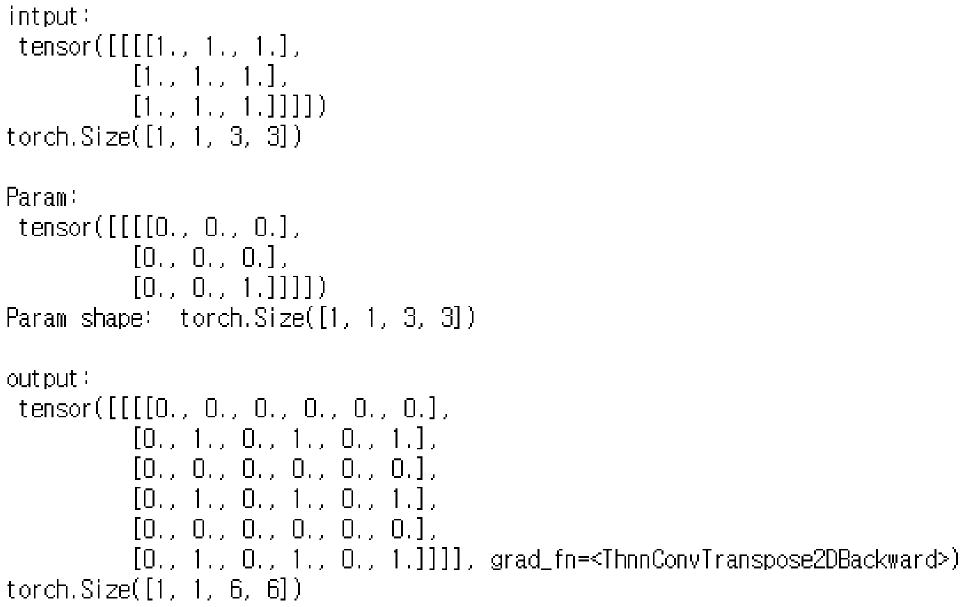
=

0	0	0	0	0	0
0	1	0	1	0	1
0	0	0	0	0	0
0	1	0	1	0	1
0	0	0	0	0	0
0	1	0	1	0	1

1	0	0
0	0	0
0	0	0

0	0	0	0	0	0
0	1	0	1	0	0
0	0	0	0	0	0
0	1	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Python result

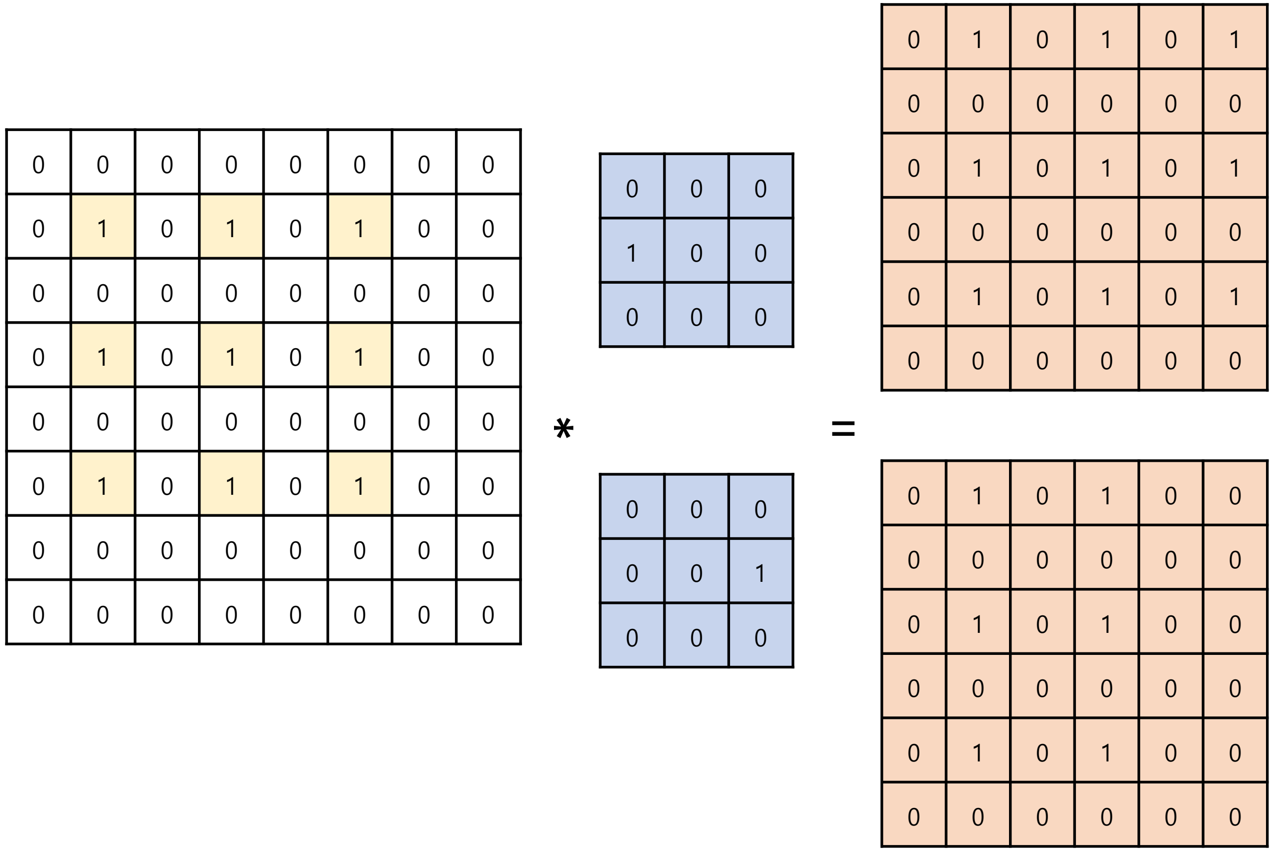


```
intput:
 tensor([[[[1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.]]]])
torch.Size([1, 1, 3, 3])

Param:
 tensor([[[[0., 0., 0.],
          [0., 0., 0.],
          [0., 0., 1.]]]])
Param shape:  torch.Size([1, 1, 3, 3])

output:
 tensor([[[[0., 0., 0., 0., 0., 0.],
          [0., 1., 0., 1., 0., 1.],
          [0., 0., 0., 0., 0., 0.],
          [0., 1., 0., 1., 0., 1.],
          [0., 0., 0., 0., 0., 0.],
          [0., 1., 0., 1., 0., 1.]]]], grad_fn=<ThnnConvTranspose2DBackward>)
torch.Size([1, 1, 6, 6])
```

2. 실제 nn.ConvTranspose2d 연산 결과 비교

* nn.ConvTranspose2d(1, 1, 3, stride=(2,2), padding=(1,1), output_padding=(1,1), bias = False)

Jihyun's reuslt



0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

*

0	0	0
1	0	0
0	0	0

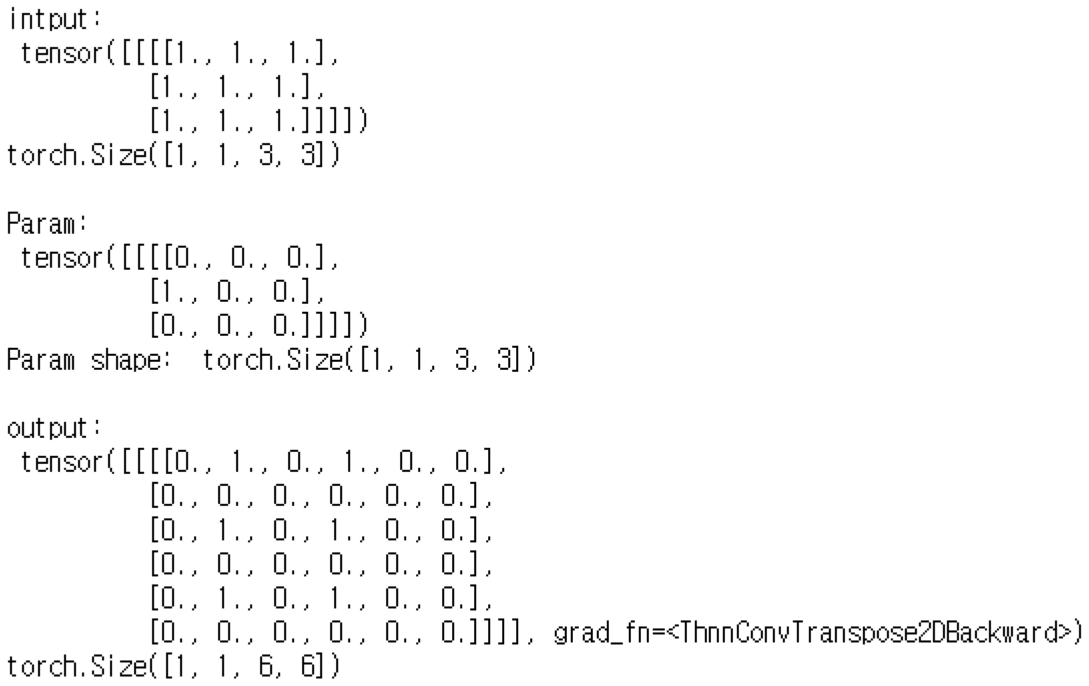
0	0	0
0	0	1
0	0	0

=

0	1	0	1	0	1
0	0	0	0	0	0
0	1	0	1	0	1
0	0	0	0	0	0
0	1	0	1	0	1
0	0	0	0	0	0

0	1	0	1	0	0
0	0	0	0	0	0
0	1	0	1	0	0
0	0	0	0	0	0
0	1	0	1	0	0
0	0	0	0	0	0

Python result



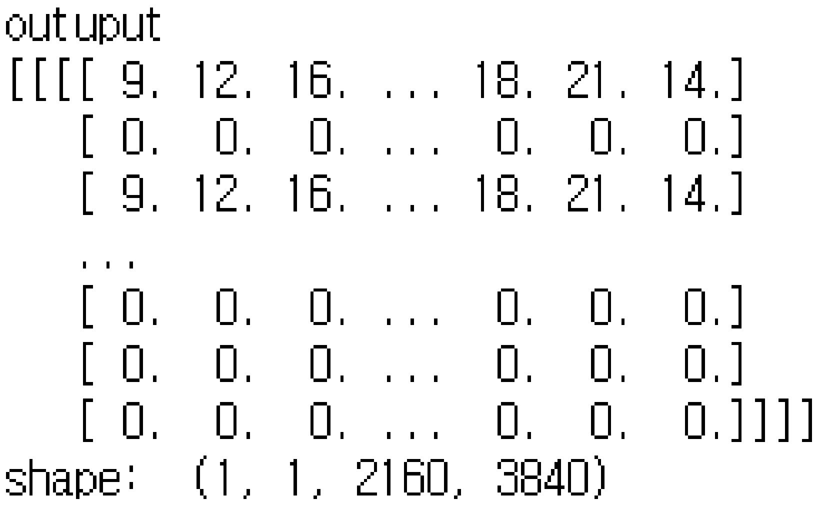
```
intput:
 tensor([[[[1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.]]]])
torch.Size([1, 1, 3, 3])

Param:
 tensor([[[[0., 0., 0.],
          [1., 0., 0.],
          [0., 0., 0.]]]])
Param shape:  torch.Size([1, 1, 3, 3])

output:
 tensor([[[[0., 1., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0.],
          [0., 1., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0.],
          [0., 1., 0., 1., 0., 0.],
          [0., 0., 0., 0., 0., 0.]]]], grad_fn=<ThnnConvTranspose2DBackward>)
torch.Size([1, 1, 6, 6])
```

2. 실제 nn.ConvTranspose2d 연산 결과 비교

* nn.ConvTranspose2d(1, 1, 9, stride=(2,2), padding=(4,4), output_padding=(1,1), bias = False)

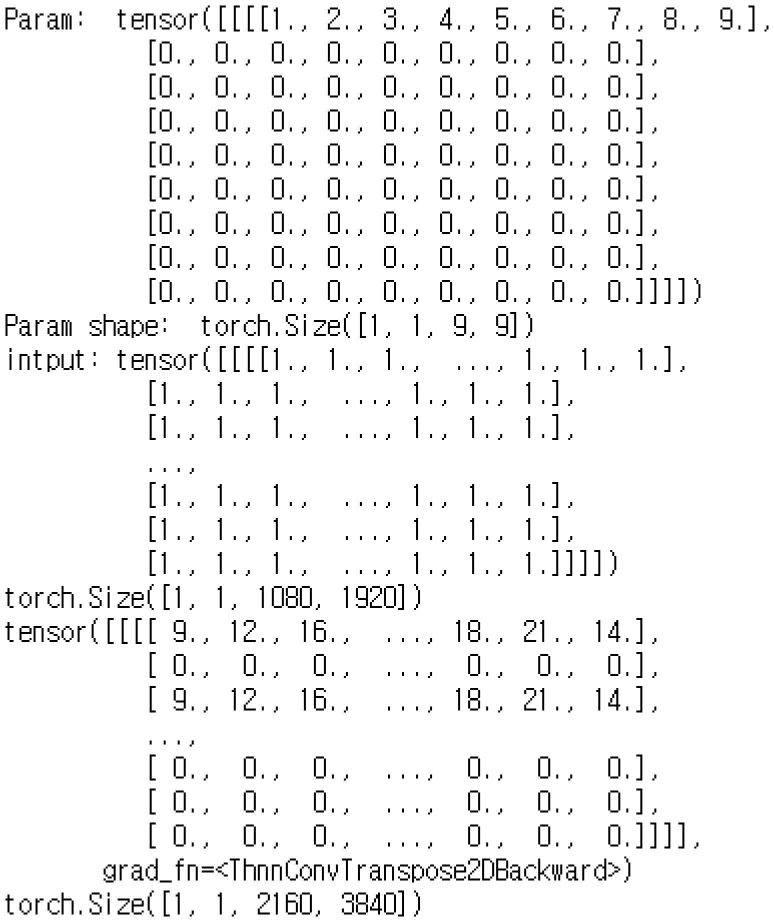
(1920 * 1080) → (3840 * 2160)

Jihyun’s reuslt



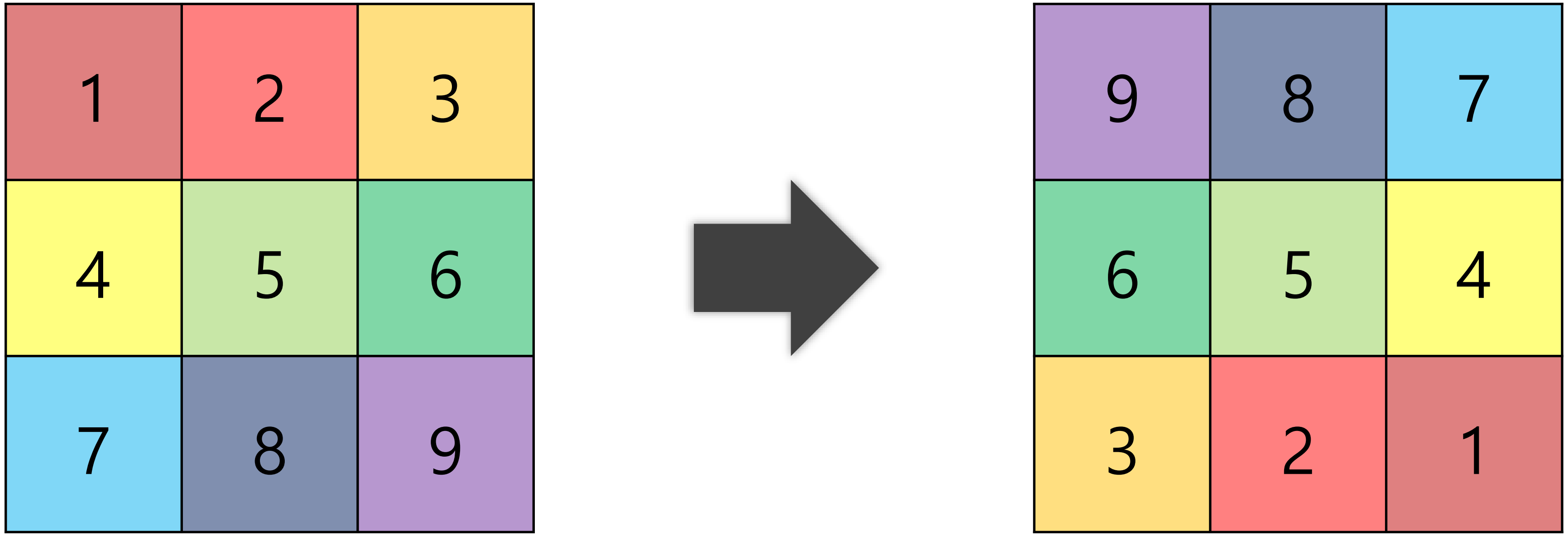
```
outuput
[[[[ 9. 12. 16. ... 18. 21. 14.]
   [ 0.  0.  0. ...  0.  0.  0.]
   [ 9. 12. 16. ... 18. 21. 14.]
   ...
   [ 0.  0.  0. ...  0.  0.  0.]
   [ 0.  0.  0. ...  0.  0.  0.]
   [ 0.  0.  0. ...  0.  0.  0.]]]]
shape:  (1, 1, 2160, 3840)
```

Python result



```
Param:  tensor([[[[1., 2., 3., 4., 5., 6., 7., 8., 9.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0., 0., 0., 0., 0.]]]])
Param shape:  torch.Size([1, 1, 9, 9])
intput: tensor([[[[1., 1., 1.,  ..., 1., 1., 1.],
          [1., 1., 1.,  ..., 1., 1., 1.],
          [1., 1., 1.,  ..., 1., 1., 1.],
          ...,
          [1., 1., 1.,  ..., 1., 1., 1.],
          [1., 1., 1.,  ..., 1., 1., 1.],
          [1., 1., 1.,  ..., 1., 1., 1.]]]])
torch.Size([1, 1, 1080, 1920])
tensor([[[[ 9., 12., 16.,  ..., 18., 21., 14.],
          [ 0.,  0.,  0.,  ...,  0.,  0.,  0.],
          [ 9., 12., 16.,  ..., 18., 21., 14.],
          ...,
          [ 0.,  0.,  0.,  ...,  0.,  0.,  0.],
          [ 0.,  0.,  0.,  ...,  0.,  0.,  0.],
          [ 0.,  0.,  0.,  ...,  0.,  0.,  0.]]]],
       grad_fn=<ThnnConvTranspose2DBackward>)
torch.Size([1, 1, 2160, 3840])
```

3. 결과 분석



1	2	3
4	5	6
7	8	9

9	8	7
6	5	4
3	2	1

결론: Kernel을 상하좌우 바꾼 후 convolution 연산을 진행해야 함

4. 피드백