

7.2

How Convolutional Neural Networks Work

Part 5: Controlling The Output Size With Padding

Sebastian Raschka and the Lightning AI Team

How do we calculate the feature map size?

The diagram shows the formula for calculating the output width O of a feature map. The formula is $O = \frac{W - K + 2P}{S} + 1$. Arrows point from labels to the variables in the formula: 'input width' points to W , 'kernel width' points to K , 'padding' points to P , 'output width' points to O , and 'stride' points to S .

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

How do we calculate the feature map size?

The diagram shows the formula for calculating the output width O of a feature map. The formula is $O = \frac{W - K + 2P}{S} + 1$. Arrows point from labels to the variables in the formula: 'input width' points to W , 'kernel width' points to K , 'padding' points to P , 'stride' points to S , and 'output width' points to O .

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

(The same formula works for "height")

Example 1

The diagram illustrates the calculation of the output width O for a 1D convolution. The formula is $O = \frac{\text{input width} - \text{kernel width} + \text{padding}}{\text{stride}} + 1$. In this example, the input width is 100, the kernel width is 3, the padding is 0, and the stride is 1. The output width O is calculated as $\frac{100 - 3 + 0}{1} + 1 = 98$. Arrows point from the text labels to their respective values in the formula: 'input width' to 100, 'kernel width' to 3, 'padding' to 0, 'stride' to 1, and 'output width' to O .

$$O = \frac{100 - 3 + 0}{1} + 1 = 98$$

Labels and arrows in the diagram:

- input width (points to 100)
- kernel width (points to 3)
- padding (points to 0)
- stride (points to 1)
- output width (points to O)

Example 1

input width kernel width padding

output width stride

$$O = \frac{100 - 3 + 0}{1} + 1 = 98$$

```
import torch
import torch.nn as nn

layer = nn.Conv2d(1, 1, kernel_size=3, padding=0, stride=1)

example = torch.rand(1, 100, 100)
layer(example).shape

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

Example 2

The diagram illustrates the 1D convolution formula with the following components labeled:

- input width**: points to the value 100.
- kernel width**: points to the value 5.
- padding**: points to the value 0.
- stride**: points to the value 2.
- output width**: points to the variable O .

$$O = \frac{100 - 5 + 0}{2} + 1 = 48.5$$

Example 2

input width kernel width padding

output width stride

$$O = \frac{100 - 5 + 0}{2} + 1 = 48.5$$

```
import torch
import torch.nn as nn

layer = nn.Conv2d(1, 1, kernel_size=5, padding=0, stride=2)

example = torch.rand(1, 100, 100)
layer(example).shape

torch.Size([1, 48, 48])
```

Example 2

Diagram illustrating the calculation of output width O for a 1D convolution:

$$O = \left\lfloor \frac{\text{input width} - \text{kernel width} + \text{padding}}{\text{stride}} \right\rfloor + 1$$

Substituting the values from the diagram:

$$O = \left\lfloor \frac{100 - 5 + 0}{2} \right\rfloor + 1 = 48.5$$

The result 48.5 is then floored to 48.

```
import torch
import torch.nn as nn

layer = nn.Conv2d(1, 1, kernel_size=5, padding=0, stride=2)

example = torch.rand(1, 100, 100)
layer(example).shape

torch.Size([1, 48, 48])
```


Padding = 1


Will add a row/column of zeros to each size

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

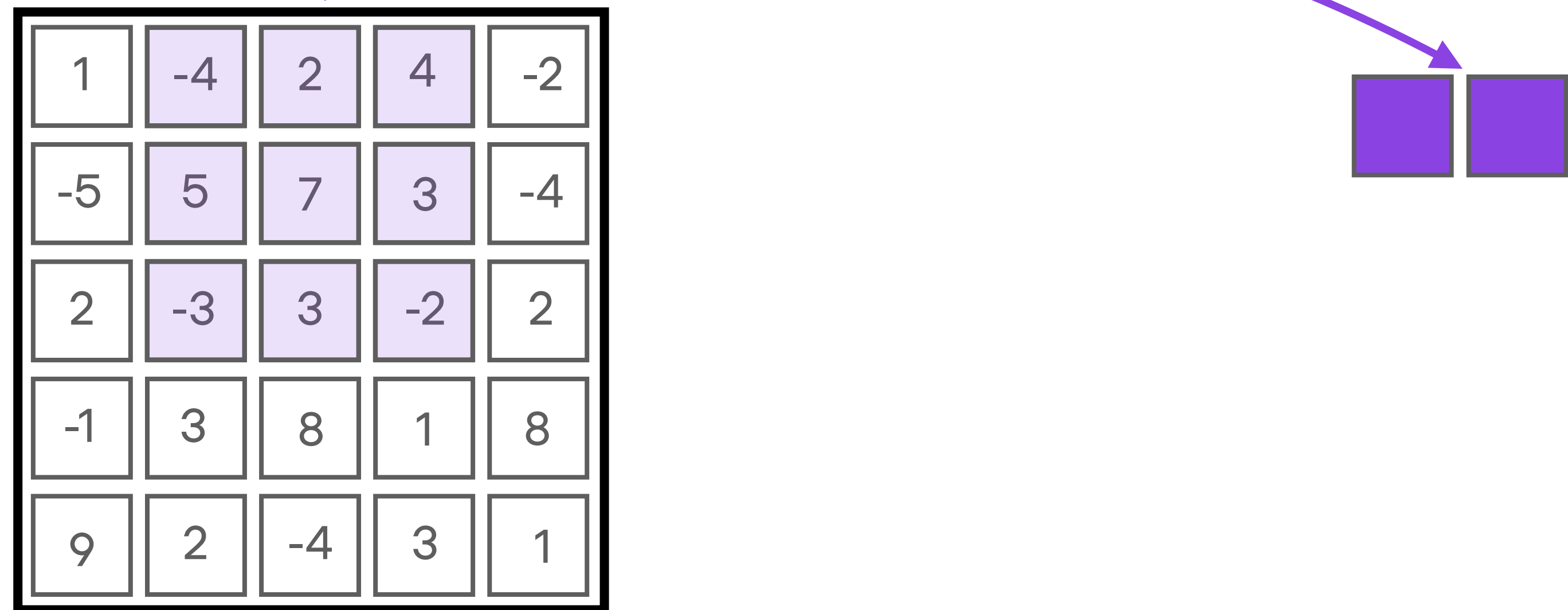
0	0	0	0	0	0	0
0	1	-4	2	4	-2	0
0	-5	5	7	3	-4	0
0	2	-3	3	-2	2	0
0	-1	3	8	1	8	0
0	9	2	-4	3	1	0
0	0	0	0	0	0	0

padding = 1
padding = (1, 1)



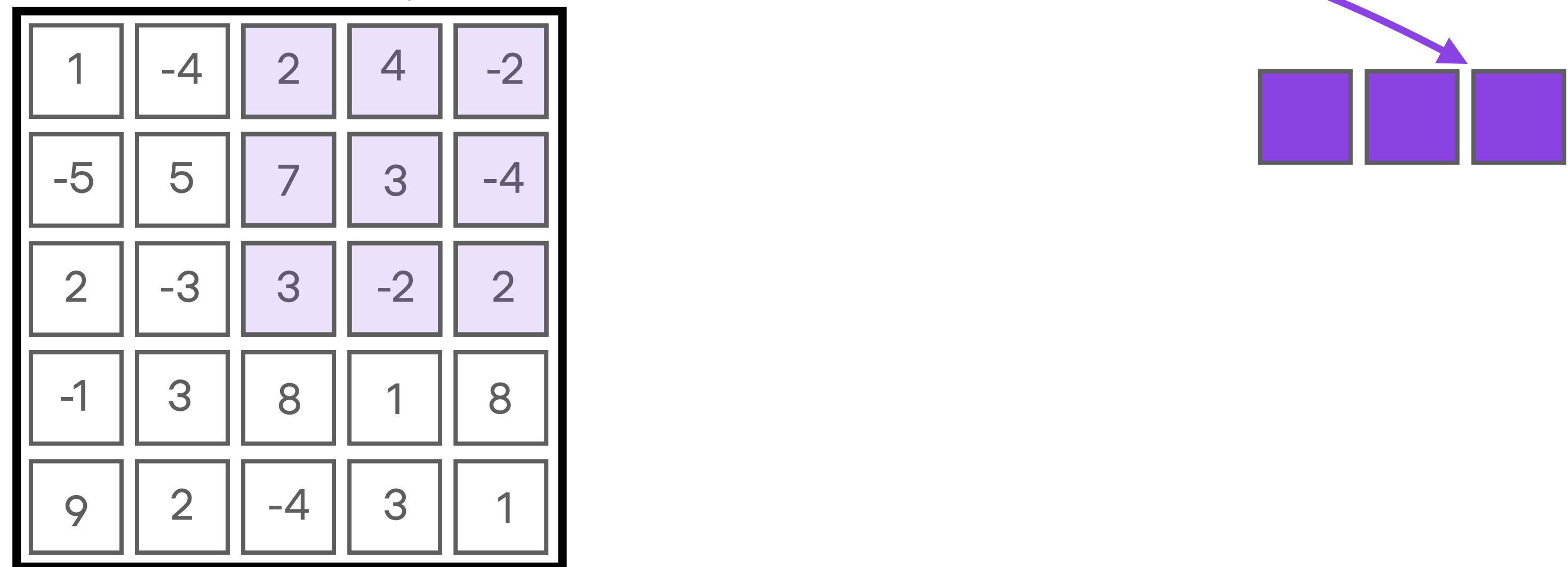
1	-4	2	4	-2
-5	5	7	3	-4
2	-3	3	-2	2
-1	3	8	1	8
9	2	-4	3	1

image with no padding



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

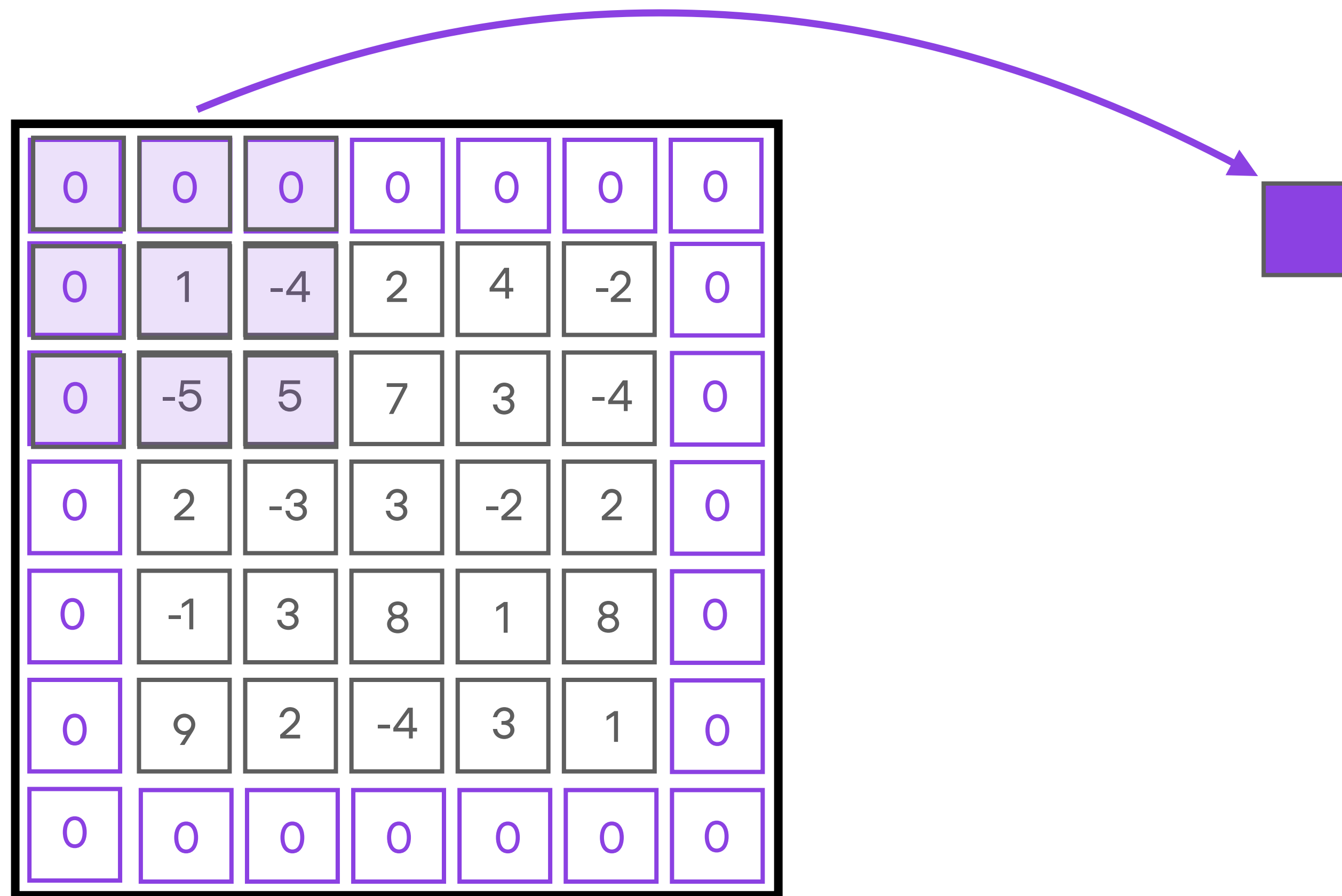


1	-4	2	4	-2
-5	5	7	3	-4
2	-3	3	-2	2
-1	3	8	1	8
9	2	-4	3	1

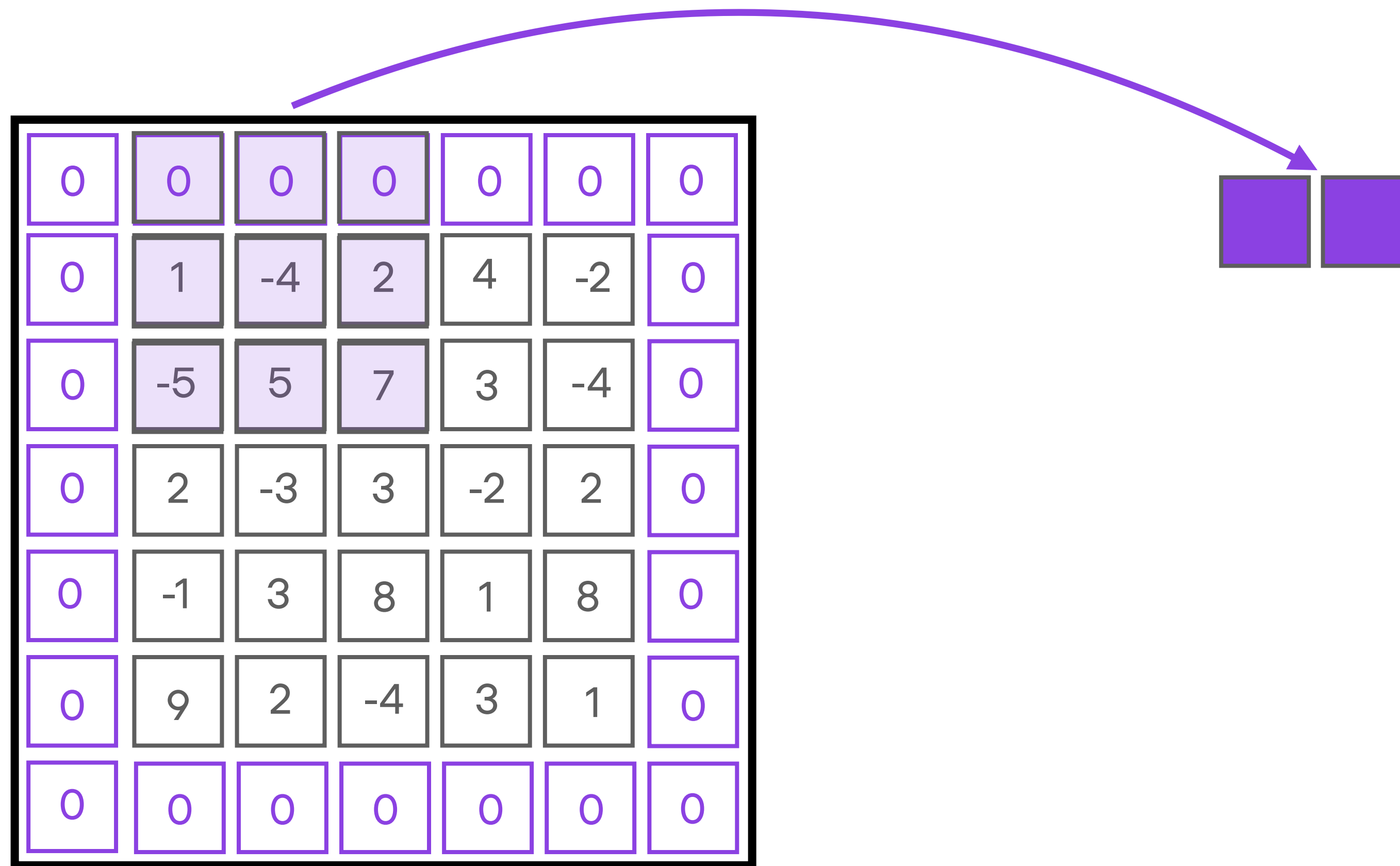
image with no padding

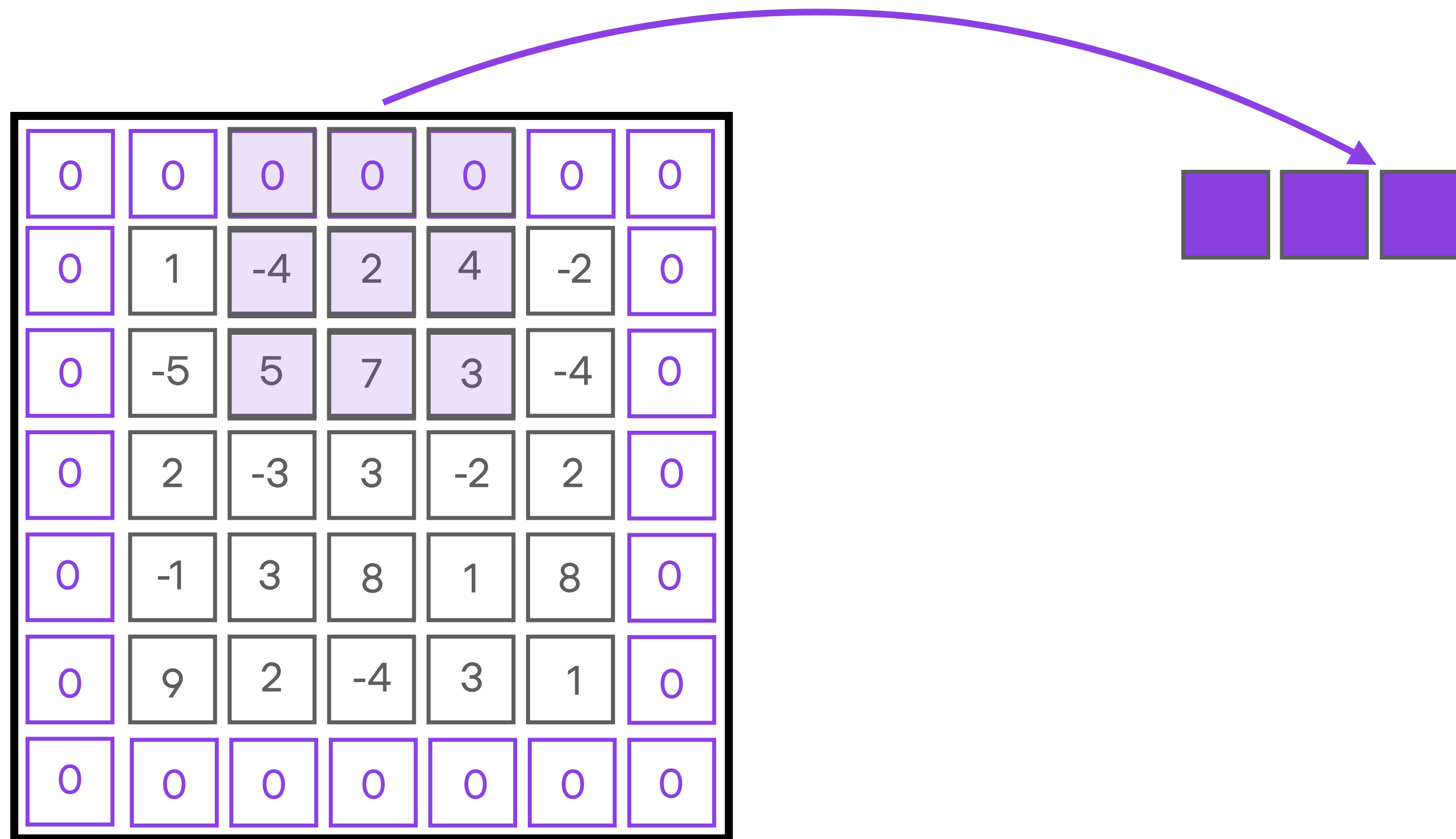
0	0	0	0	0	0	0
0	1	-4	2	4	-2	0
0	-5	5	7	3	-4	0
0	2	-3	3	-2	2	0
0	-1	3	8	1	8	0
0	9	2	-4	3	1	0
0	0	0	0	0	0	0

padding = 1
padding = (1, 1)



padding = 1
padding = (1, 1)





padding = 1
padding = (1, 1)

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 = (1, 1)

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 = (1, 1)

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 = (1, 1)

Or, instead of doing the math, we can use
padding = "same"

**Next: Let's take a look at some common CNN
architectures**