PAT 463/563 (Fall 2025)

Music & AI

Lecture 9: Convolutional Neural Networks

Instructor: Hao-Wen Dong

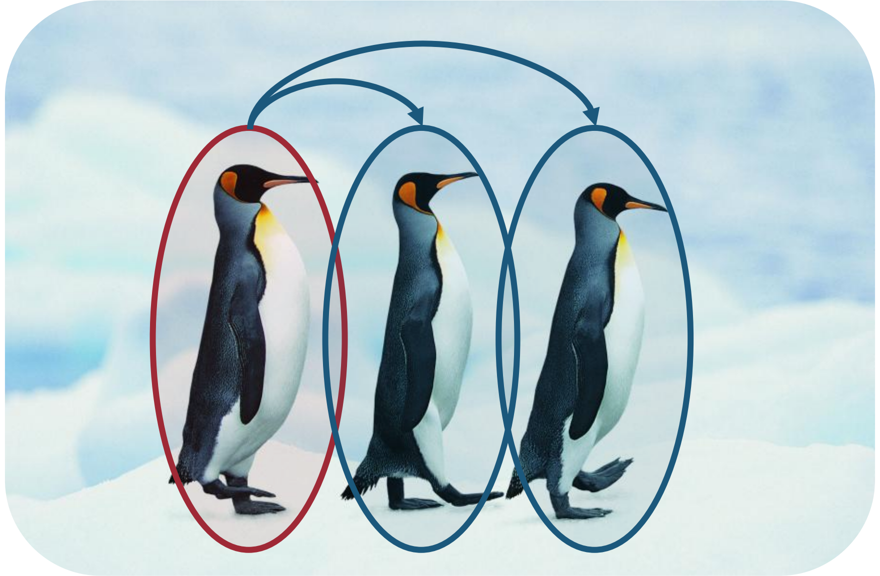
UNIVERSITY OF MICHIGAN

Convolutional Neural Networks (CNNs)

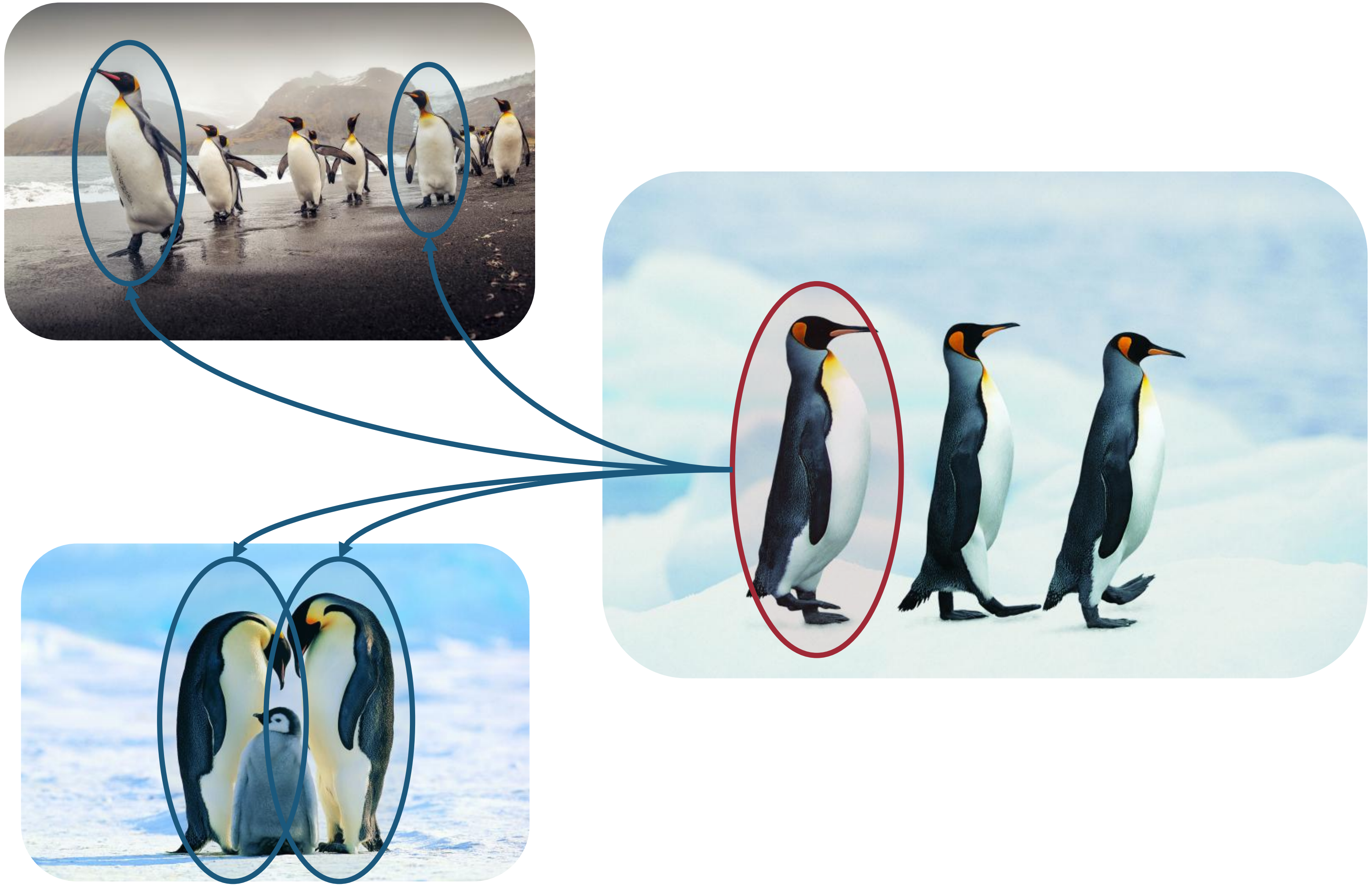
Convolutional Neural Networks (CNNs)

- **Intuition**: Learn **reusable** **local pattern detector**
- Widely used in **computer vision**
- Also used for music and audio
  - Representing music as **piano rolls**
  - Representing audio as **spectrograms**

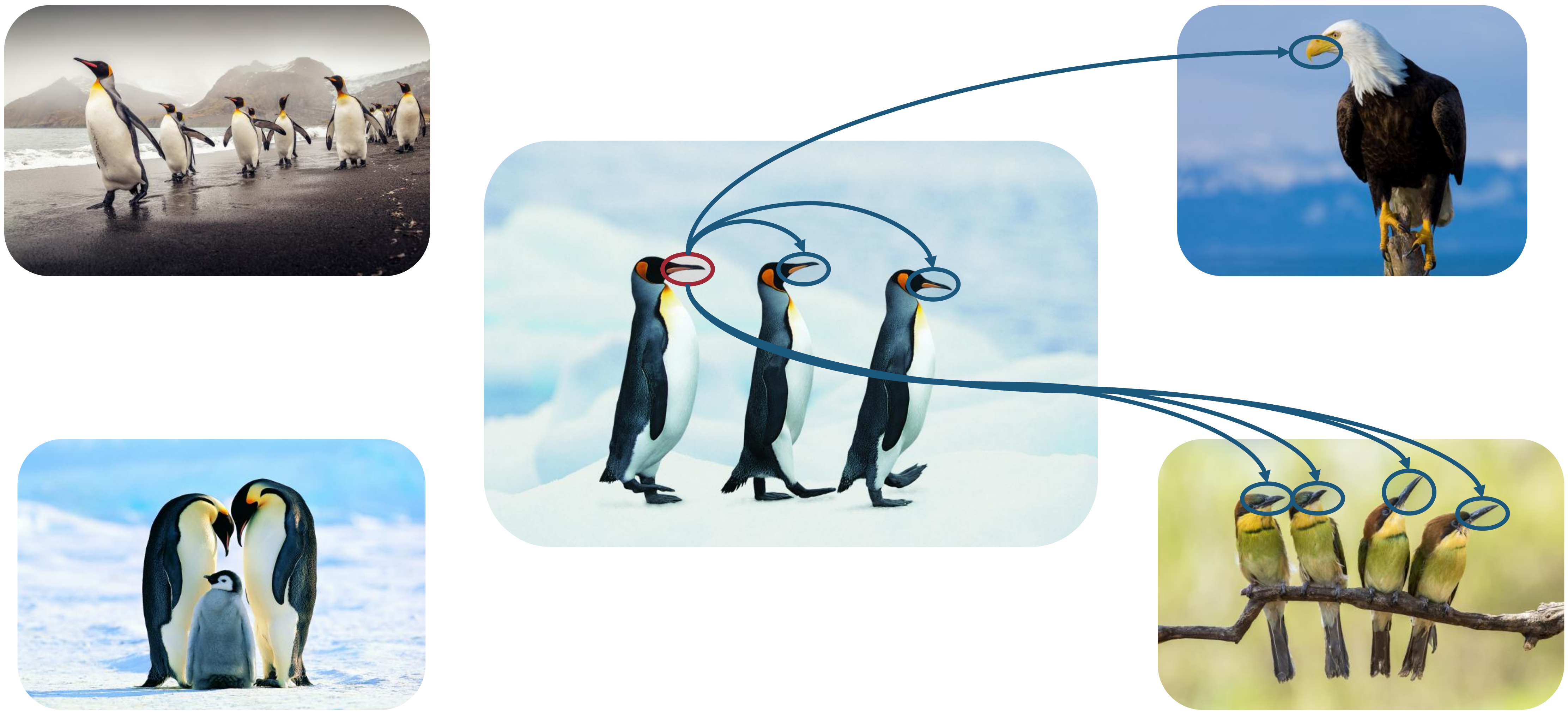
Reusable Pattern Detectors



Reusable Pattern Detectors

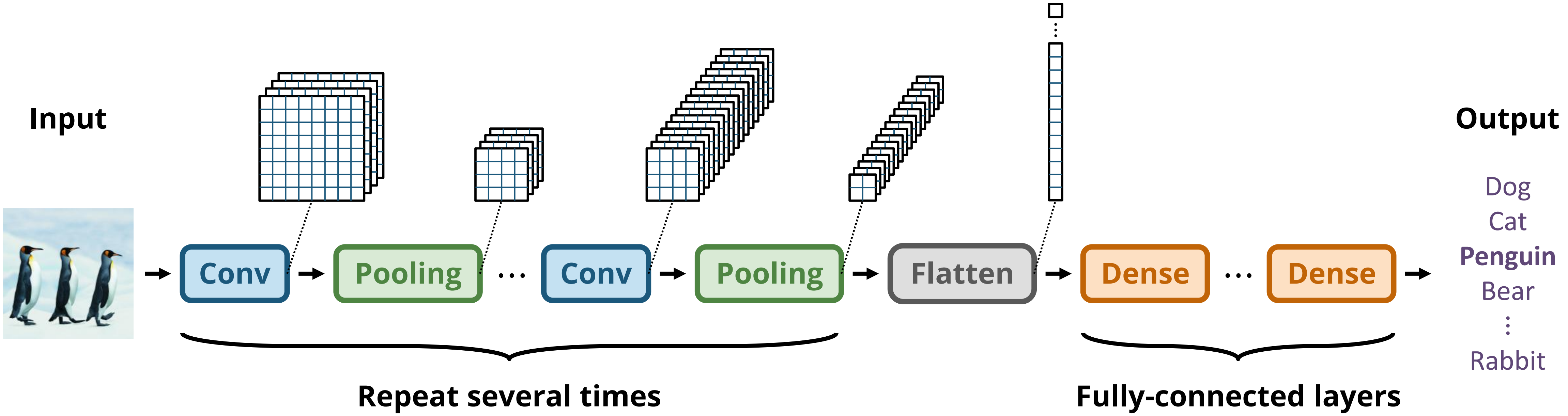


Reusable Pattern Detectors



Reusable Pattern Detectors

Convolutional Neural Network (CNNs)

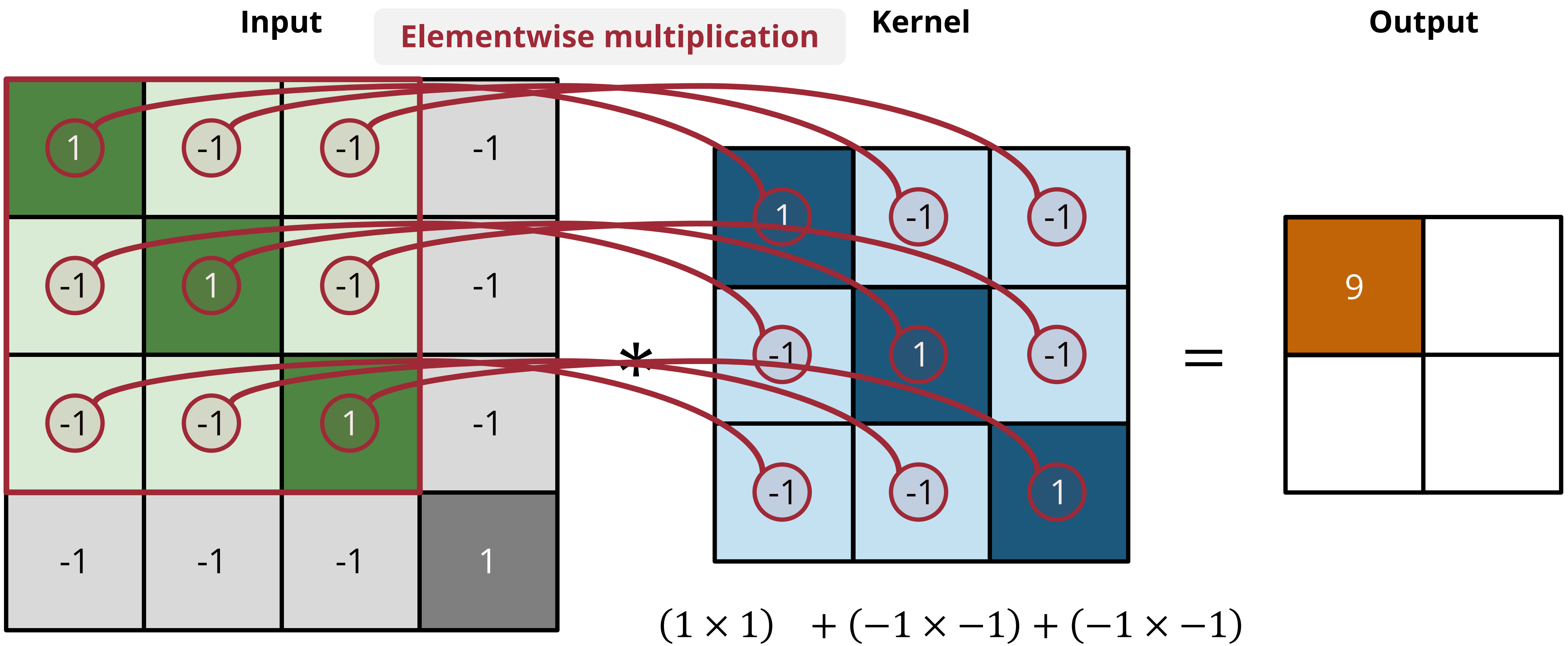


2D Convolution

Input **Kernel** **Output**

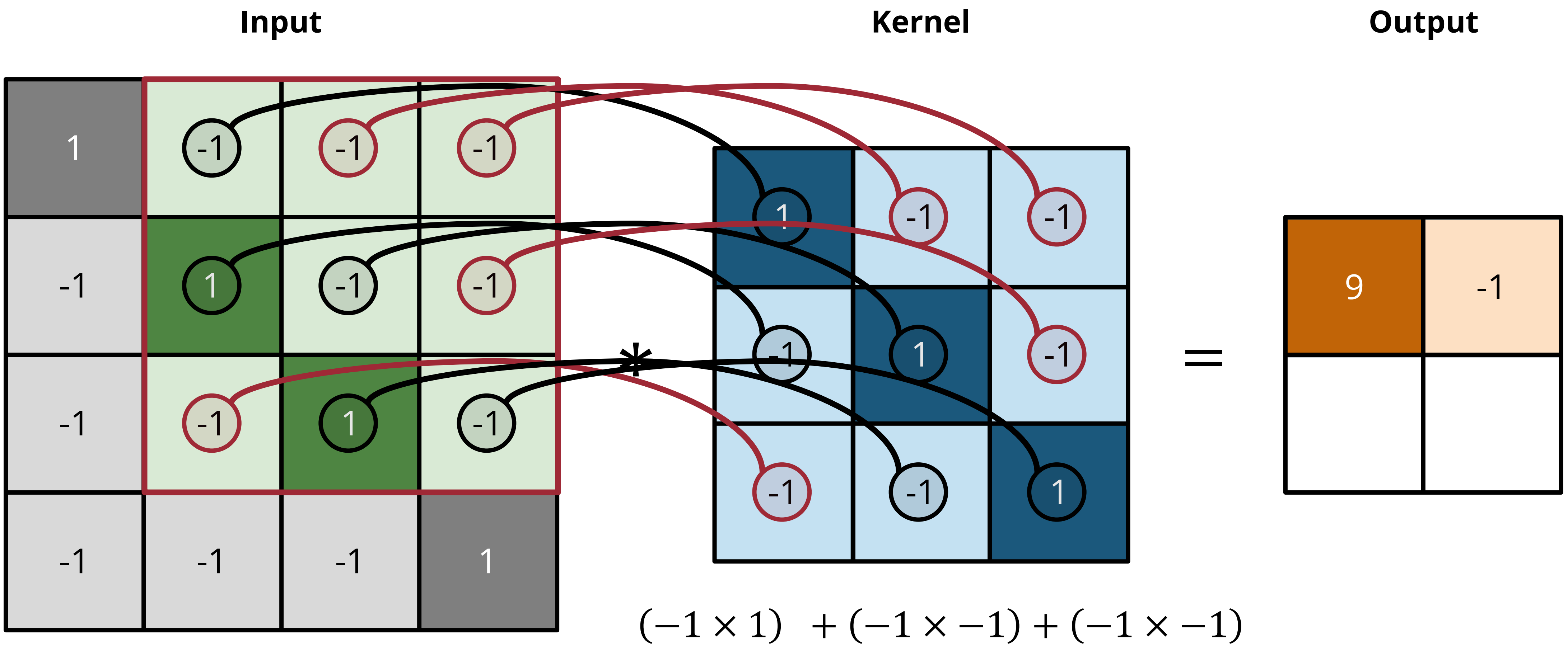
$$\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$

2D Convolution



$$\begin{aligned}&(1\times 1) + (-1\times -1) + (-1\times -1)\\ &+(-1\times -1) + (1\times 1) + (-1\times -1) \quad = \mathbf{9}\\ &+(-1\times -1) + (-1\times -1) + (1\times 1)\end{aligned}$$

2D Convolution



$$
\begin{aligned}
&(-1\times 1) + (-1\times -1) + (-1\times -1)\\
+\ &(-1\times 1) + (-1\times 1) + (-1\times -1) \quad = \mathbf{-1}\\
+&(-1\times -1) + (1\times -1) + (-1\times 1)
\end{aligned}
$$

2D Convolution

Input **Kernel** **Output**

$$\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 9 & -1 \\ -1 & \end{bmatrix}$$

$$\begin{aligned} &(-1 \times 1) + (1 \times -1) + (-1 \times -1) \\ &+ (-1 \times -1) + (-1 \times 1) + (1 \times -1) \\ &+ (-1 \times -1) + (-1 \times -1) + (-1 \times 1) \end{aligned} = \mathbf{-1}$$

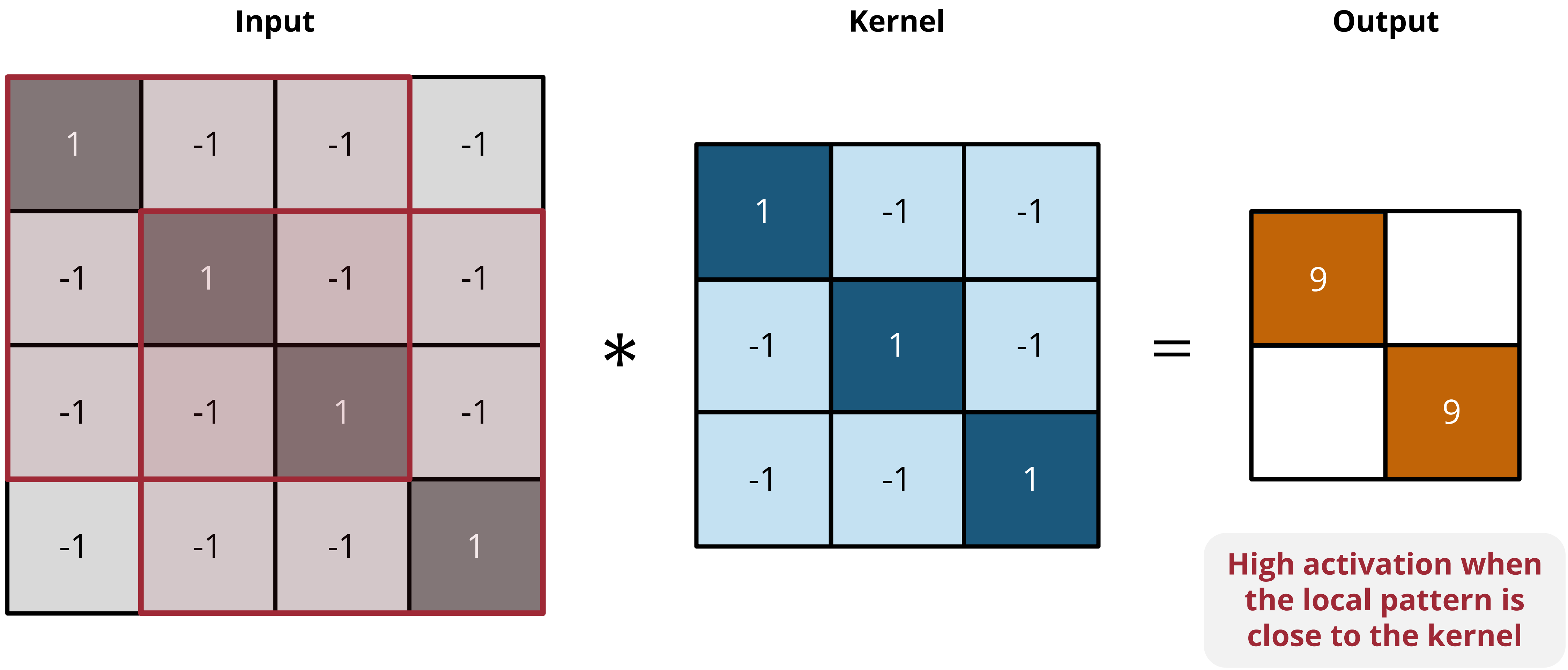
2D Convolution

Input **Kernel** **Output**

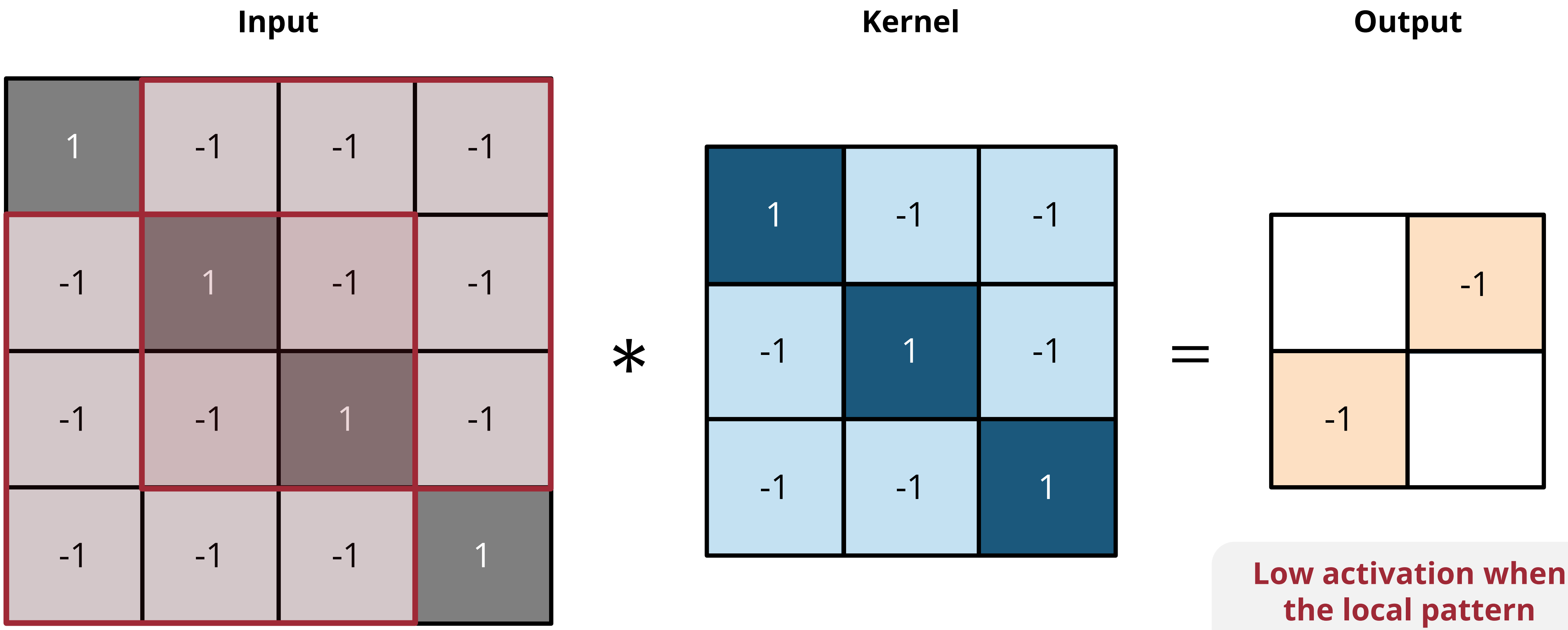
$$\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 9 & -1 \\ -1 & 9 \end{bmatrix}$$

$$\begin{aligned} &(1\times 1) + (-1\times -1) + (-1\times -1) \\ &+(-1\times -1) + (1\times 1) + (-1\times -1) \\ &+(-1\times -1) + (-1\times -1) + (1\times 1) \end{aligned} = \mathbf{9}$$

2D Convolution



2D Convolution



Low activation when the local pattern differs from the kernel

2D Convolution

$$\begin{array}{ccccc}
\textbf{Input} & & \textbf{Kernel} & & \textbf{Output} \\
\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} & * & \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} & = & \begin{bmatrix} 9 & -1 \\ -1 & 9 \end{bmatrix}
\end{array}$$

2D Convolution

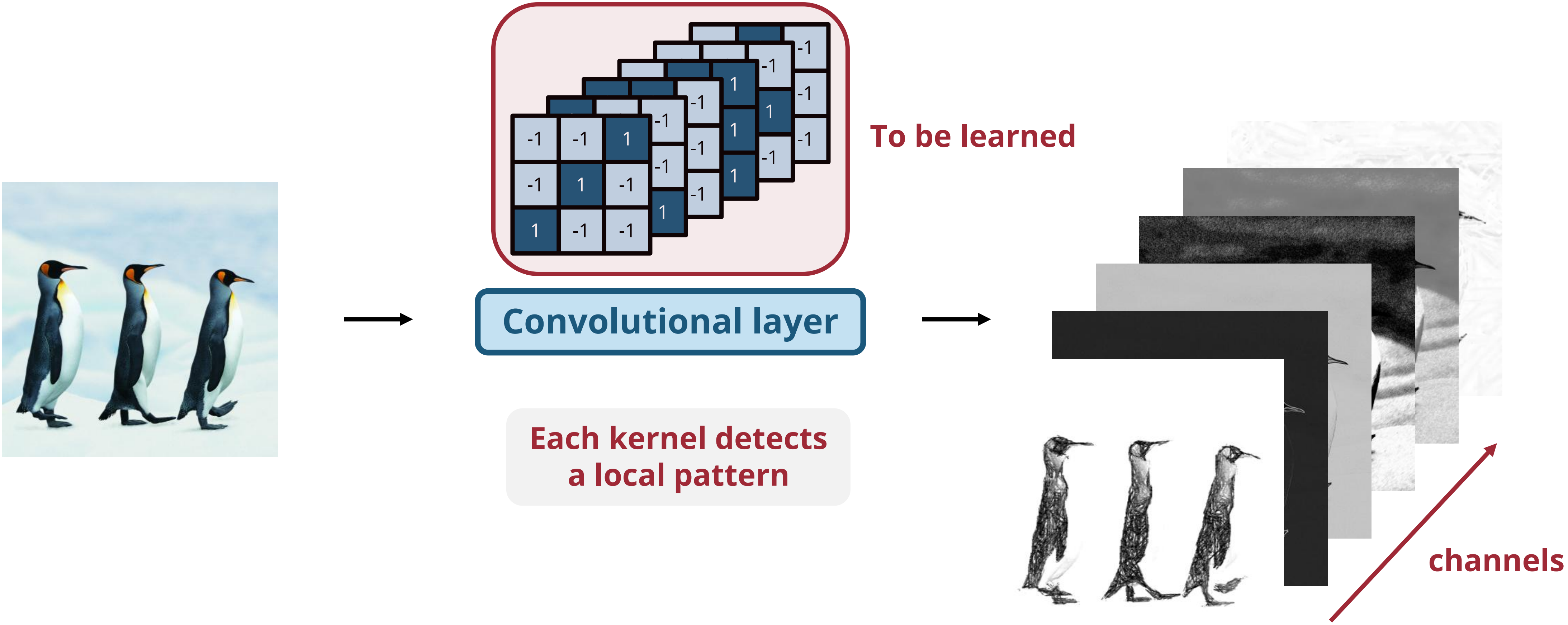
$$\underset{\textbf{Input}}{\begin{bmatrix} -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}} * \underset{\textbf{Kernel}}{\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}} = \underset{\textbf{Output}}{\begin{bmatrix} 1 & 1 \\ 1 & 5 \end{bmatrix}}$$

2D Convolution

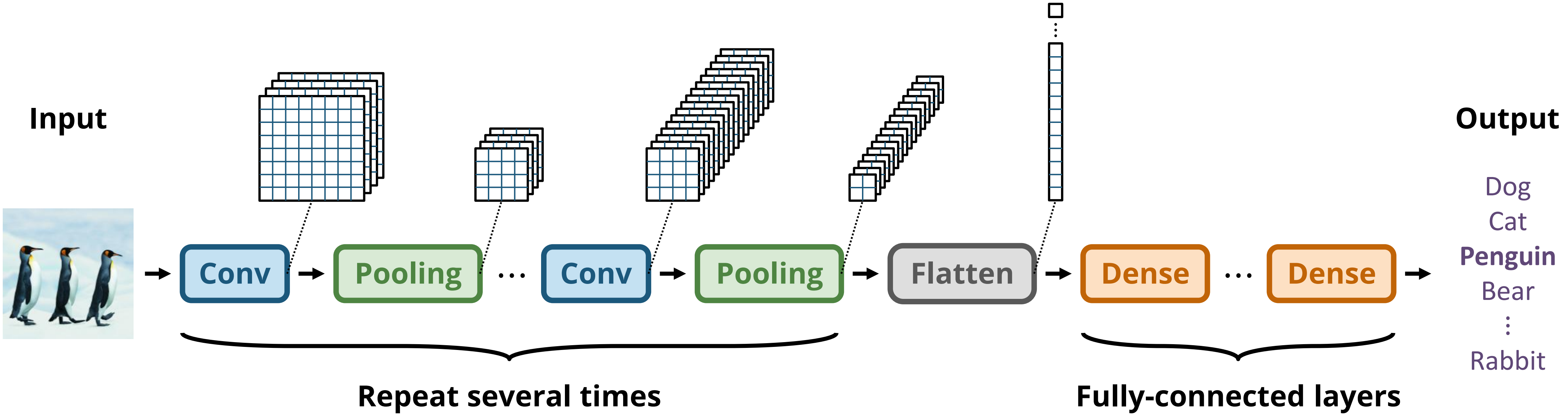
$$
\underset{\textbf{Input}}{\begin{bmatrix} -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}} * \underset{\textbf{Kernel}}{\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}} = \underset{\textbf{Output}}{\begin{bmatrix} 9 & -1 \\ -1 & 1 \end{bmatrix}}
$$

Convolutional Layer

- A convolutional layer consists of many **learnable kernels** (channels)



Convolutional Neural Network (CNNs)



Padding

`padding=“valid”`

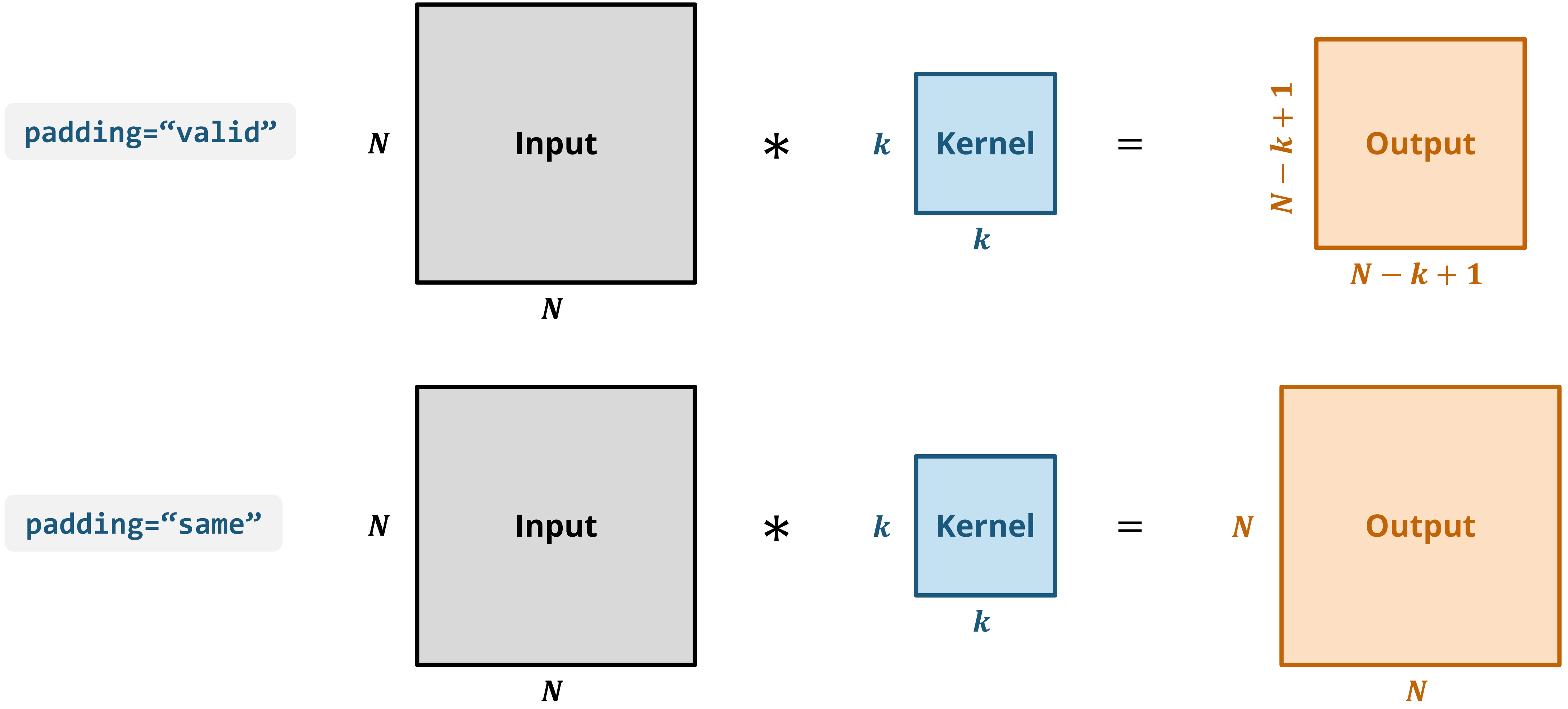
$$\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 9 & -1 \\ -1 & 9 \end{bmatrix}$$

`padding=“same”`

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & -1 & 0 \\ 0 & -1 & 1 & -1 & -1 & 0 \\ 0 & -1 & -1 & 1 & -1 & 0 \\ 0 & -1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & -2 & 0 & 2 \\ -2 & 9 & -1 & 0 \\ 0 & -1 & 9 & -2 \\ 2 & 0 & -2 & 4 \end{bmatrix}$$

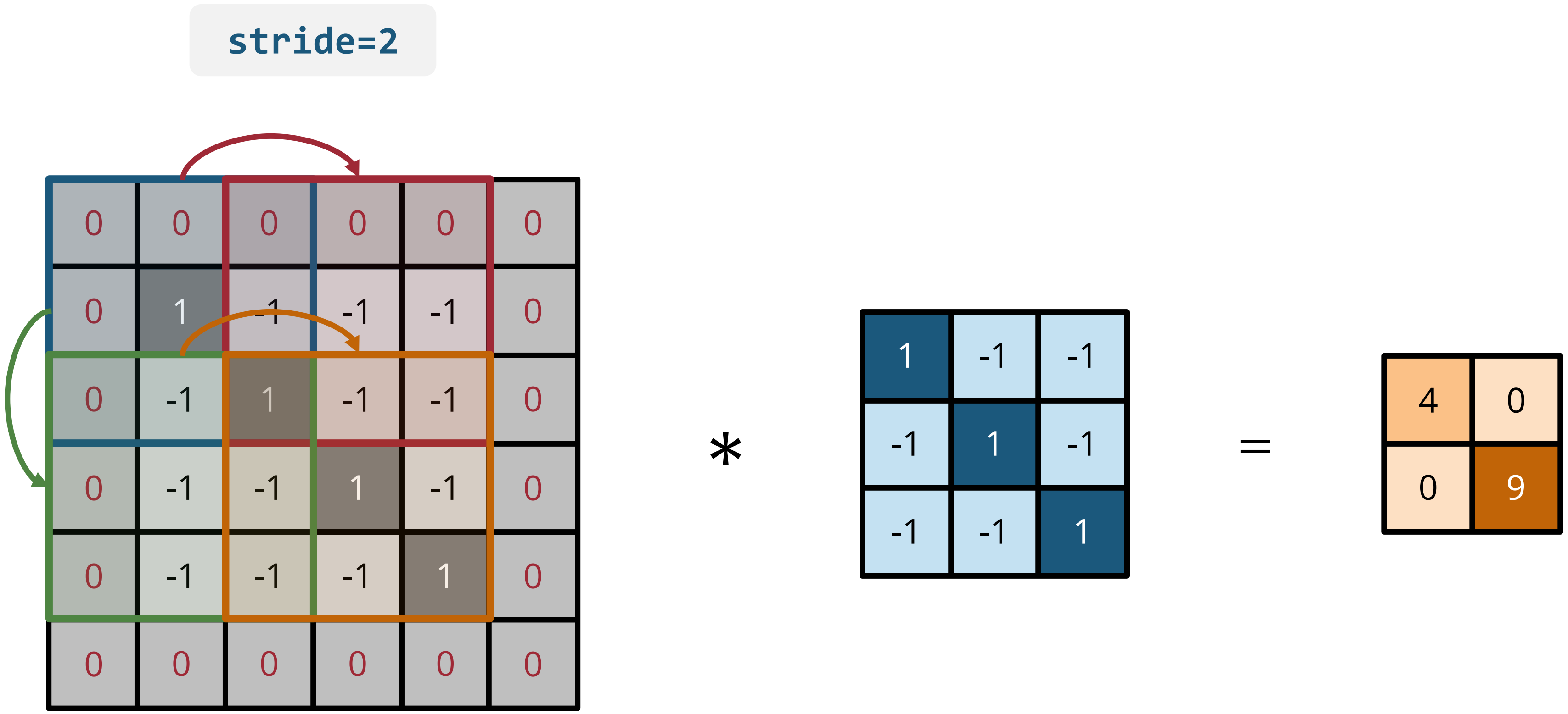
Keep the output of the same size as the input

Shapes

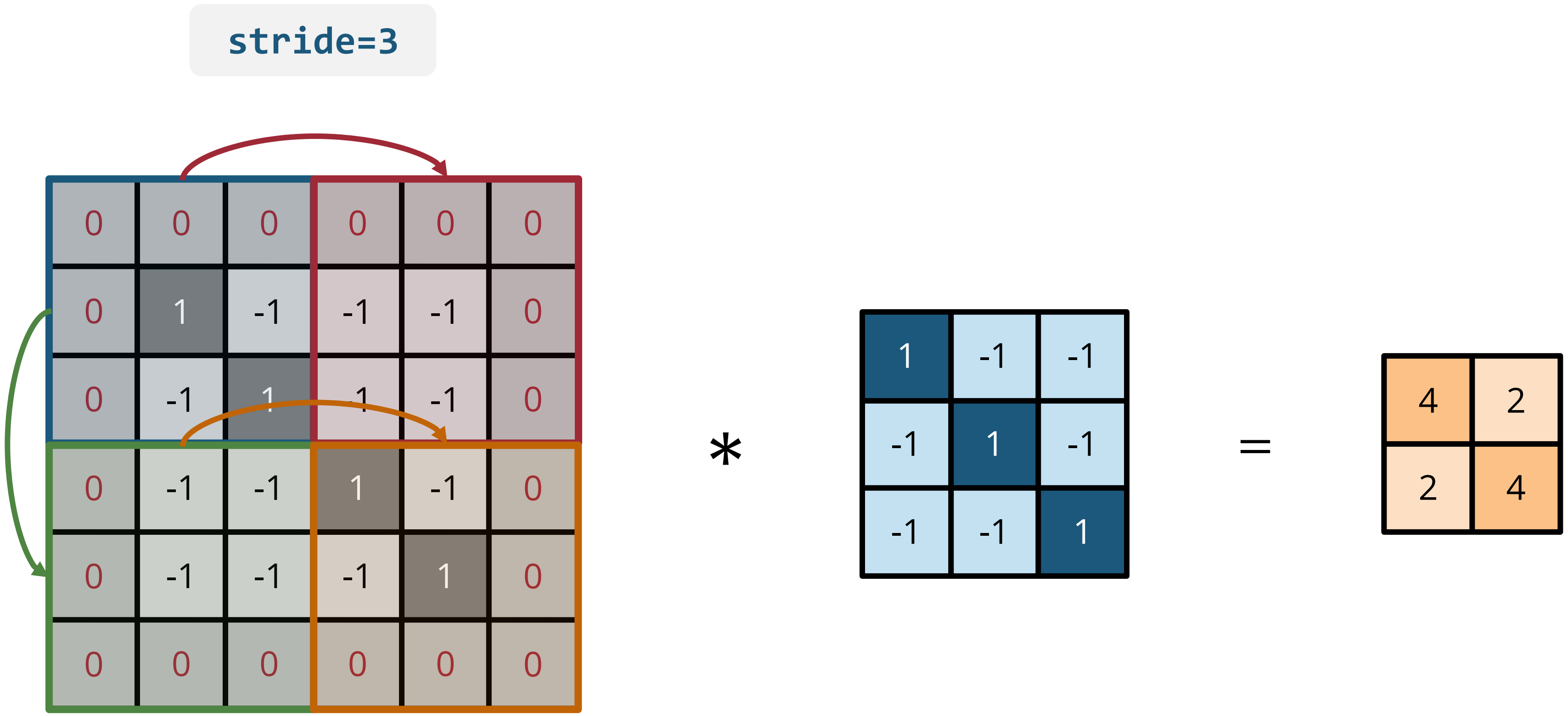


padding="valid"
N
Input
N
*
k
Kernel
k
=
N − k + 1
Output
N − k + 1
padding="same"
N
Input
N
*
k
Kernel
k
=
N
Output
N

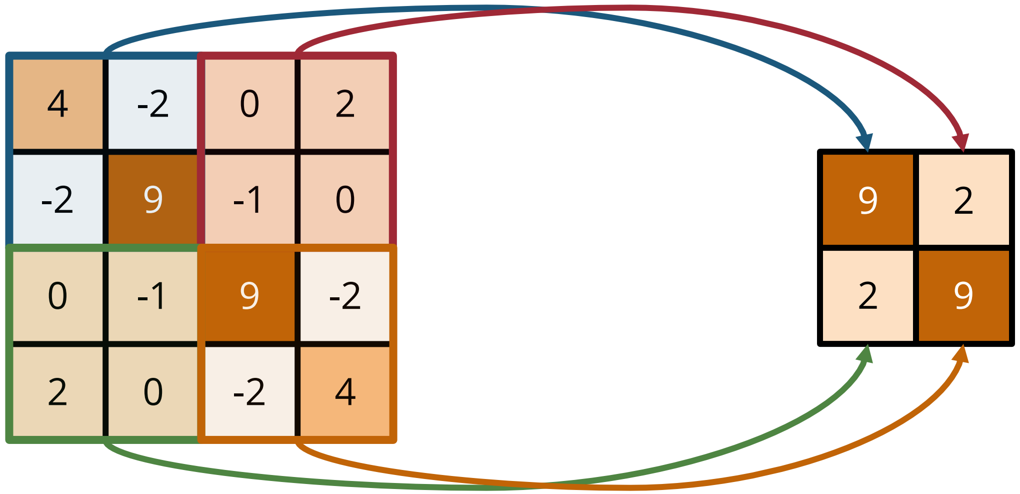
Striding


stride=2
0 0 0 0 0 0
0 1 -1 -1 -1 0
0 -1 1 -1 -1 0
0 -1 -1 1 -1 0
0 -1 -1 -1 1 0
0 0 0 0 0 0
*
1 -1 -1
-1 1 -1
-1 -1 1
=
4 0
0 9

Striding

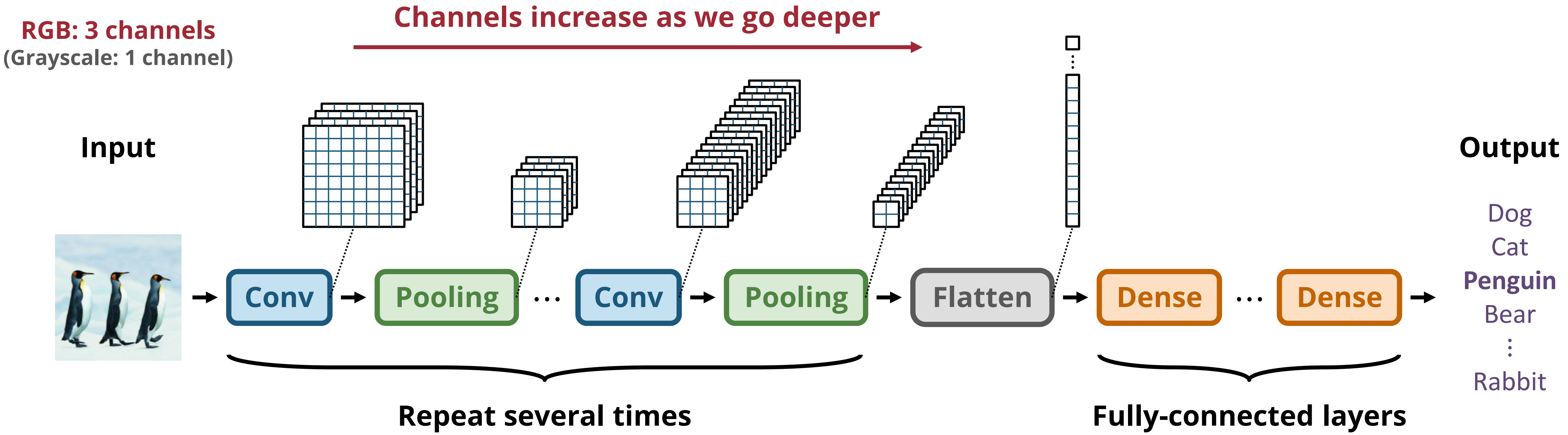

stride=3
0 0 0 0 0 0
0 1 -1 -1 -1 0
0 -1 1 1 -1 0
0 -1 -1 1 -1 0
0 -1 -1 -1 1 0
0 0 0 0 0 0
*
1 -1 -1
-1 1 -1
-1 -1 1
=
4 2
2 4

Max Pooling Layer

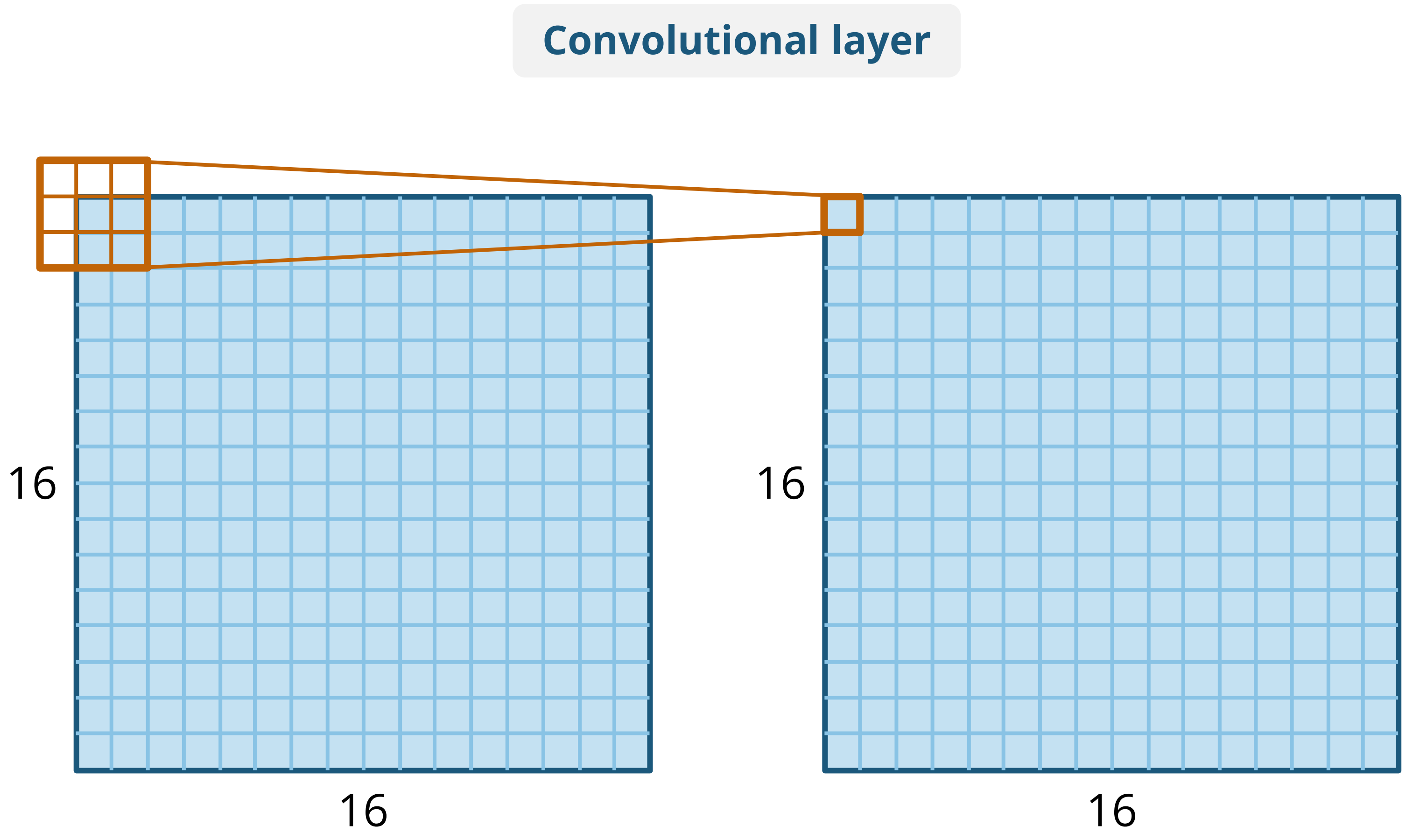


Downsample and keep the strongest activation in each block

Convolutional Neural Network (CNNs)



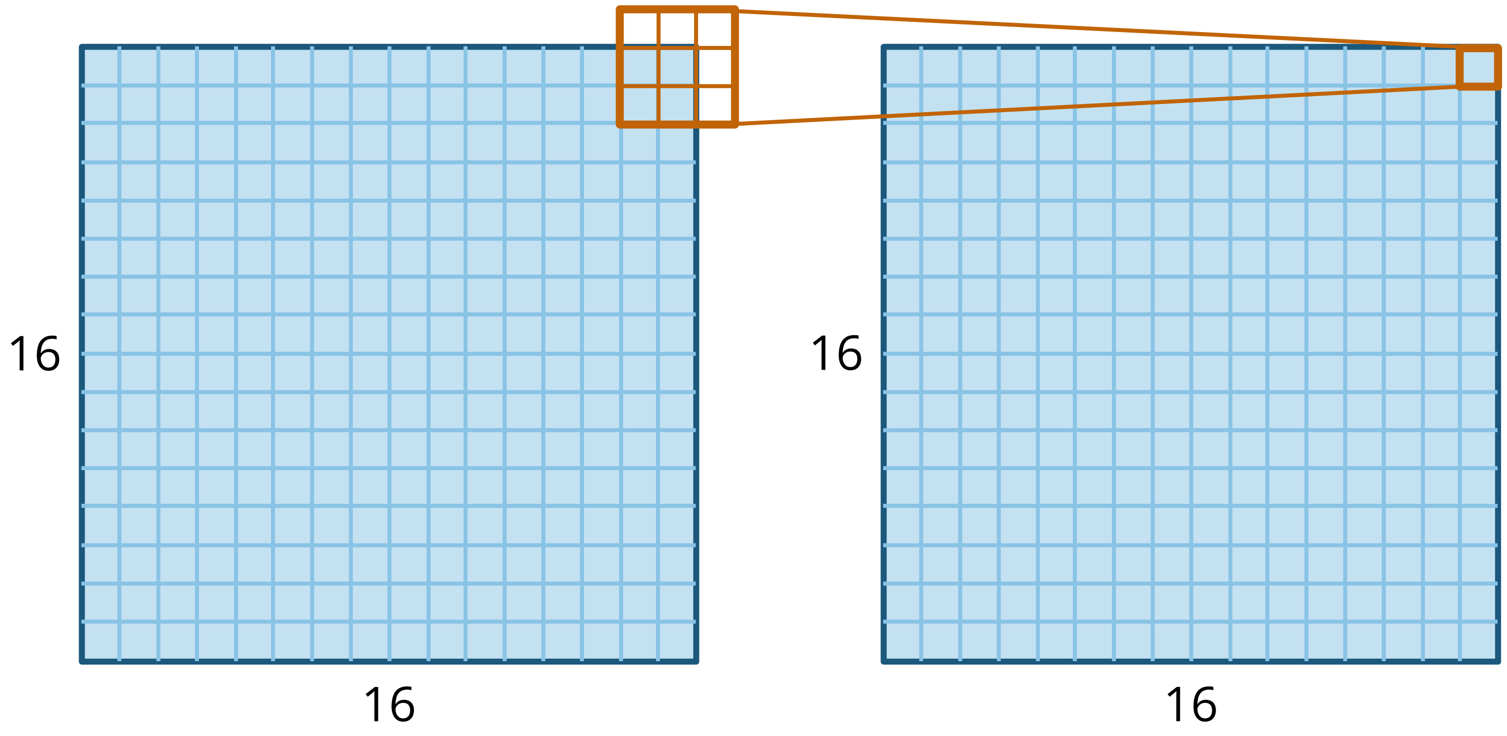
A Toy Example




```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

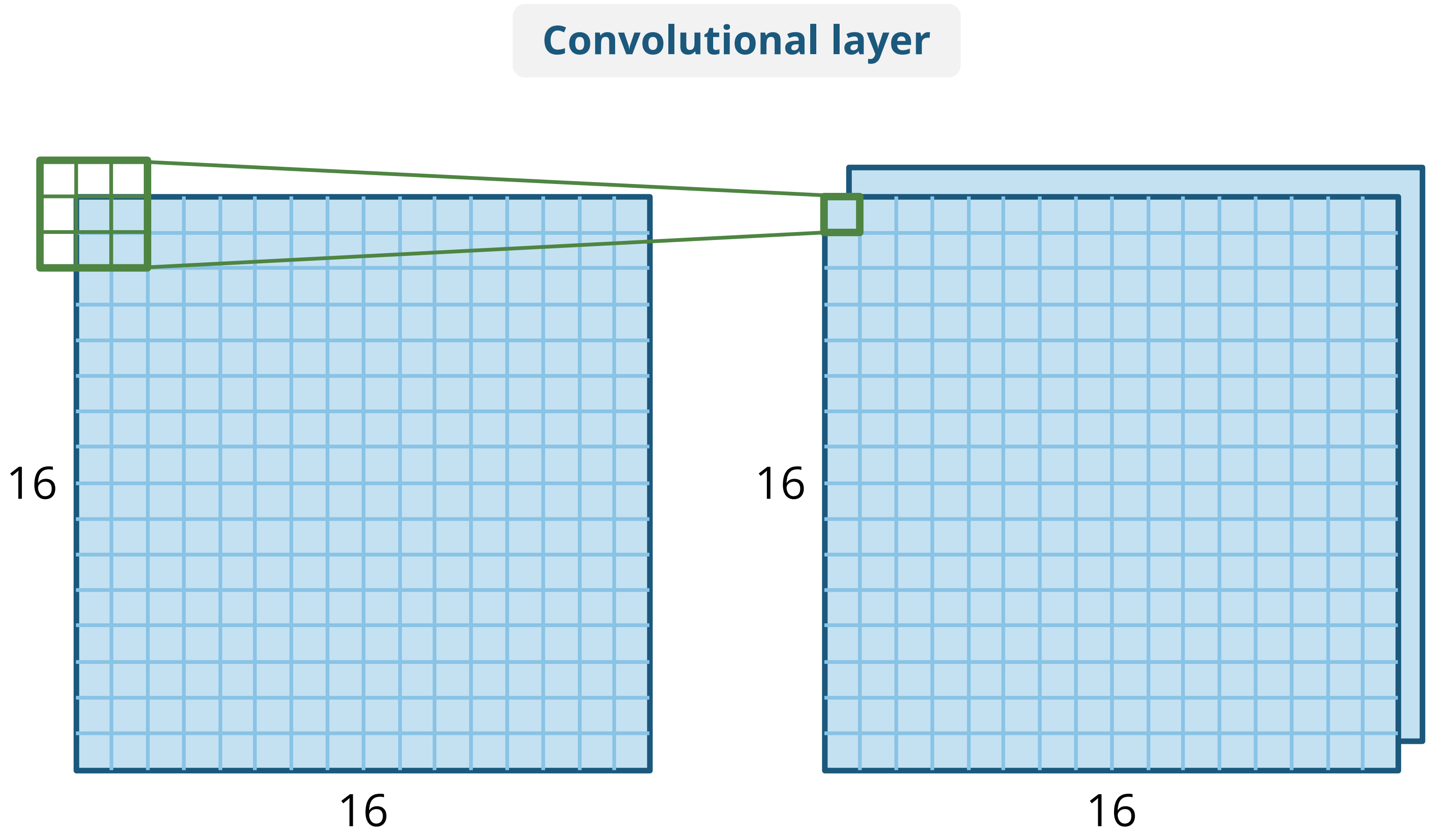
A Toy Example

Convolutional layer



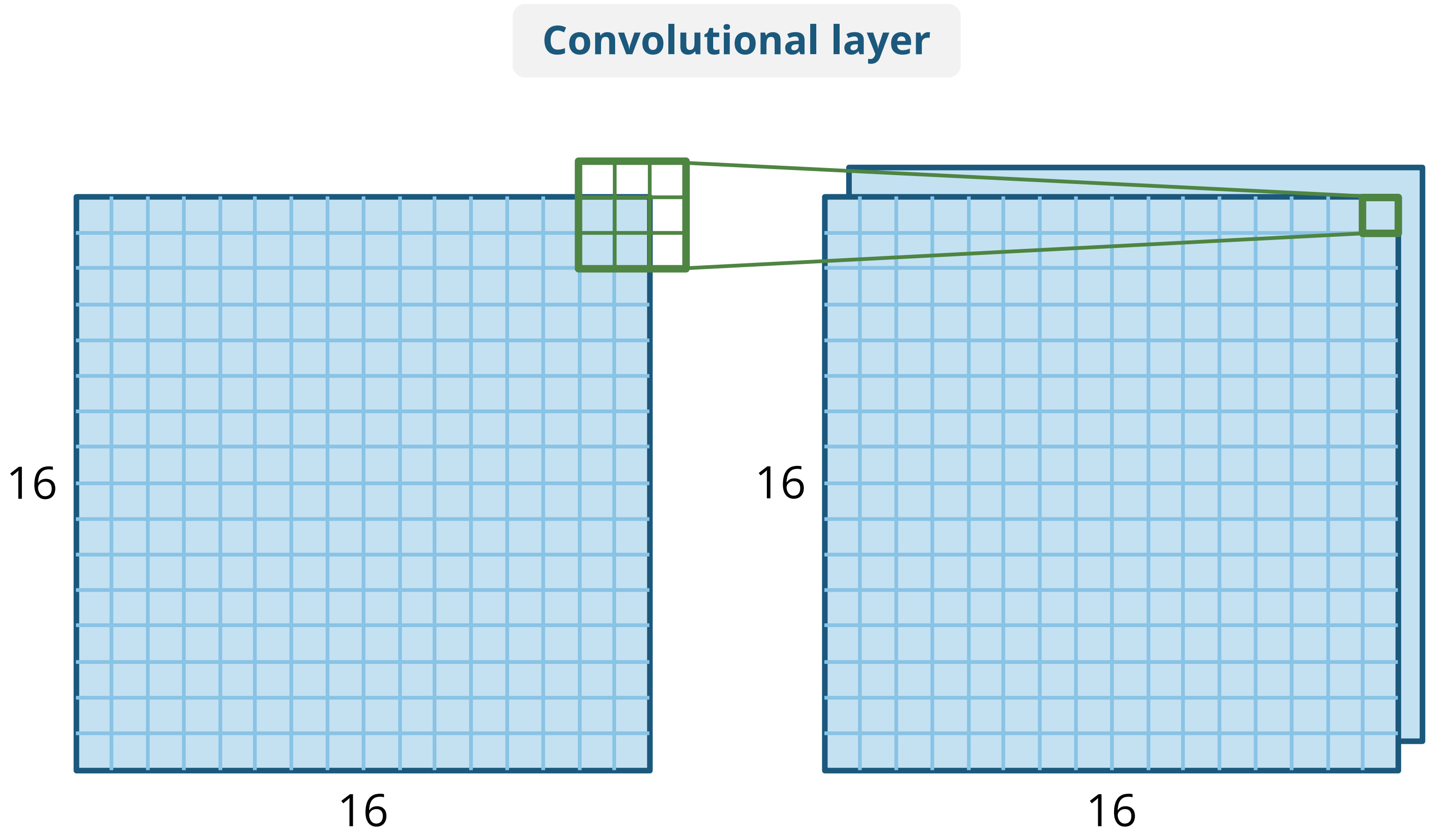

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example



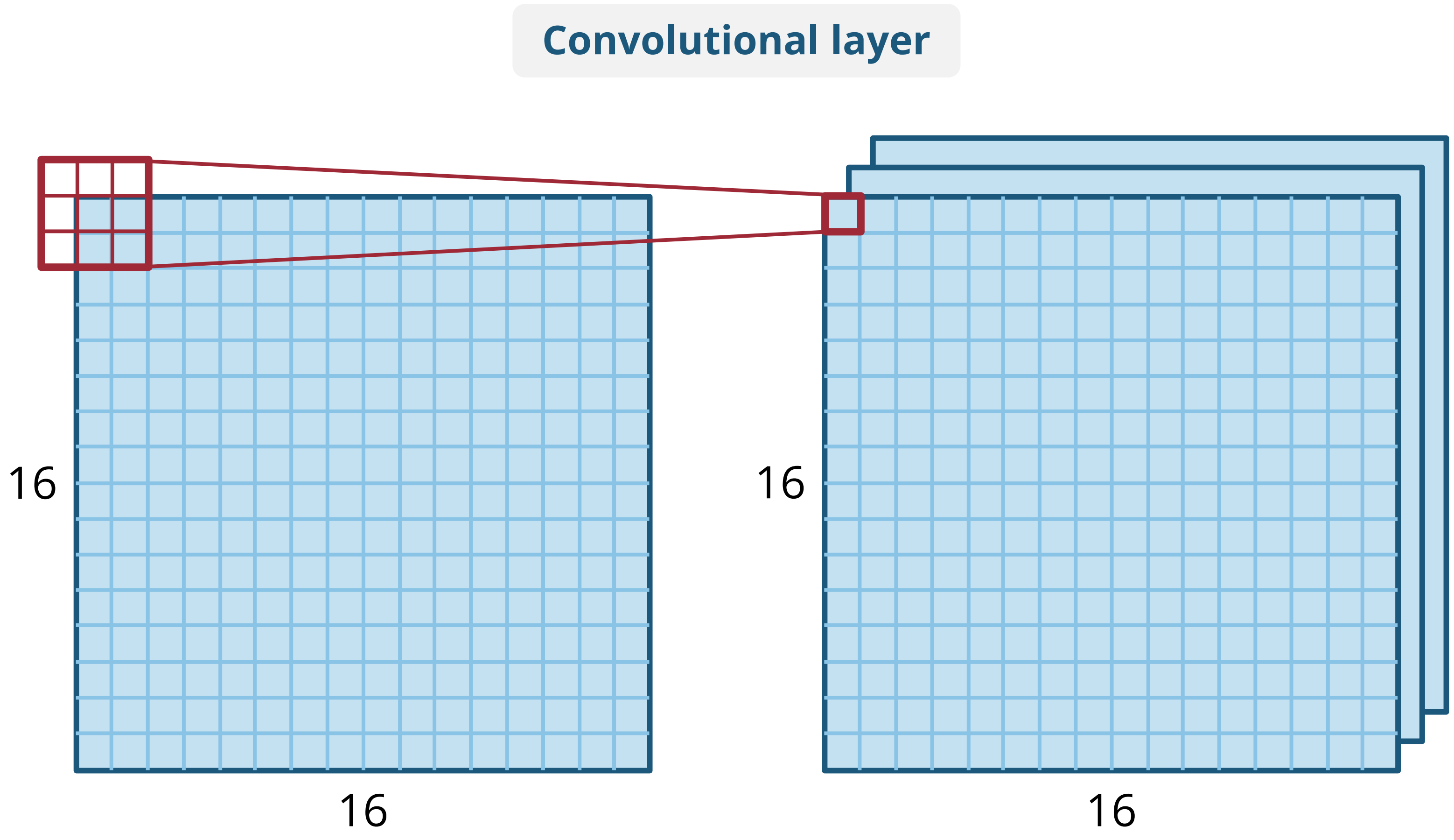

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example



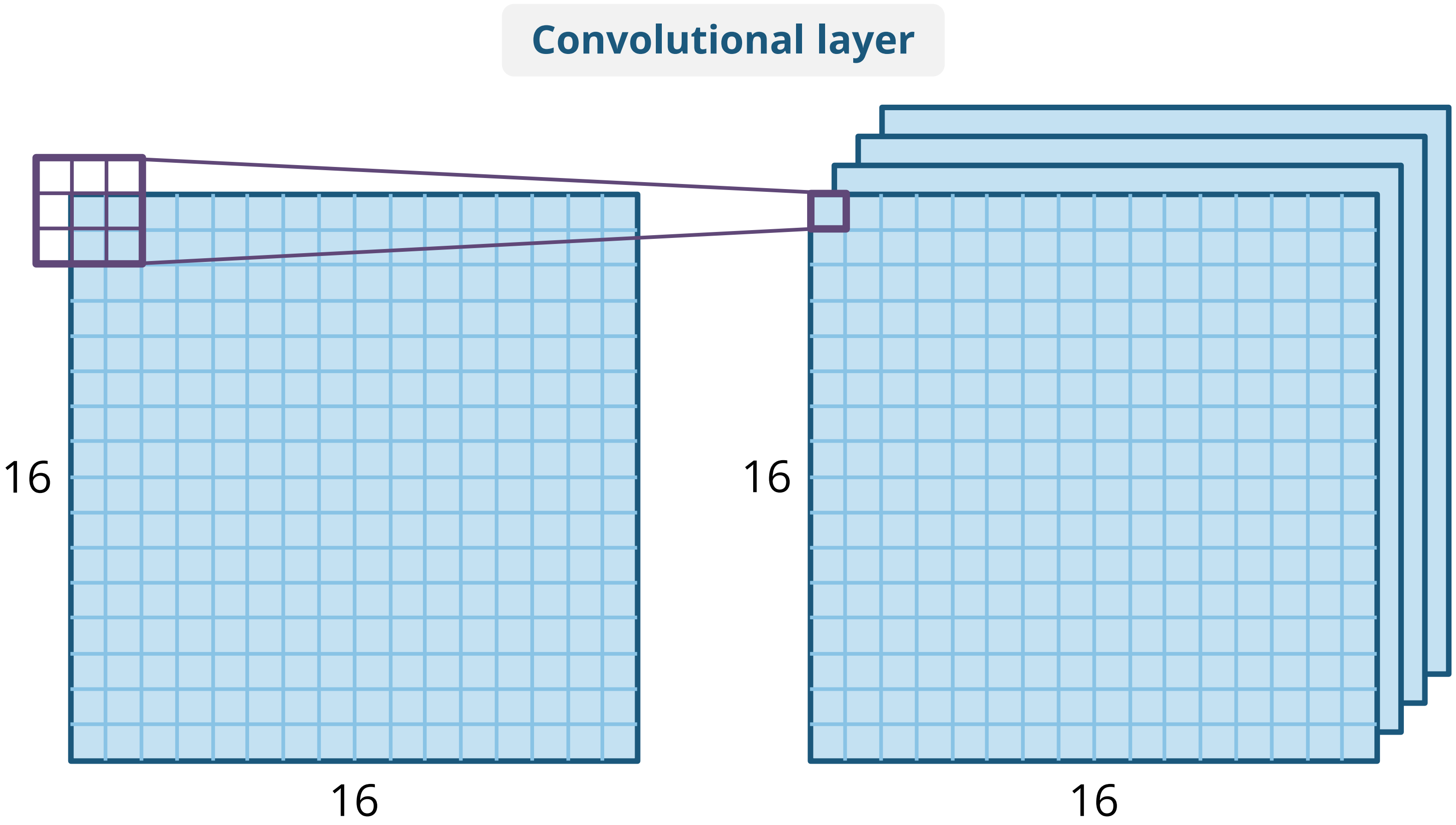

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example



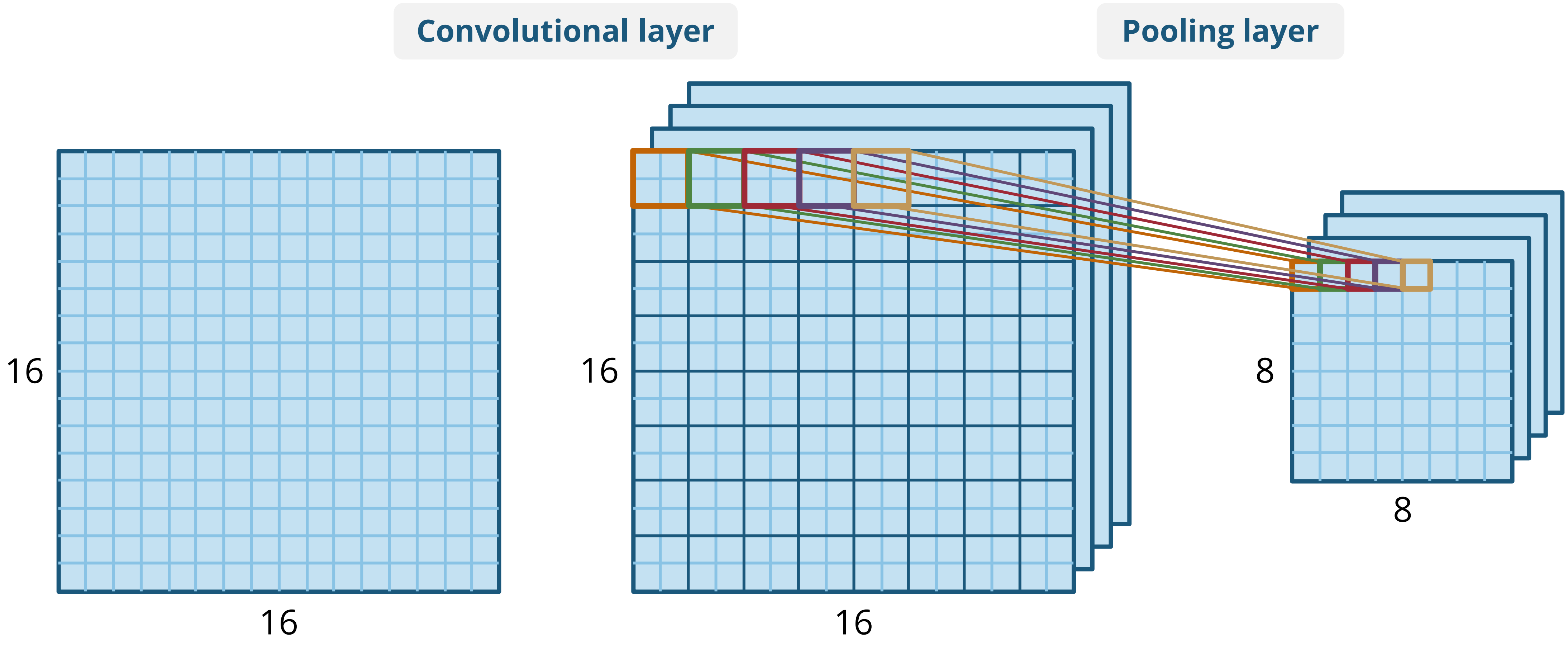

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example



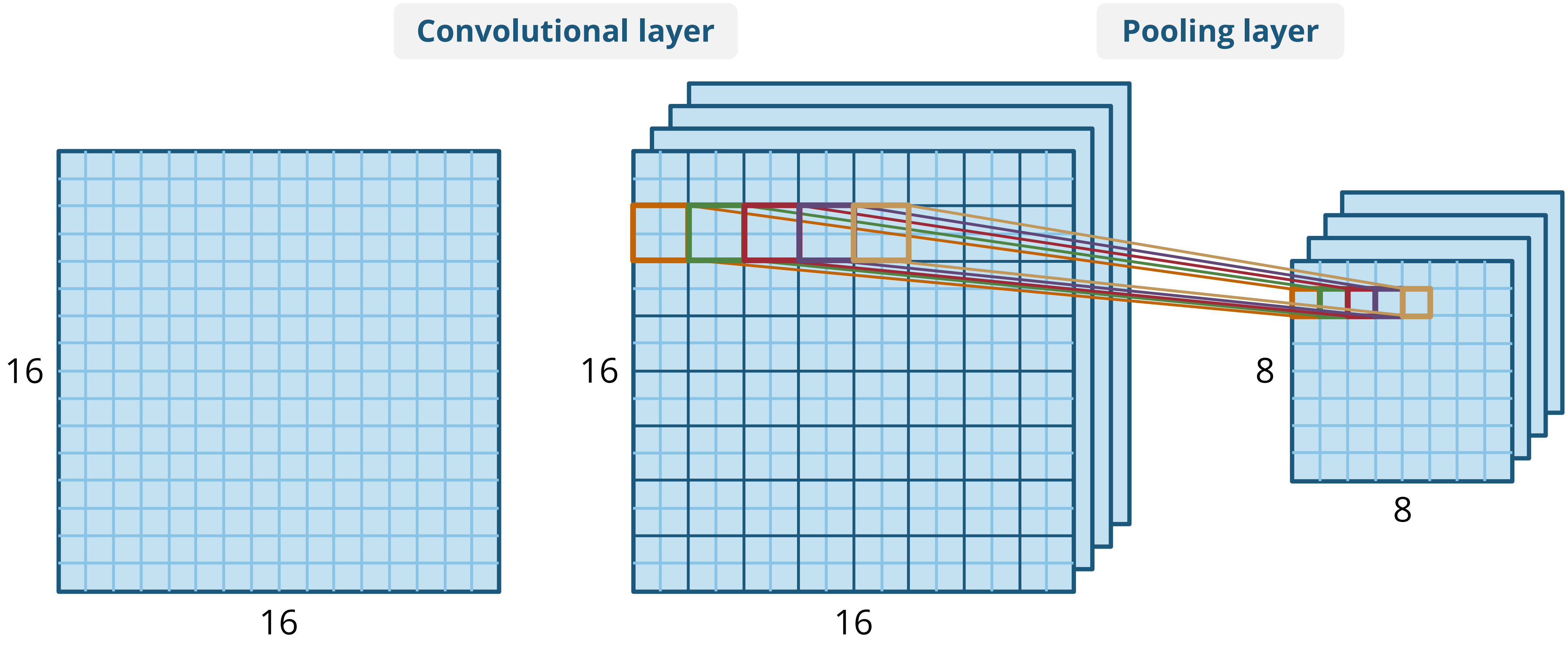

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example



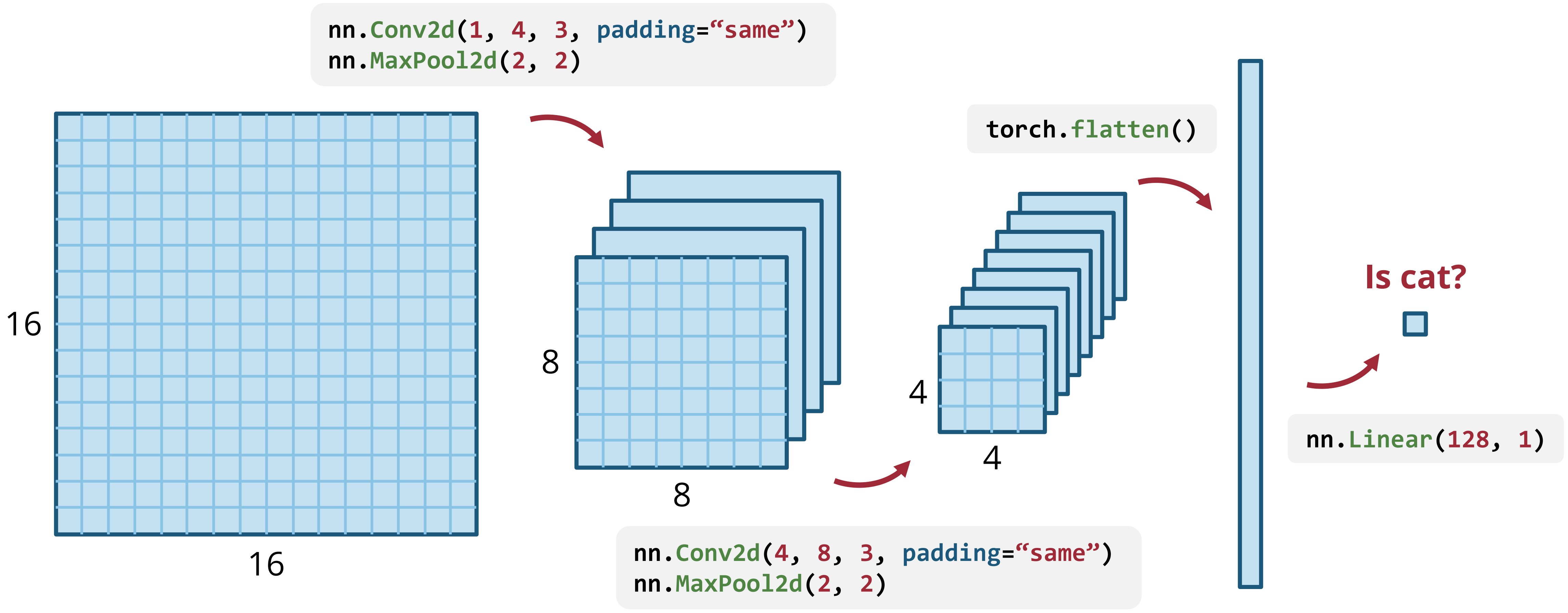

```
nn.MaxPool2d(kernel_size=2, stride=2)
```

A Toy Example

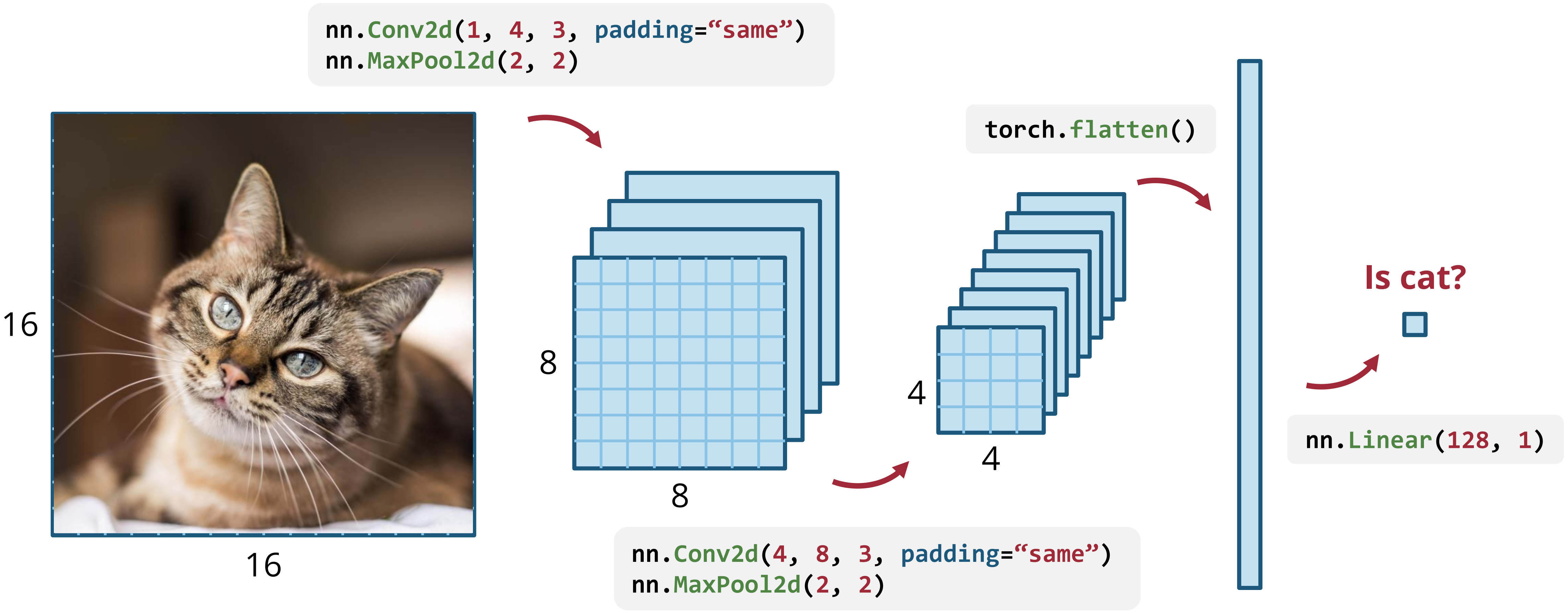



```
nn.MaxPool2d(kernel_size=2, stride=2)
```

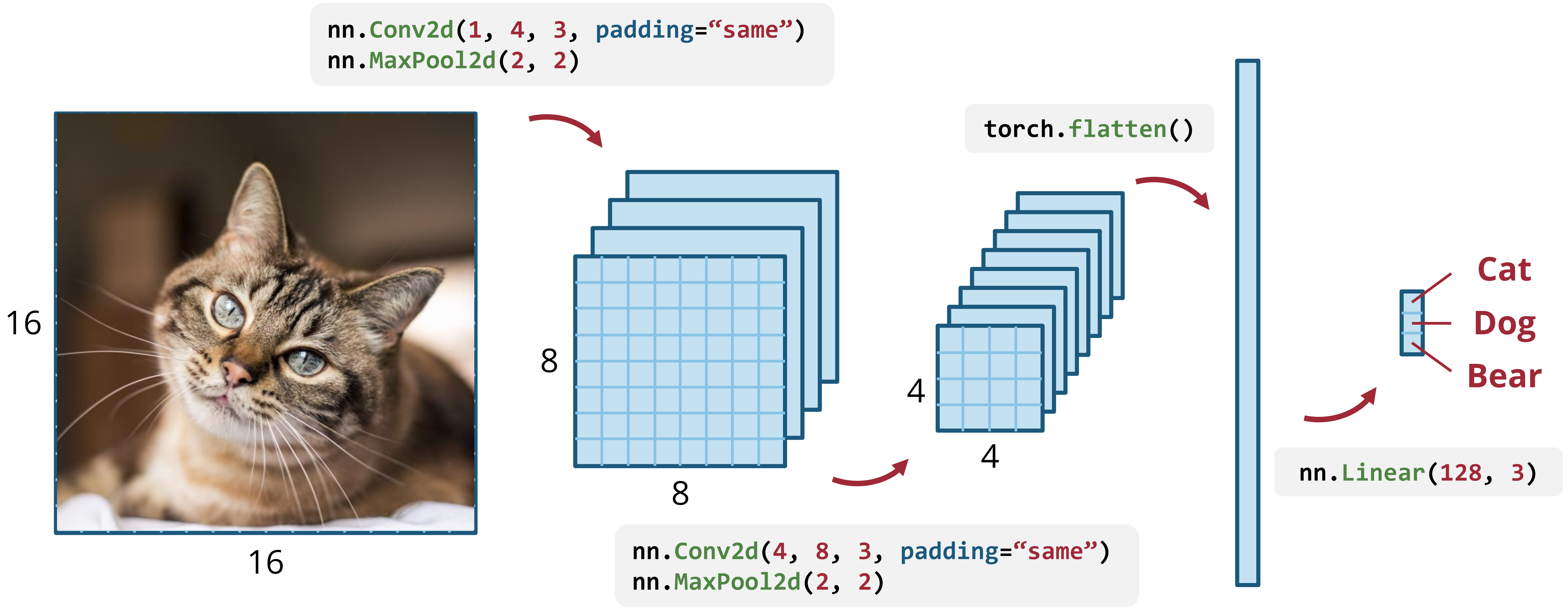
A Toy Example



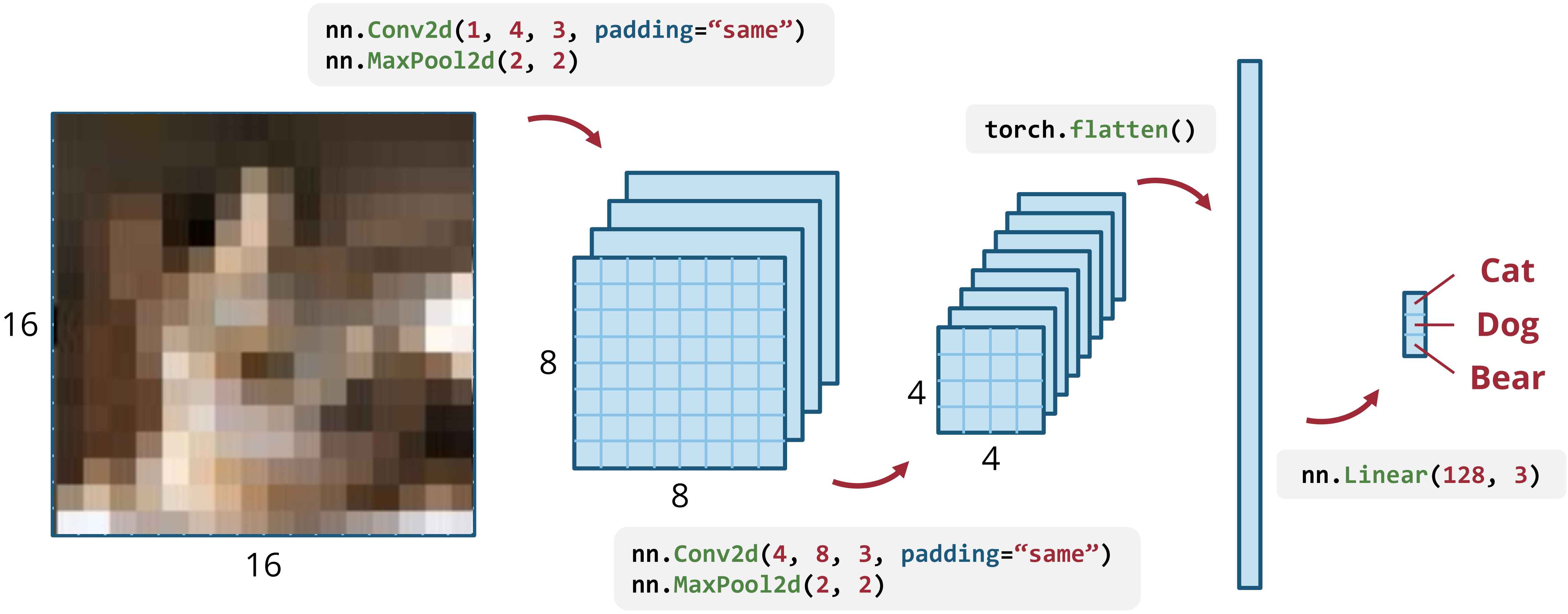
A Toy Example



A Toy Example



A Toy Example

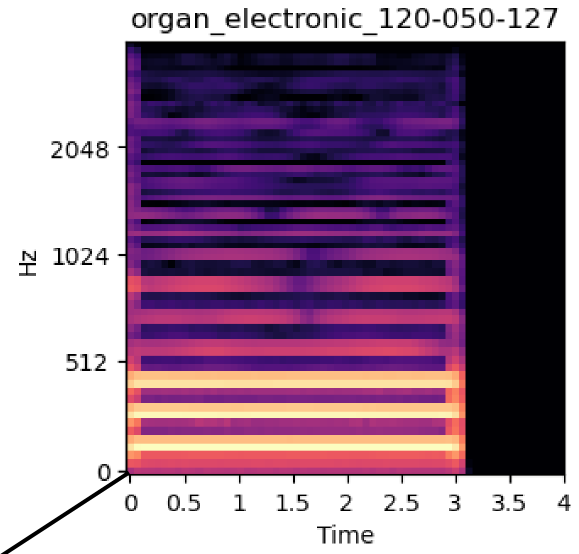


A Real Example

Input channels   Output channels   Kernel size

```python
class CNN(nn.Module):
    """A basic convolutional neural network."""
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding="same")
        self.conv2 = nn.Conv2d(16, 32, 3, padding="same")
        self.conv3 = nn.Conv2d(32, 64, 3, padding="same")
        self.conv4 = nn.Conv2d(64, 128, 3, padding="same")
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(128 * 4 * 4, n_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```



Input: 1	x 64 x 64		
16	x 32 x 32	=	16384
32	x 16 x 16	=	8192
64	x 8 x 8	=	4096
128	x 4 x 4	=	2048

More channels, lower resolution as we go deeper

Total number of features decrease as we go deeper

A Real Example

Input channels

Output channels

Kernel size

```python
class CNN(nn.Module):
    """A basic convolutional neural network."""
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding="same")
        self.conv2 = nn.Conv2d(16, 32, 3, padding="same")
        self.conv3 = nn.Conv2d(32, 64, 3, padding="same")
        self.conv4 = nn.Conv2d(64, 128, 3, padding="same")
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(128 * 4 * 4, n_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

How many parameters do we have in each layer?

$(3 \times 3 \times 1 + 1) \times 16 = 160$

$(3 \times 3 \times 16 + 1) \times 32 = 4640$

$(3 \times 3 \times 32 + 1) \times 64 = 18496$

$(3 \times 3 \times 64 + 1) \times 128 = 73856$

$(2048 + 1) \times 11 = 22539$

Benefits of CNNs

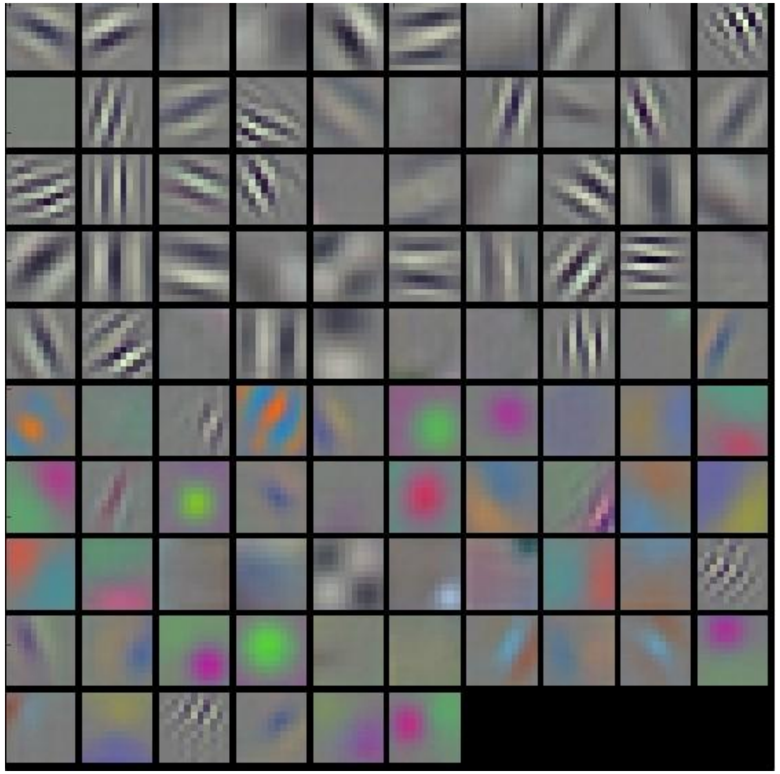
- Learn **local patterns**
- **Invariant to shifts**
  - Also called *translational invariance*
- **Reuse the learned filters** across
  - Different parts of the image
  - Across different images
- **Higher parameter-efficiency** against fully-connected neural network

What does a CNN Learn?

Learned CNN Kernels in a Trained AlexNet

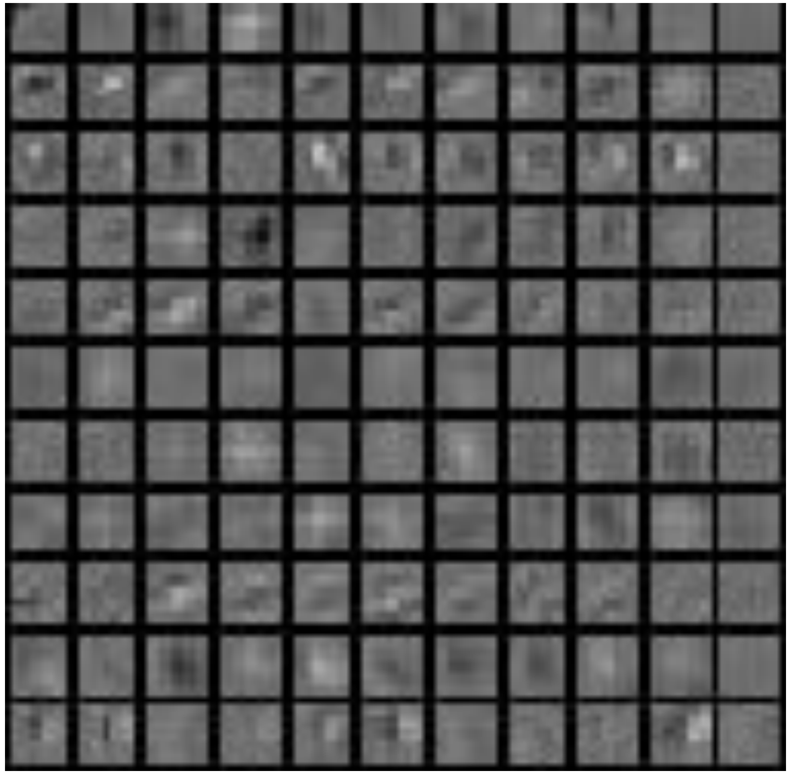
1st convolutional layer

11x11 kernels

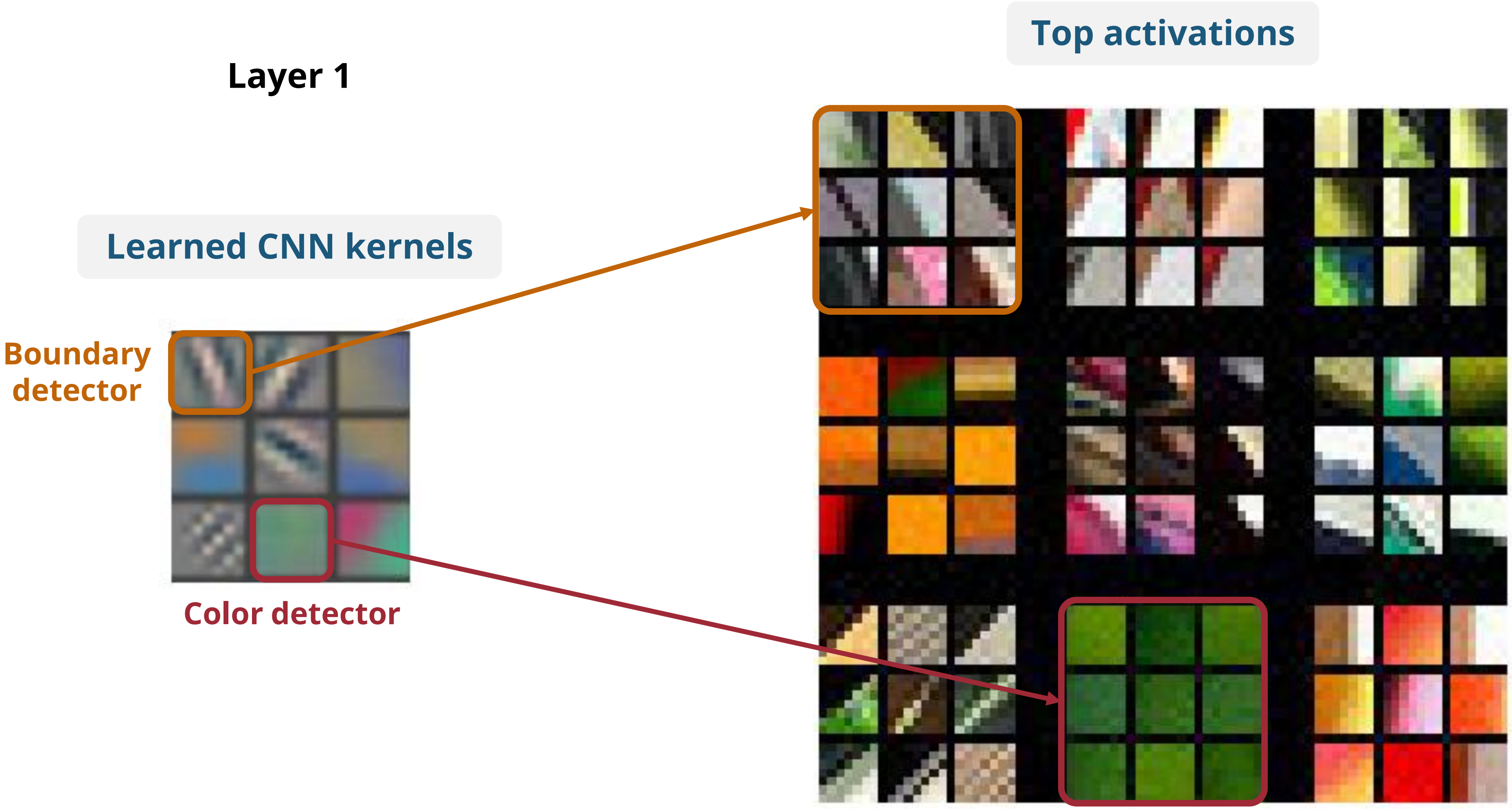


2nd convolutional layer

5x5 kernels

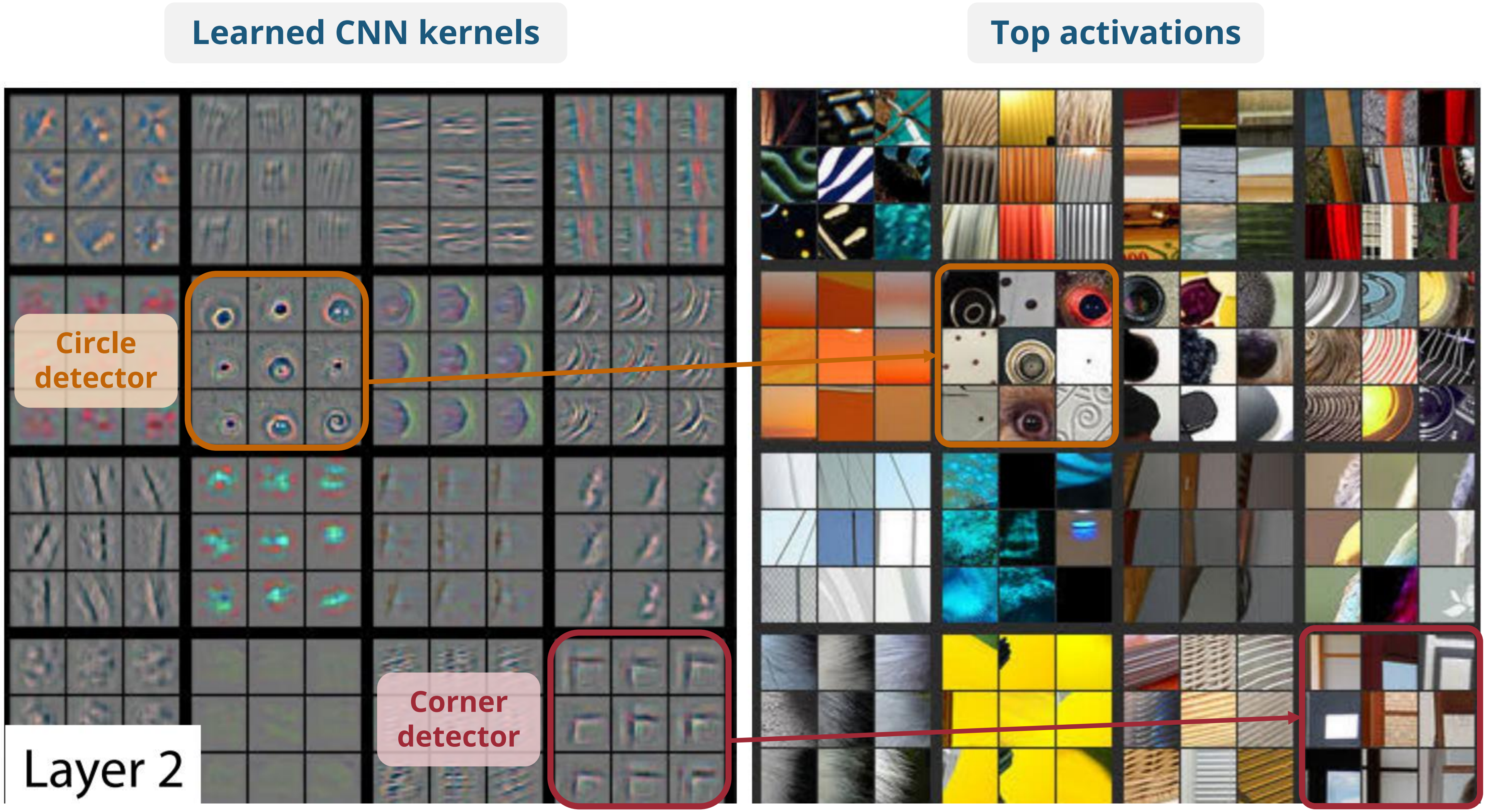


Learned CNN Kernels in a Trained AlexNet



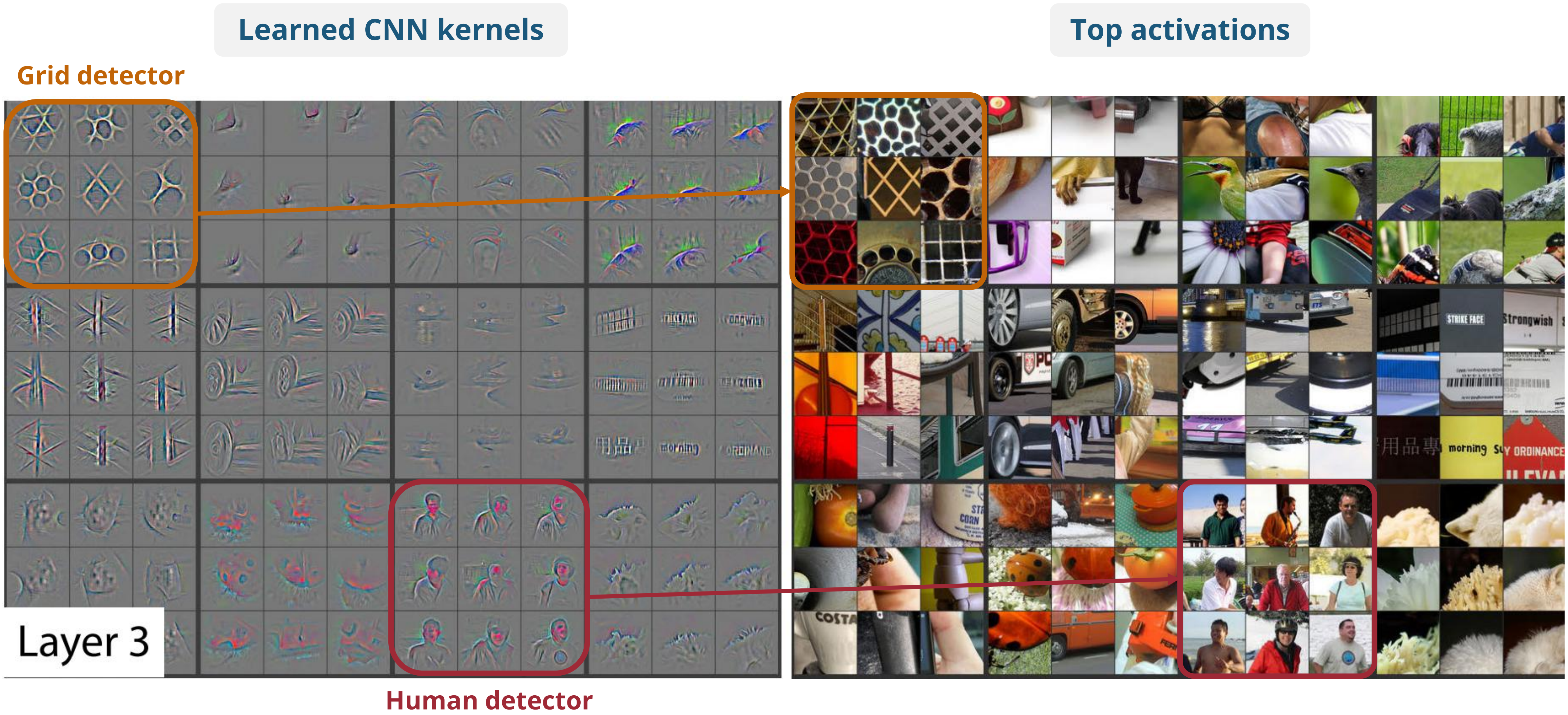
Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet



Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

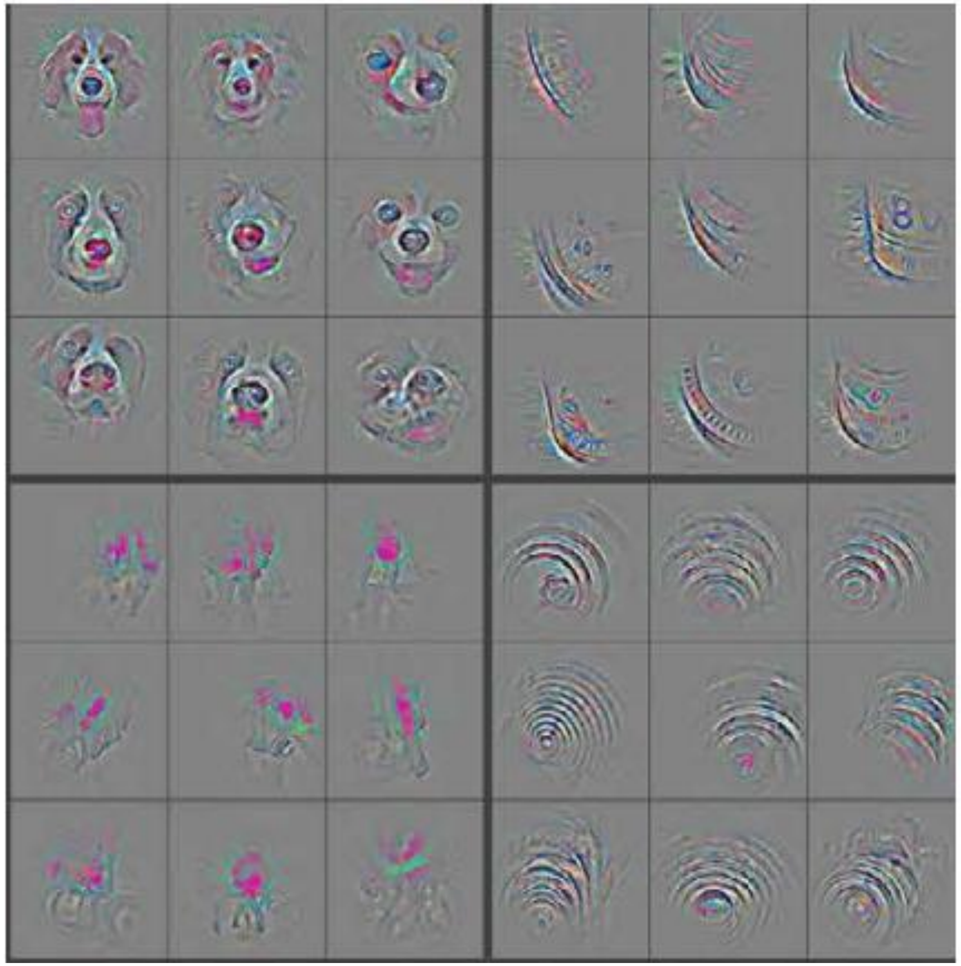
Learned CNN Kernels in a Trained AlexNet



Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet

Learned CNN kernels

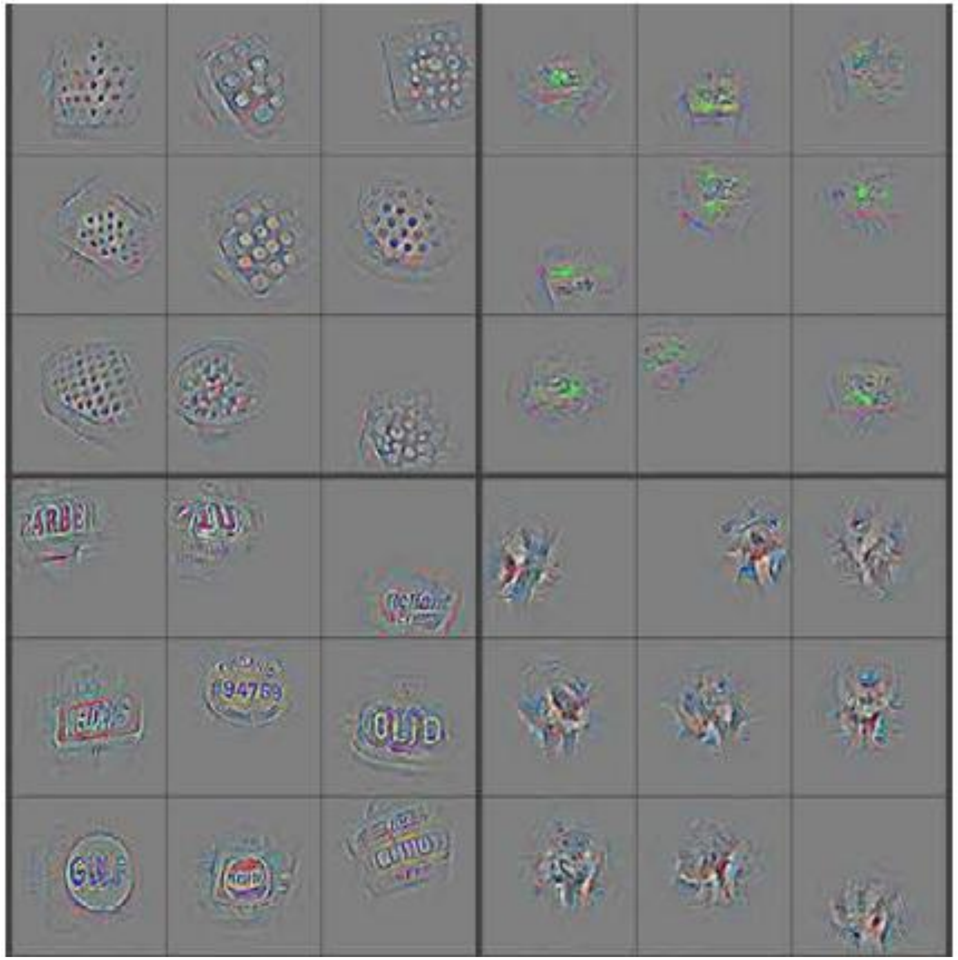


Top activations

Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet

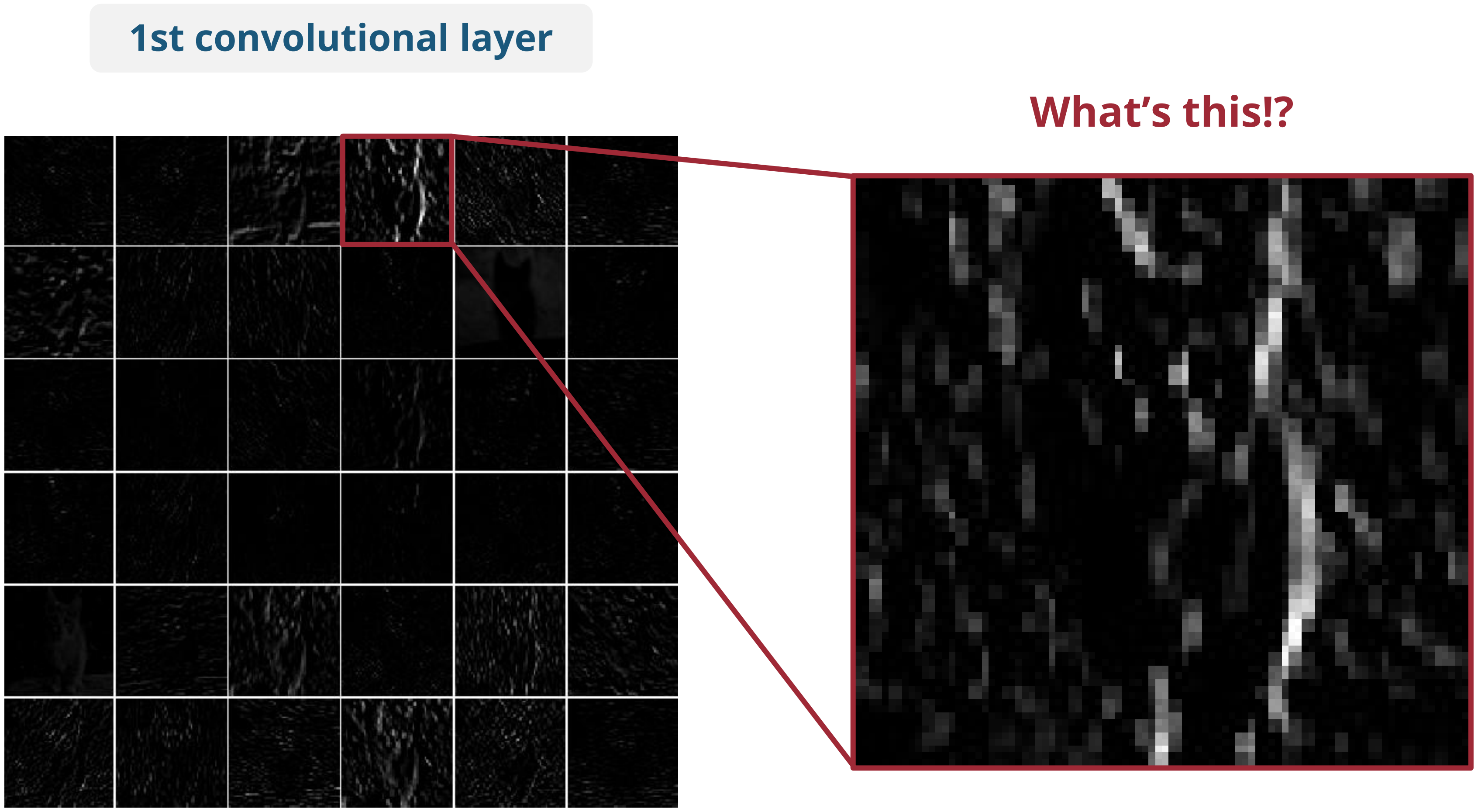
Learned CNN kernels



Top activations

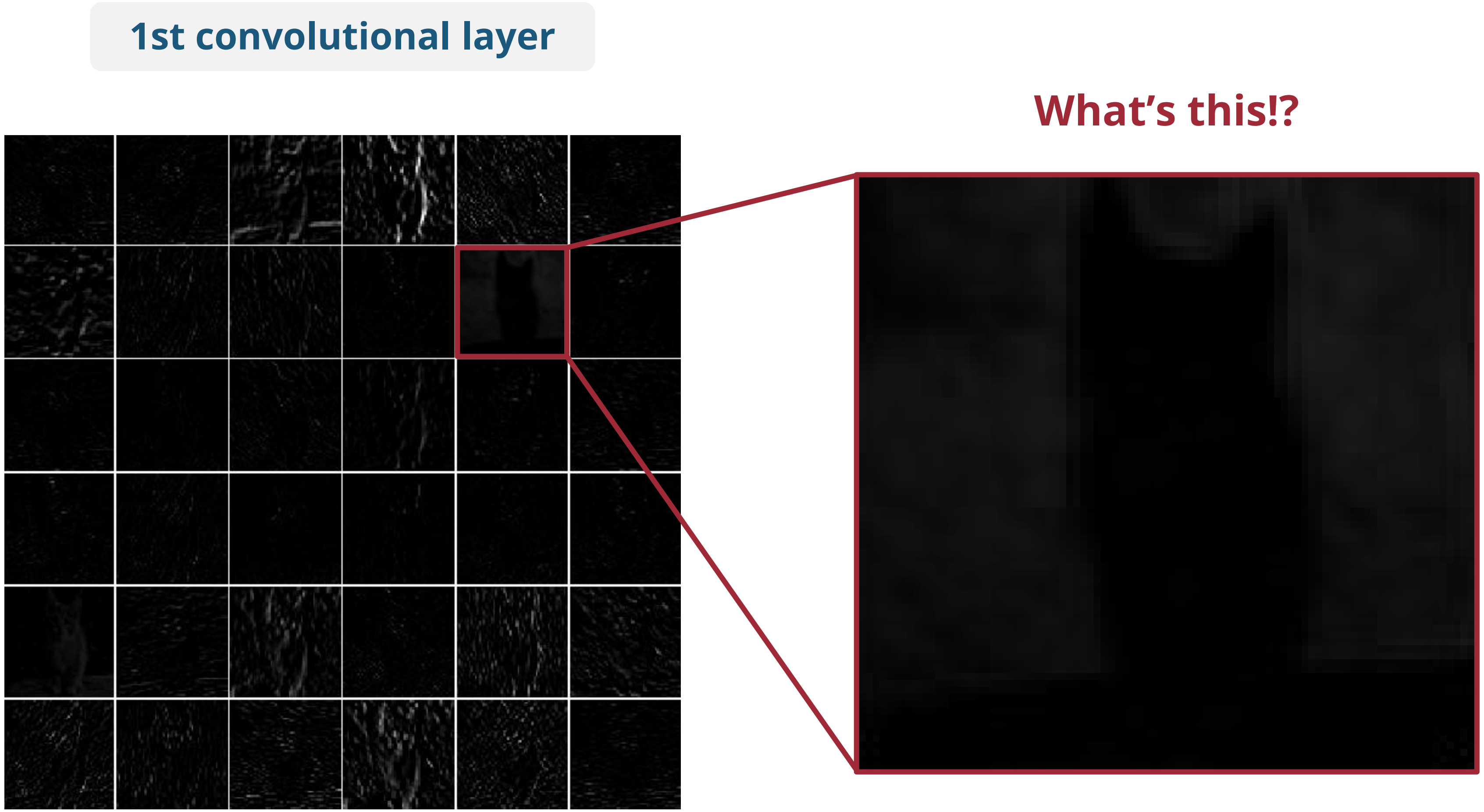
Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Activations in a Trained AlexNet



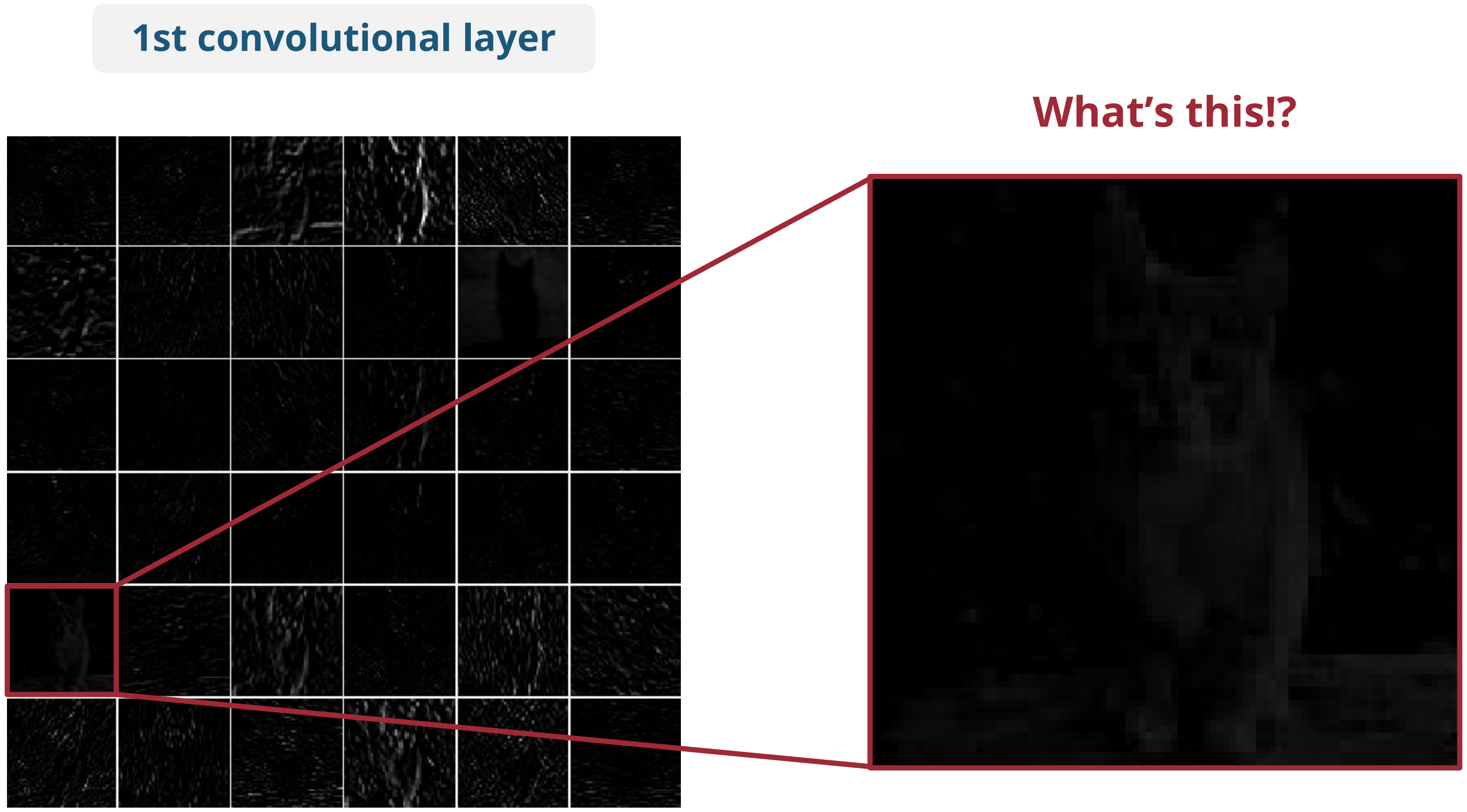
cs231n.github.io/understanding-cnn/

Activations in a Trained AlexNet



cs231n.github.io/understanding-cnn/

Activations in a Trained AlexNet

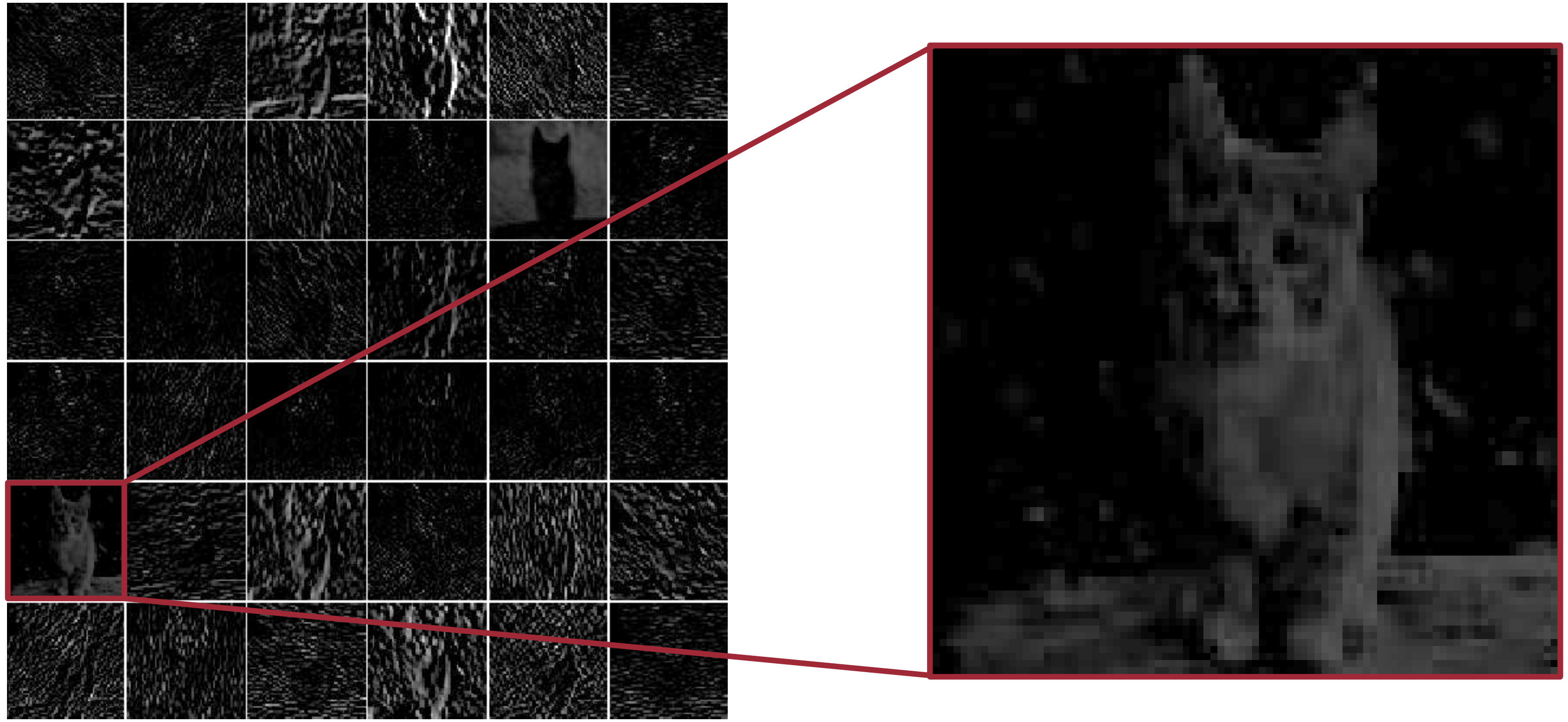


cs231n.github.io/understanding-cnn/

Activations in a Trained AlexNet

1st convolutional layer

What's this!?

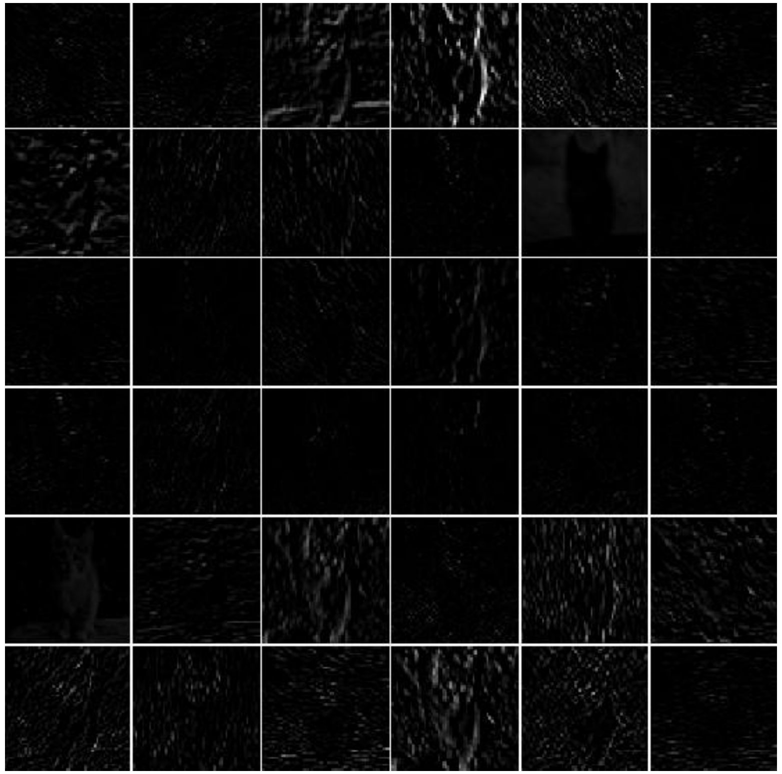


Brightness+
Contrast+

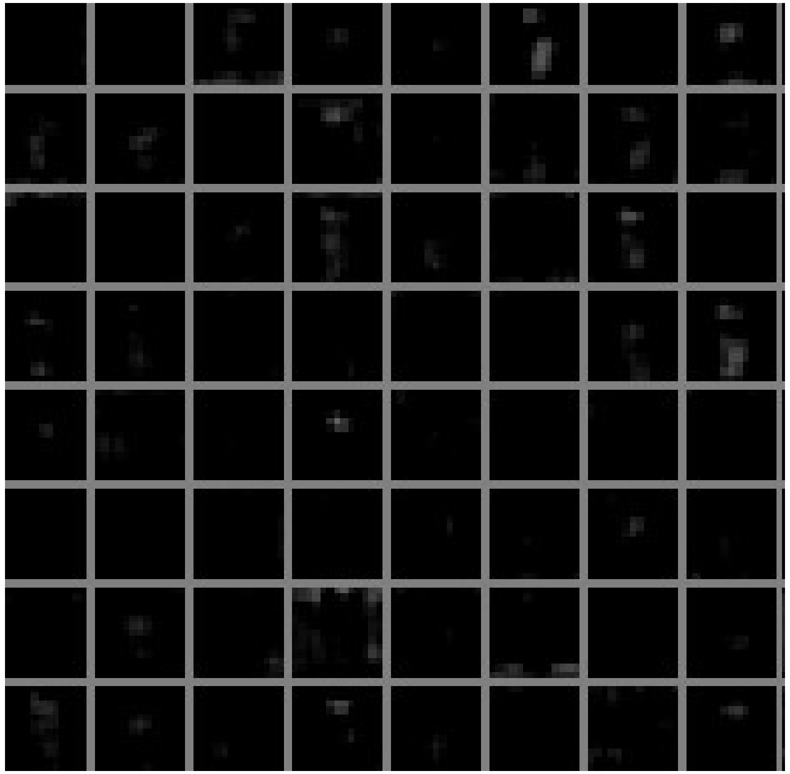
cs231n.github.io/understanding-cnn/

Activations in a Trained AlexNet

1st convolutional layer

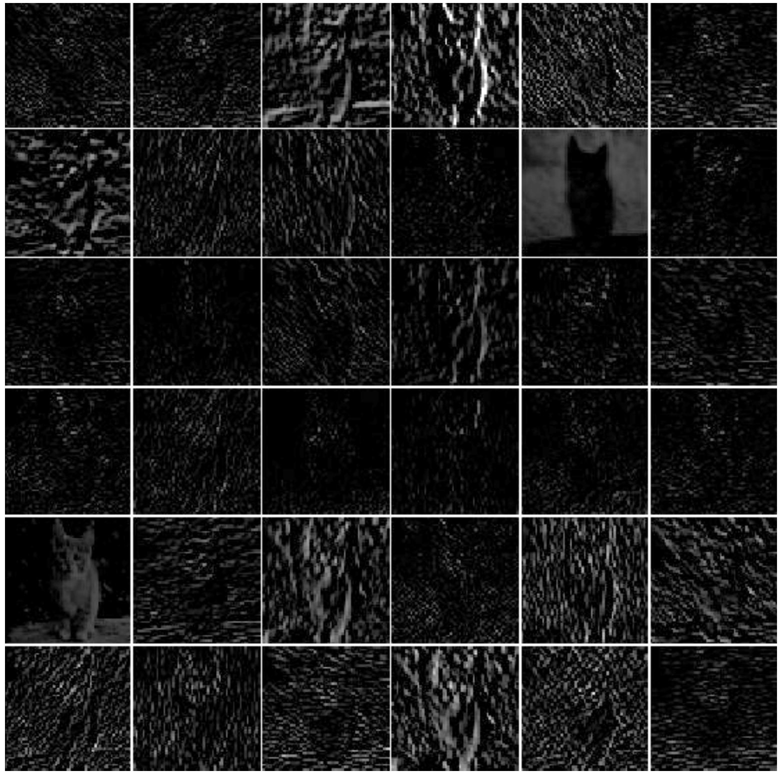


5th convolutional layer



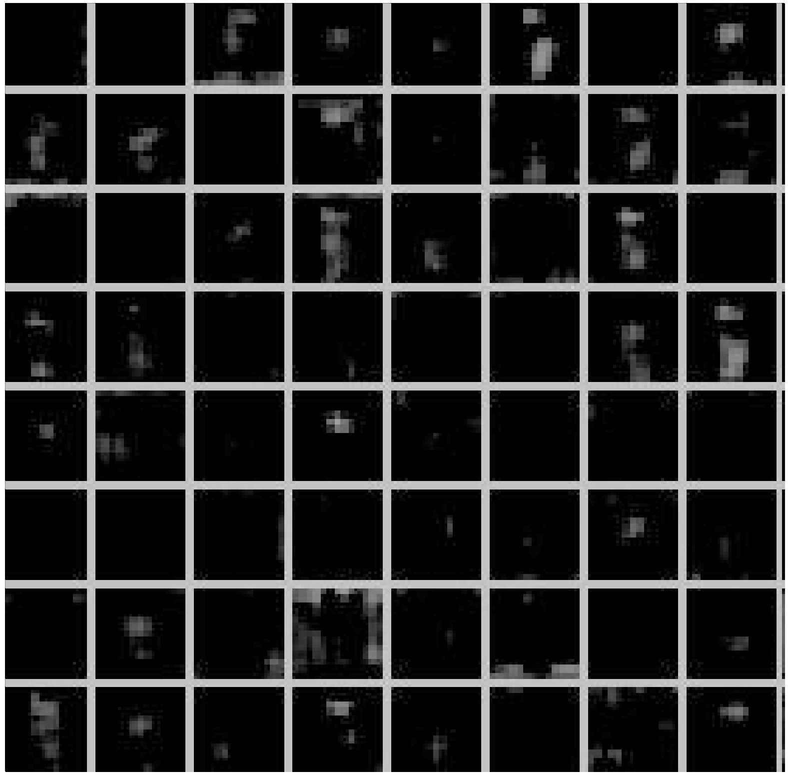
Activations in a Trained AlexNet

1st convolutional layer



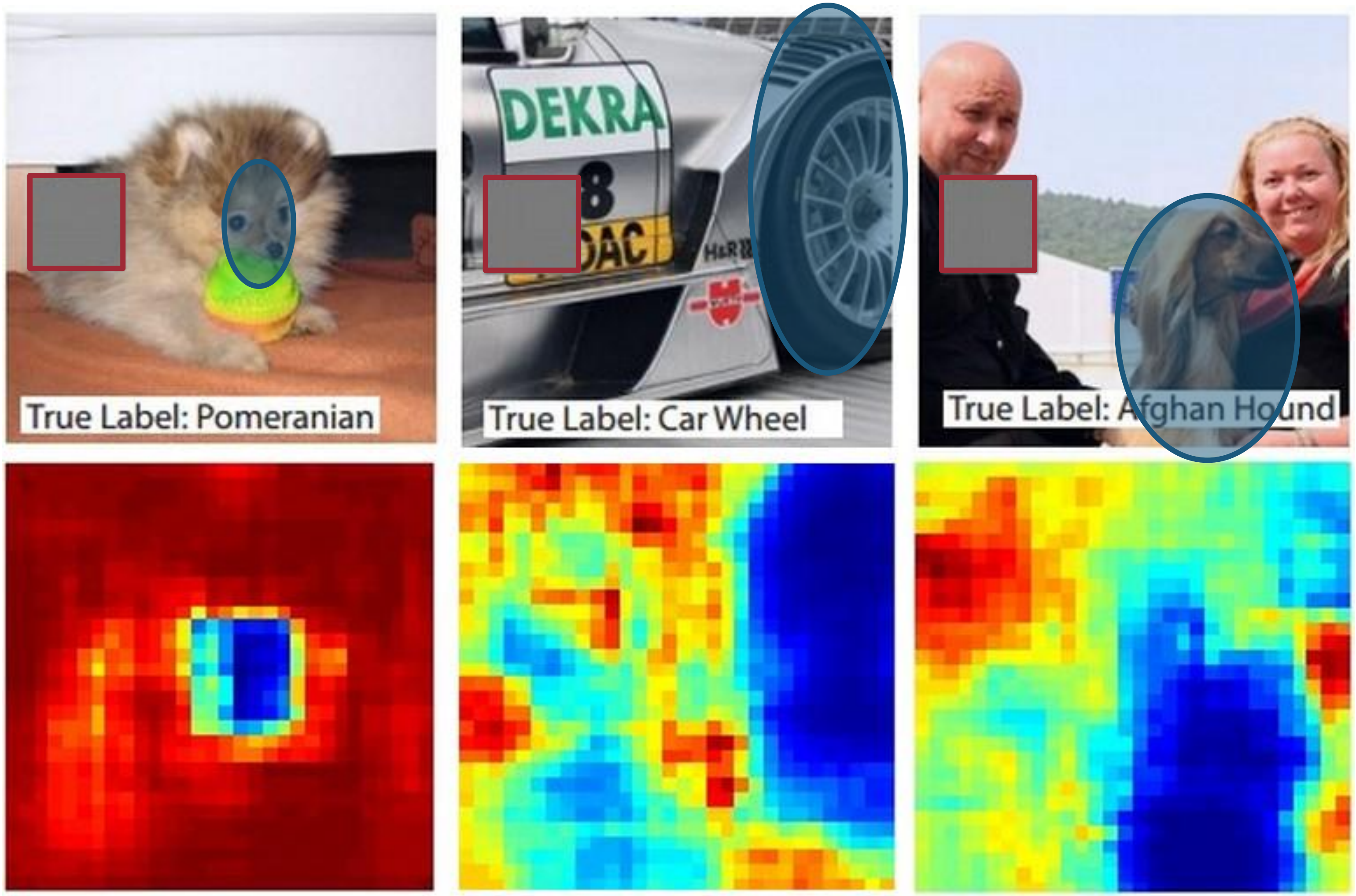
Brightness+ Contrast+

5th convolutional layer



Brightness+ Contrast+

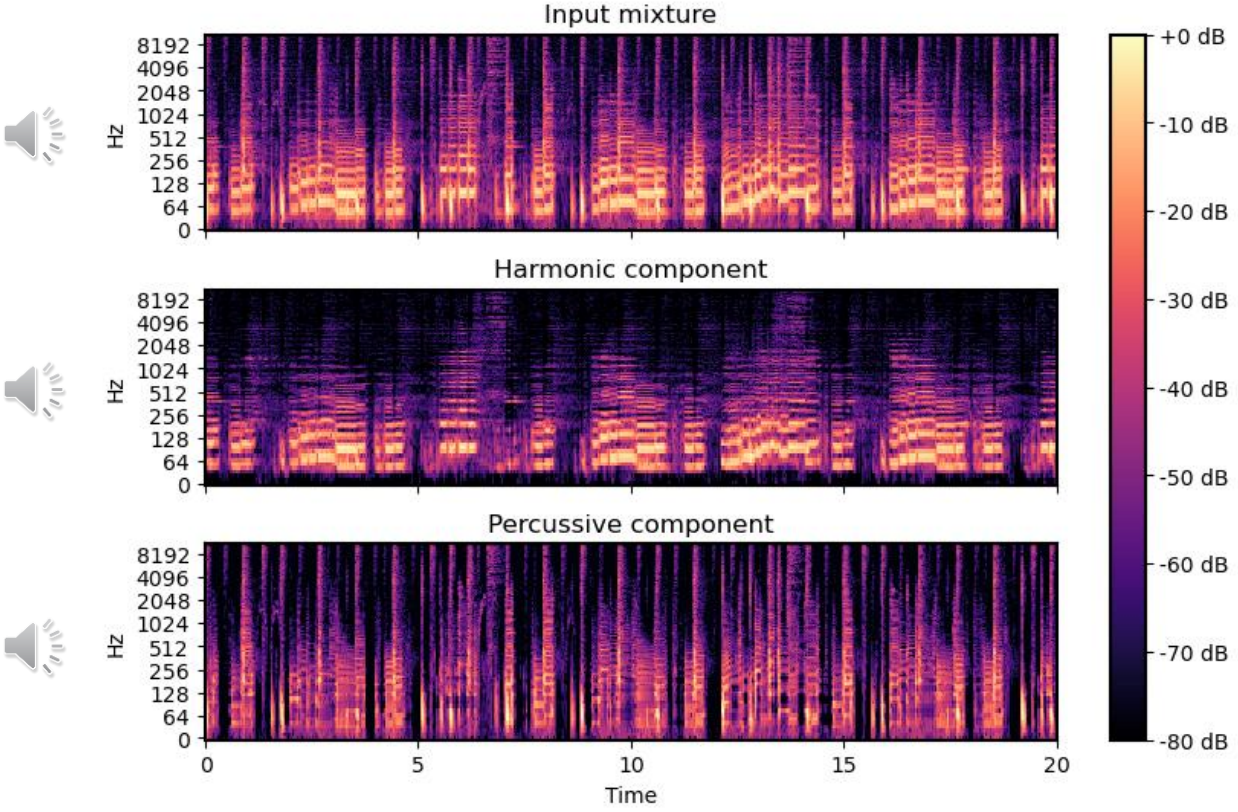
What does a CNN Learn?



cs231n.github.io/understanding-cnn/

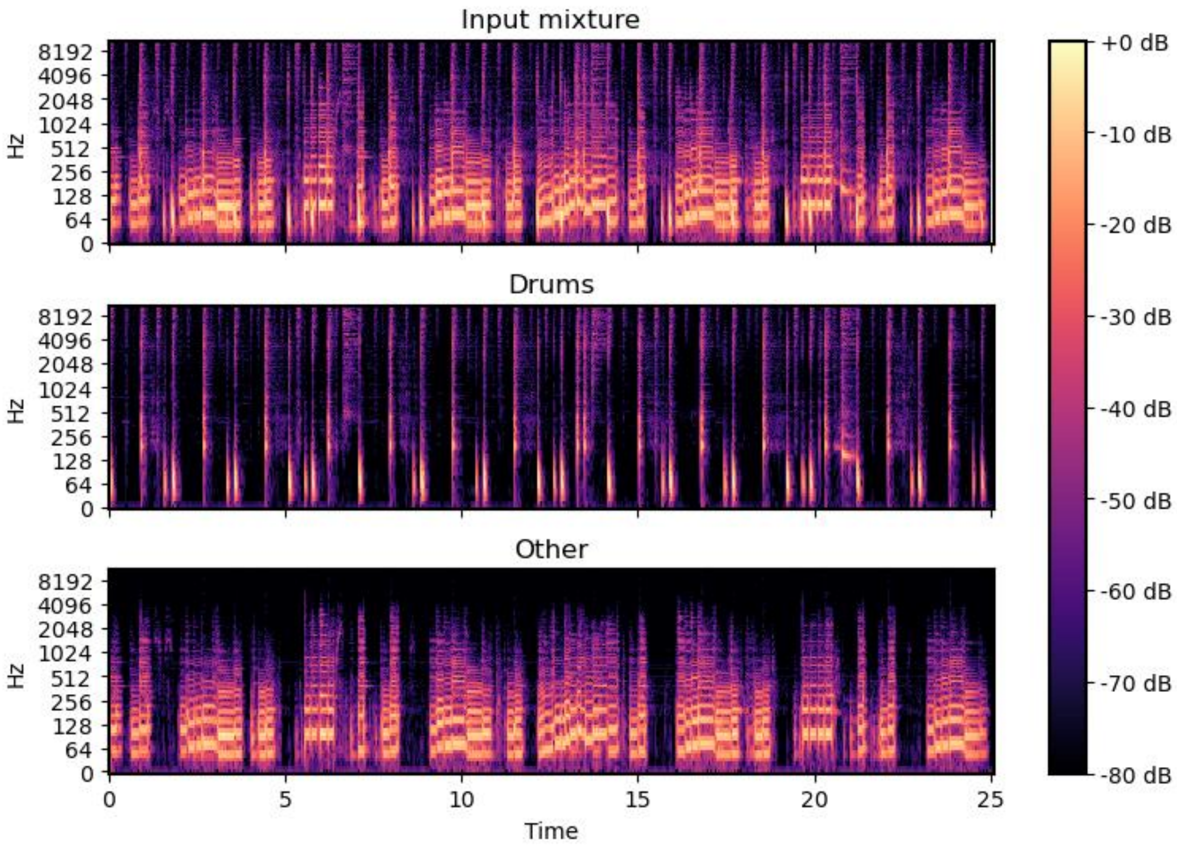
🔥 PA 3: Source Separation

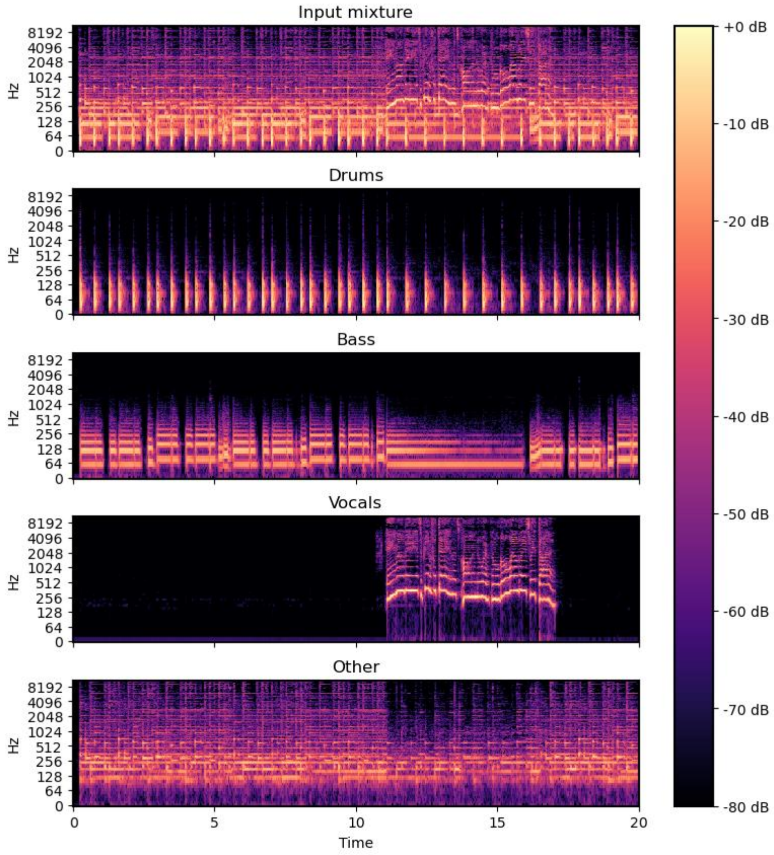
- Instructions will be sent by **emails** and released on the **course website**
- **Part 1**: Harmonic-Percussive Source Separation (HPSS) using **librosa**



🔥 PA 3: Source Separation

- **Part 2**: Music Source Separation using **Demucs**

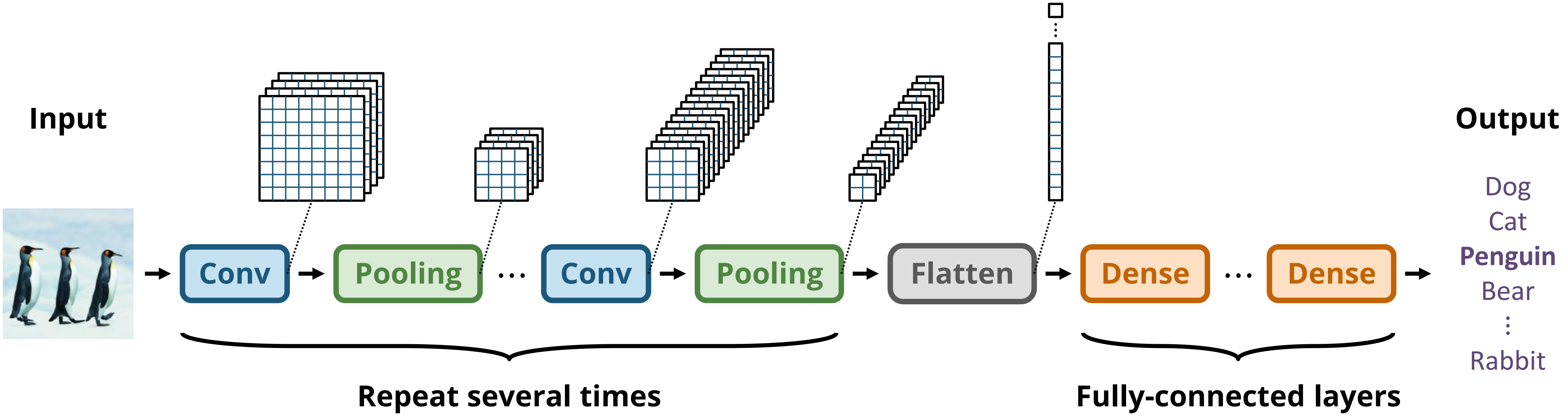




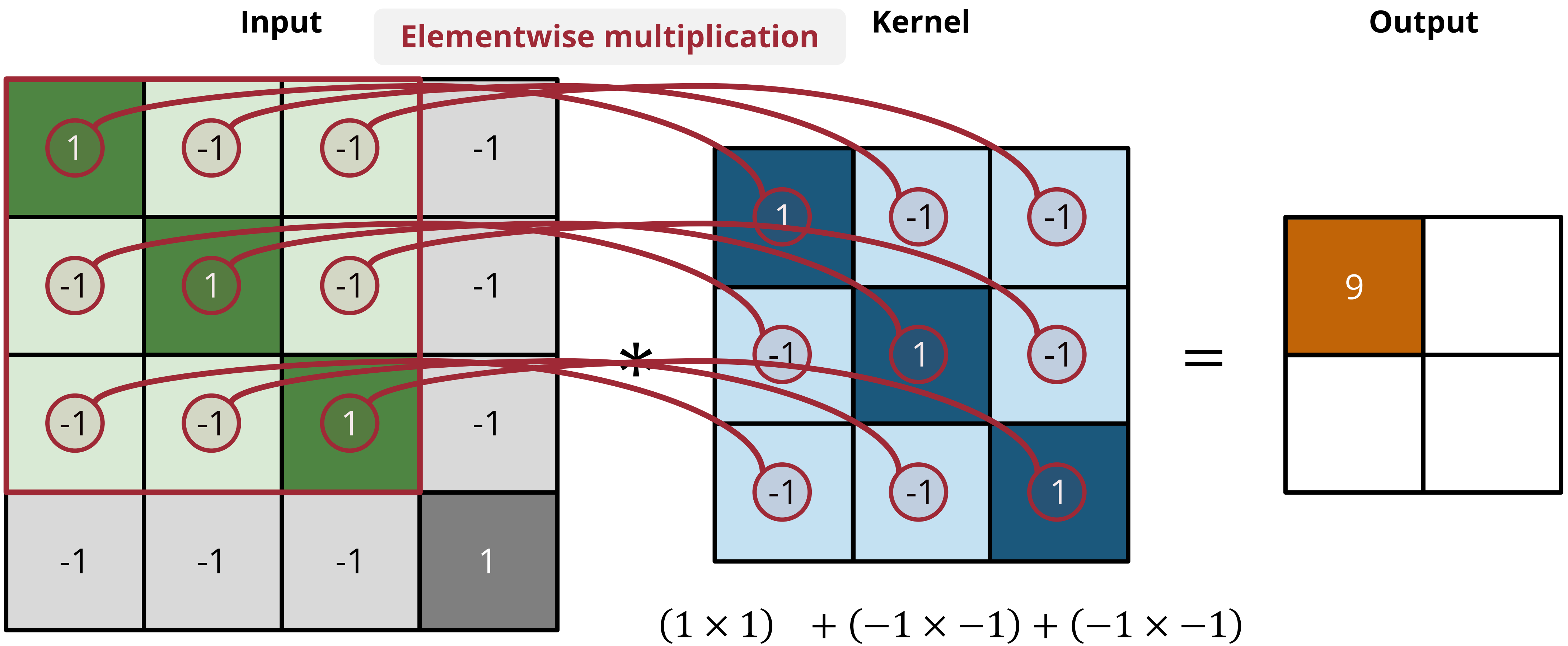
Recap

Reusable Pattern Detectors

Convolutional Neural Network (CNNs)



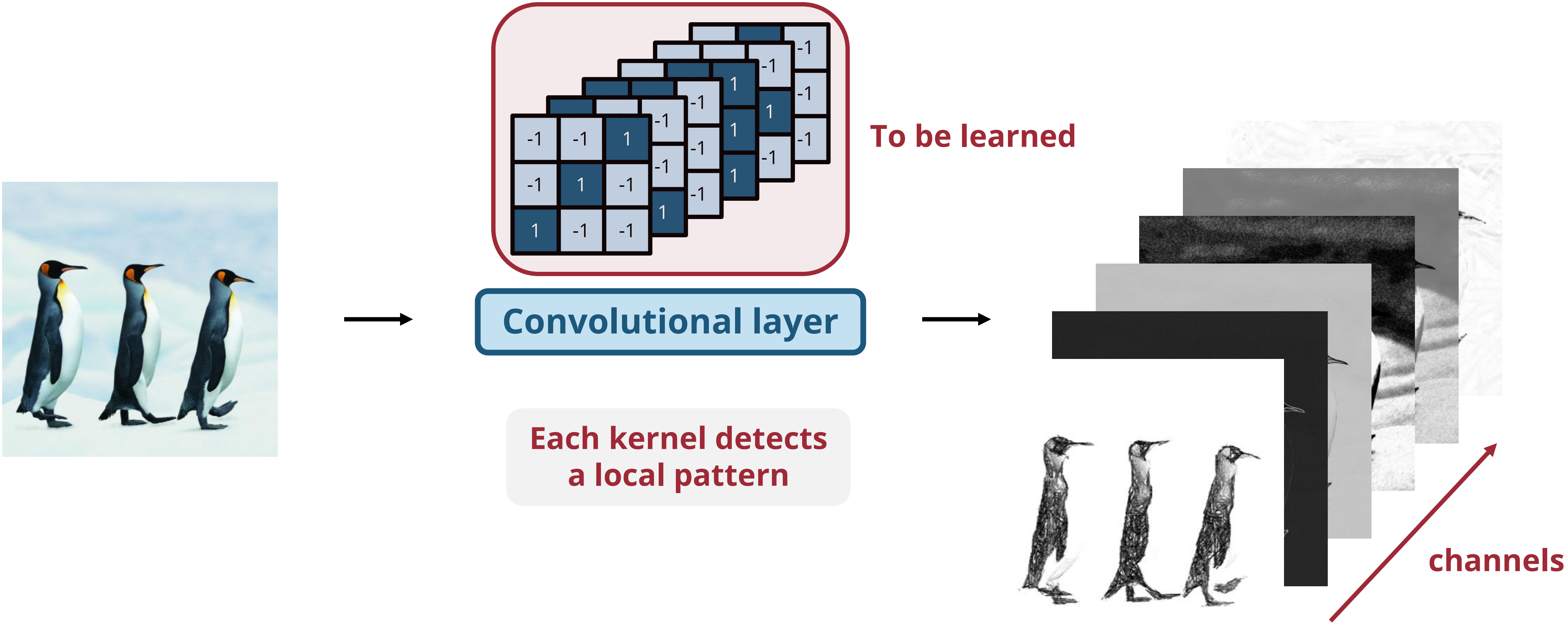
2D Convolution



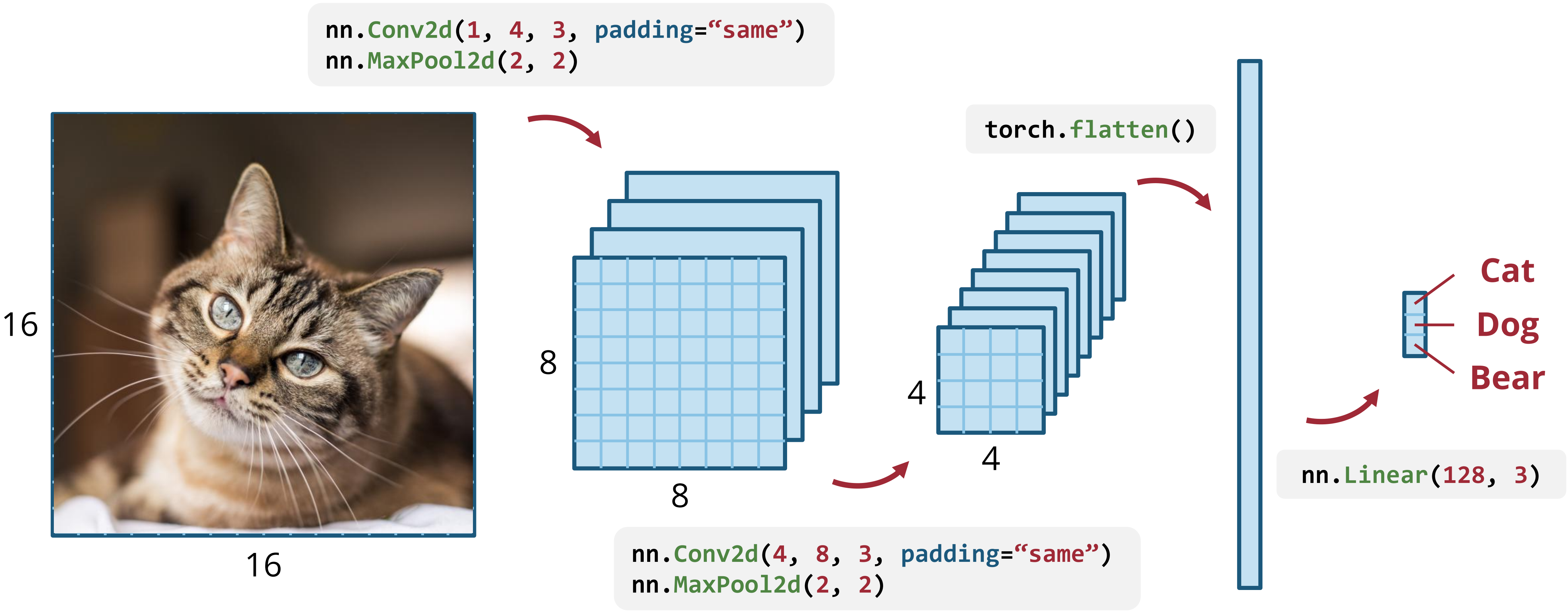
$$
\begin{aligned}
&(1\times 1) + (-1\times -1) + (-1\times -1)\\
&+(-1\times -1) + (1\times 1) + (-1\times -1) \quad = \mathbf{9}\\
&+(-1\times -1) + (-1\times -1) + (1\times 1)
\end{aligned}
$$

Convolutional Layer

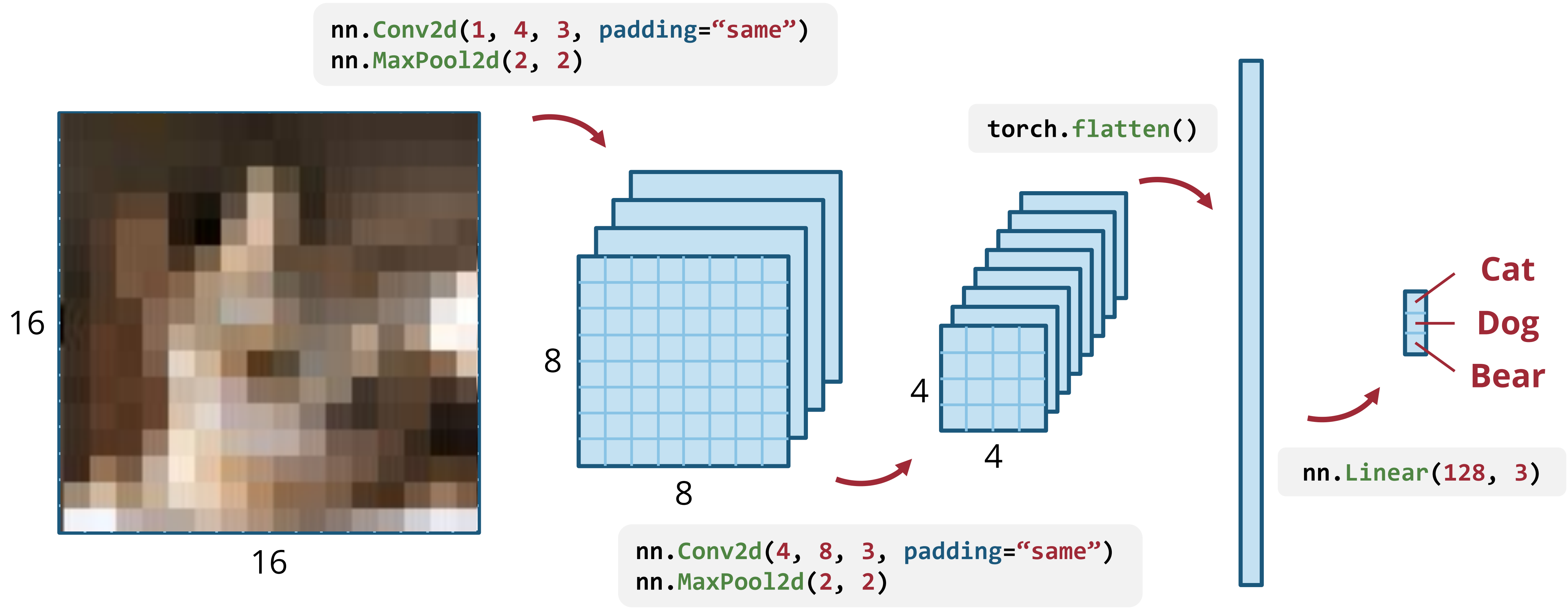
- A convolutional layer consists of many **learnable kernels** (channels)



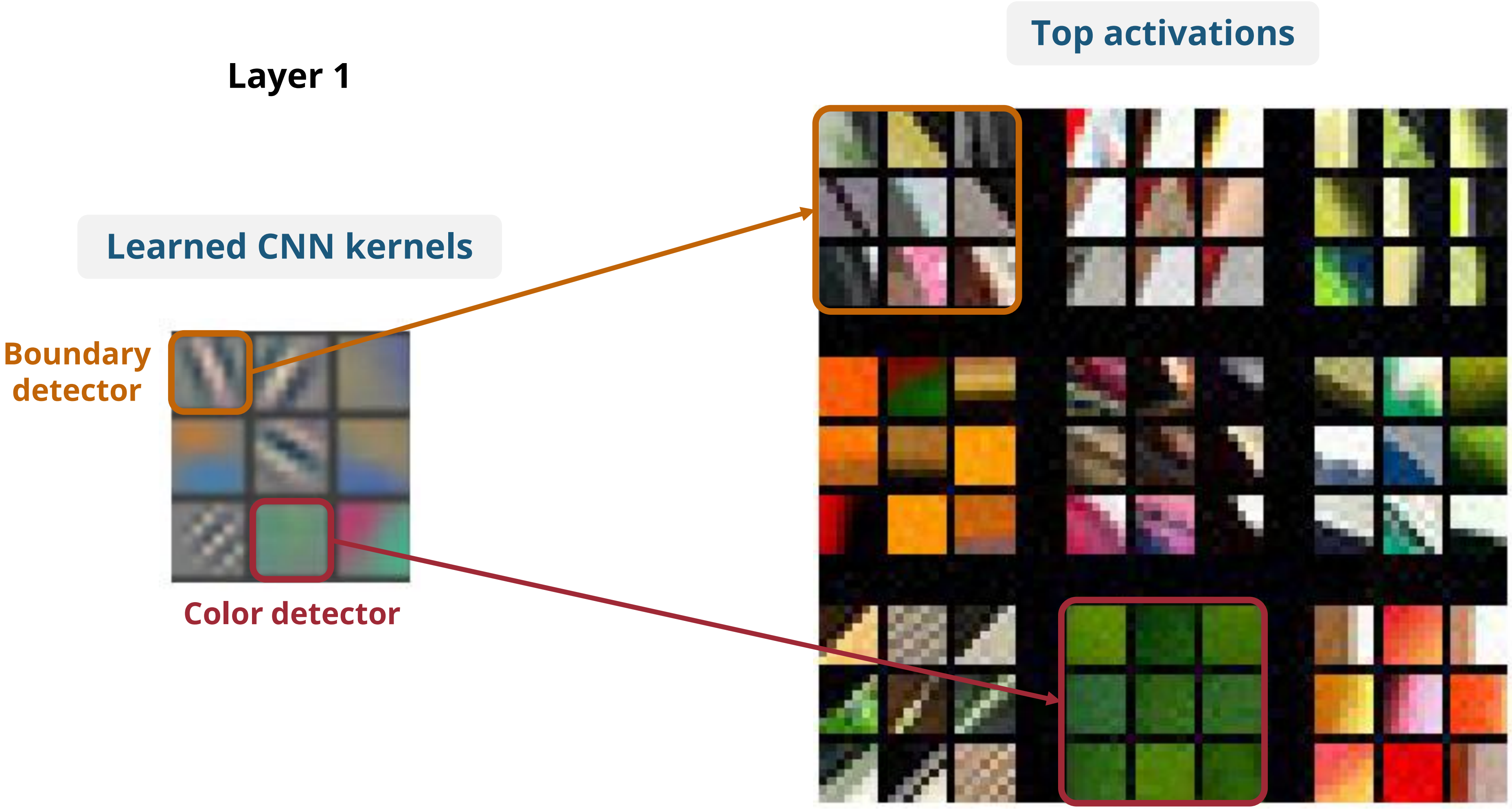
A Toy Example



A Toy Example

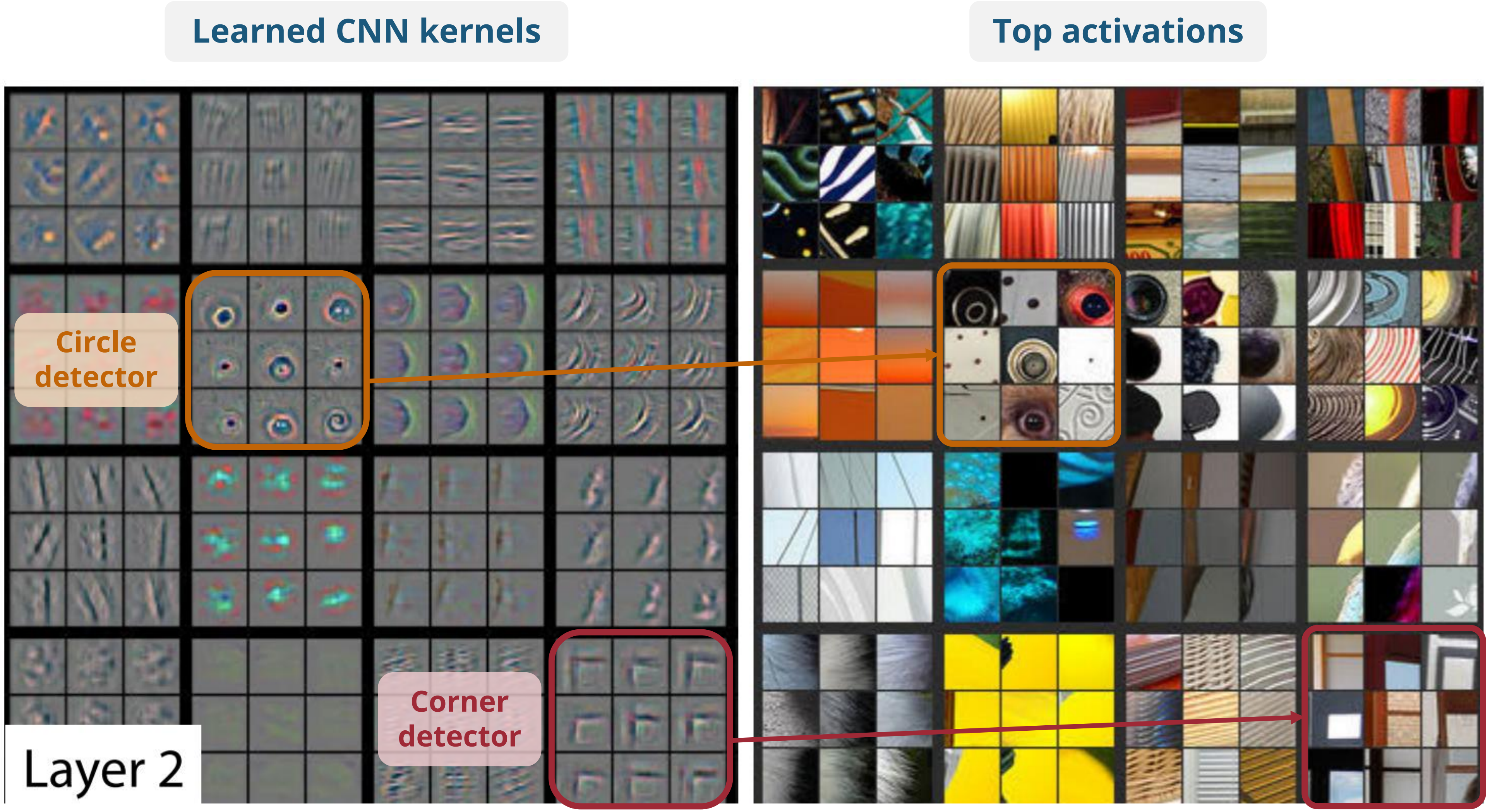


Learned CNN Kernels in a Trained AlexNet



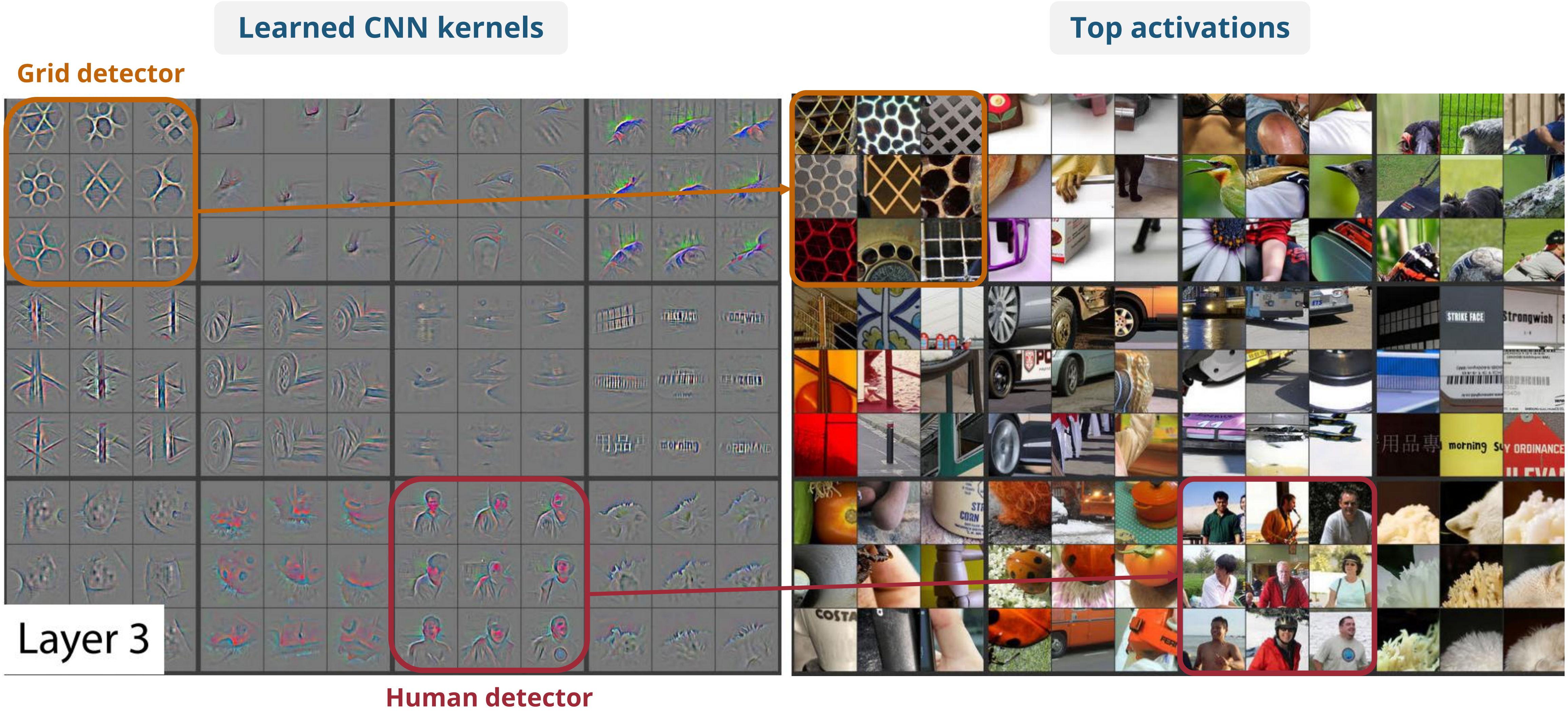
Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet



Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

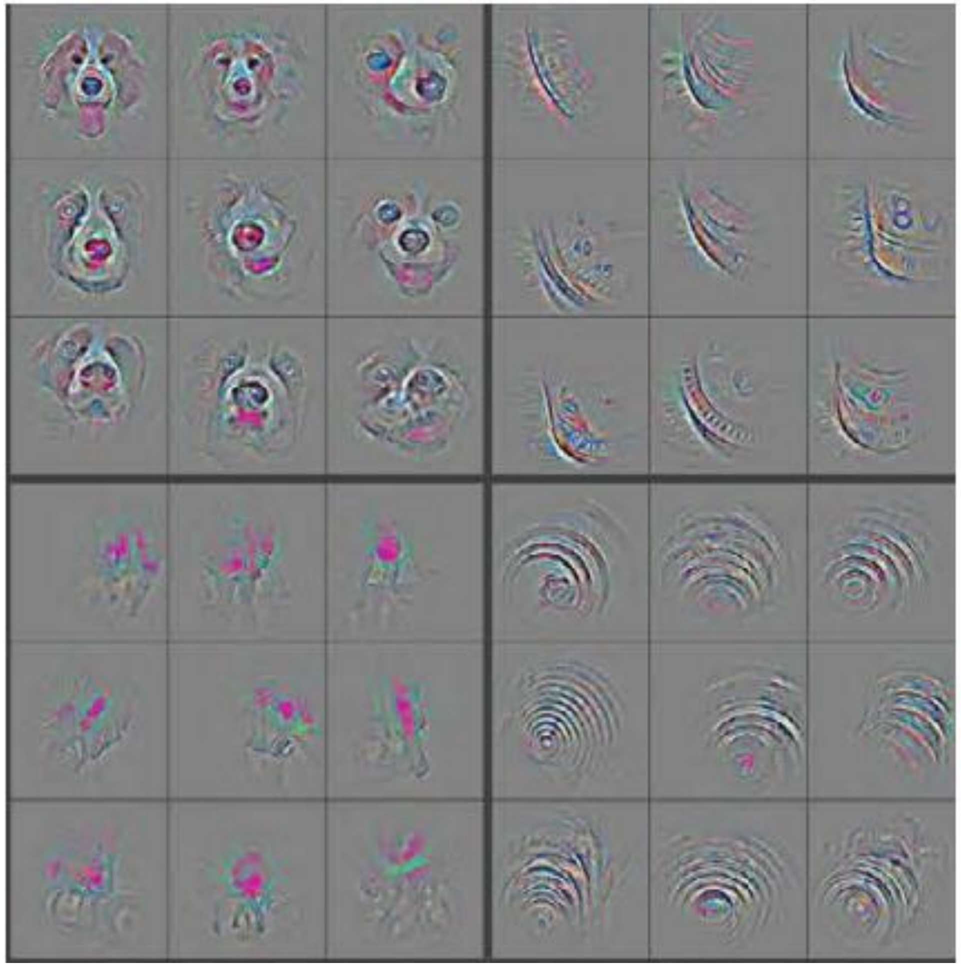
Learned CNN Kernels in a Trained AlexNet



Matthew D. Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks,” *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet

Learned CNN kernels

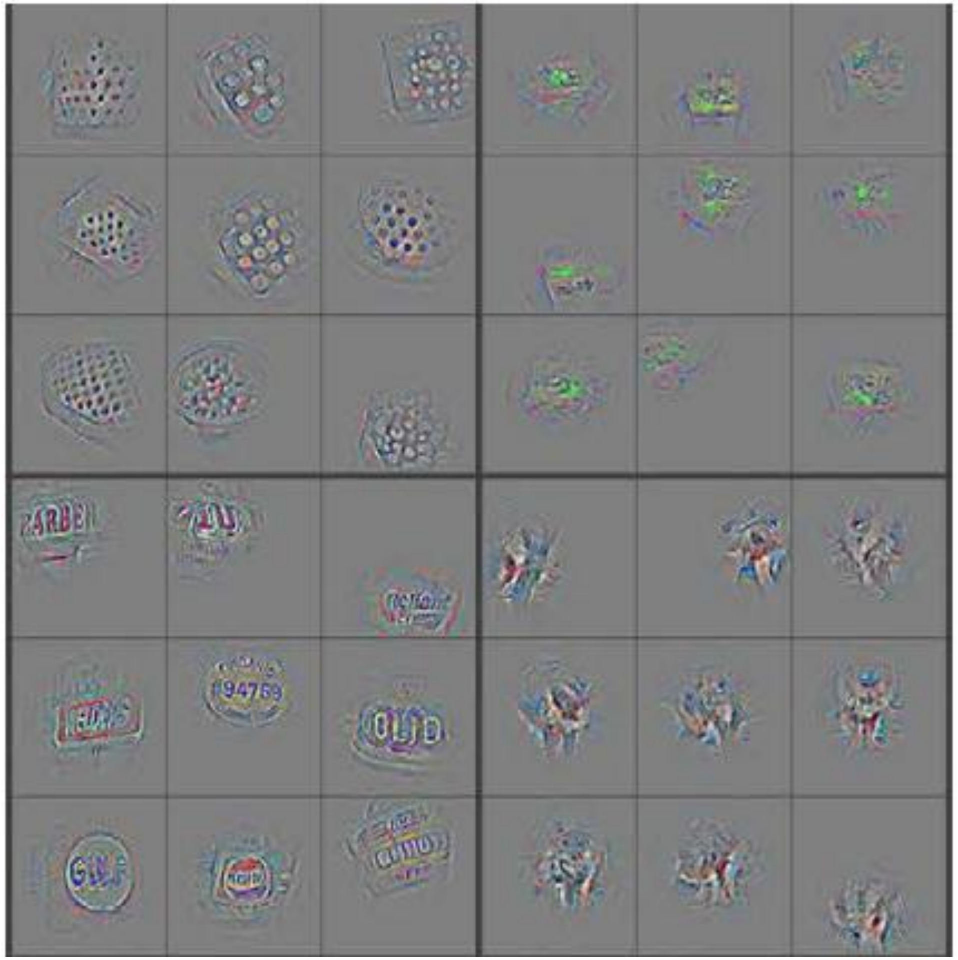


Top activations

Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet

Learned CNN kernels

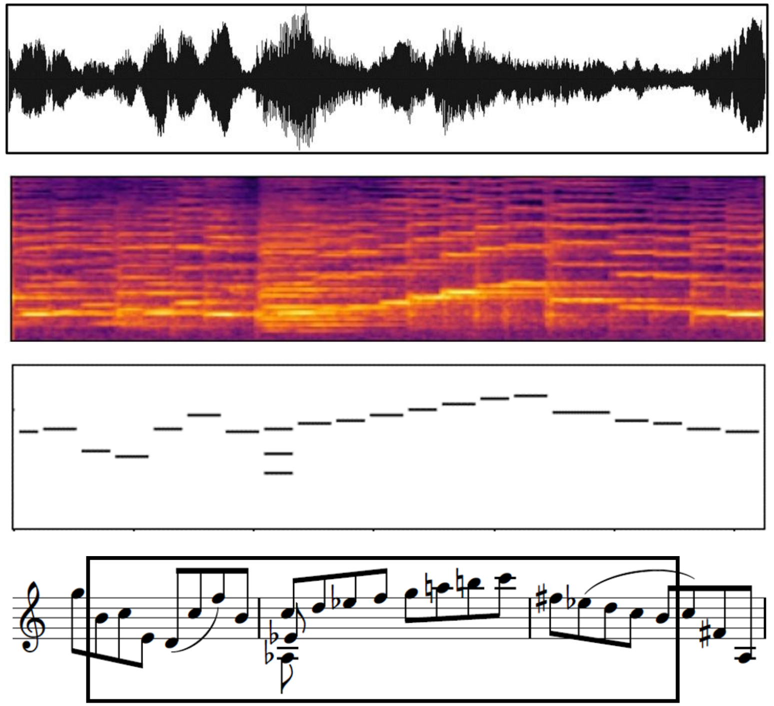


Top activations

Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Next Lecture

Music Analysis



(Source: Dong et al., 2022)