PAT 498/598 (Winter 2025)

Music & AI

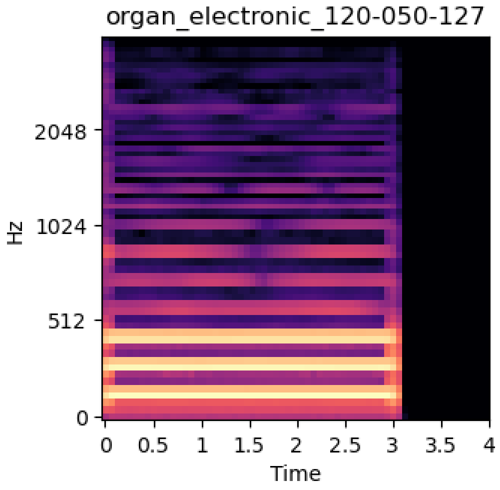
Lecture 10: Convolutional Neural Networks

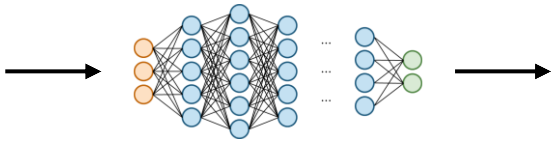
Instructor: Hao-Wen Dong

SCHOOL OF MUSIC, THEATRE & DANCE
PERFORMING ARTS TECHNOLOGY
UNIVERSITY OF MICHIGAN

Homework 3: Musical Note Classification using CNNs

- Train a CNN that can classify audio files into their **instrument families**
  - **Input**: 64x64 mel spectrogram
  - **Output**: 11 instrument classes
  - Using the **NSynth** dataset (Engel et al., 2017)



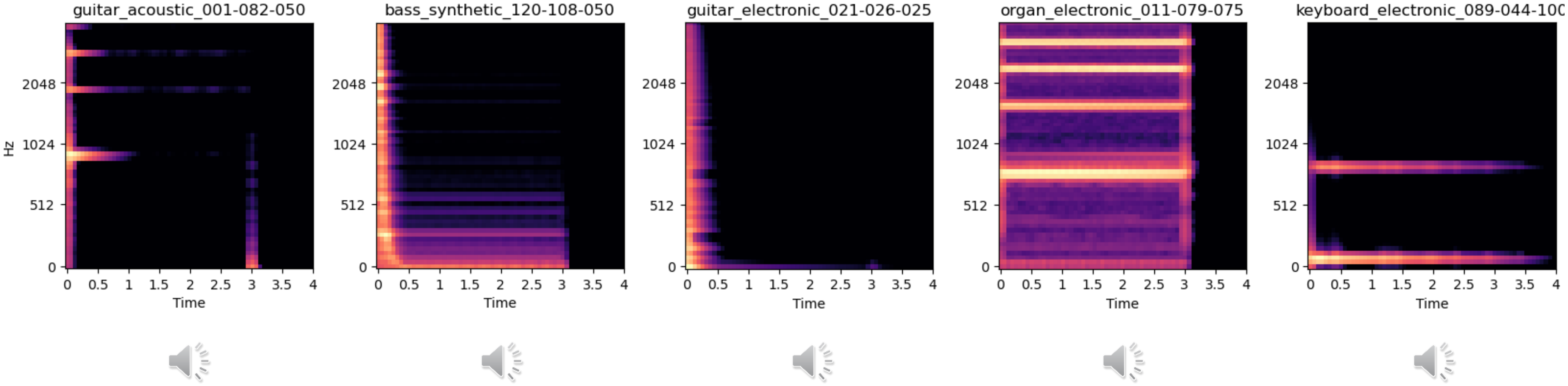


Bass
Brass
Flute
Guitar
Keyboard
Mallet
Organ
Reed
String
Synth Lead
Vocal

Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi, "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," *ICML*, 2017.

NSynth Dataset

- A collection of 305,979 **single-shot musical notes** (Engel et al., 2017)
  - Produced from 1,006 **commercial sample libraries**
  - With different **MIDI pitches** (21–108) and **velocities** (25, 50, 75, 100, 127)



Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi, "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," *ICML*, 2017.

Homework 3: Musical Note Classification using CNNs

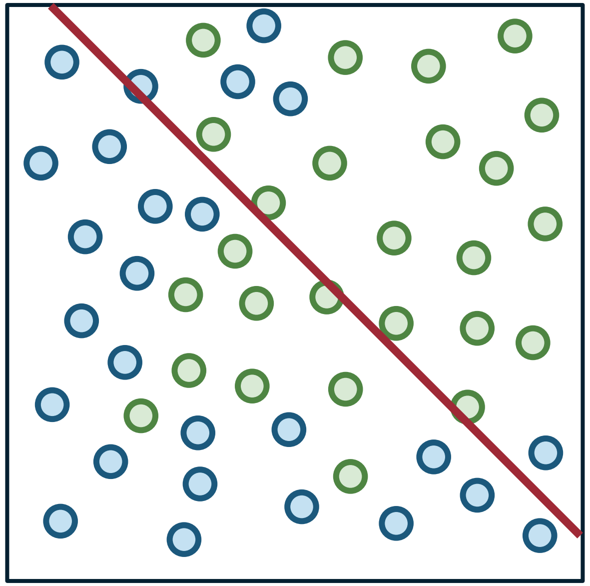
- Instructions will be released on Gradescope
- Due at **11:59pm ET** on **February 17**
- Late submissions: **1 point deducted per day**

(Recap) In-distribution vs Out-of-distribution

- **Key**: Make the training distribution closer to the target distribution
- First, we need to **define our target distribution**
- Then, we can try to
  - Collect a **diverse** dataset covering that covers different parts of the target distribution
  - Apply **data augmentation** to fill the gaps in the distribution

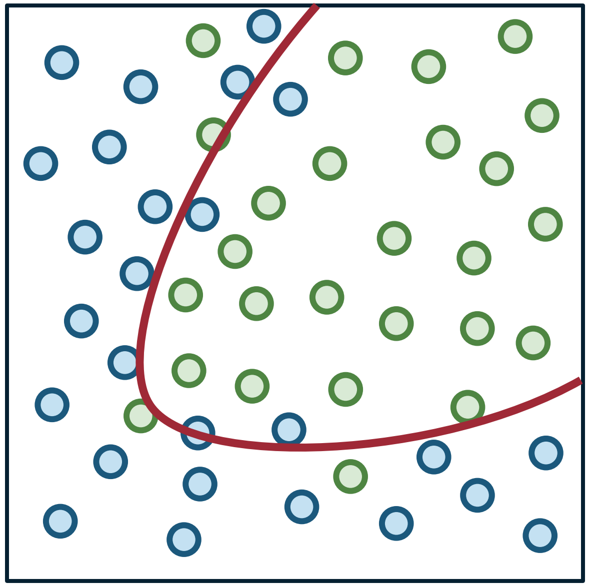
(Recap) Overfitting & Underfitting

Underfitting

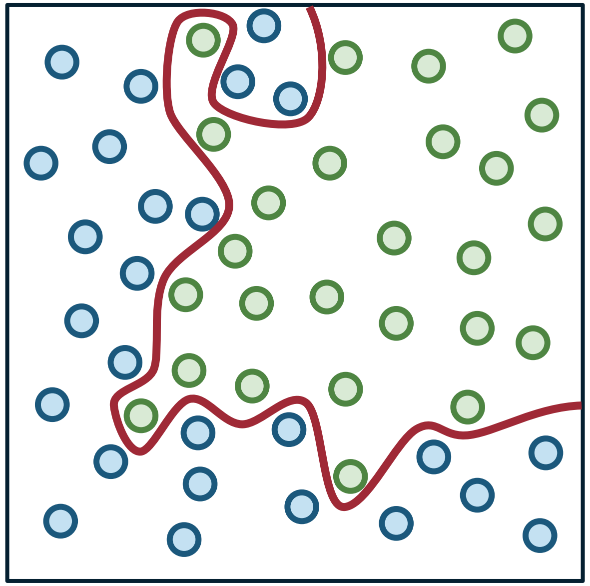


Model too inexpressive

Good fit!



Overfitting

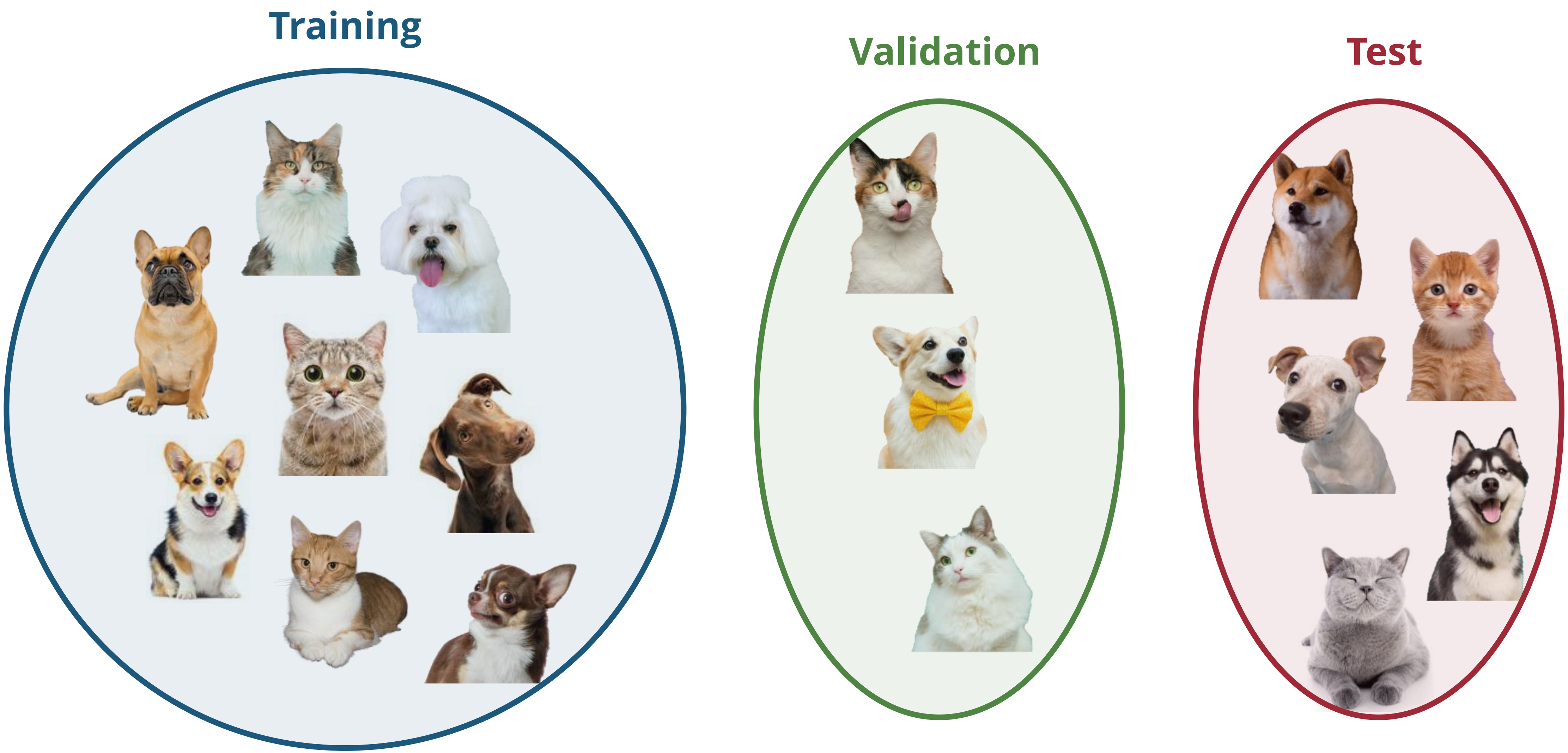


Model too expressive

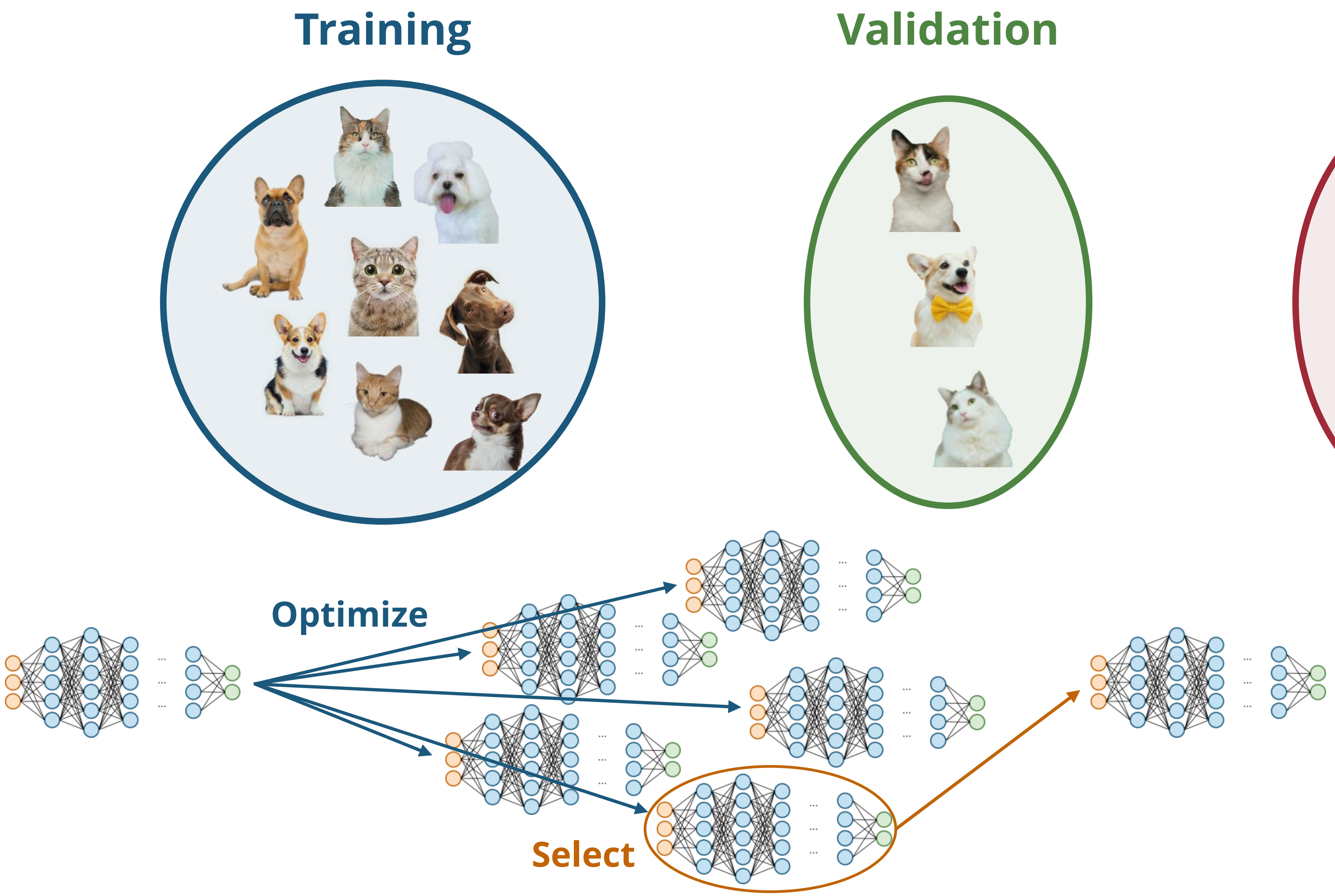
(Recap) Train–Validation–Test Split

(Recap) Train–Validation–Test Split

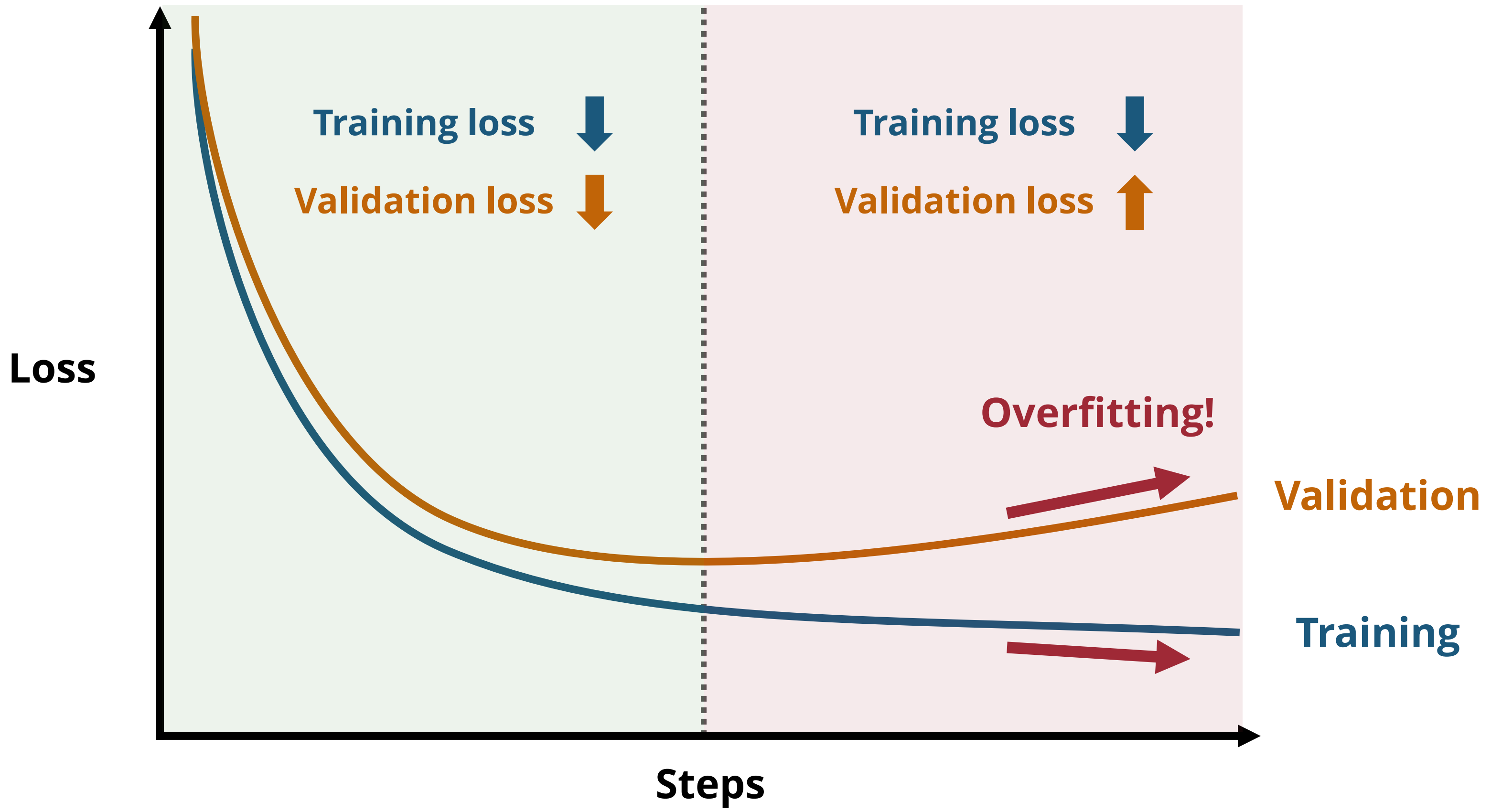
(Recap) Train–Validation–Test Split



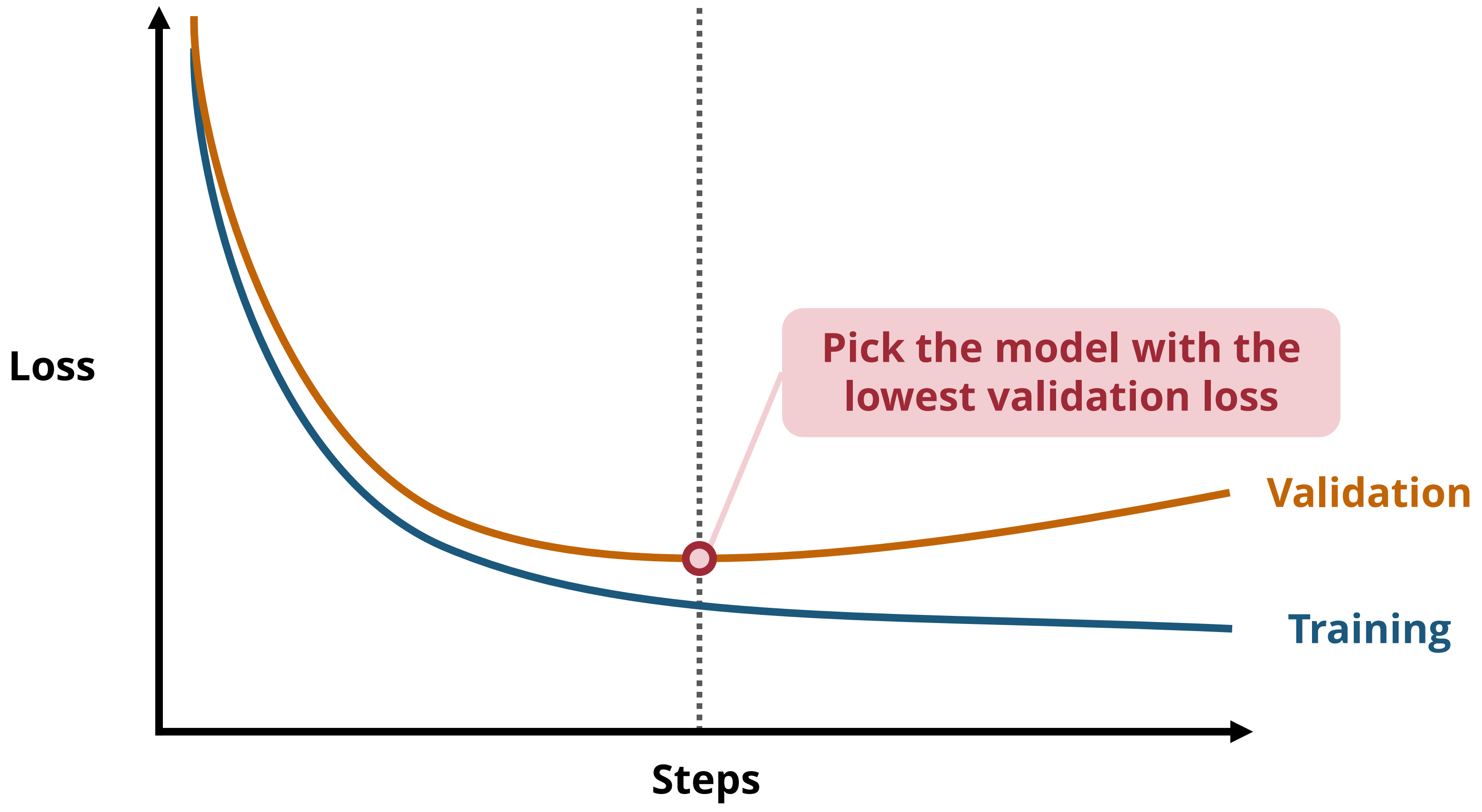
(Recap) Training–Validation–Test Pipeline



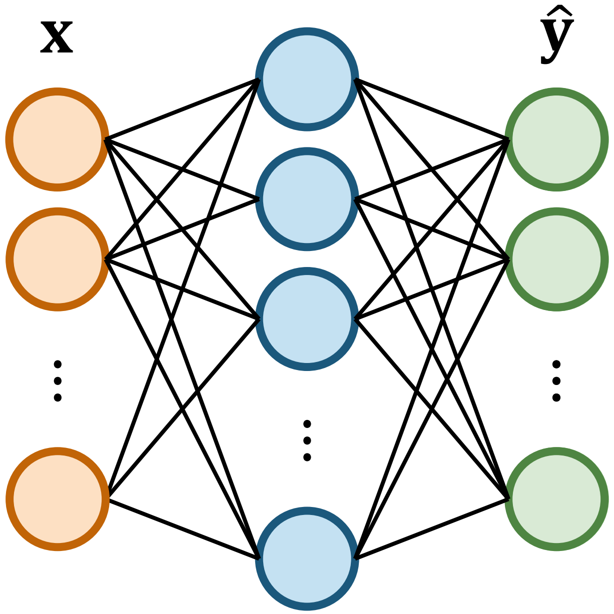
(Recap) Training vs Validation Losses



(Recap) Training vs Validation Losses

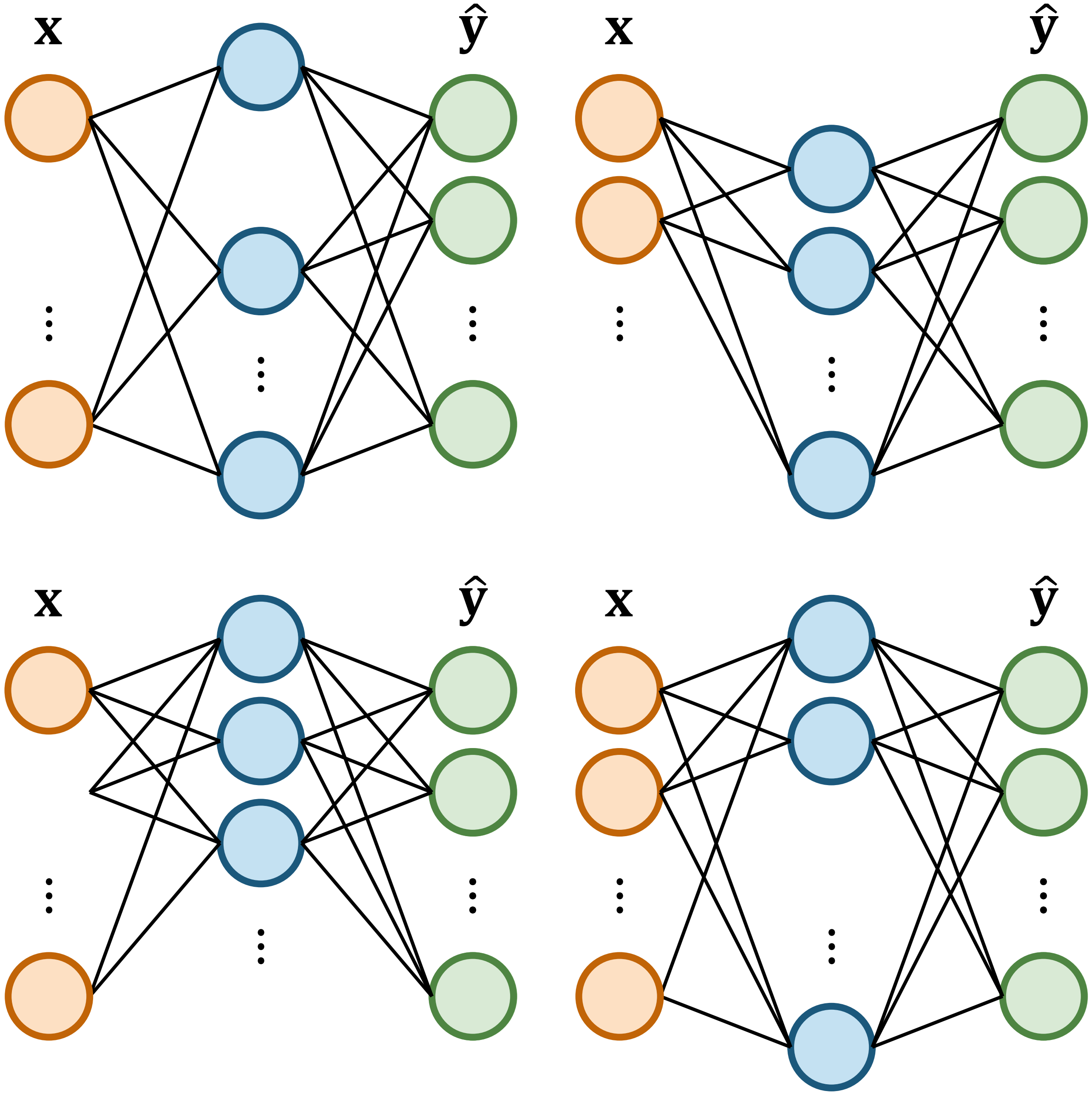


(Recap) Dropout

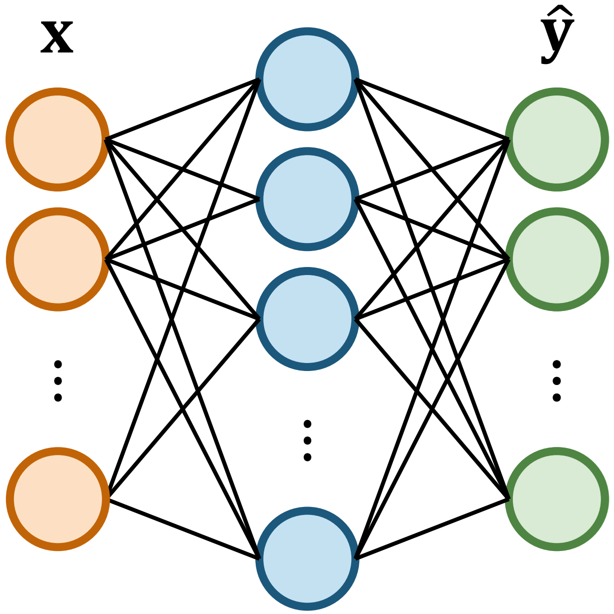


Each neuron may be removed with probability p during training

Dropout rate

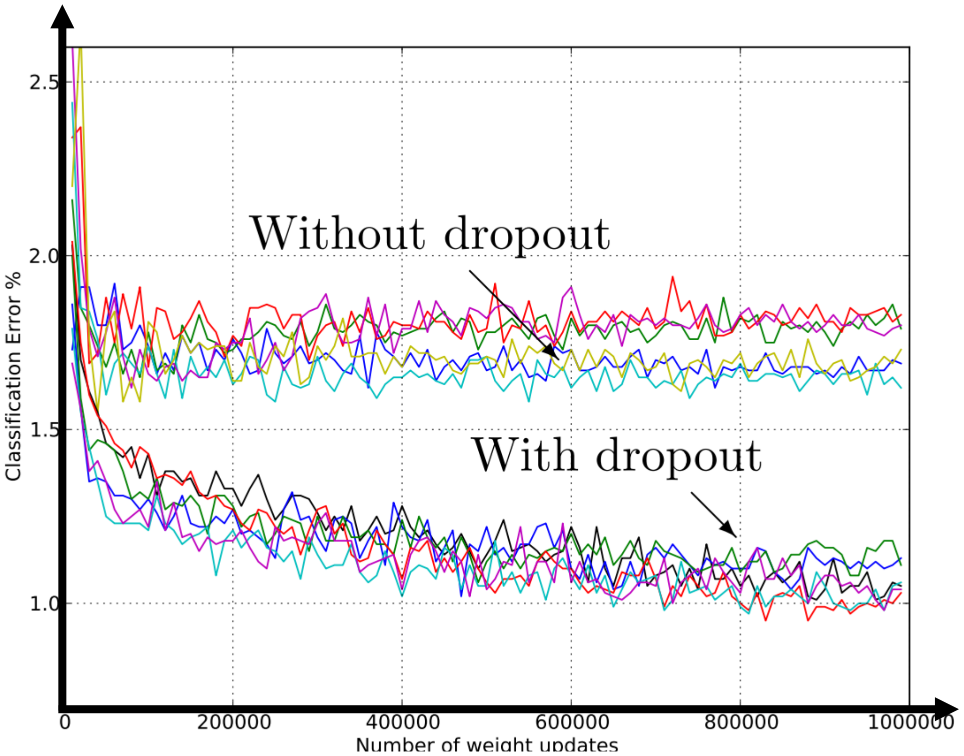


(Recap) Dropout



Each neuron may be removed with probability p during training

Test error rate



Weight updates

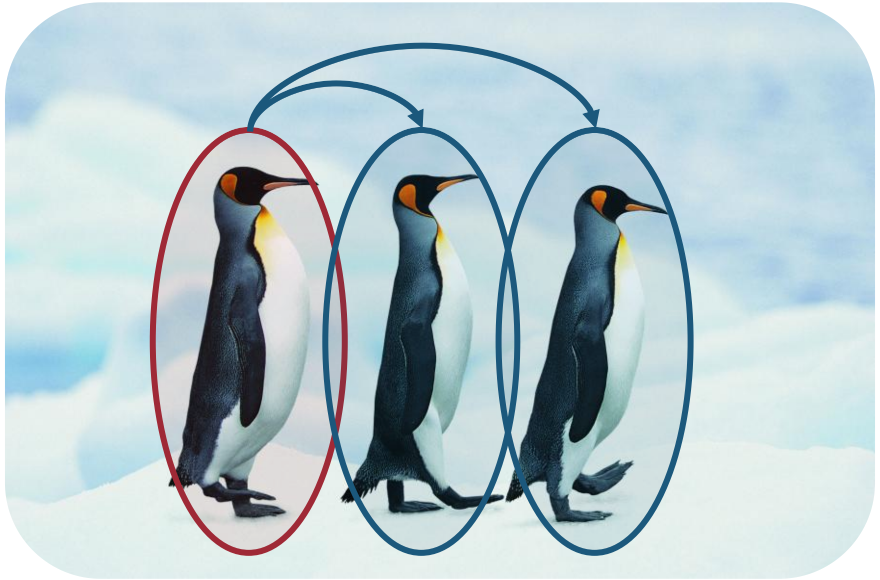
Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *JMLR*, 2014.

Convolutional Neural Networks (CNNs)

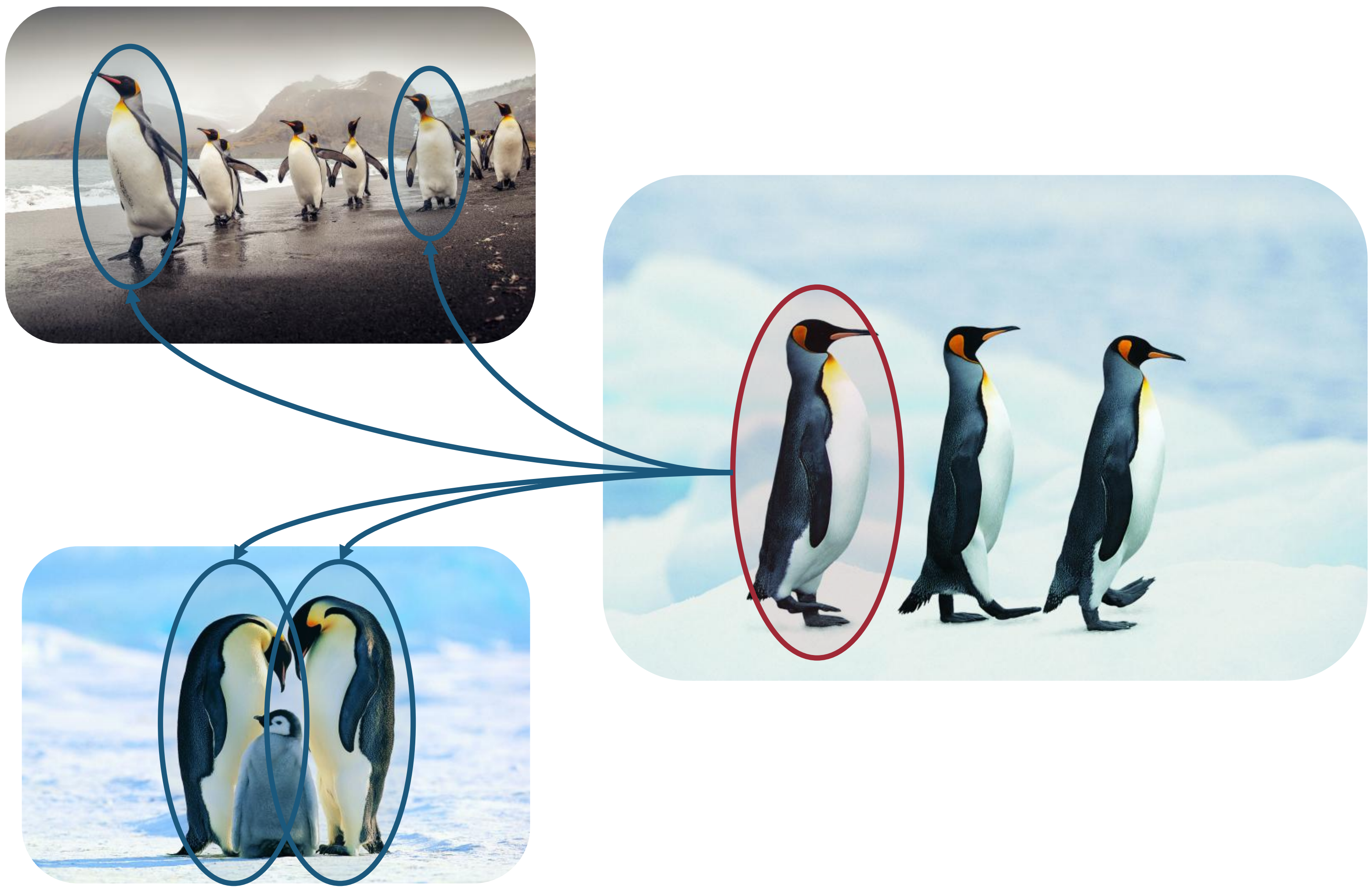
Convolutional Neural Networks (CNNs)

- **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**

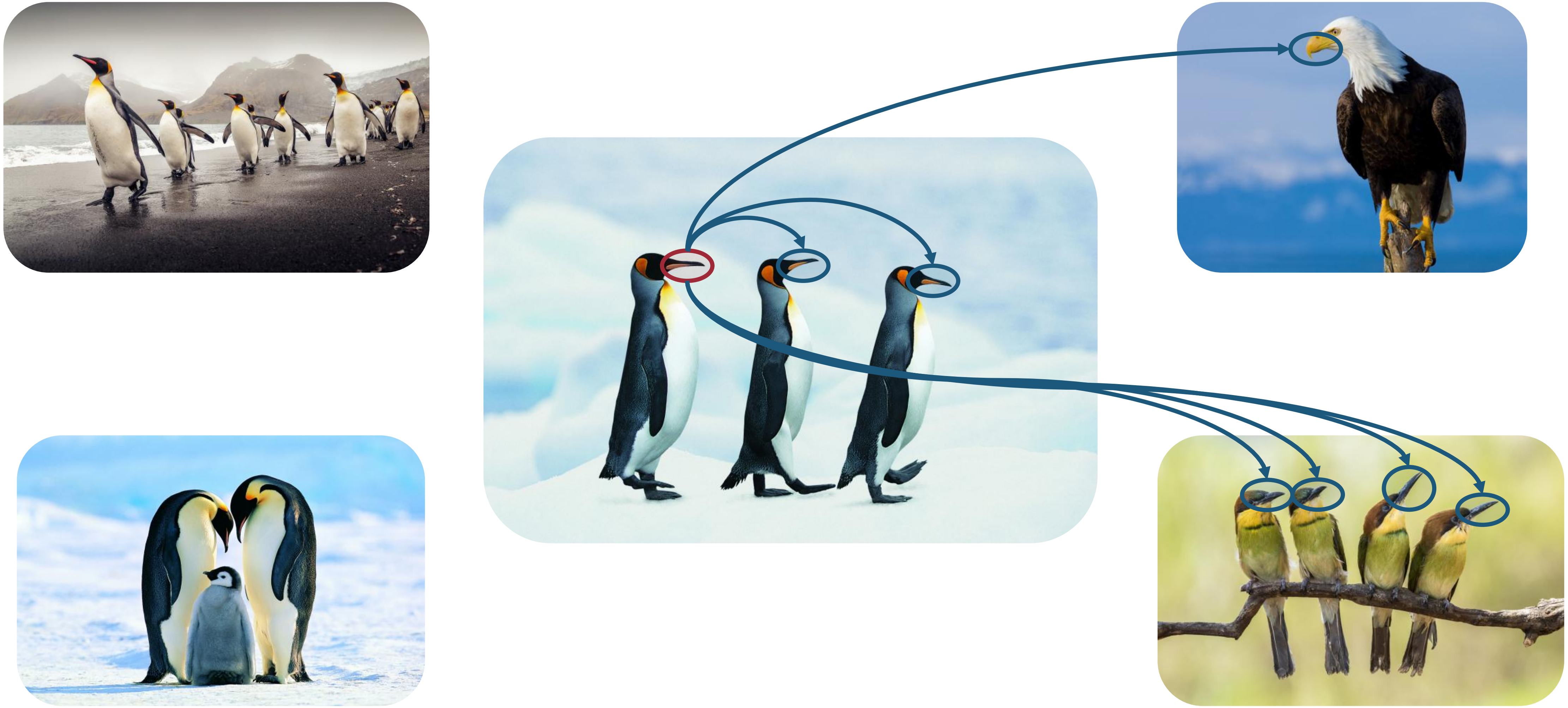
Reusable Pattern Detectors



Reusable Pattern Detectors

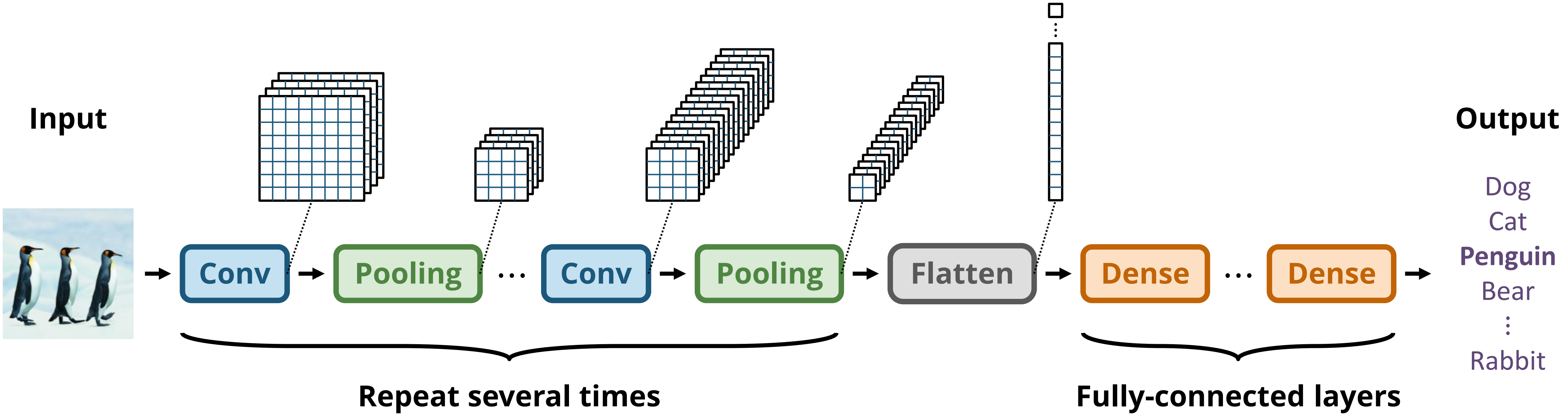


Reusable Pattern Detectors



Reusable Pattern Detectors

Convolutional Neural Network (CNNs)

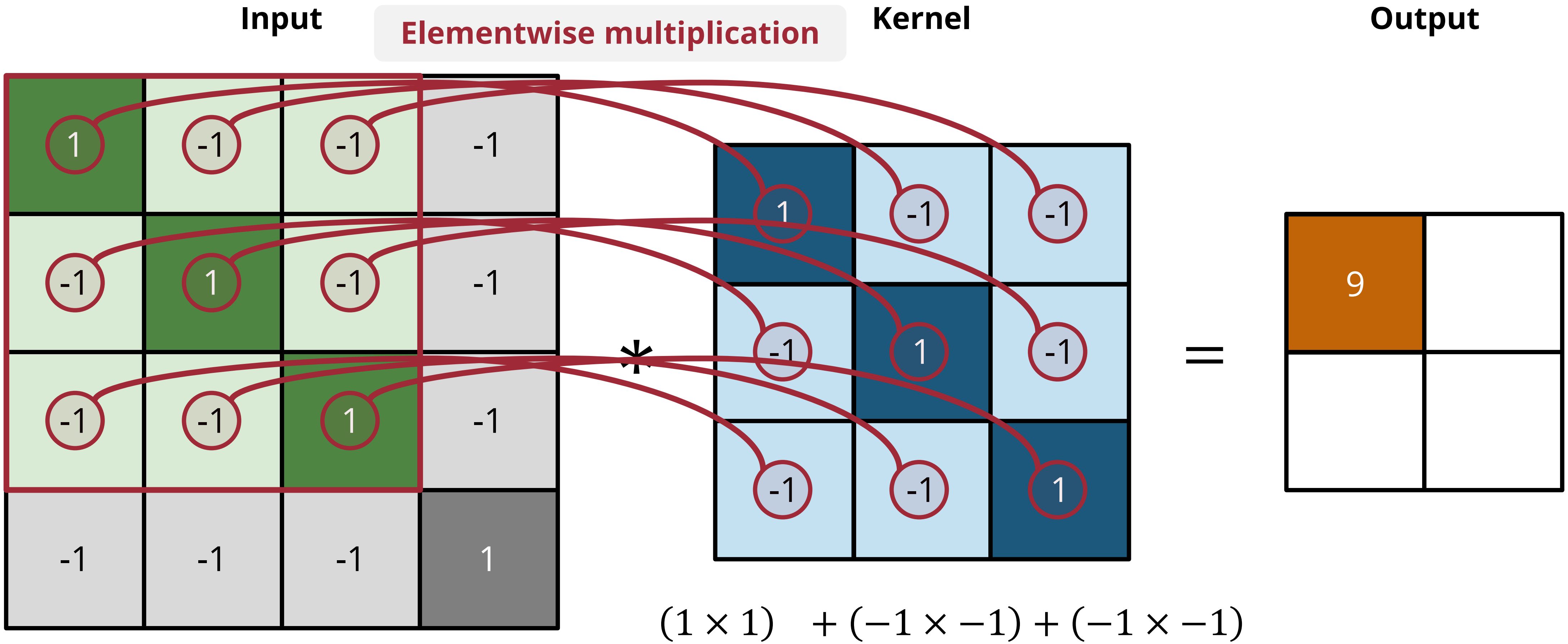


2D Convolution

Input / **Kernel** / **Output**

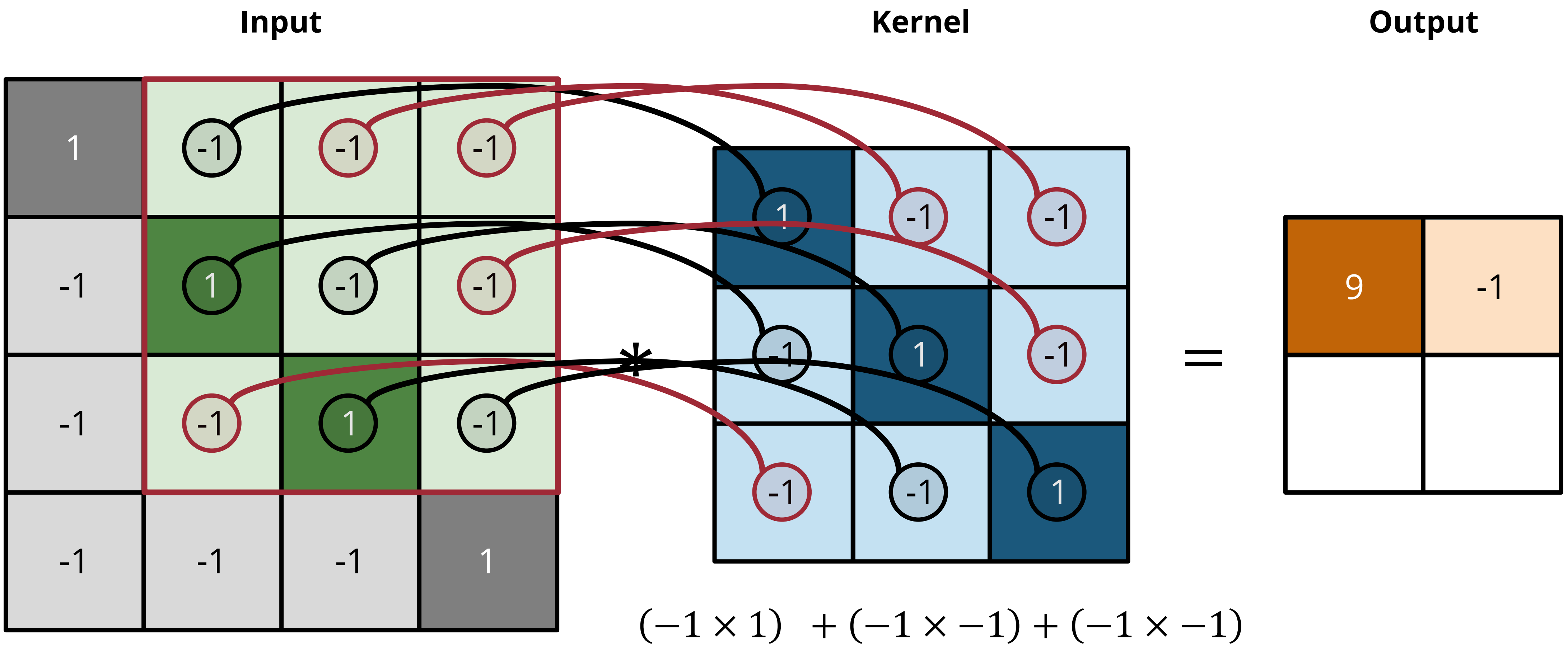
$$
\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}
*
\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}
=
\begin{bmatrix} & \\ & \end{bmatrix}
$$

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



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

2D Convolution

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

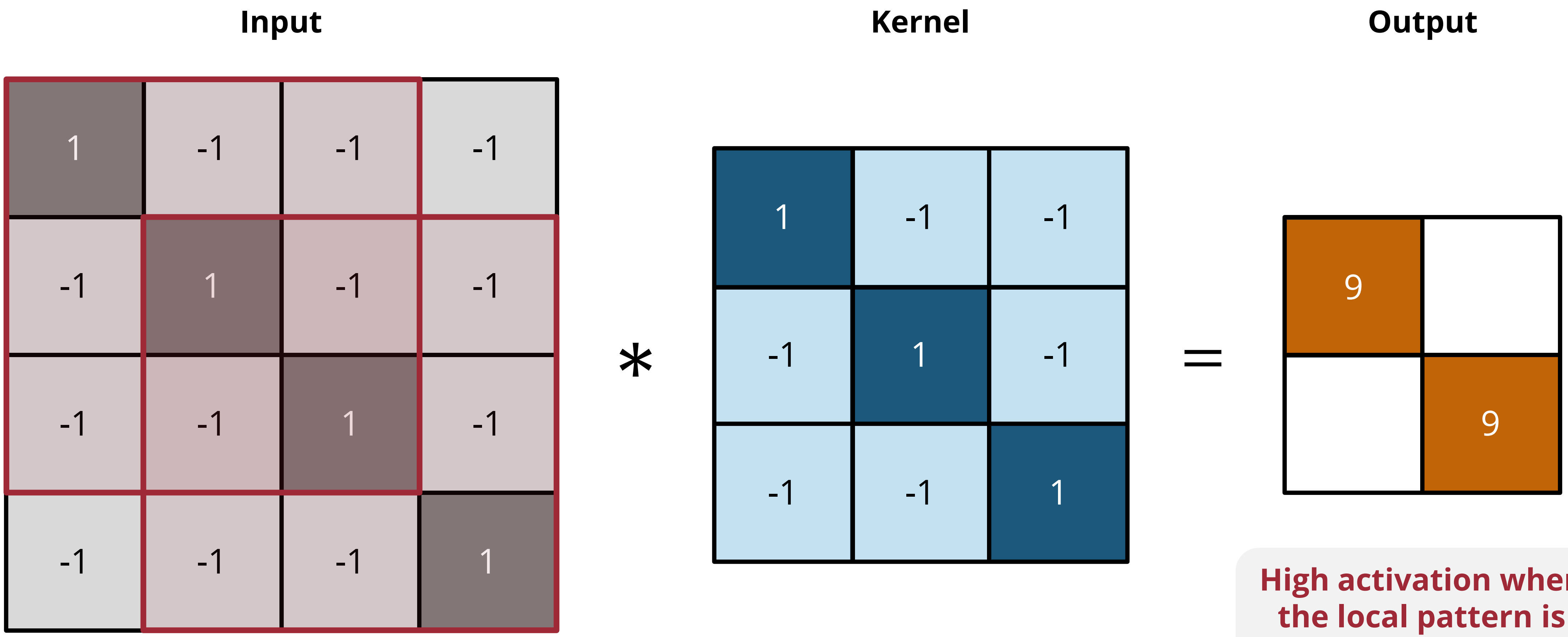
$$
\begin{aligned}
&(-1 \times 1) + (1 \times -1) + (-1 \times -1) \\
&+(-1 \times -1) + (-1 \times 1) + (1 \times -1) \quad = \mathbf{-1} \\
&+(-1 \times -1) + (-1 \times -1) + (-1 \times 1)
\end{aligned}
$$

2D Convolution

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

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

2D Convolution



High activation when the local pattern is close to the kernel

2D Convolution

Input **Kernel** **Output**

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

Low activation when the local pattern differs from the kernel

2D Convolution

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

2D Convolution

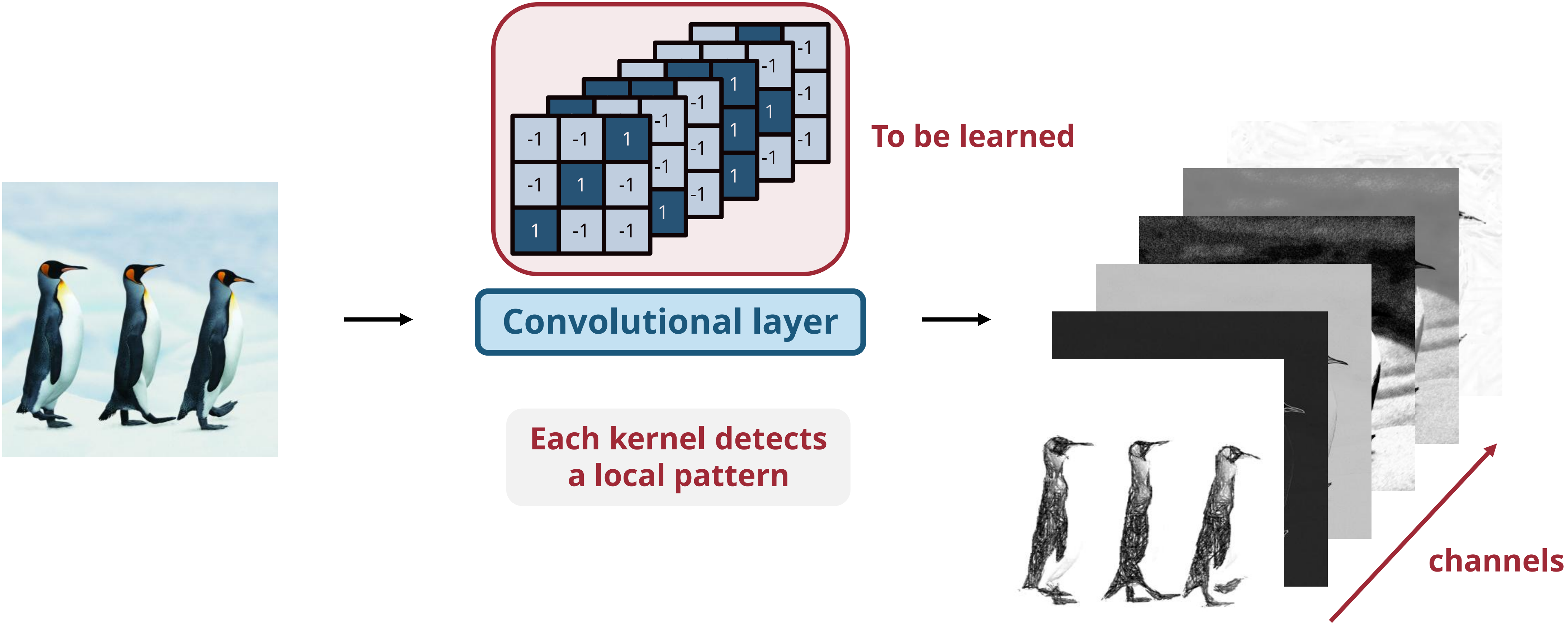
$$
\underbrace{\begin{bmatrix} -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}}_{\textbf{Input}} * \underbrace{\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}}_{\textbf{Kernel}} = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 5 \end{bmatrix}}_{\textbf{Output}}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

2D Convolution

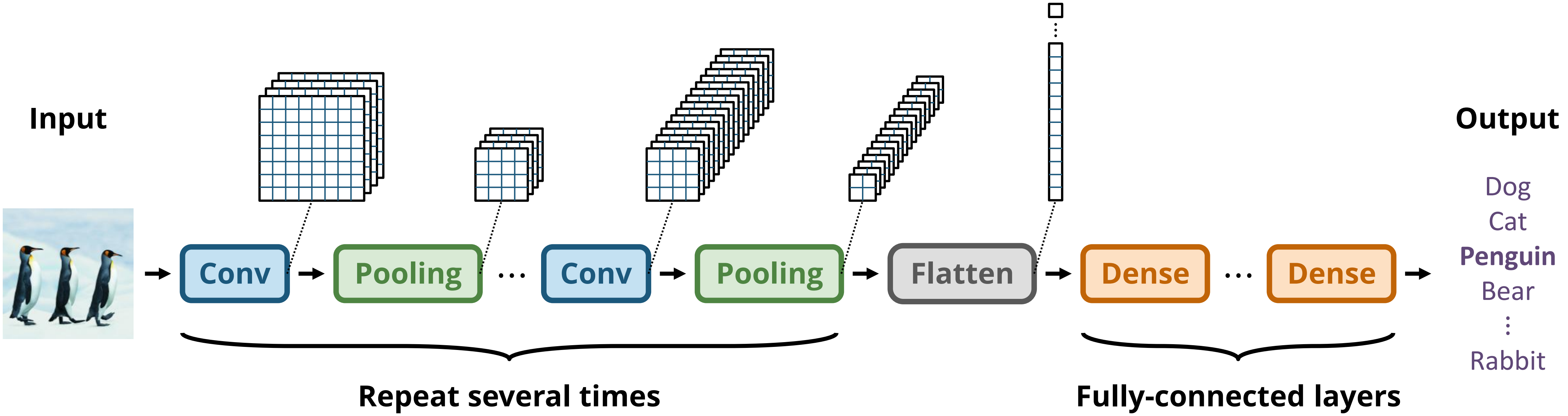
$$
\overset{\textbf{Input}}{\begin{bmatrix} -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}}
*
\overset{\textbf{Kernel}}{\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}}
=
\overset{\textbf{Output}}{\begin{bmatrix} 9 & -1 \\ -1 & 1 \end{bmatrix}}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

Convolutional Layer

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



Convolutional Neural Network (CNNs)



Padding

`padding="valid"`

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

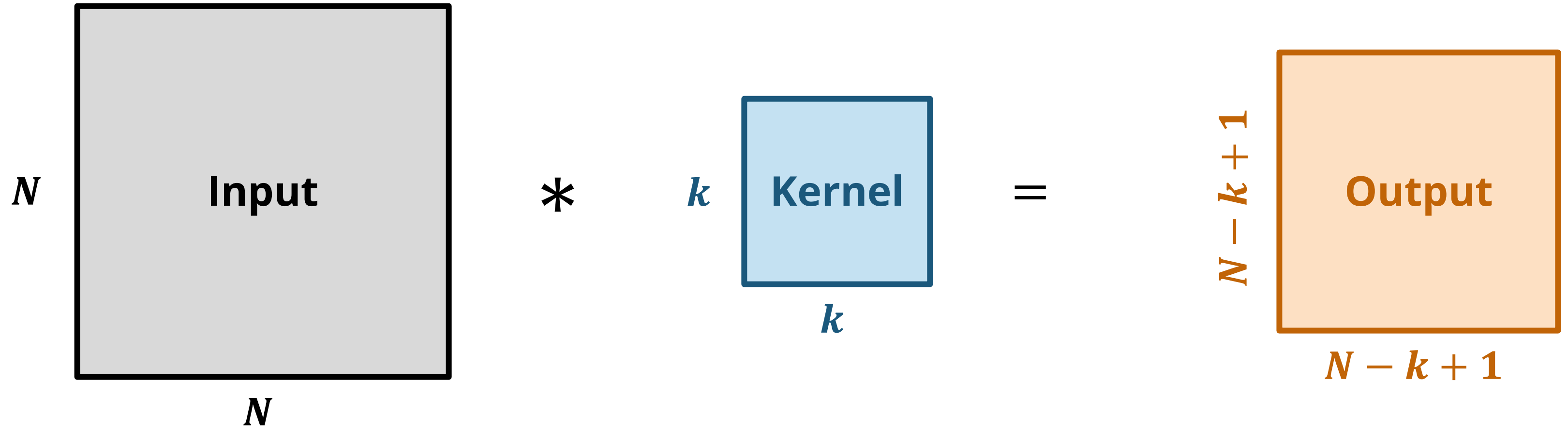
`padding="same"`

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

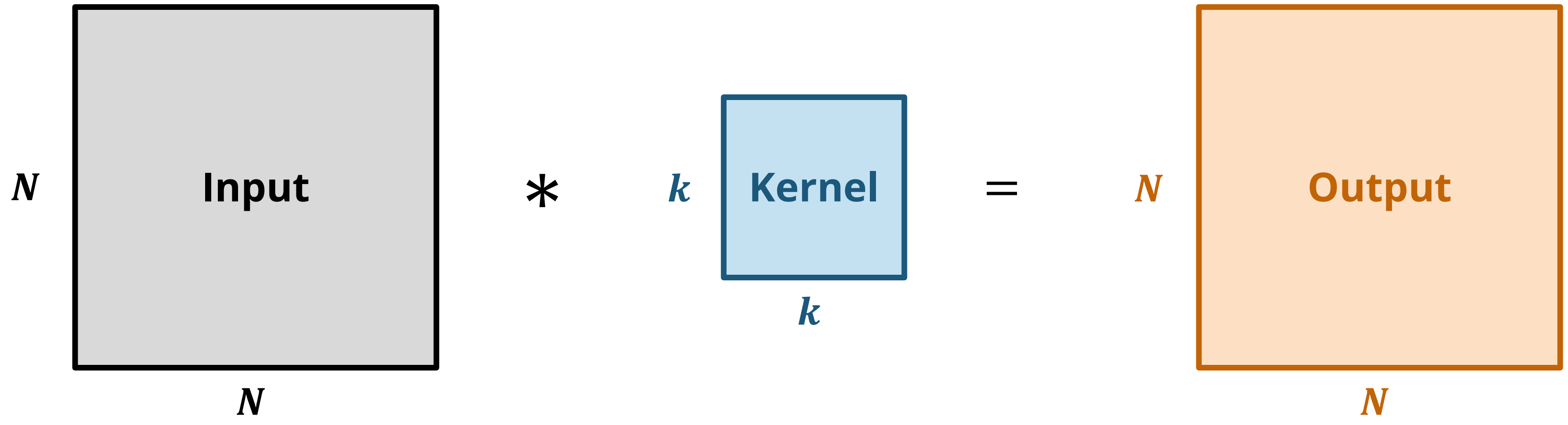
Keep the output of the same size as the input

Shapes

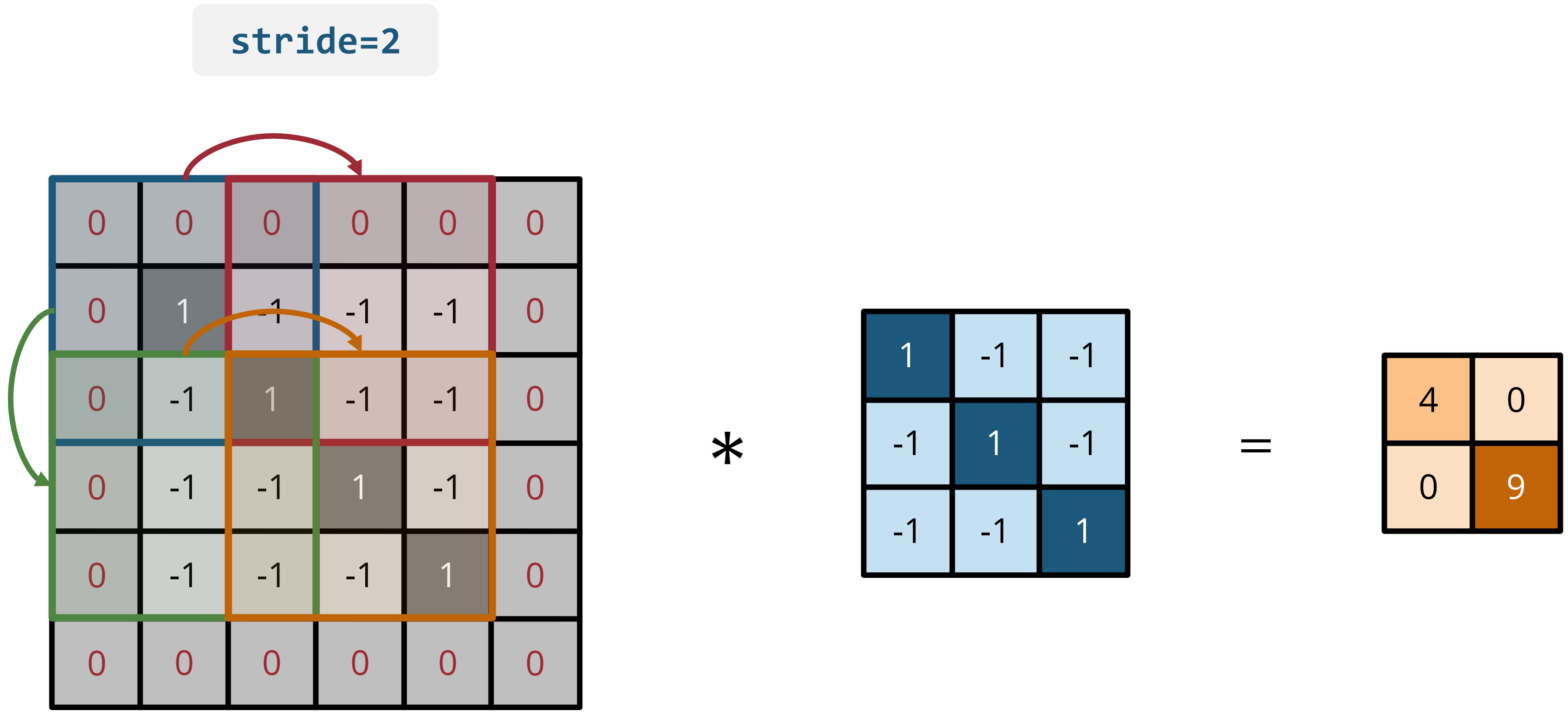
`padding="valid"`



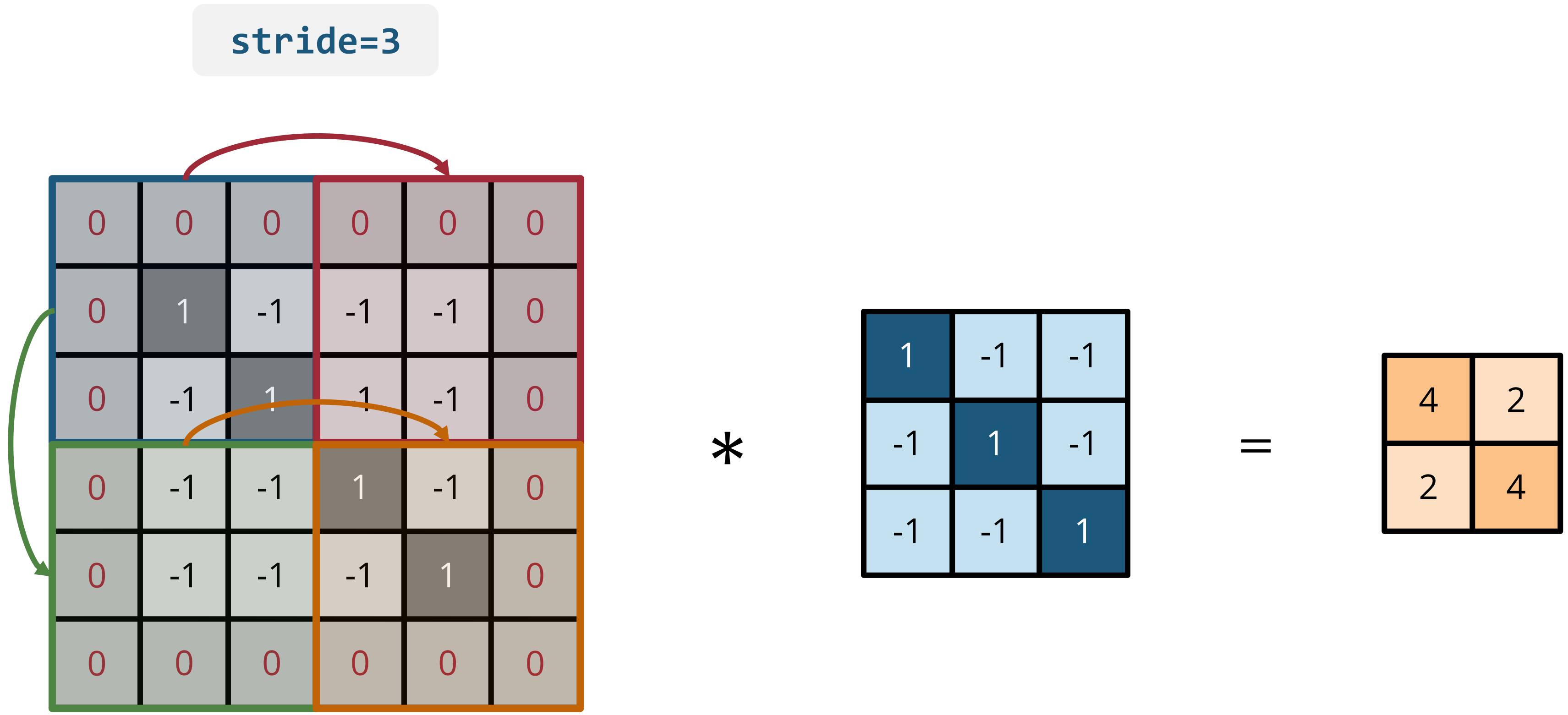
`padding="same"`



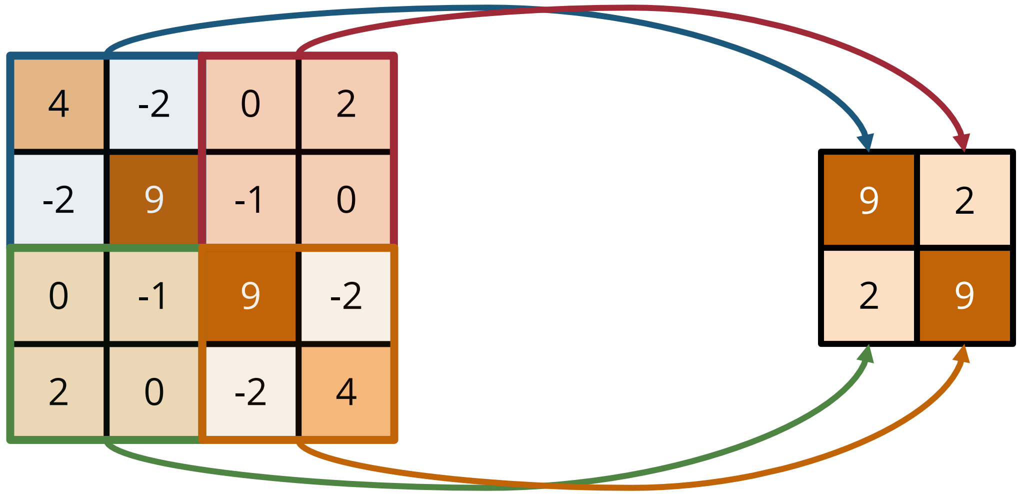
Striding


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

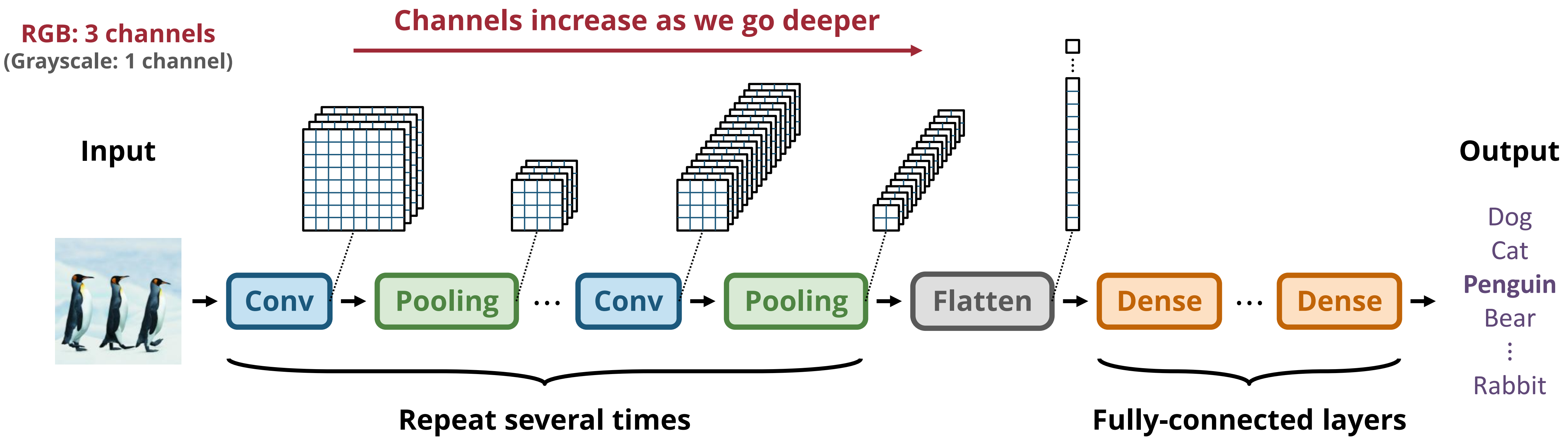


Max Pooling Layer



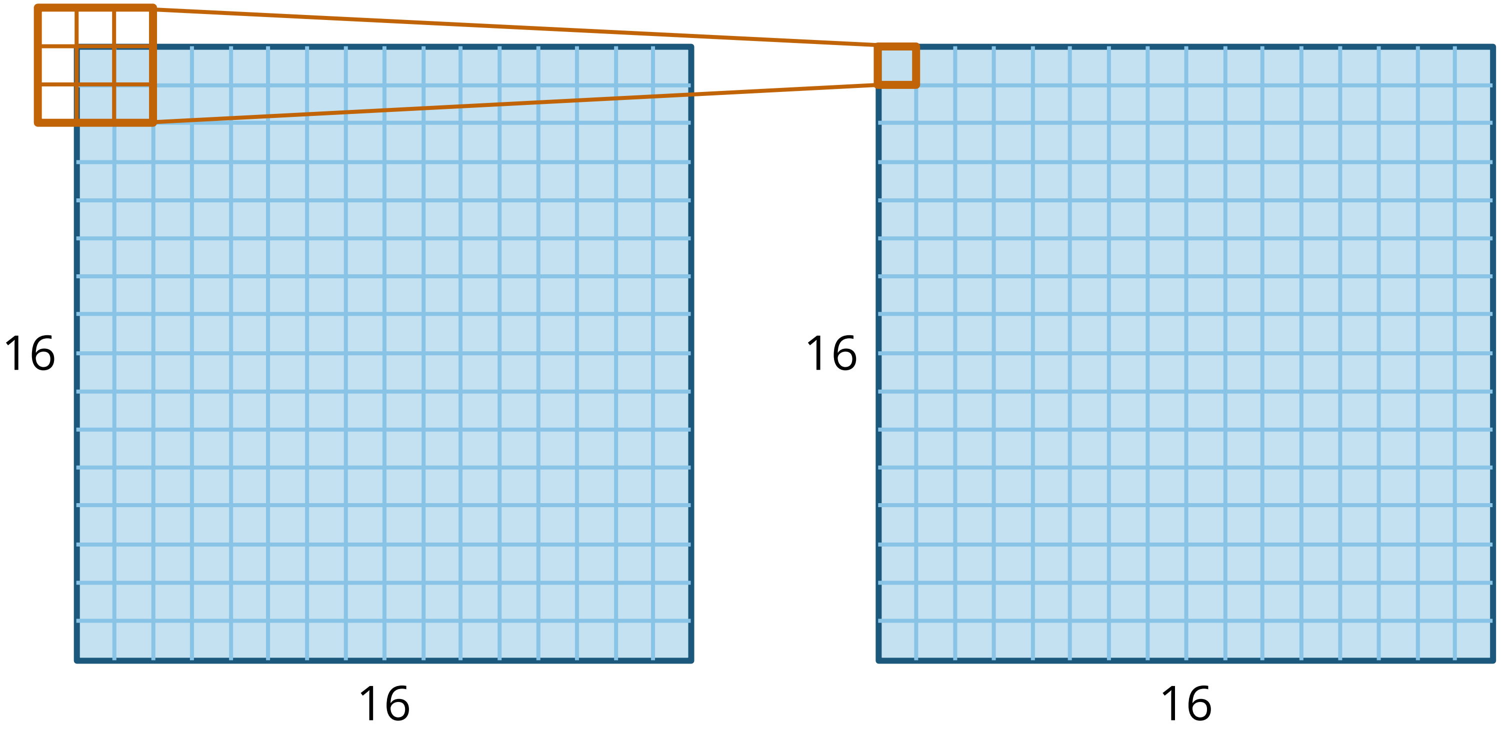
Downsample and keep the strongest activation in each block

Convolutional Neural Network (CNNs)



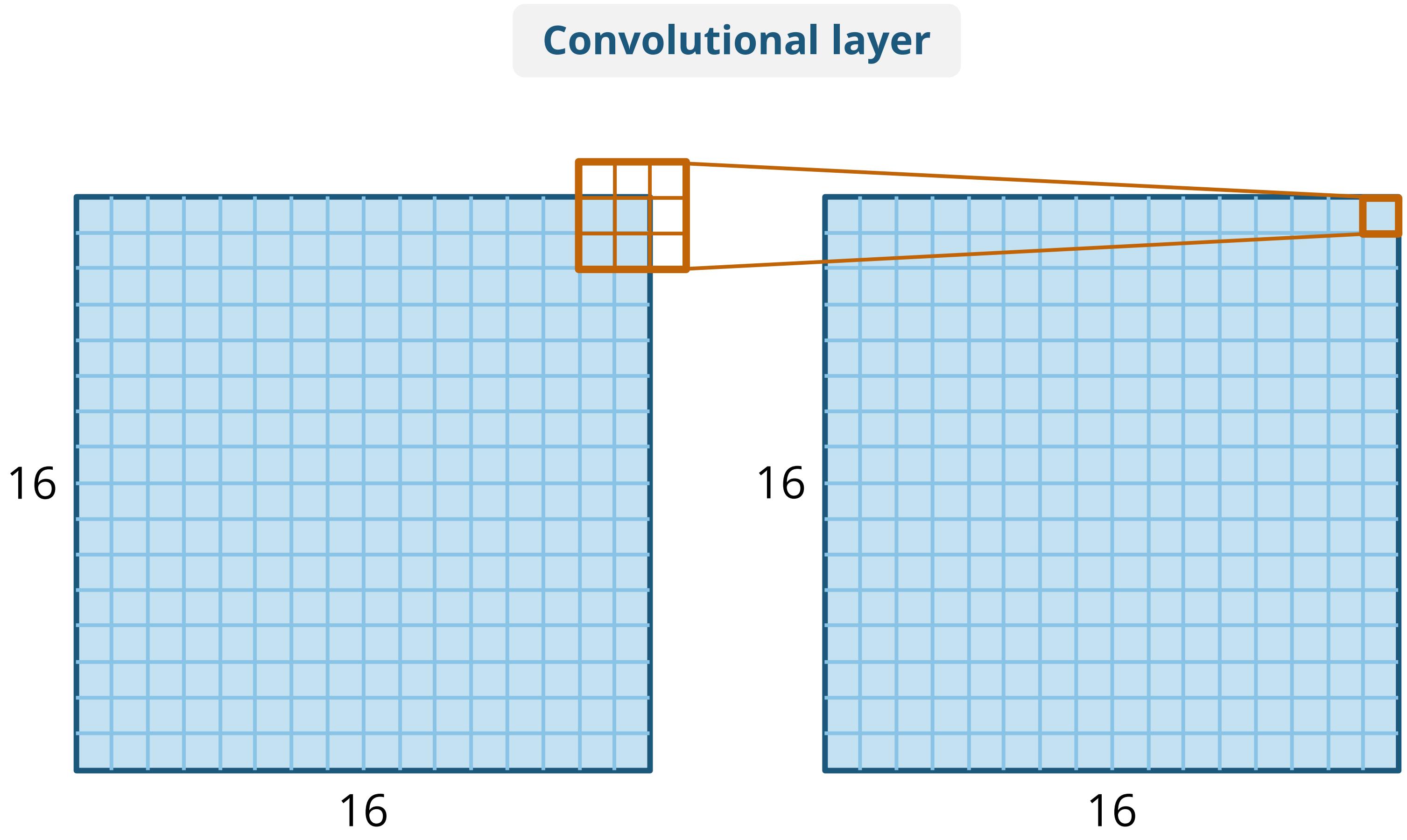
A Toy Example

Convolutional layer



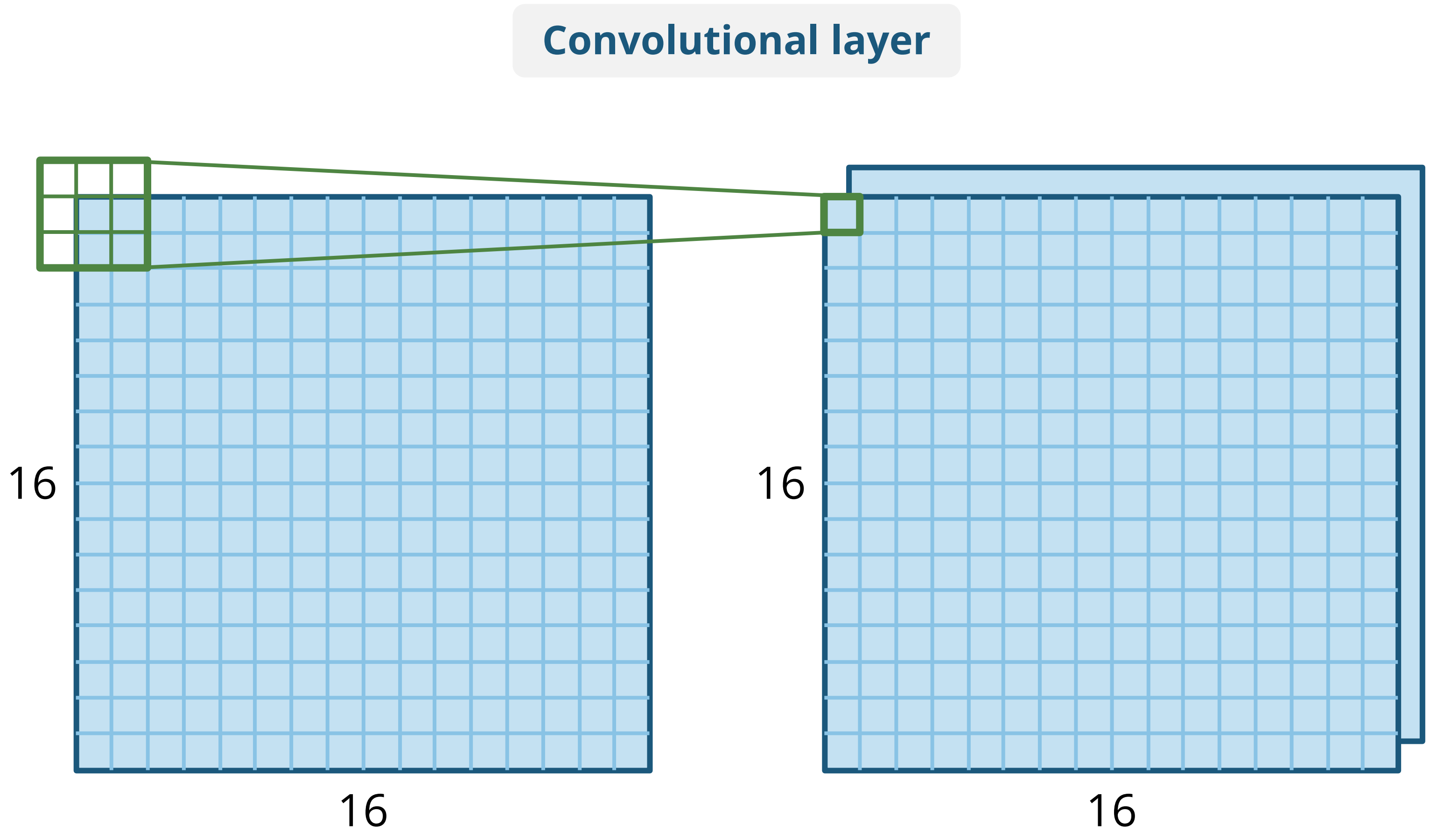

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example




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

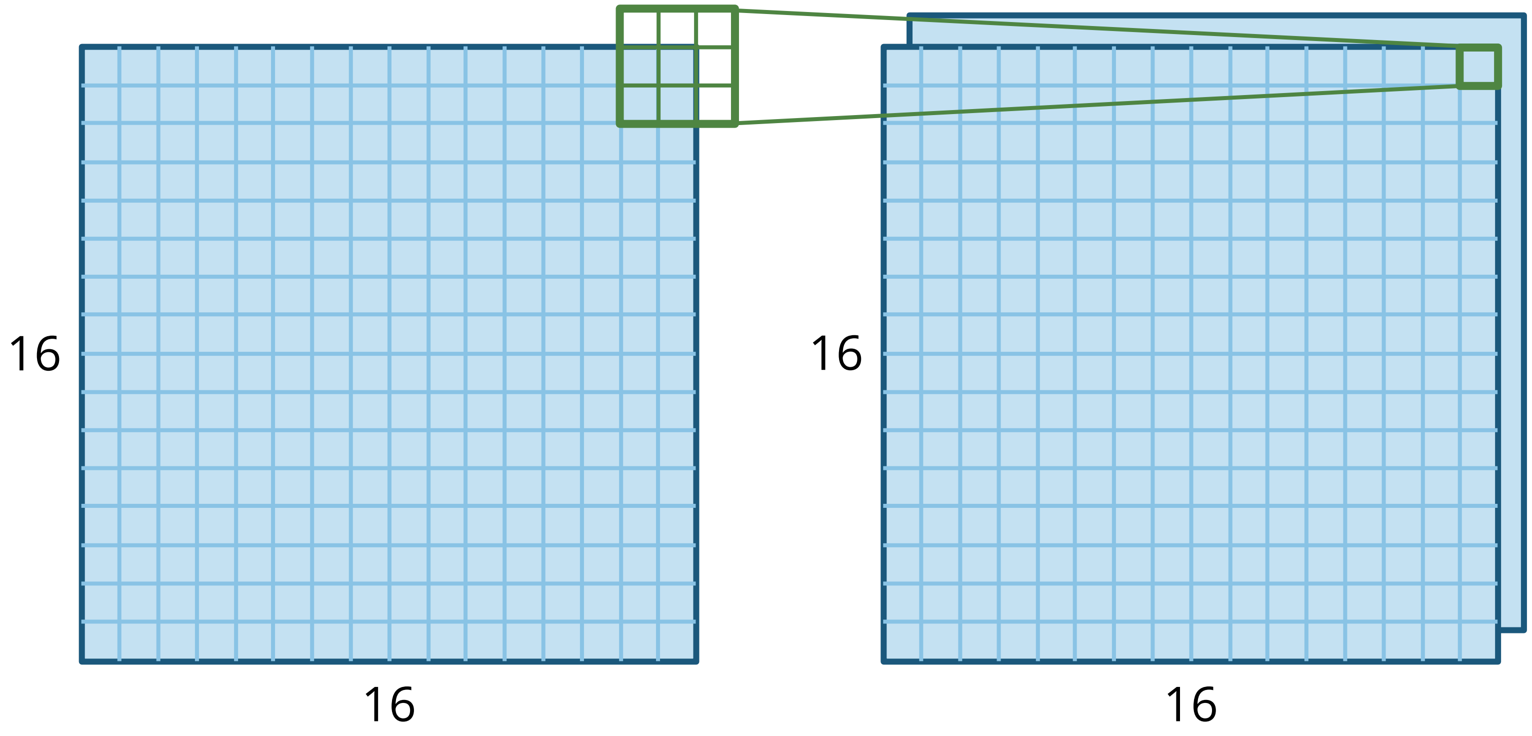
A Toy Example




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

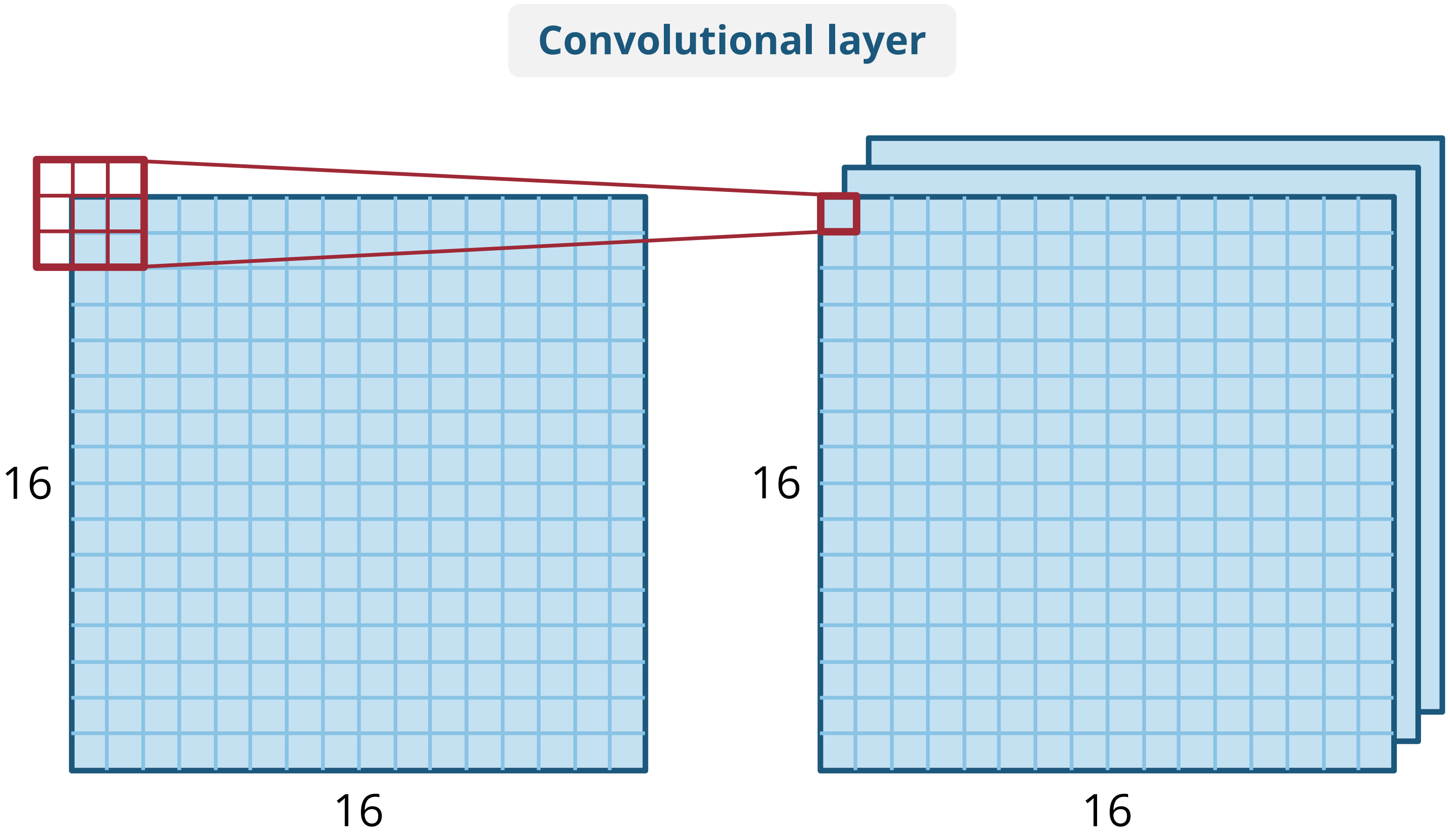
A Toy Example

Convolutional layer



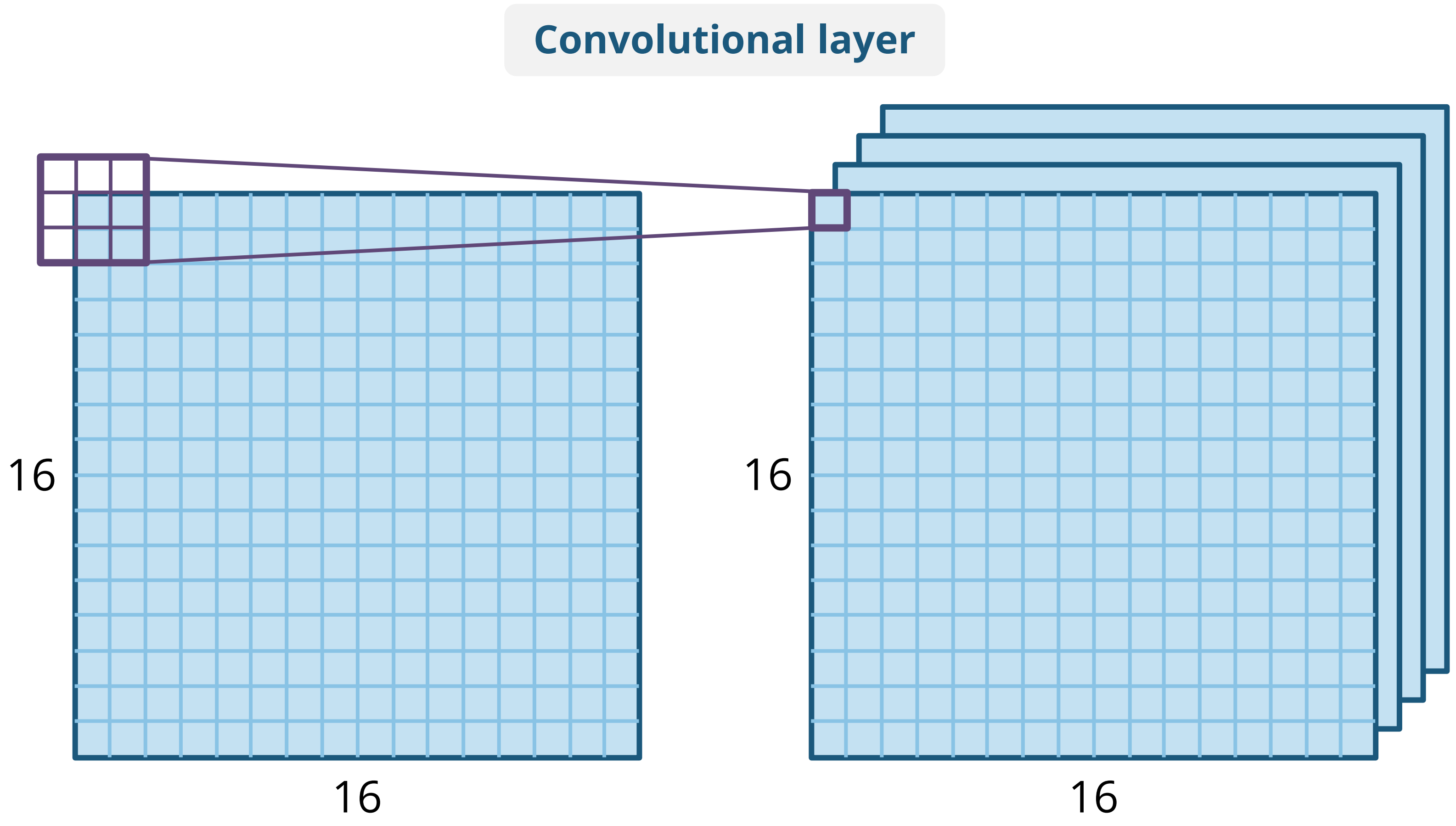

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example



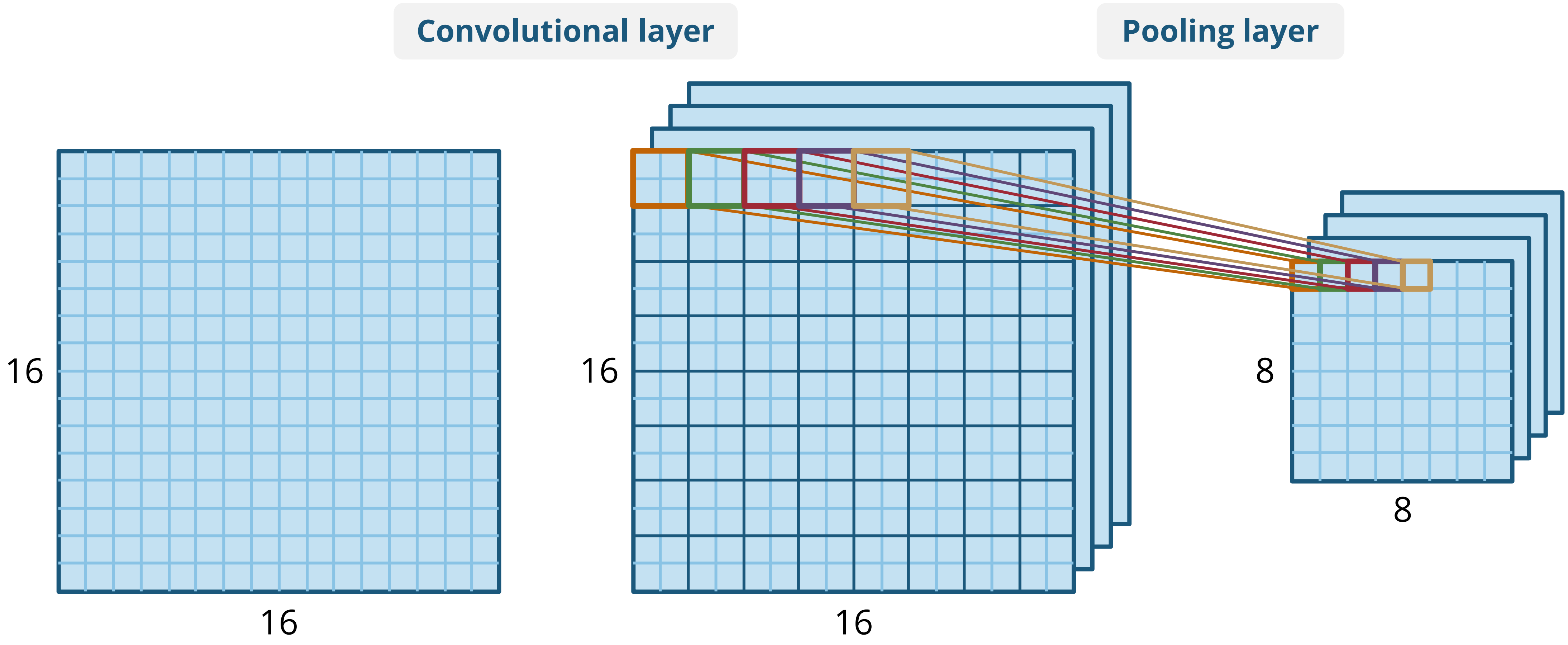

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example



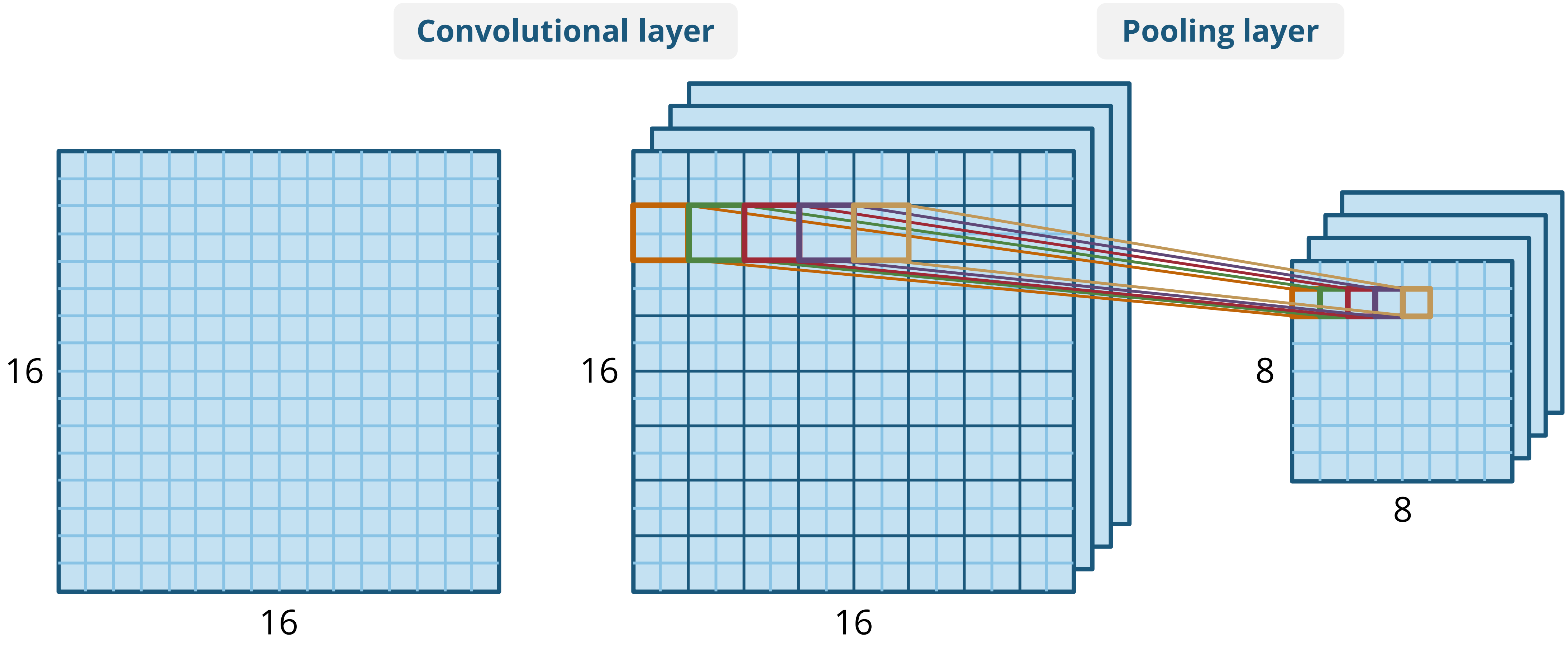

```
nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, padding="same")
```

A Toy Example



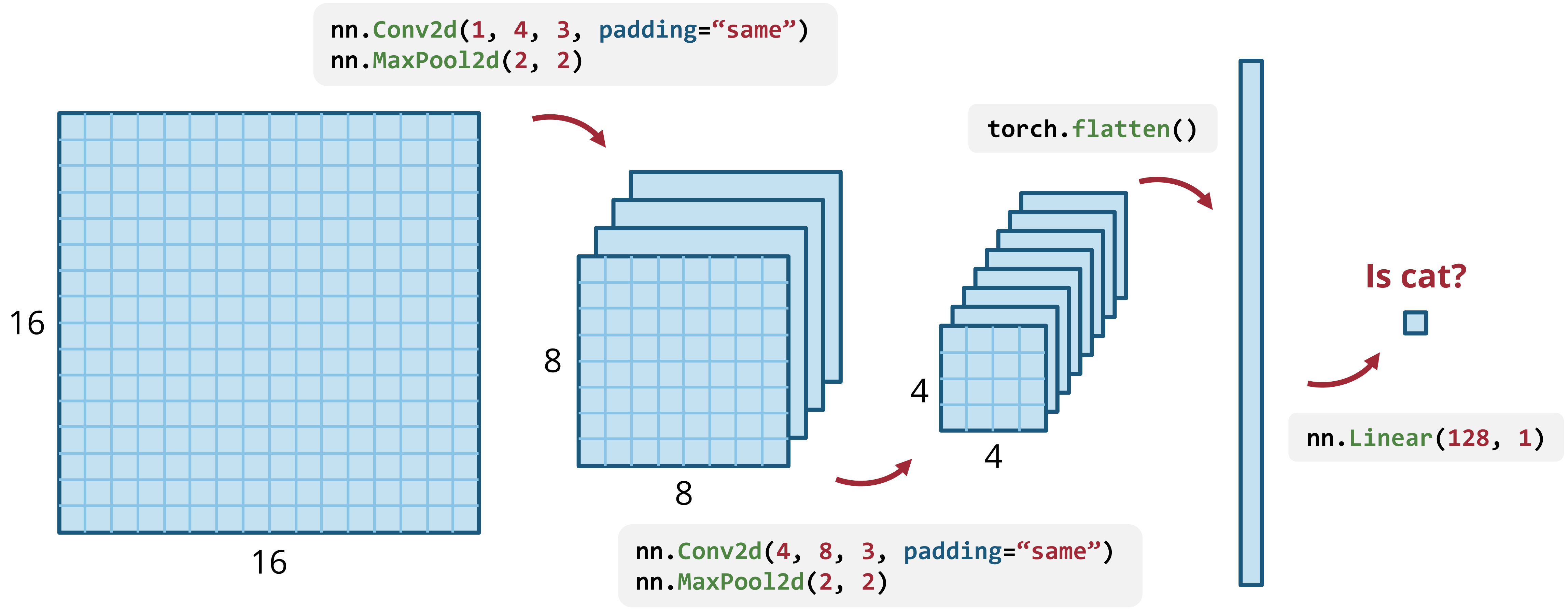

```
nn.MaxPool2d(kernel_size=2, stride=2)
```

A Toy Example

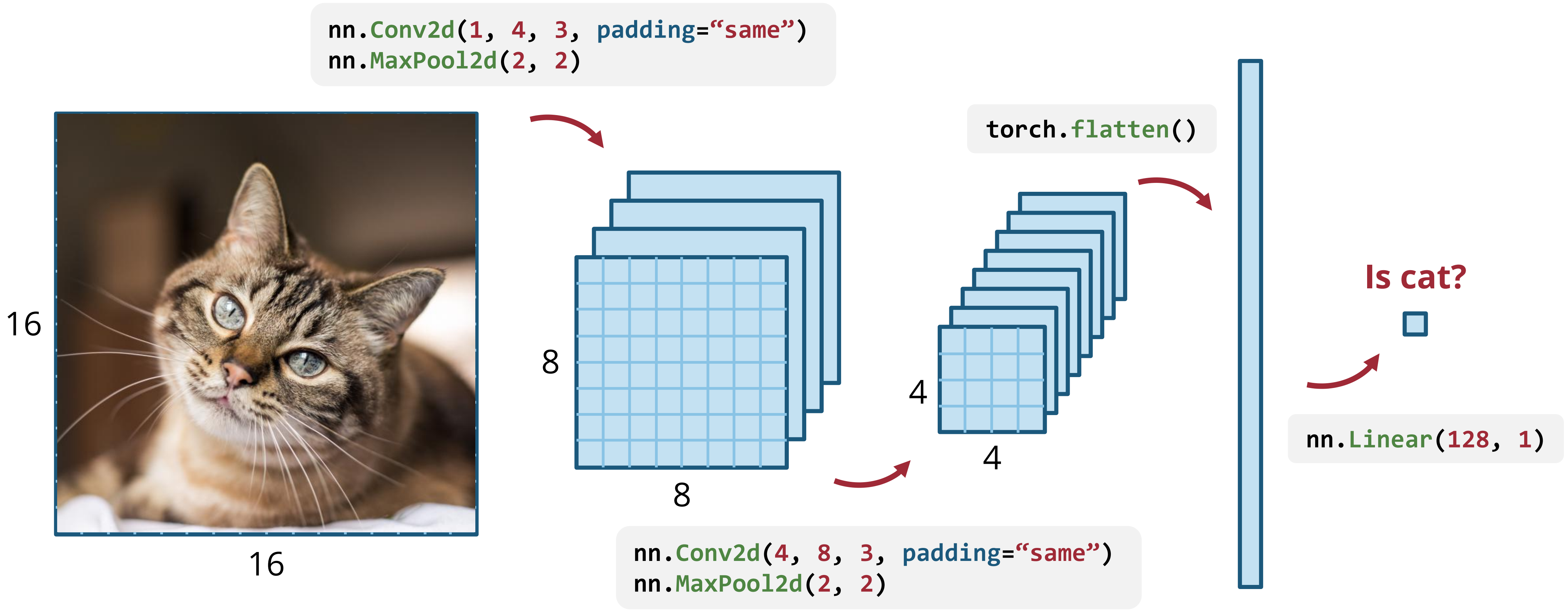



```
nn.MaxPool2d(kernel_size=2, stride=2)
```

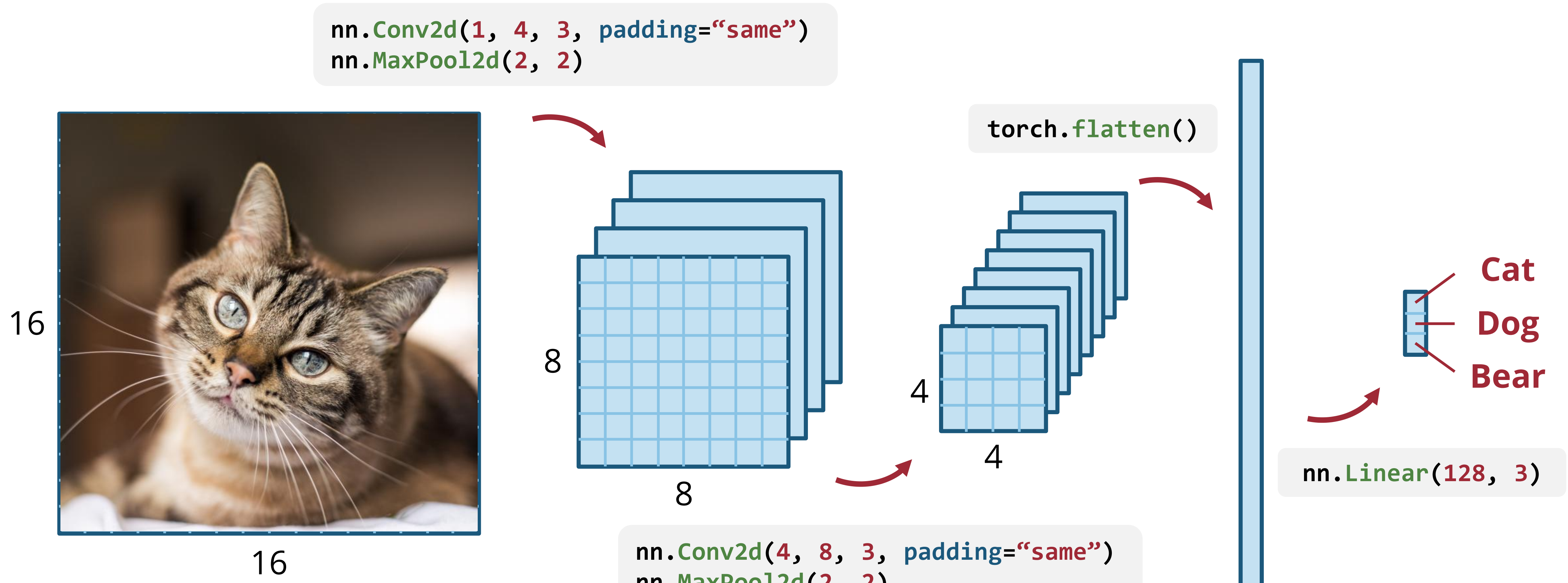
A Toy Example



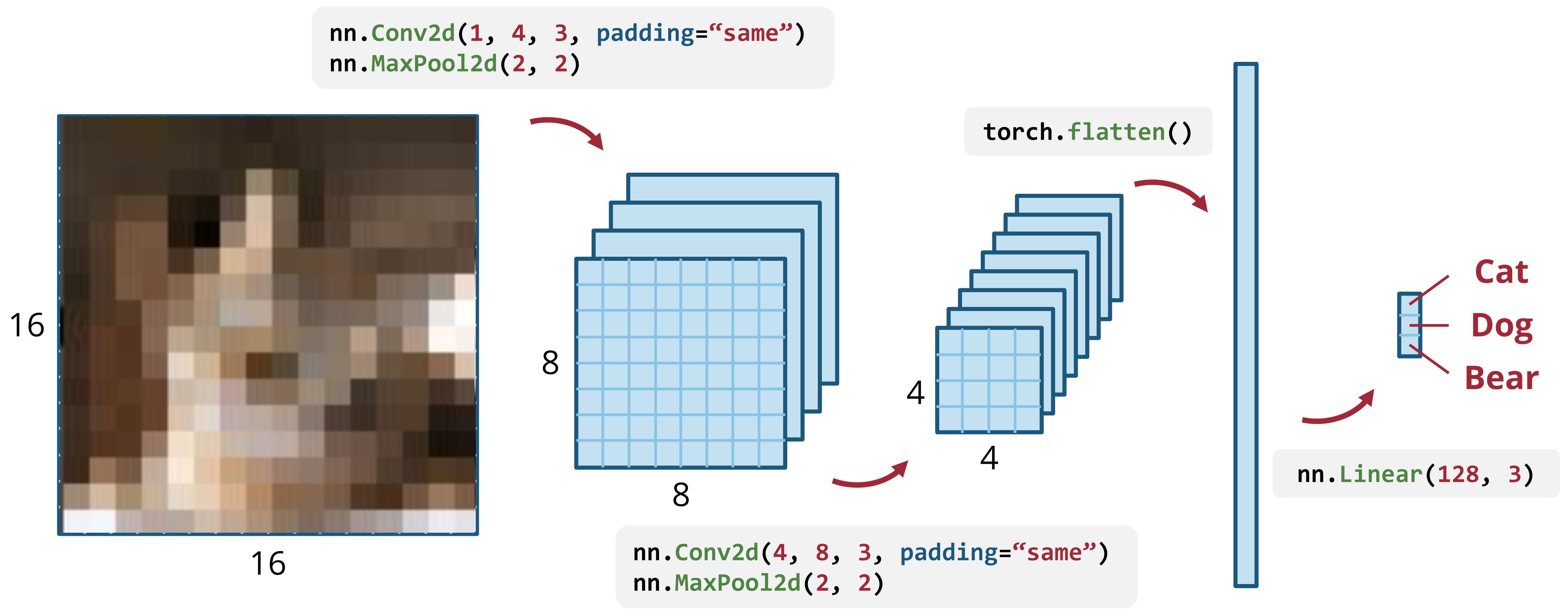
A Toy Example



A Toy Example



A Toy Example



A Real Example

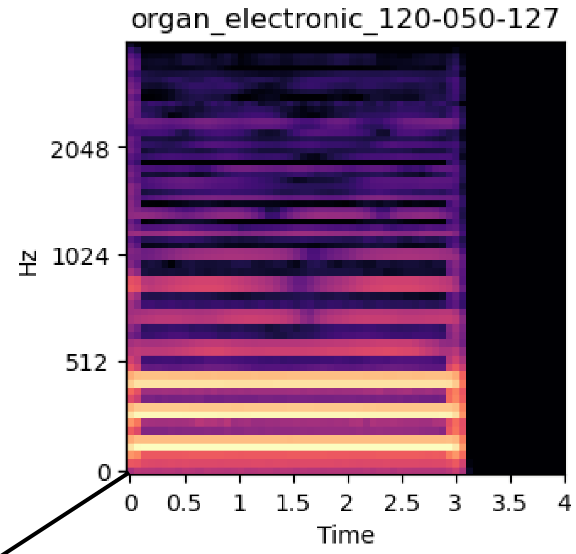
Input channels

Output channels

Kernel size

```
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?

$\rightarrow$ (3 x 3 x 1 + 1) x 16 = 160

$\rightarrow$ (3 x 3 x 16 + 1) x 32 = 4640

$\rightarrow$ (3 x 3 x 32 + 1) x 64 = 18496

$\rightarrow$ (3 x 3 x 64 + 1) x 128 = 73856

$\rightarrow$ (2048 + 1) x 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

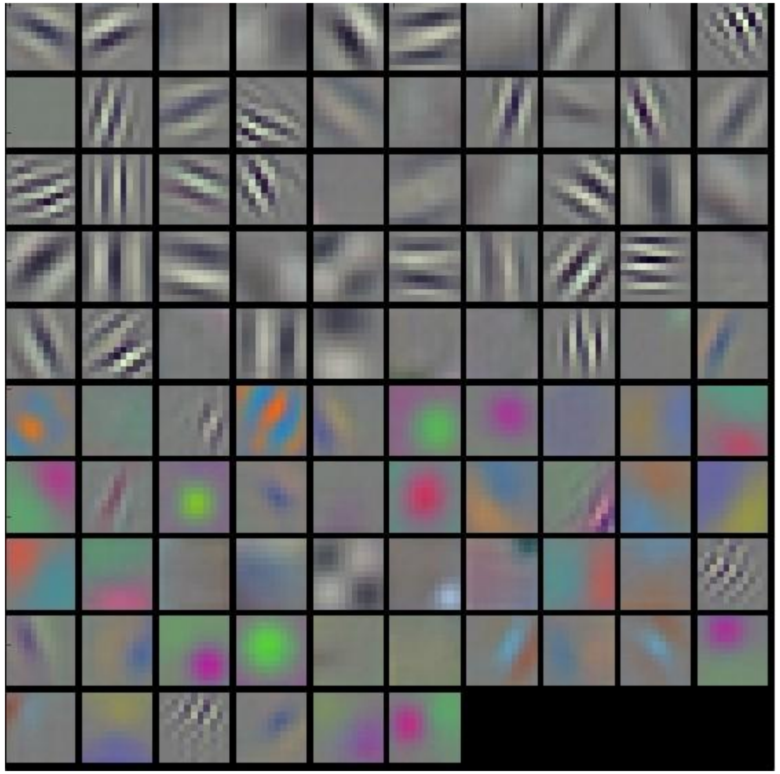
What does a CNN Learn?

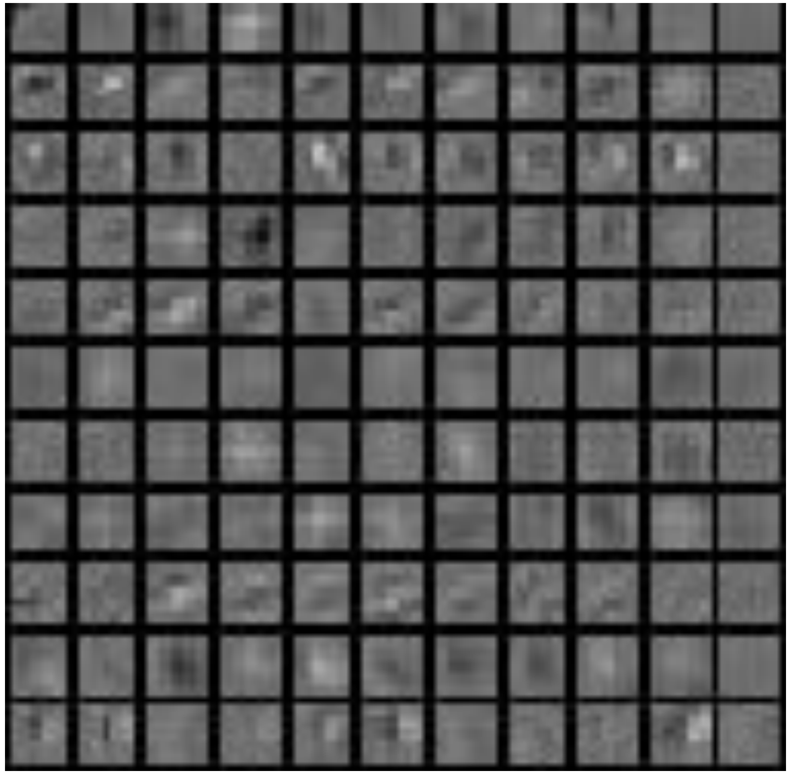
Learned CNN Kernels in a Trained AlexNet

1st convolutional layer

2nd convolutional layer

11x11 kernels

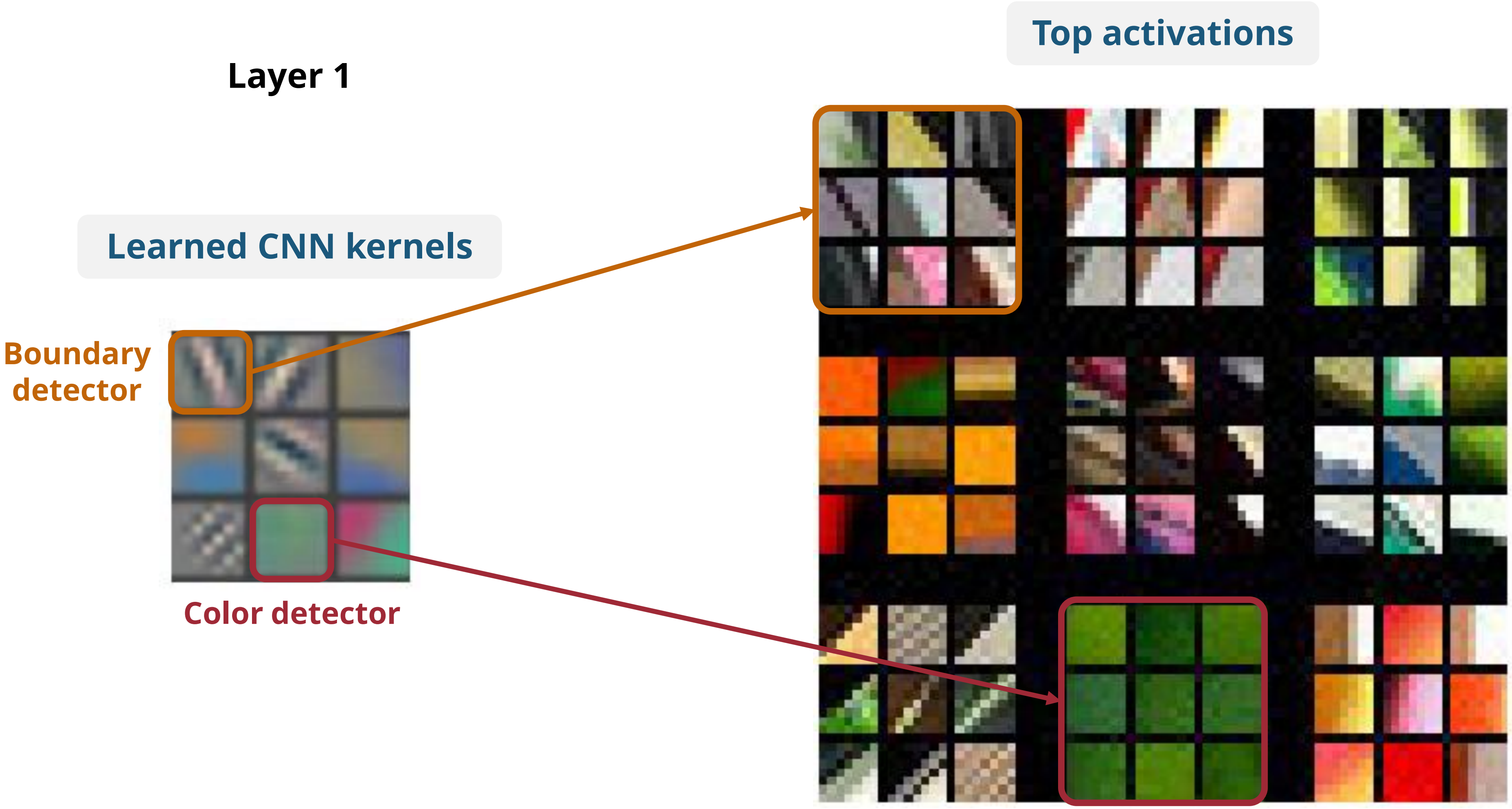




5x5 kernels

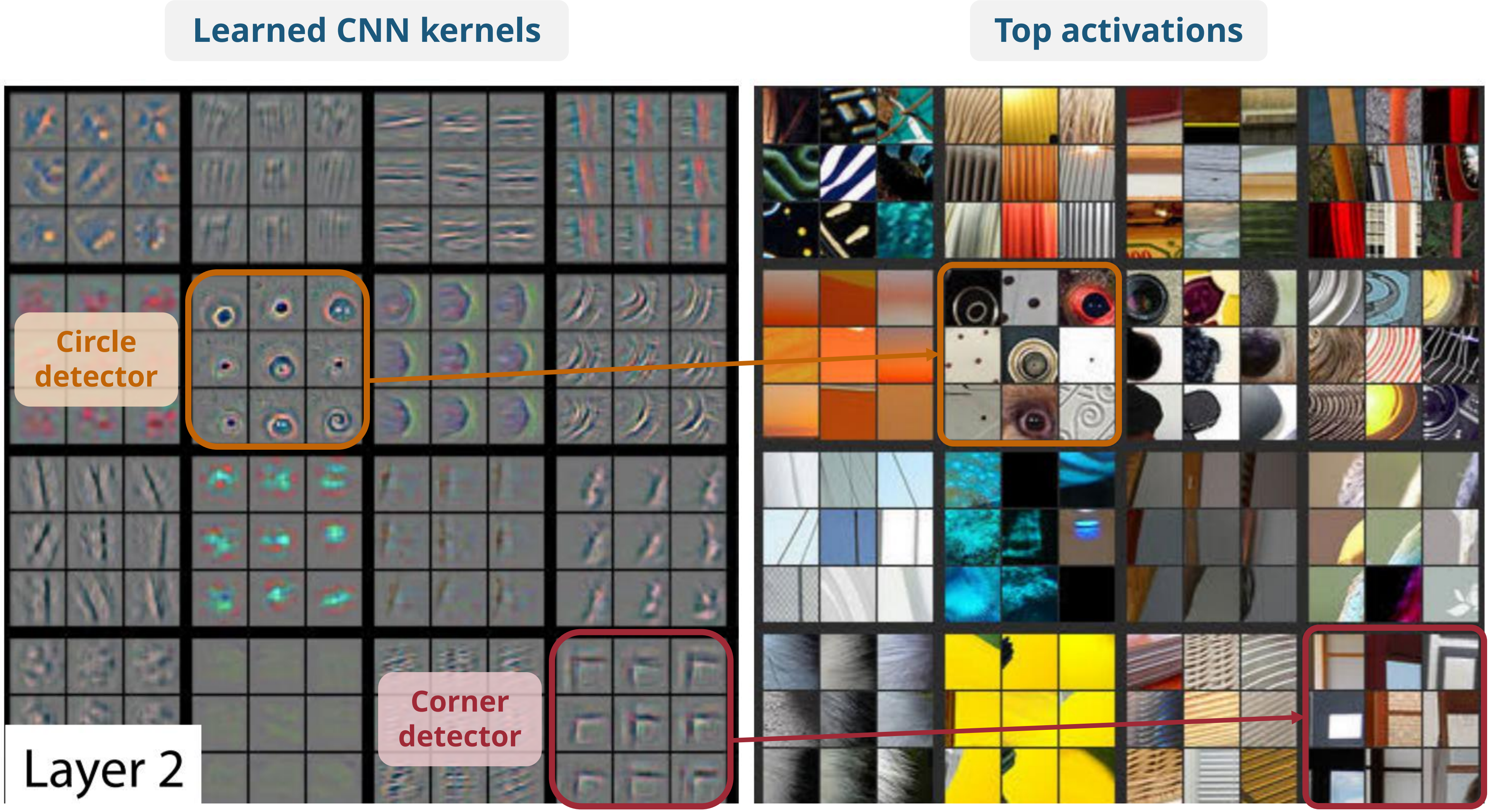
cs231n.github.io/understanding-cnn/

Learned CNN Kernels in a Trained AlexNet



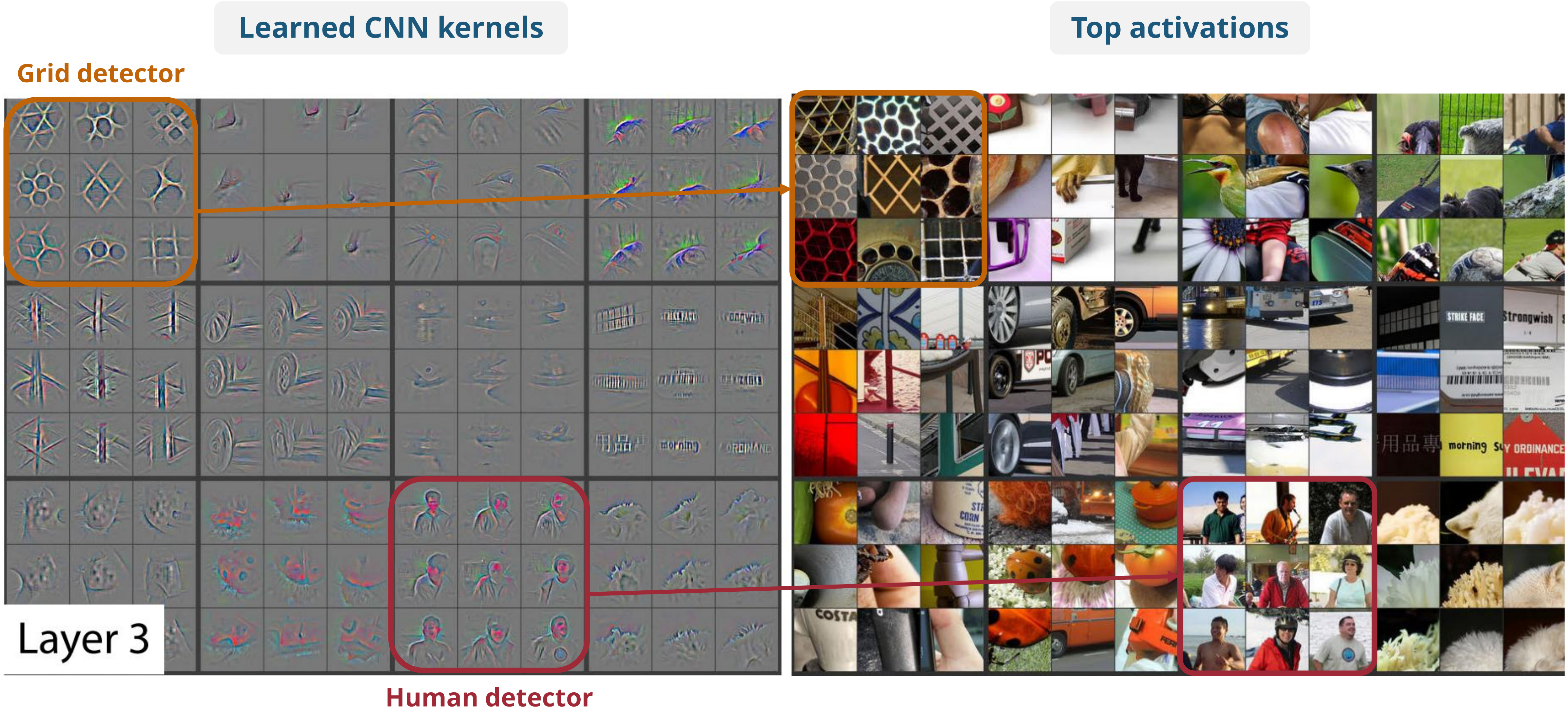
Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet



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

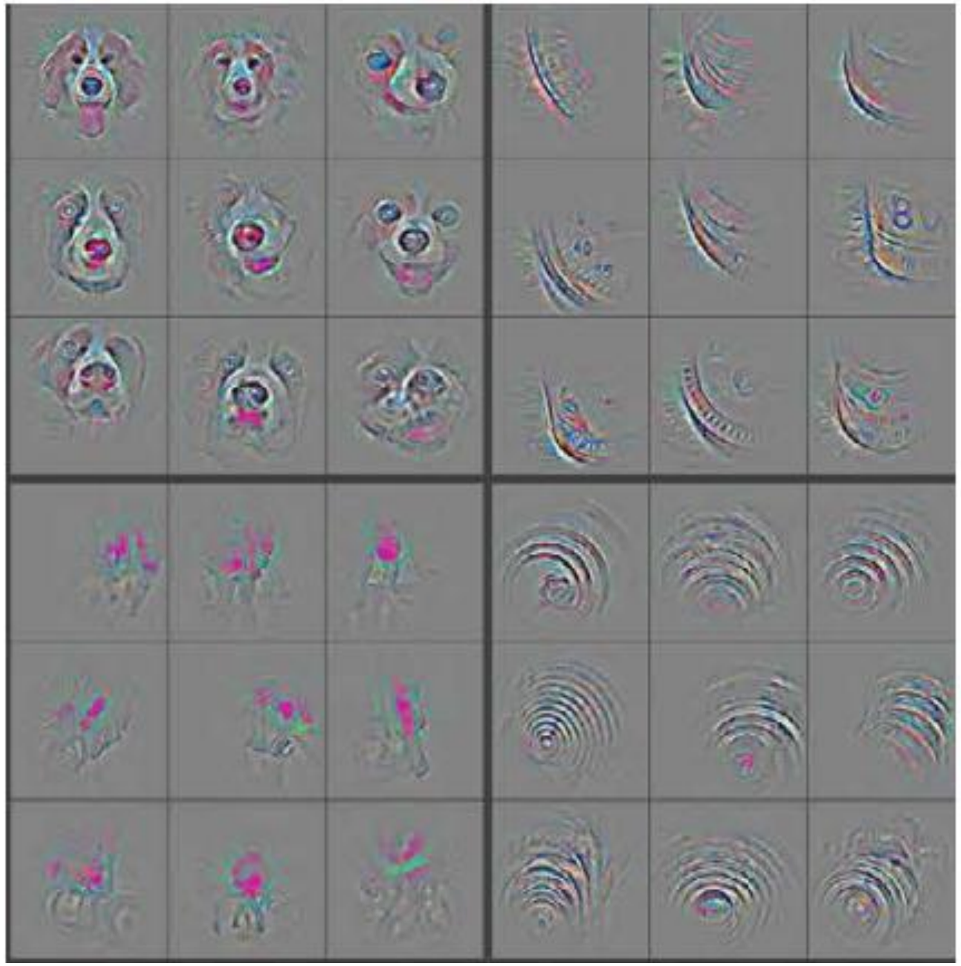
Learned CNN Kernels in a Trained AlexNet



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

Learned CNN Kernels in a Trained AlexNet

Learned CNN kernels

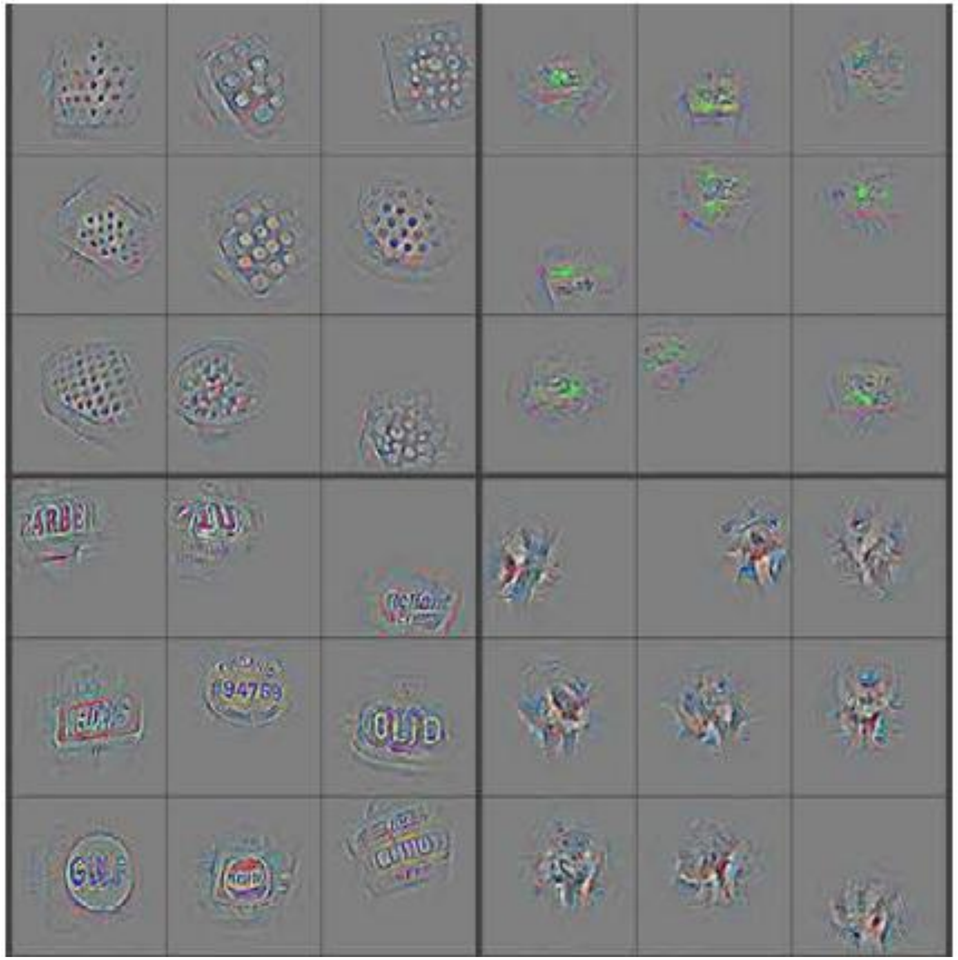


Top activations

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

Learned CNN Kernels in a Trained AlexNet

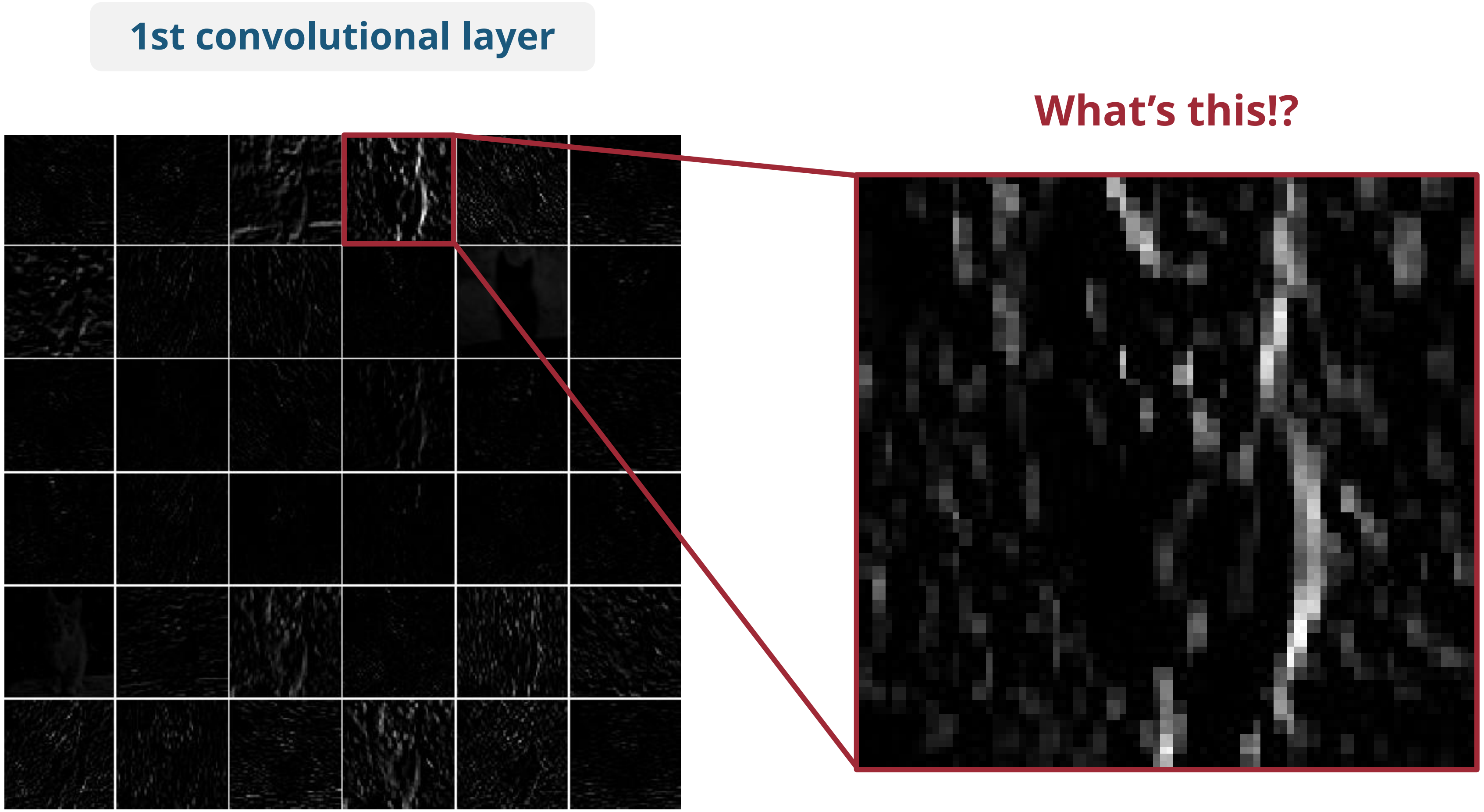
Learned CNN kernels



Top activations

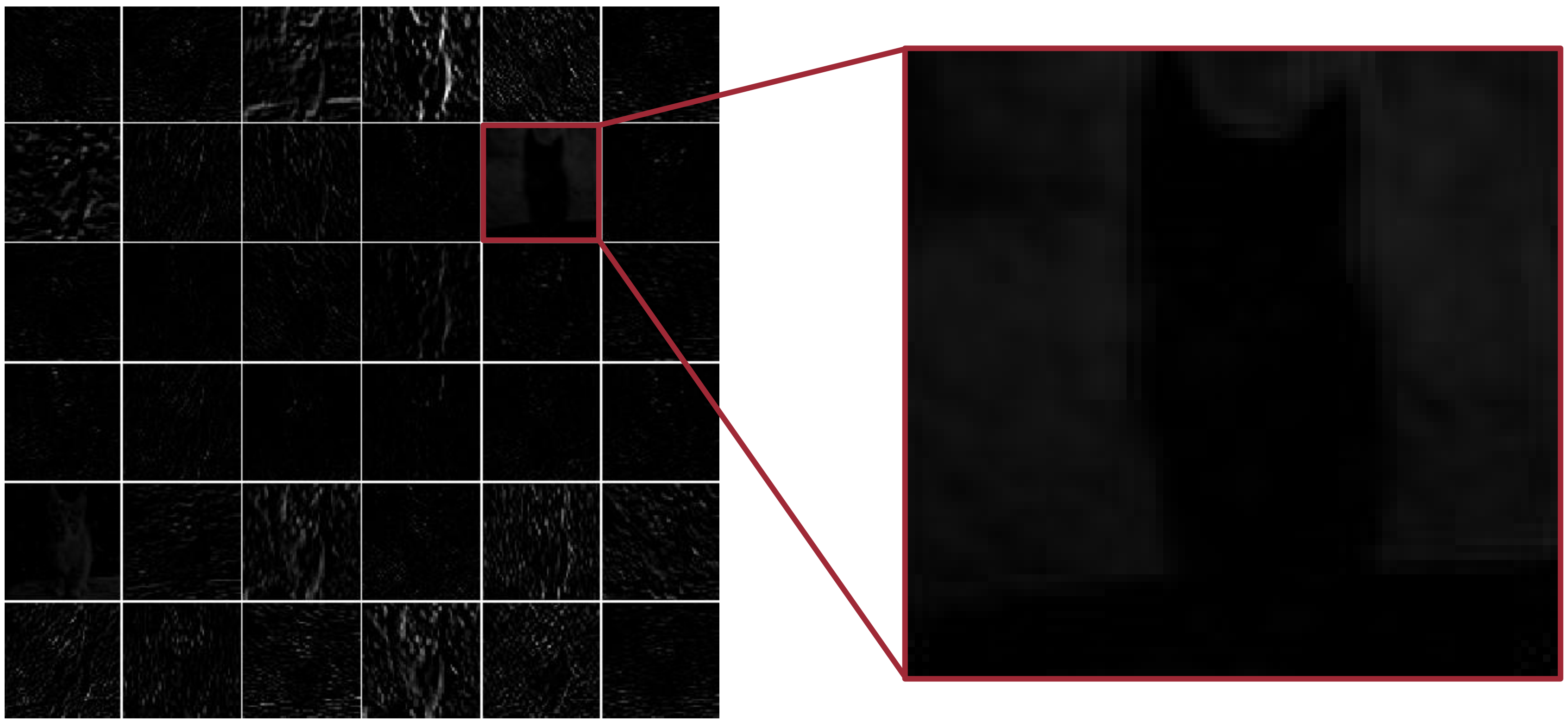
Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Activations in a Trained AlexNet

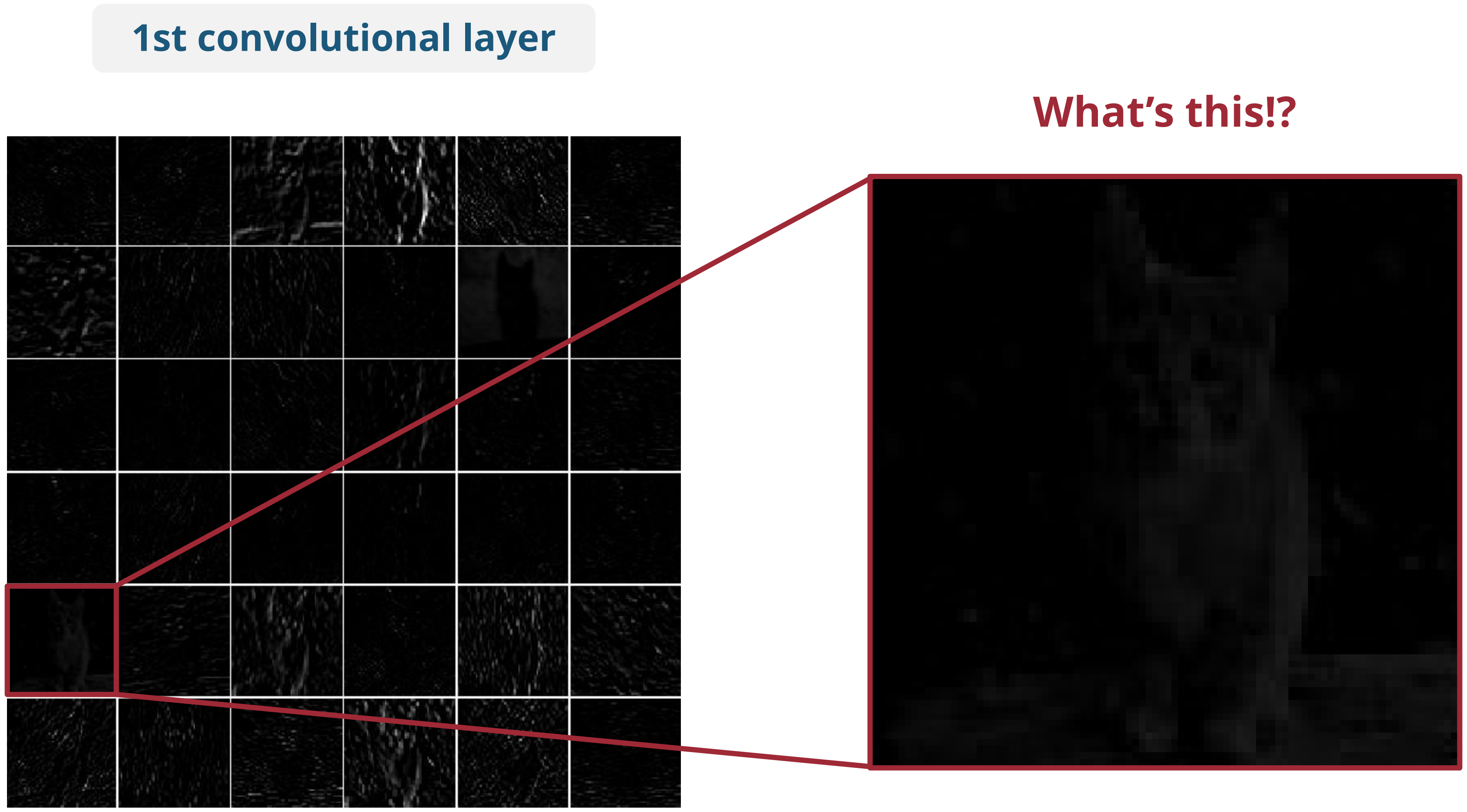


Activations in a Trained AlexNet

1st convolutional layer



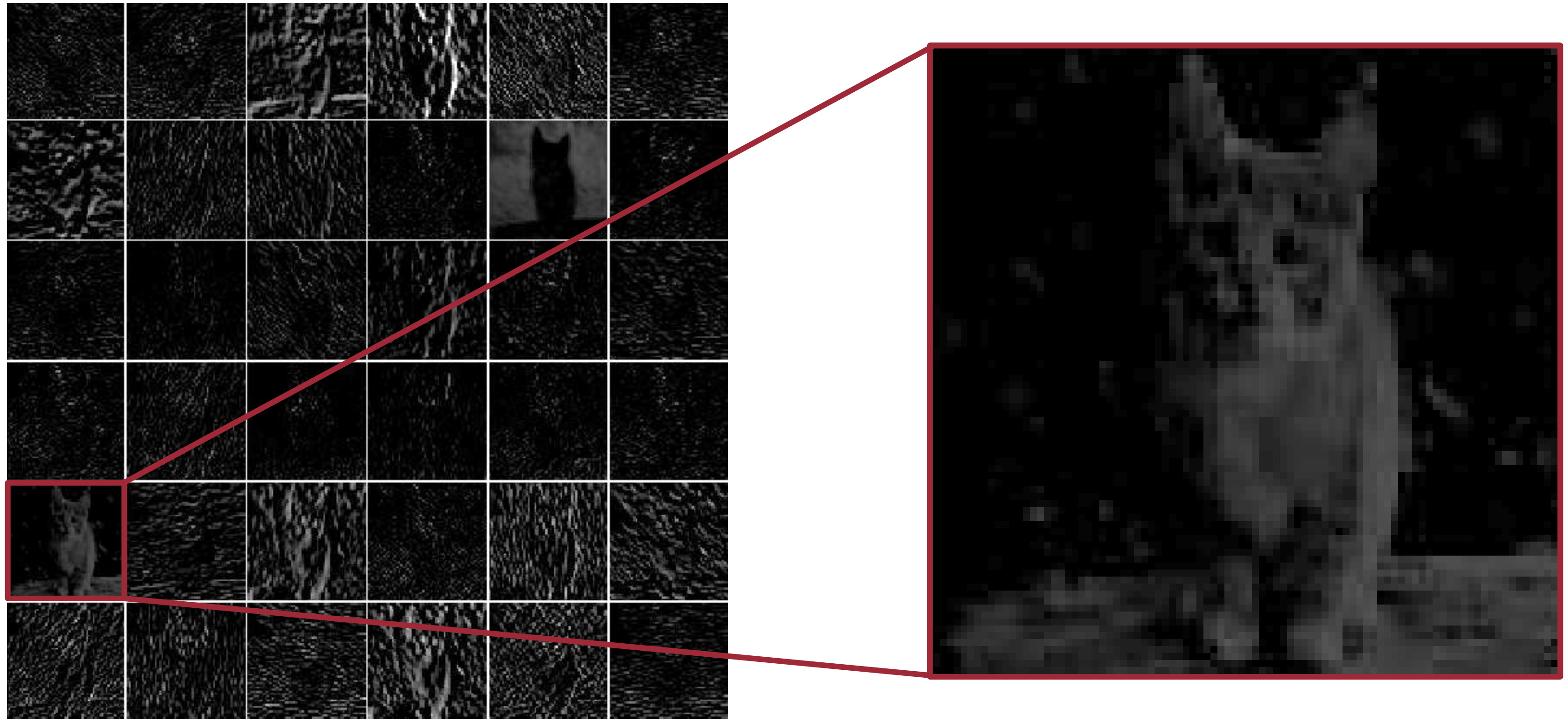
Activations in a Trained AlexNet



Activations in a Trained AlexNet

1st convolutional layer

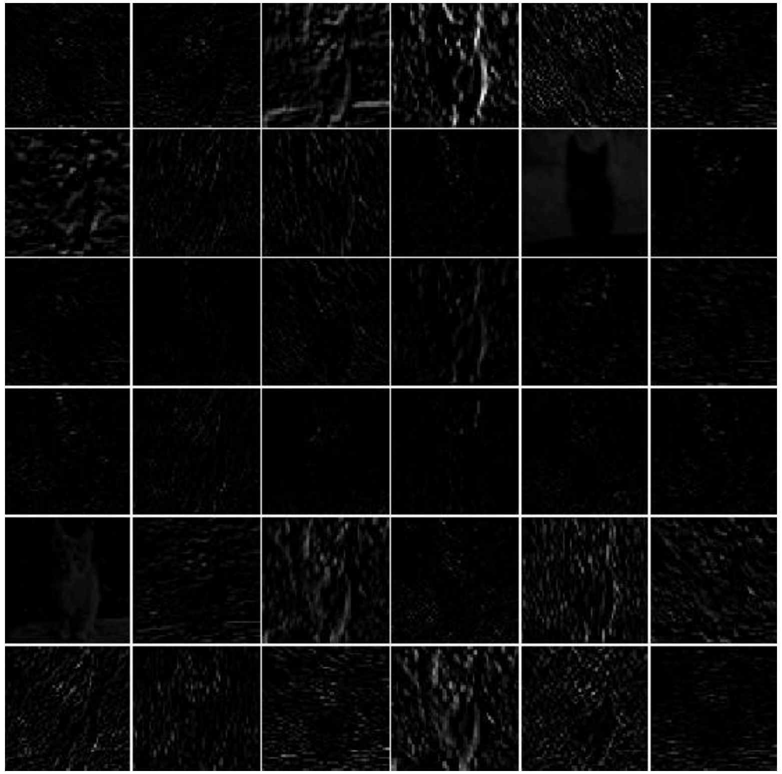
What's this!?



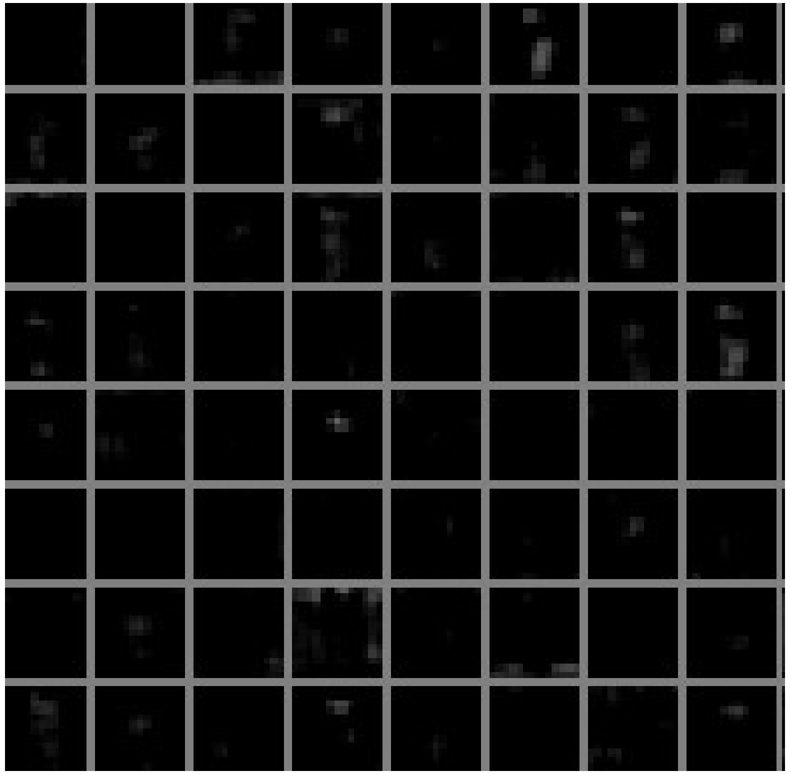
Brightness+
Contrast+

Activations in a Trained AlexNet

1st convolutional layer



5th convolutional layer

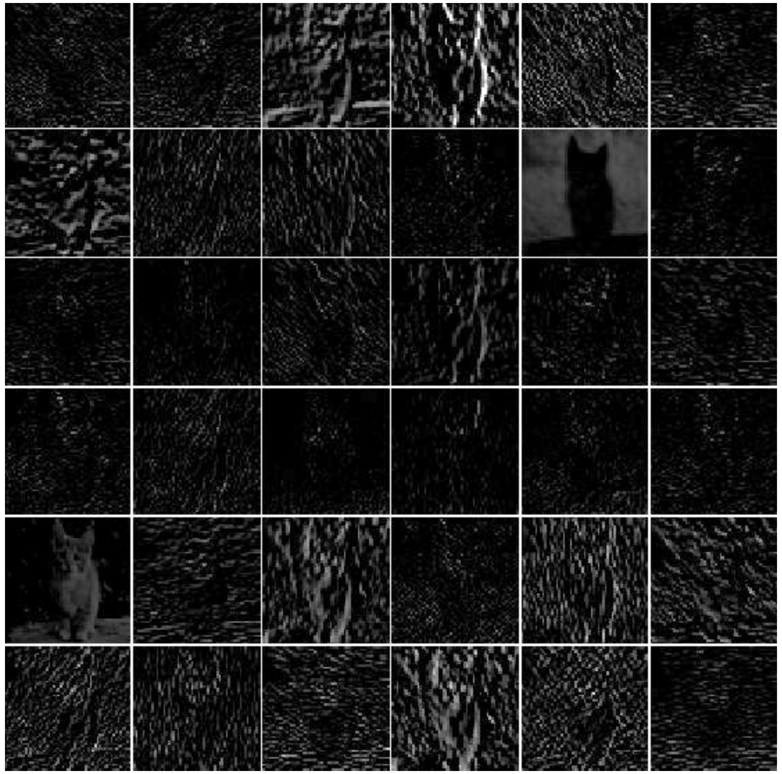


cs231n.github.io/understanding-cnn/

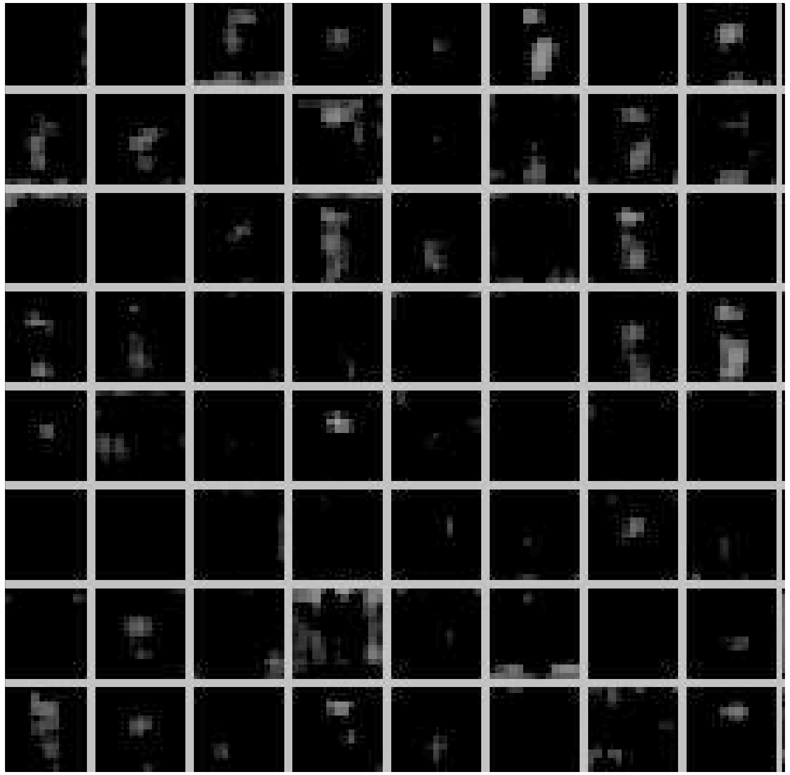
Activations in a Trained AlexNet

1st convolutional layer

Brightness+ Contrast+



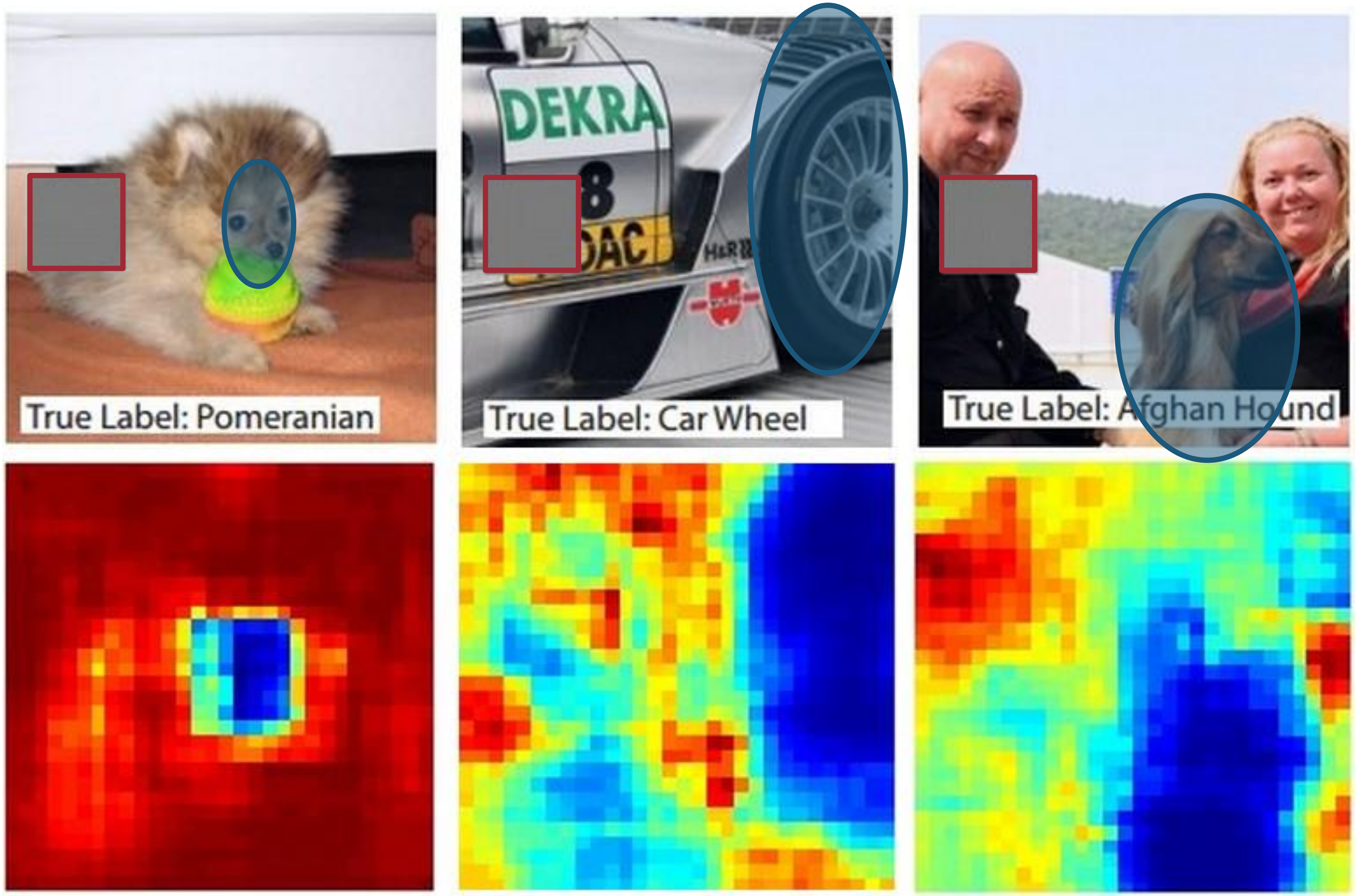
5th convolutional layer



Brightness+ Contrast+

cs231n.github.io/understanding-cnn/

What does a CNN Learn?



cs231n.github.io/understanding-cnn/