PAT 498/598 (Fall 2024)

Special Topics:
Generative AI for Music and Audio Creation

Lecture 7: CNNs

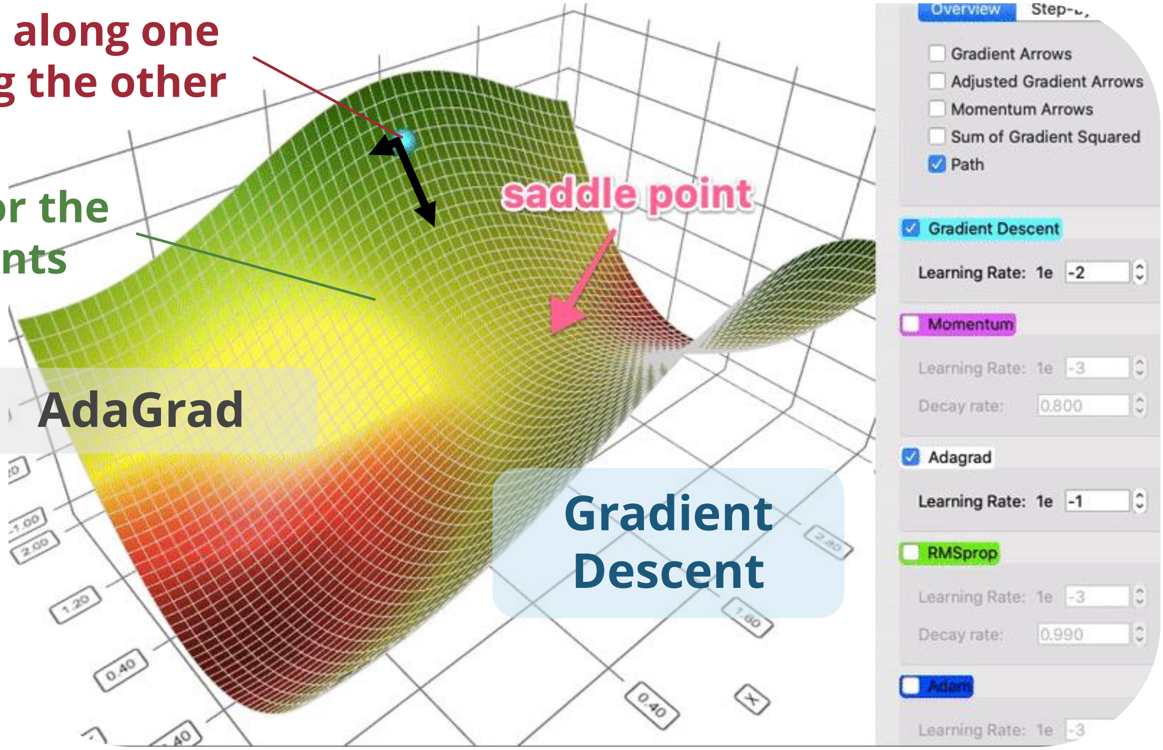
Instructor: Hao-Wen Dong

(Recap) Gradient-based Adaptive Learning Rate

- **Intuition**: **Compensate axis that has little progress** by comparing the current gradients to the previous gradients

Large gradients along one axis, small along the other

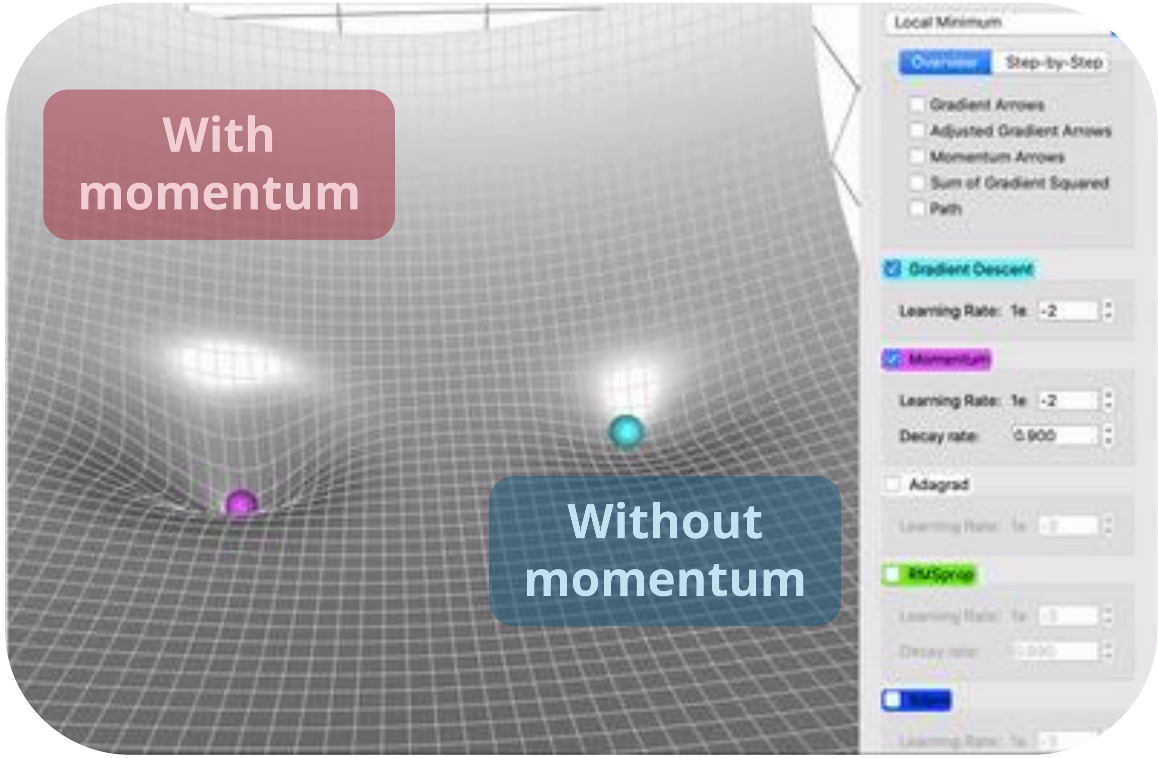
Use larger learning rate for the axis with smaller gradients



towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

(Recap) Momentum

- **Intuition**: Maintain the momentum to **escape from local minima**

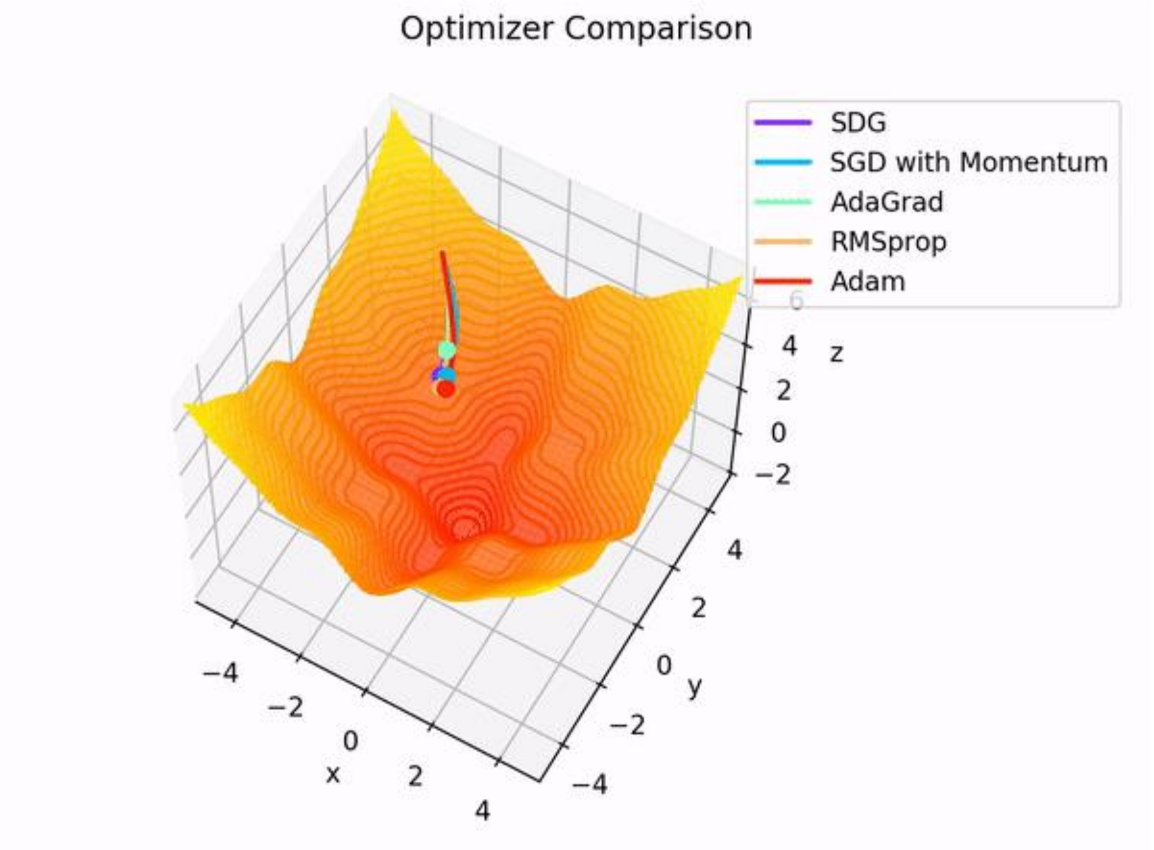


towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

(Recap) Comparison of Optimizers

- **Momentum**
  - Gets you out of spurious local minima
  - Allows the model to explore around

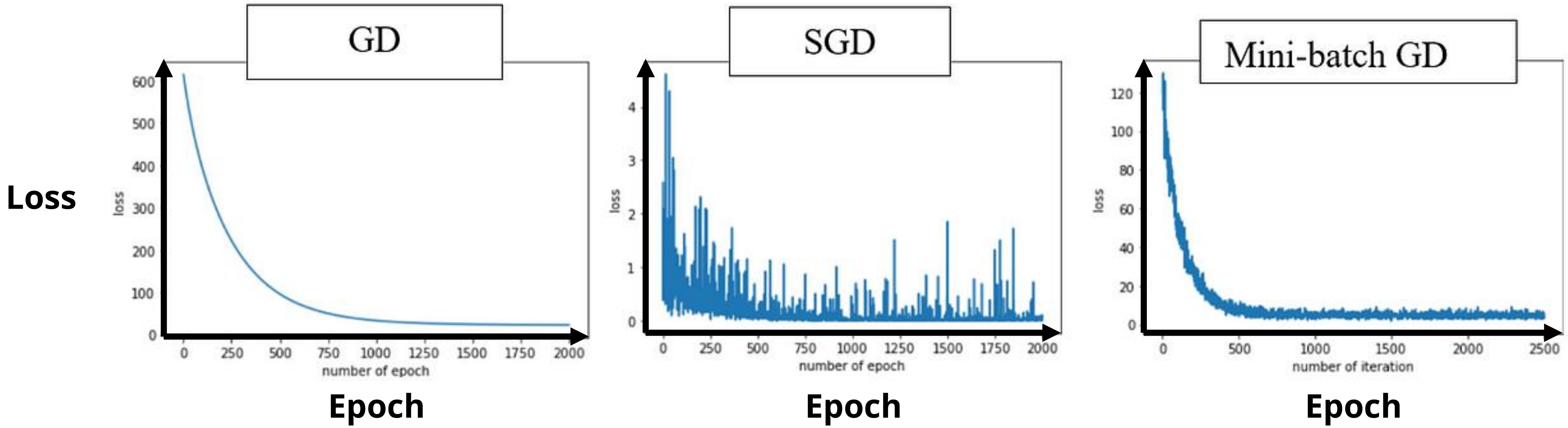
- **Gradient-based adaption**
  - Maintains steady improvement
  - Allows faster convergence



medium.com/@LayanSA/complete-guide-to-adam-optimization-1e5f29532c3d

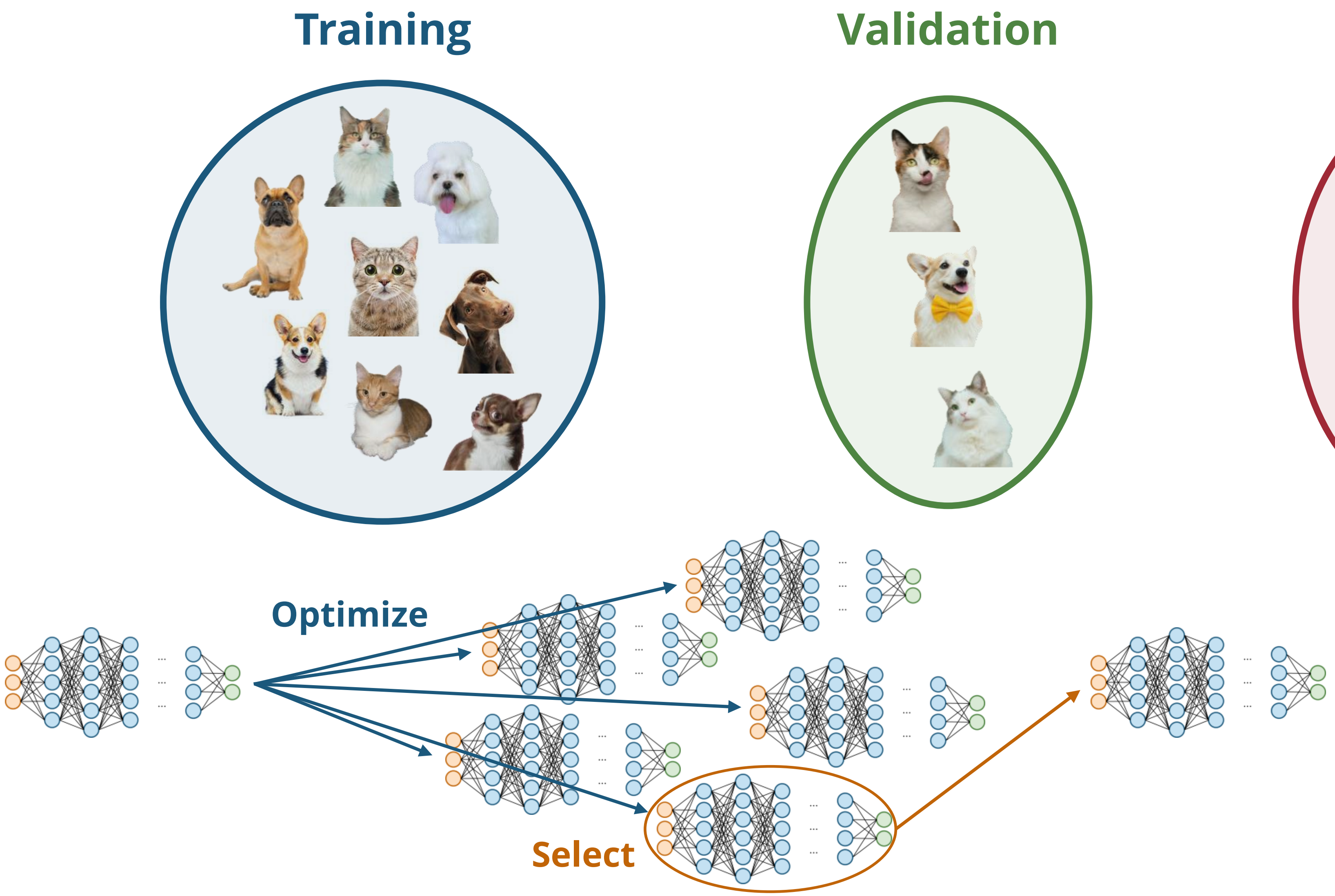
(Recap) Mini-batch Gradient Descent

- **Intuition: Estimate** the gradient using **several random training samples**

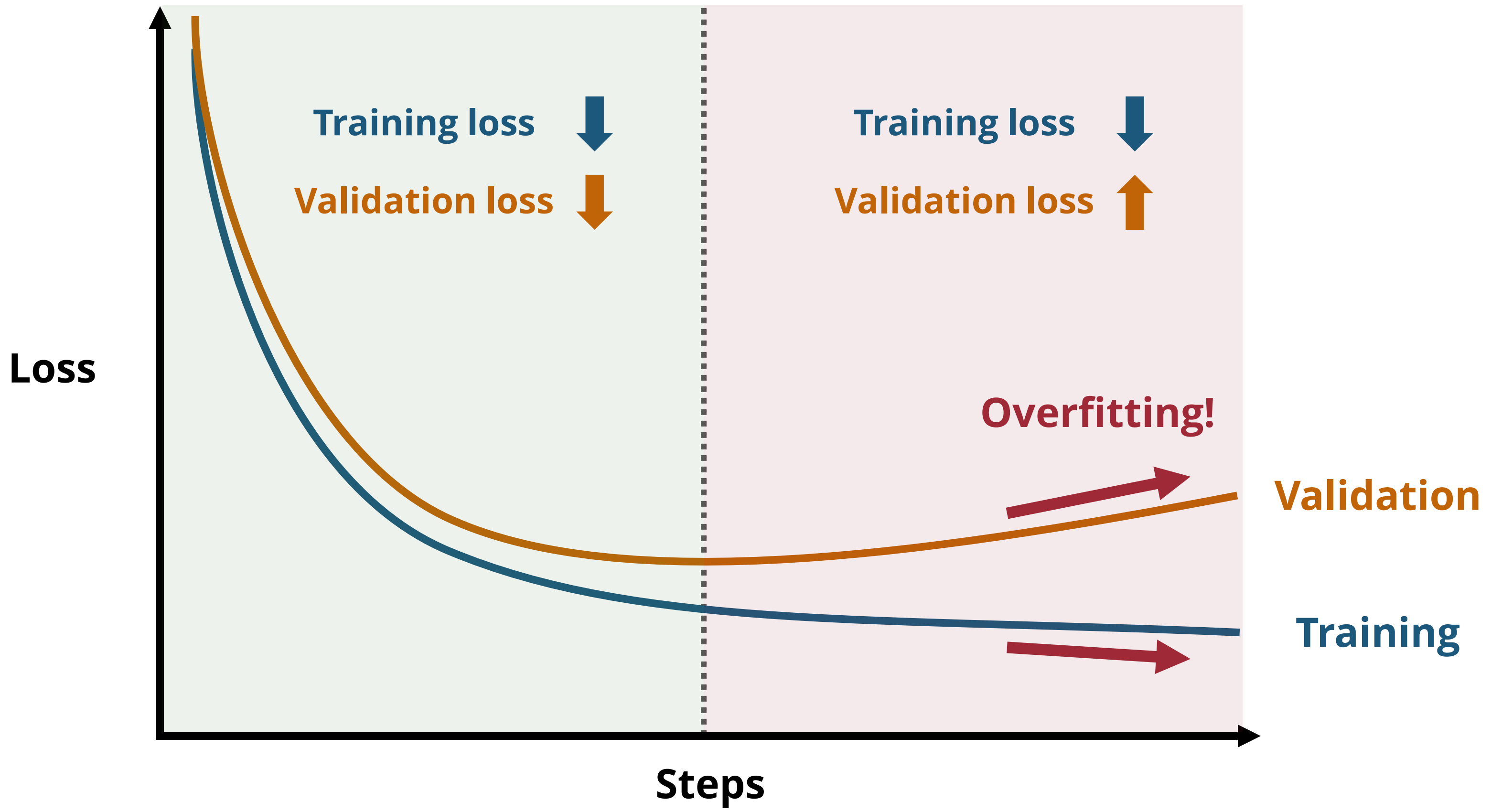


batch size $= N$ batch size $= 1$ $1 <$ batch size $< N$

(Recap) Training–Validation–Test Pipeline



(Recap) Training vs Validation Losses

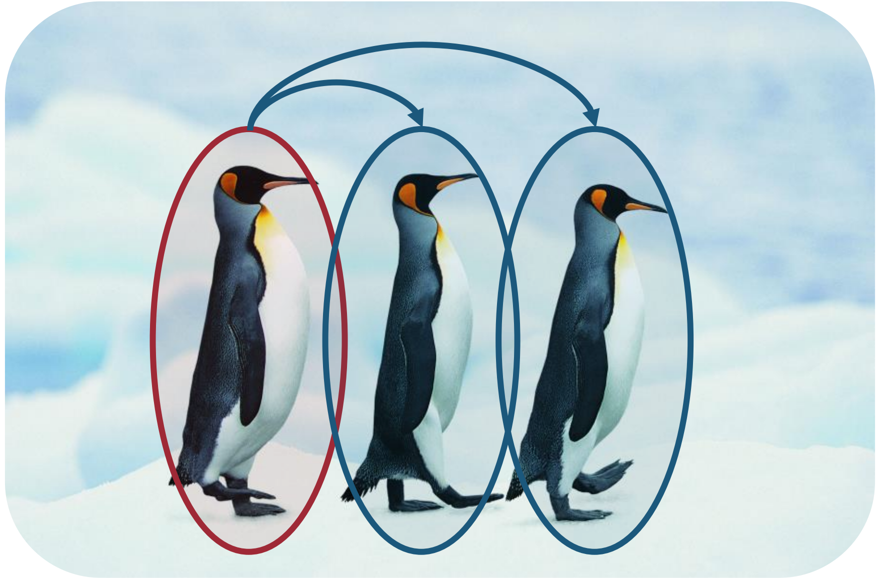


Convolutional Neural Networks (CNNs)

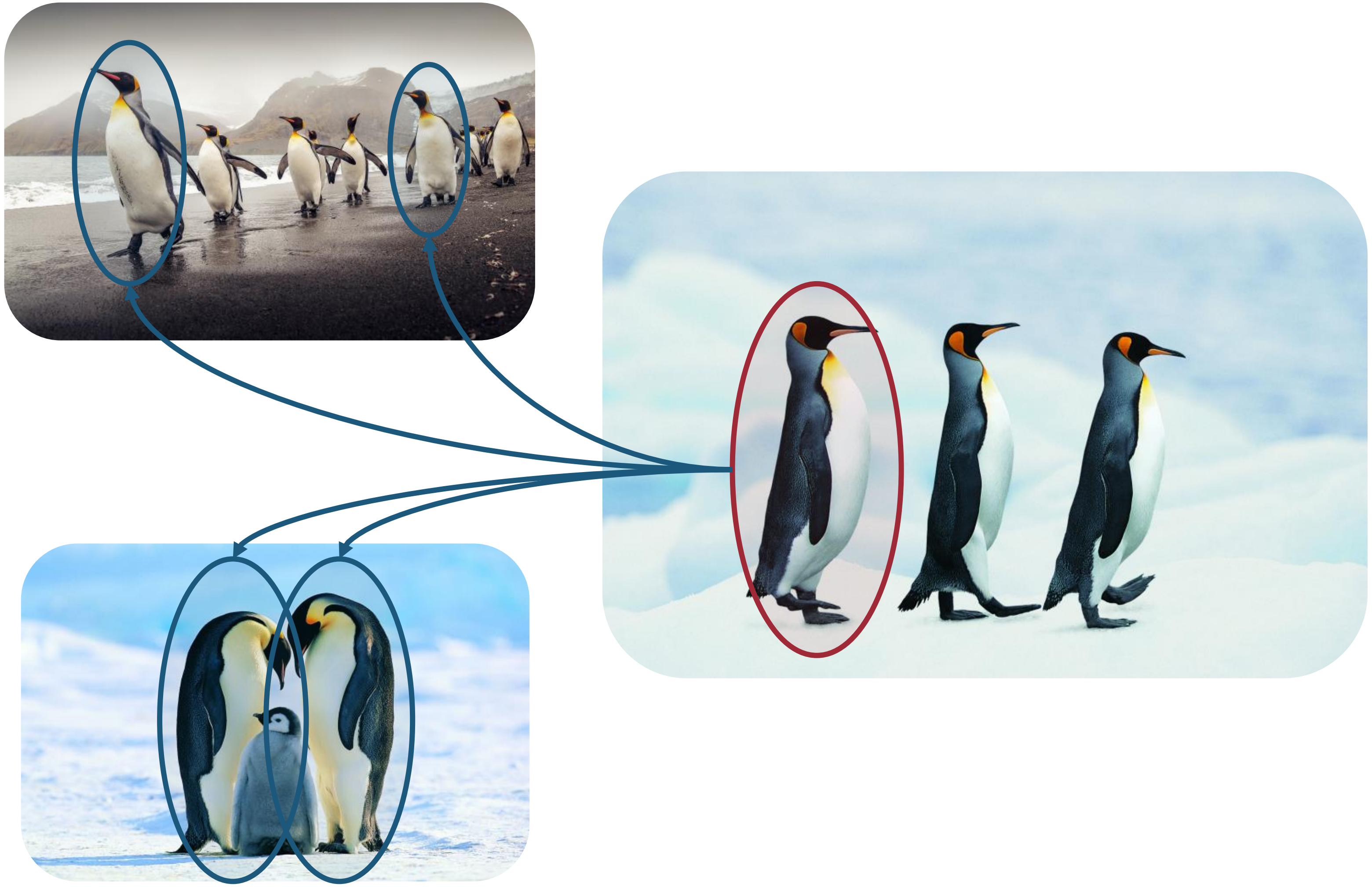
Convolutional Neural Networks (CNNs)

- **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**

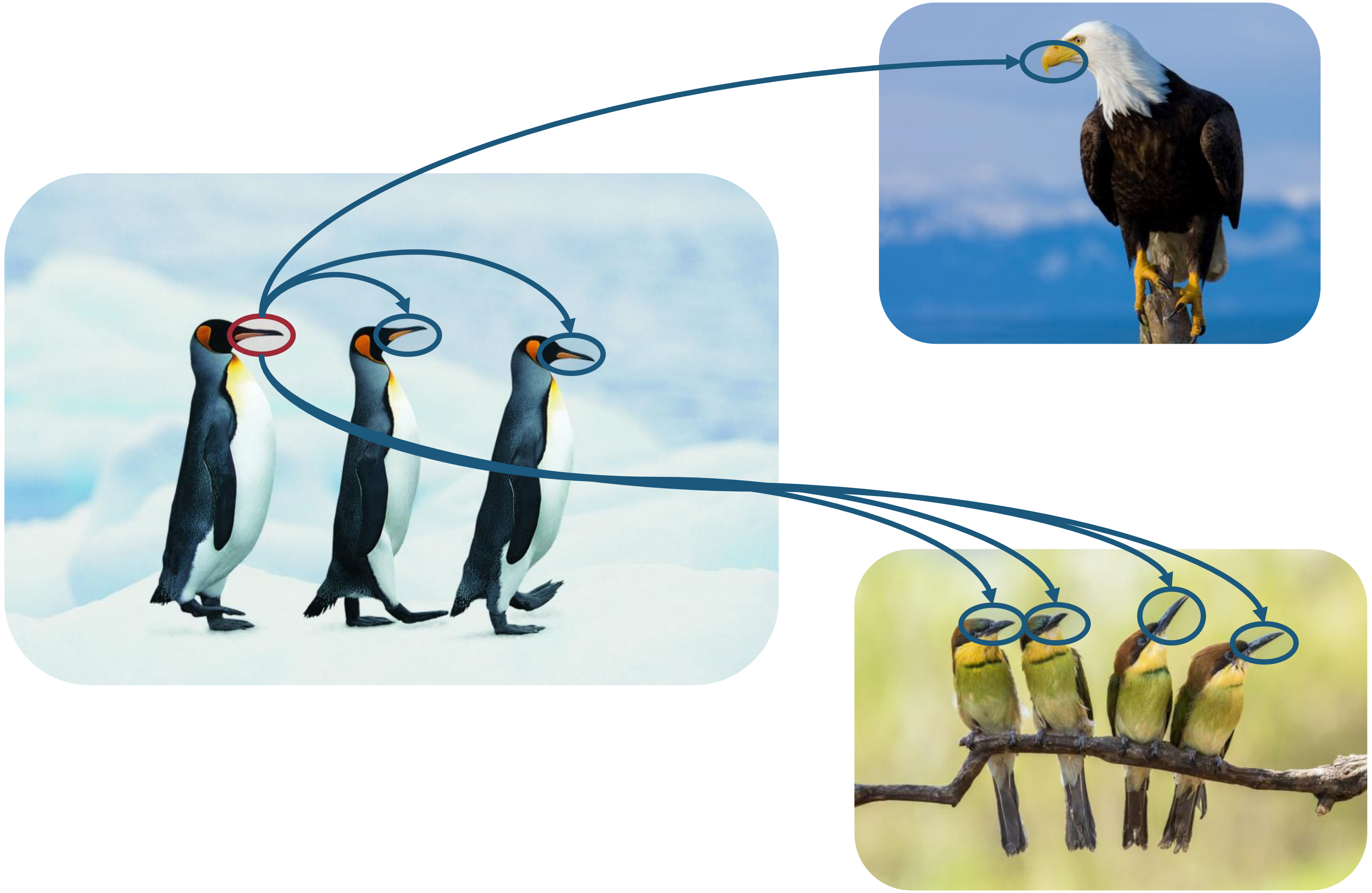
Reusable Pattern Detectors



Reusable Pattern Detectors

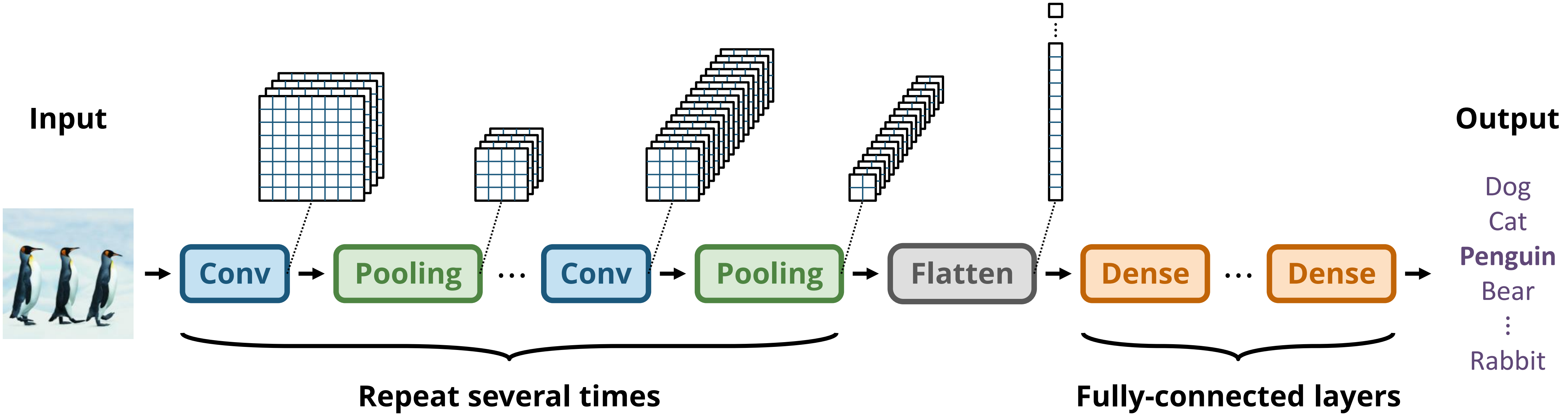


Reusable Pattern Detectors



Reusable Pattern Detectors

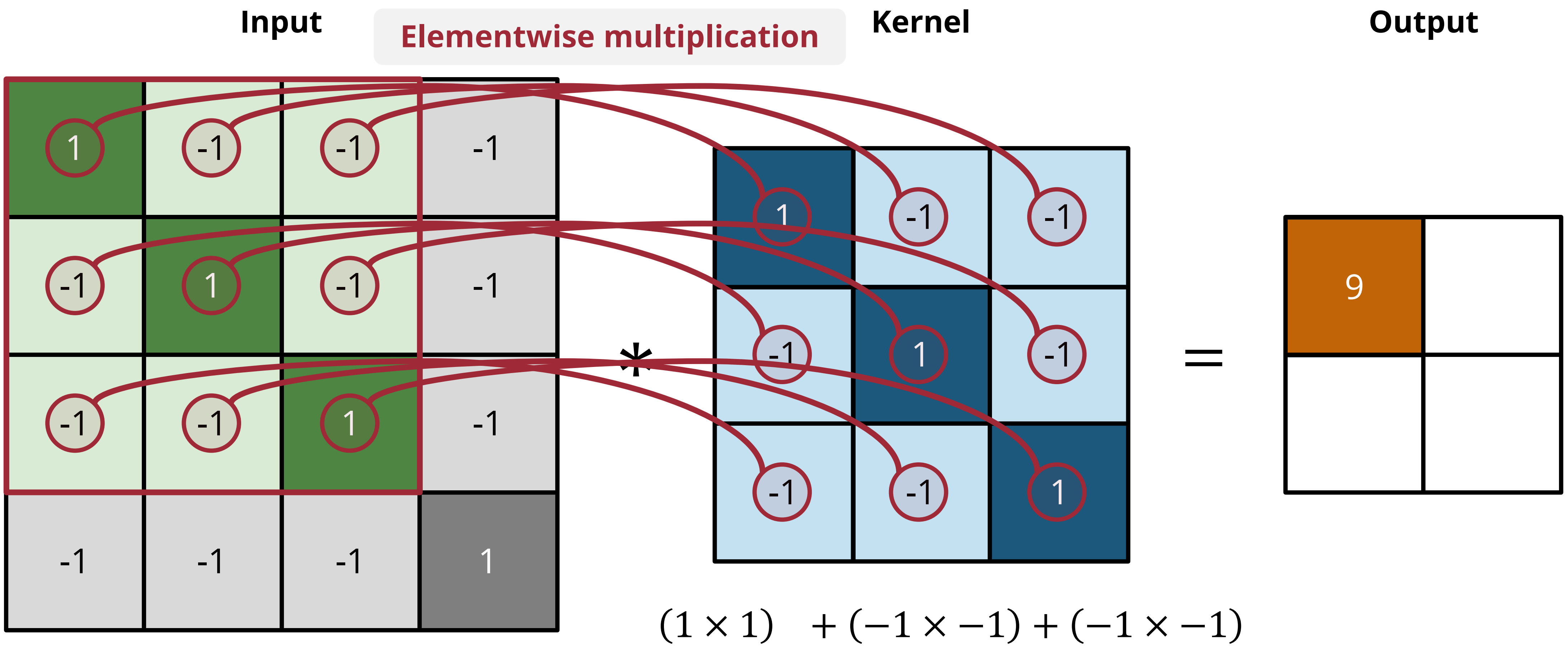
Convolutional Neural Network (CNNs)



2D Convolution

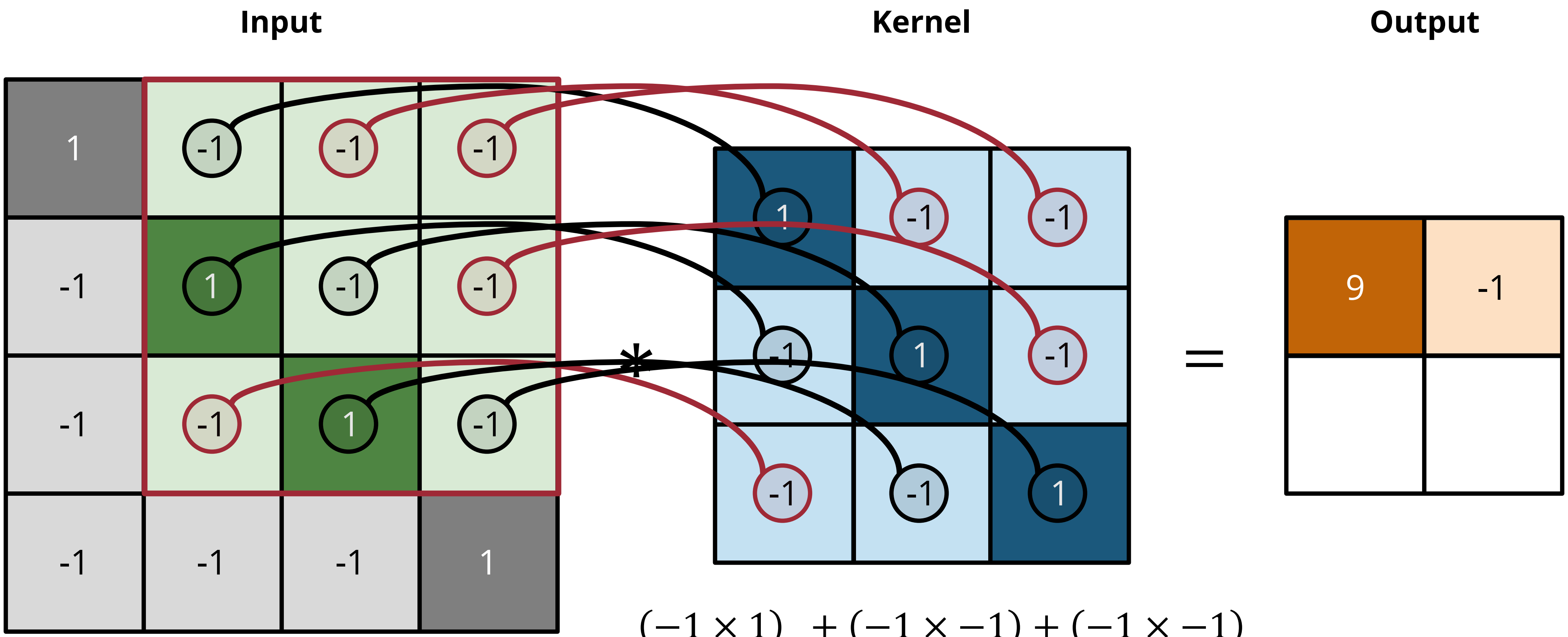
$$\underset{\textbf{Input}}{\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}} * \underset{\textbf{Kernel}}{\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}} = \underset{\textbf{Output}}{\begin{bmatrix} & \\ & \end{bmatrix}}$$

2D Convolution



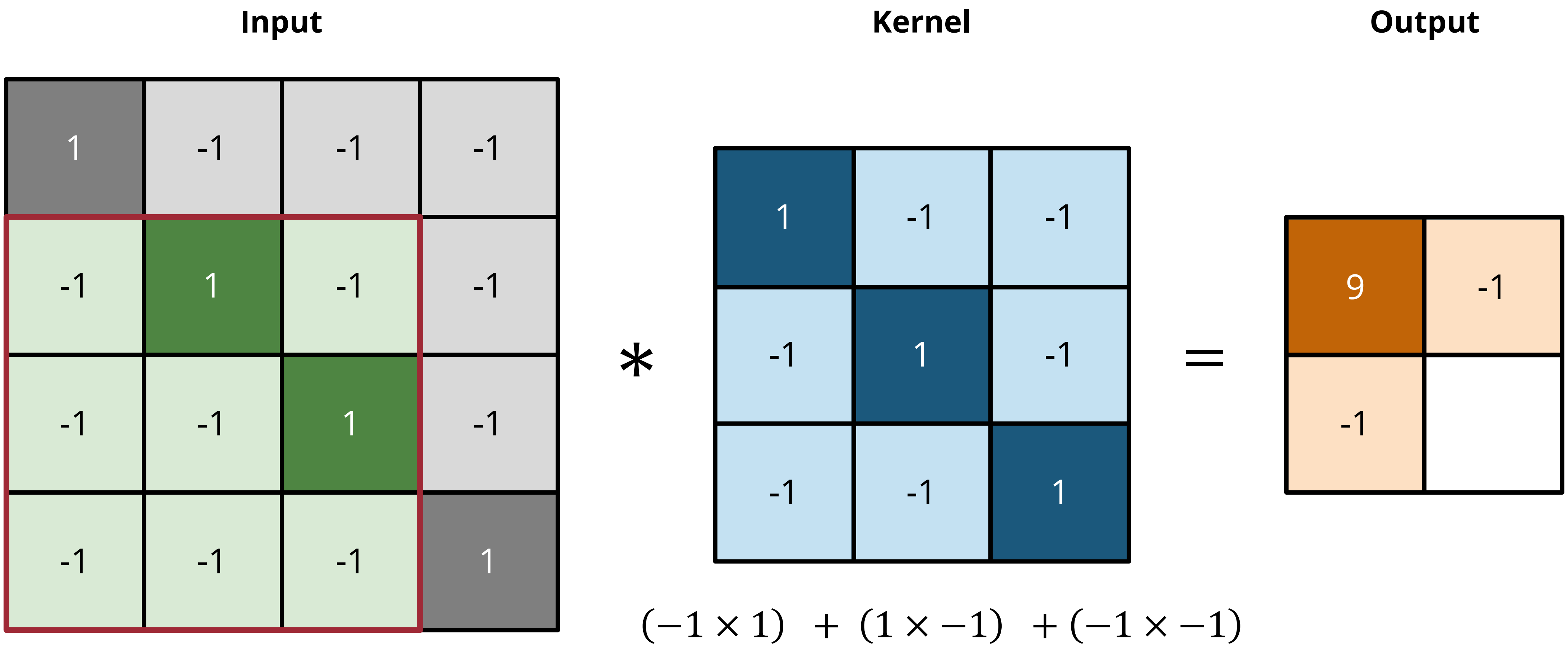
$$
\begin{aligned}
&(1 \times 1) + (-1 \times -1) + (-1 \times -1) \\
&+(-1 \times -1) + (1 \times 1) + (-1 \times -1) \quad = \mathbf{9} \\
&+(-1 \times -1) + (-1 \times -1) + (1 \times 1)
\end{aligned}
$$

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



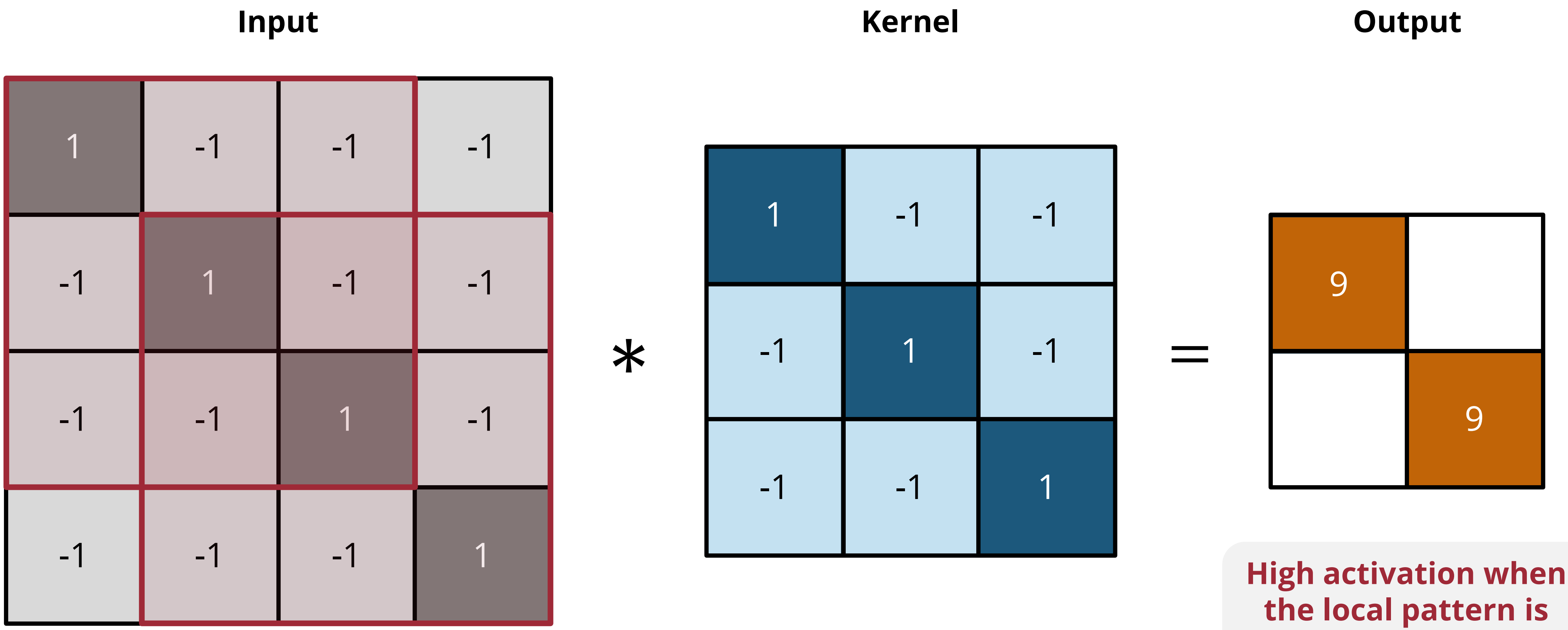
$$\begin{aligned}&(-1\times 1) + (1\times -1) + (-1\times -1)\\ &+(-1\times -1) + (-1\times 1) + (1\times -1)\\ &+(-1\times -1) + (-1\times -1) + (-1\times 1)\end{aligned} = \mathbf{-1}$$

2D Convolution

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

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

2D Convolution



High activation when the local pattern is close to the kernel

2D Convolution

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

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

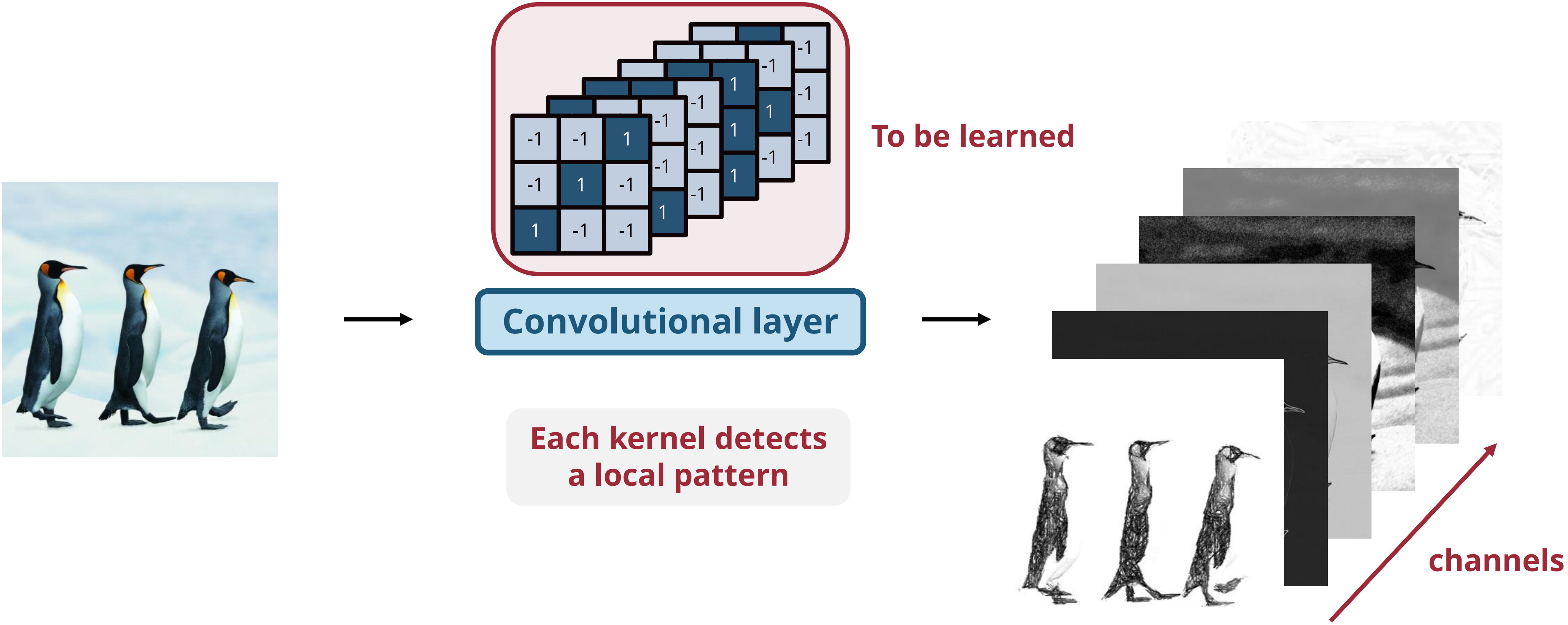
$$
\underset{\textbf{Input}}{\begin{bmatrix} -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}} * \underset{\textbf{Kernel}}{\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}} = \underset{\textbf{Output}}{\begin{bmatrix} 1 & 1 \\ 1 & 5 \end{bmatrix}}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

2D Convolution

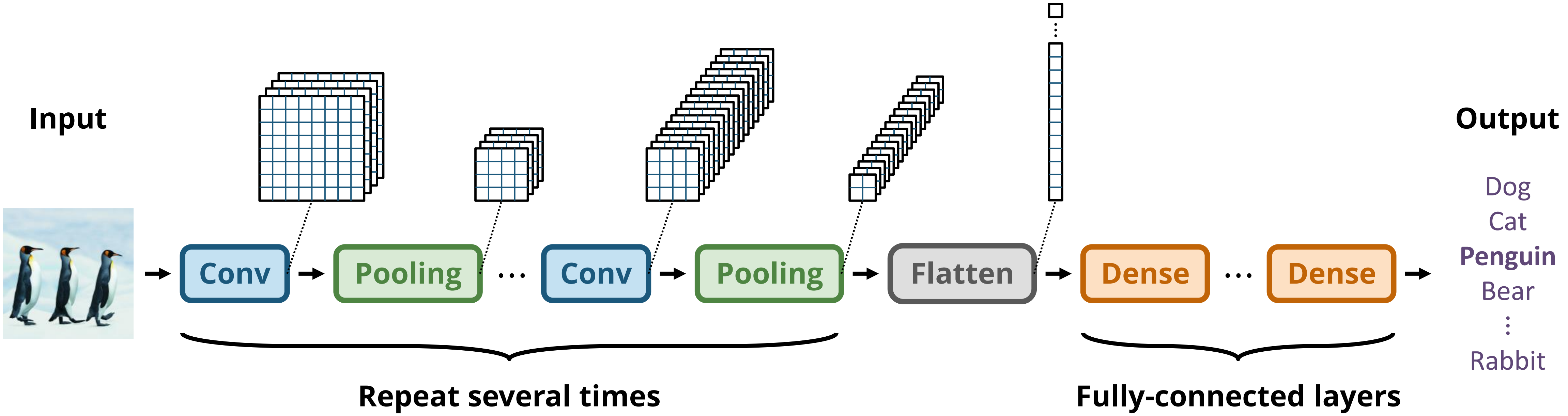
Input		Kernel		Output
$\begin{bmatrix} -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix}$	$*$	$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$	$=$	$\begin{bmatrix} 9 & -1 \\ -1 & 1 \end{bmatrix}$

Convolutional Layer

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



Convolutional Neural Network (CNNs)



Padding

`padding="valid"`

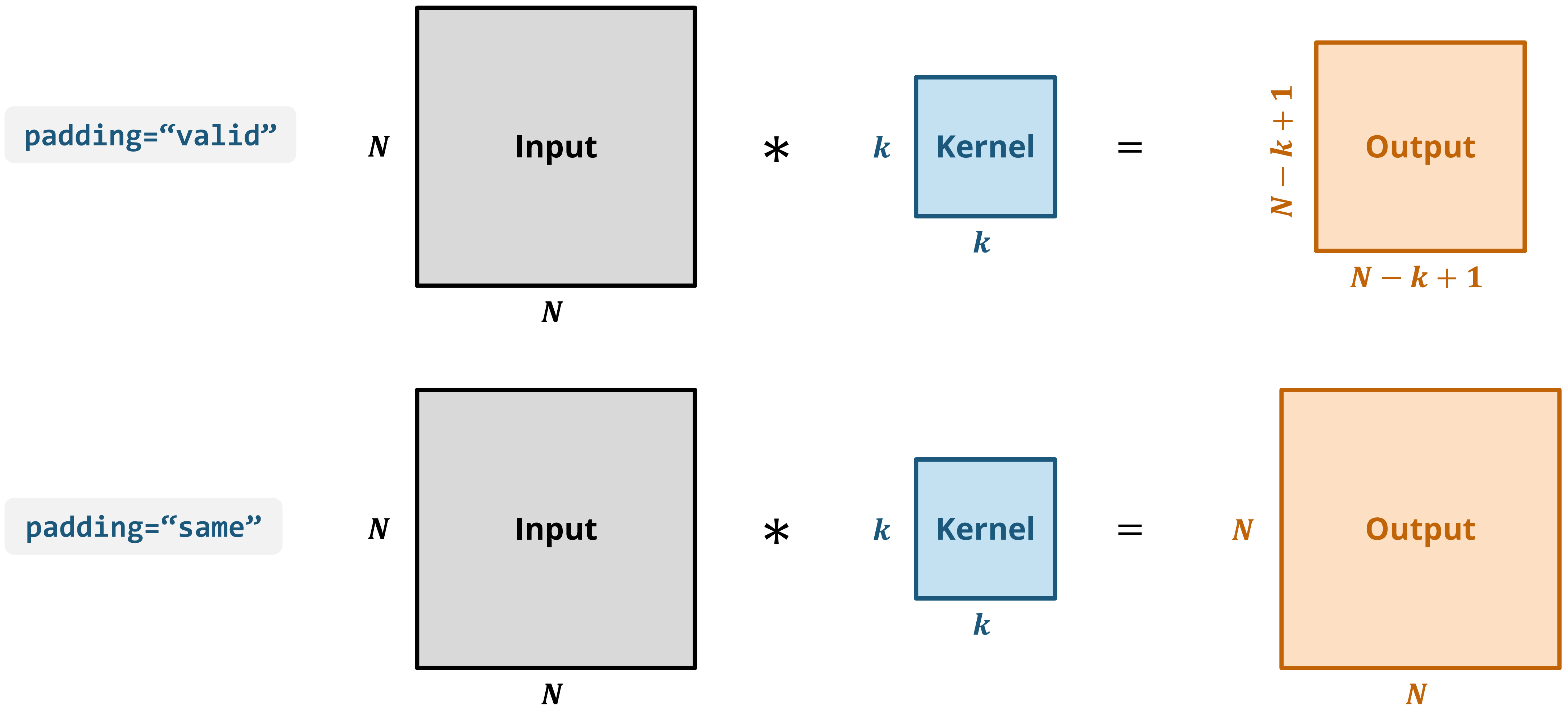
$$
\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}
*
\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}
=
\begin{bmatrix} 9 & -1 \\ -1 & 9 \end{bmatrix}
$$

`padding="same"`

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

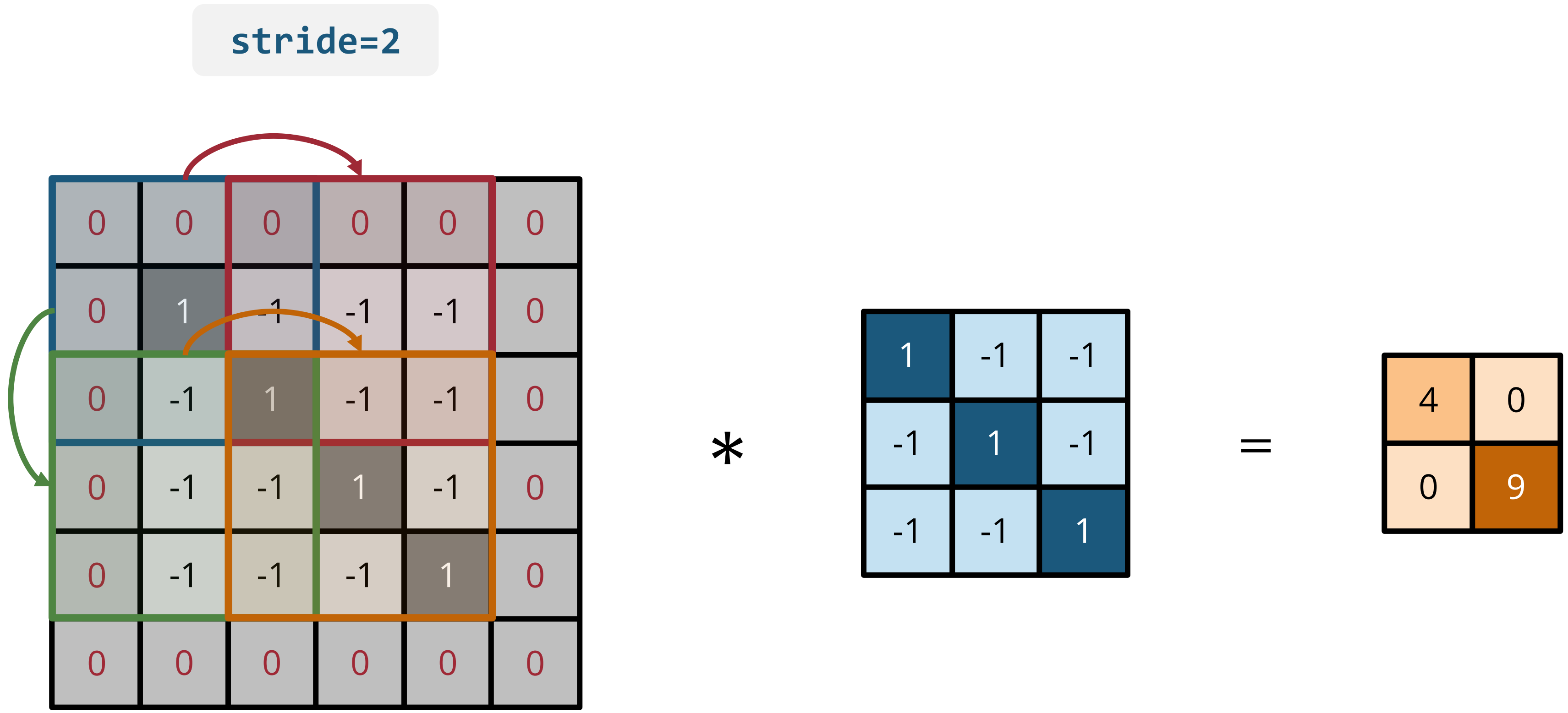
Keep the output of the same size as the input

Shapes



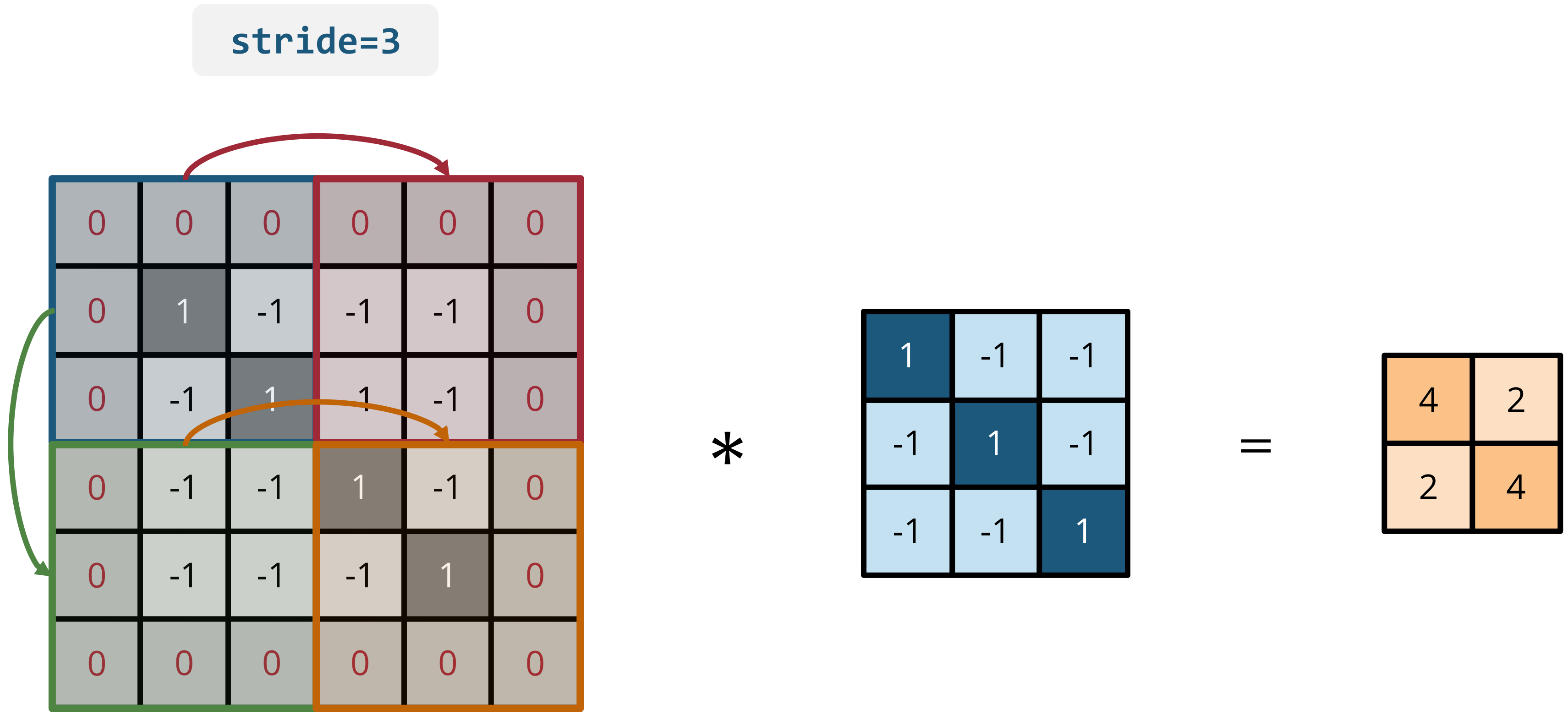
padding="valid"
N
Input
N
*
k
Kernel
k
=
N − k + 1
Output
N − k + 1
padding="same"
N
Input
N
*
k
Kernel
k
=
N
Output
N

Striding



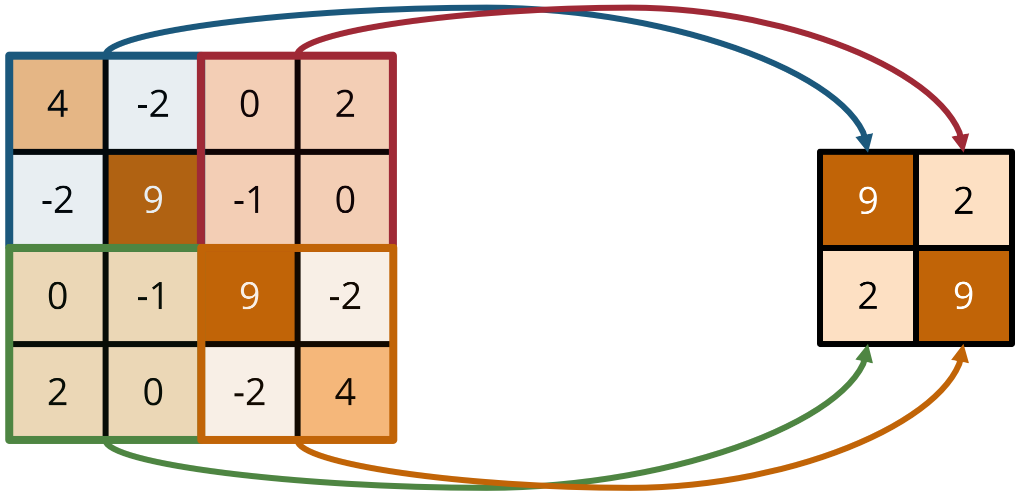
stride=2
0 0 0 0 0 0
0 1 -1 -1 -1 0
0 -1 1 -1 -1 0
0 -1 -1 1 -1 0
0 -1 -1 -1 1 0
0 0 0 0 0 0
*
1 -1 -1
-1 1 -1
-1 -1 1
=
4 0
0 9

Striding



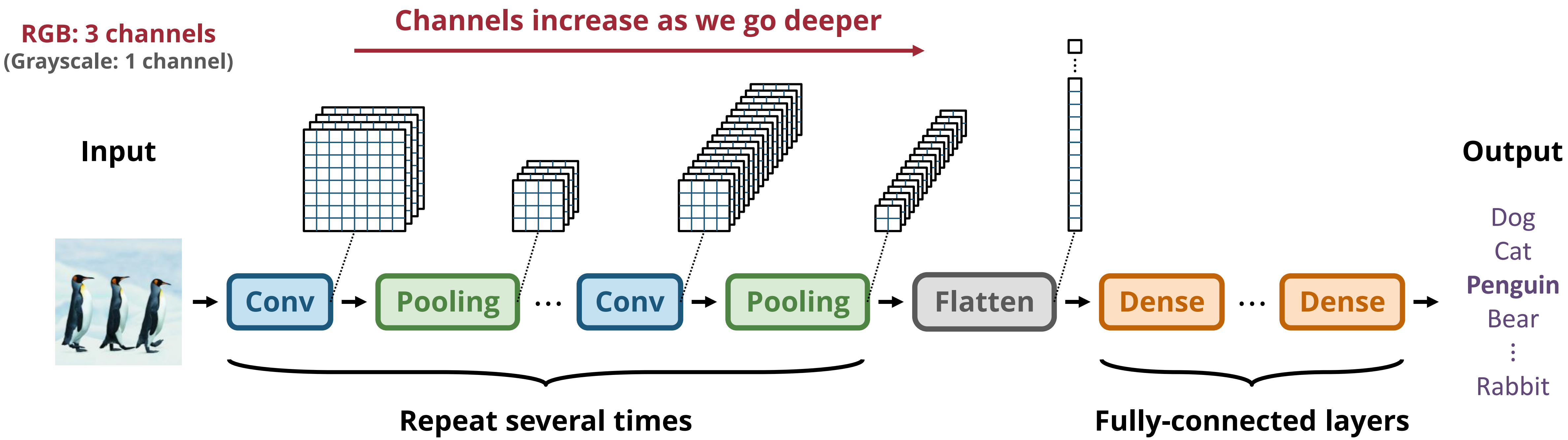
stride=3
0 0 0 0 0 0
0 1 -1 -1 -1 0
0 -1 1 -1 -1 0
0 -1 -1 1 -1 0
0 -1 -1 -1 1 0
0 0 0 0 0 0
*
1 -1 -1
-1 1 -1
-1 -1 1
=
4 2
2 4

Max Pooling Layer



Downsample and keep the strongest activation in each block

Convolutional Neural Network (CNNs)



A Real Example

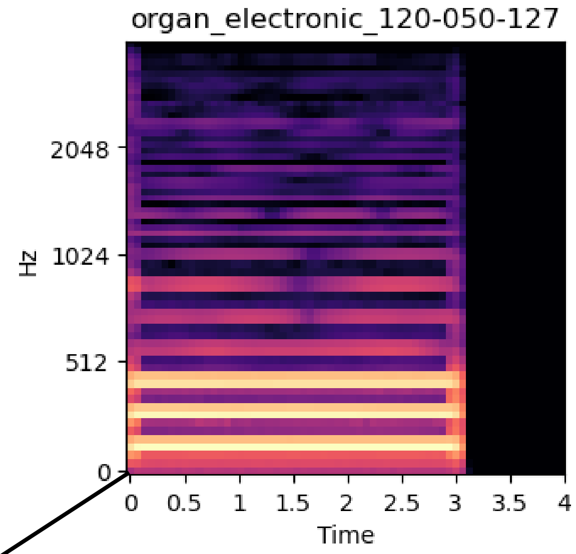
Input channels

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

```
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 x 3 x 1 + 1) x 16 = 160

(3 x 3 x 16 + 1) x 32 = 4640

(3 x 3 x 32 + 1) x 64 = 18496

(3 x 3 x 64 + 1) x 128 = 73856

(2048 + 1) x 11 = 22539

Benefits of CNNs

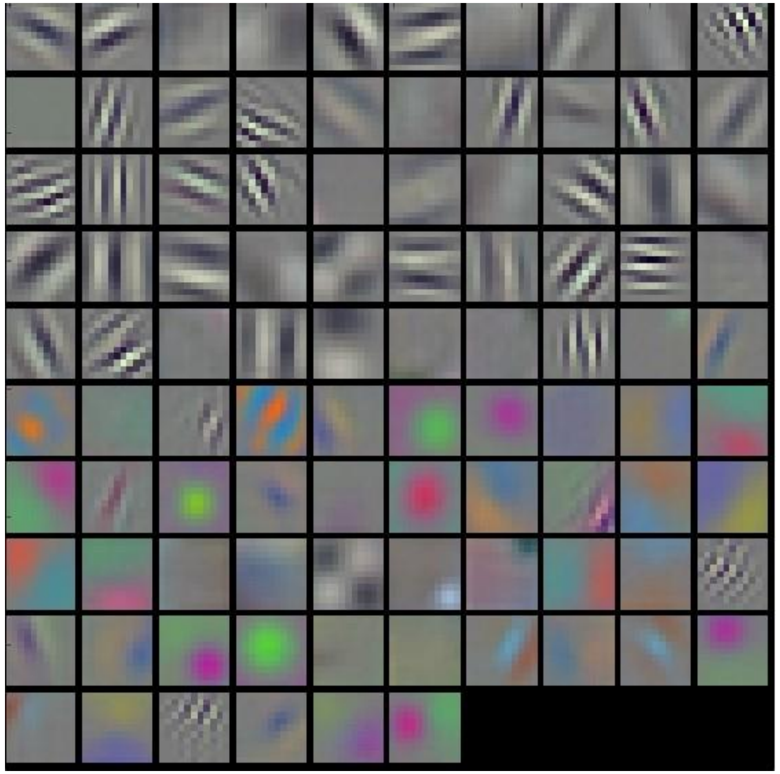
- Learn **local patterns**
- **Invariant to shifts**
  - Also called *translational invariance*
- **Reuse the learned filters** across
  - Different parts of the image
  - Across different images
- **Reduce complexity** against full-connected neural networks

What does a CNN Learn?

Learned CNN Kernels in a Trained AlexNet

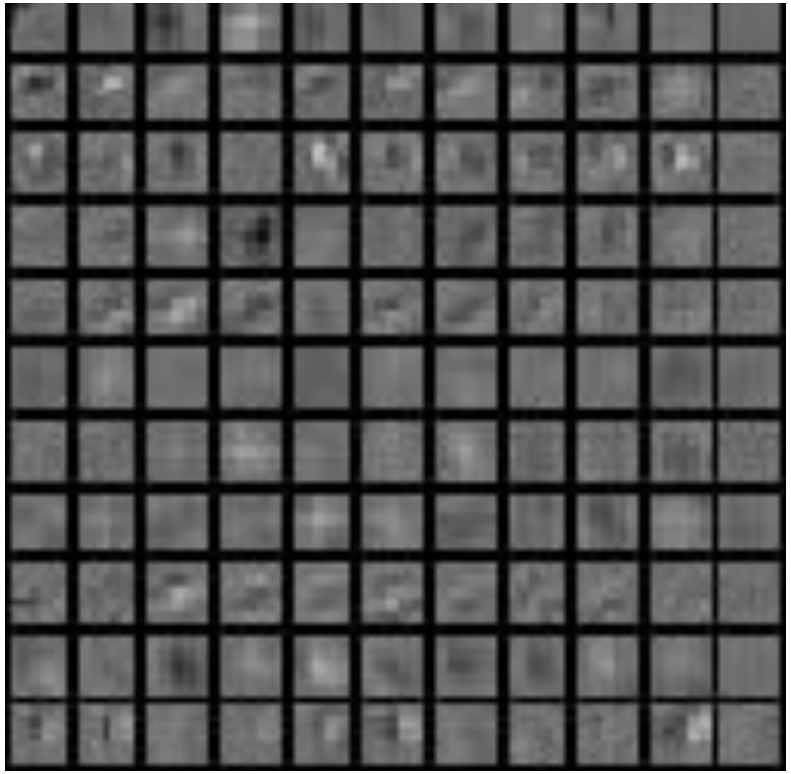
1st convolutional layer

11x11 kernels



2nd convolutional layer

5x5 kernels



cs231n.github.io/understanding-cnn/

Learned CNN Kernels in a Trained AlexNet

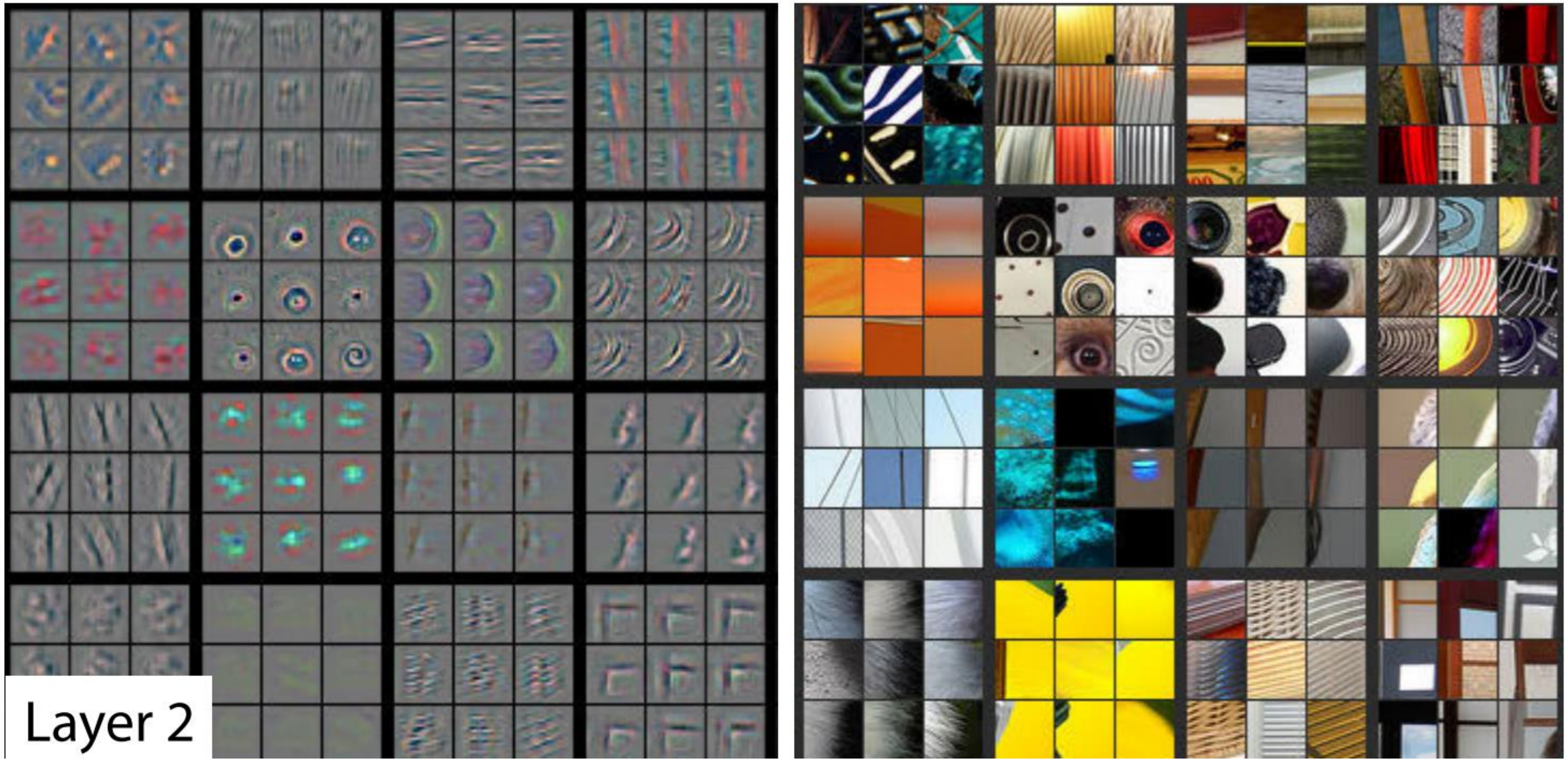
Layer 1

Learned CNN kernels

Top activations

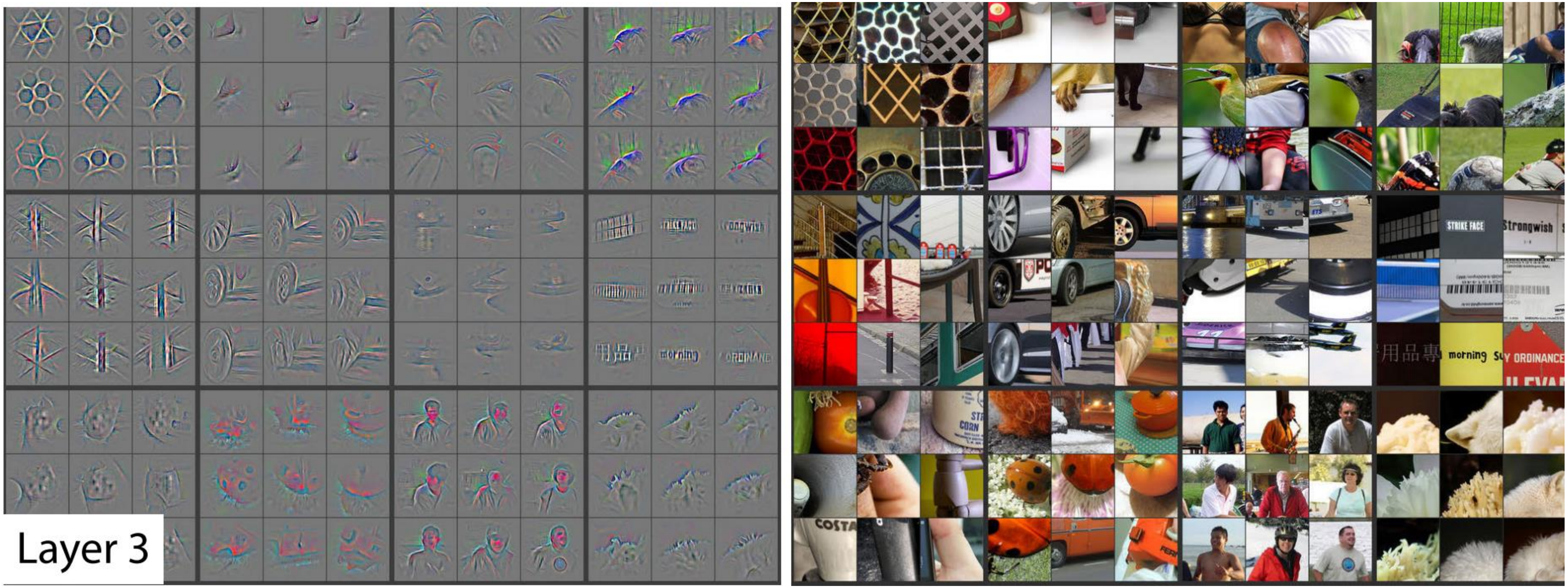
Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet



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

Learned CNN Kernels in a Trained AlexNet

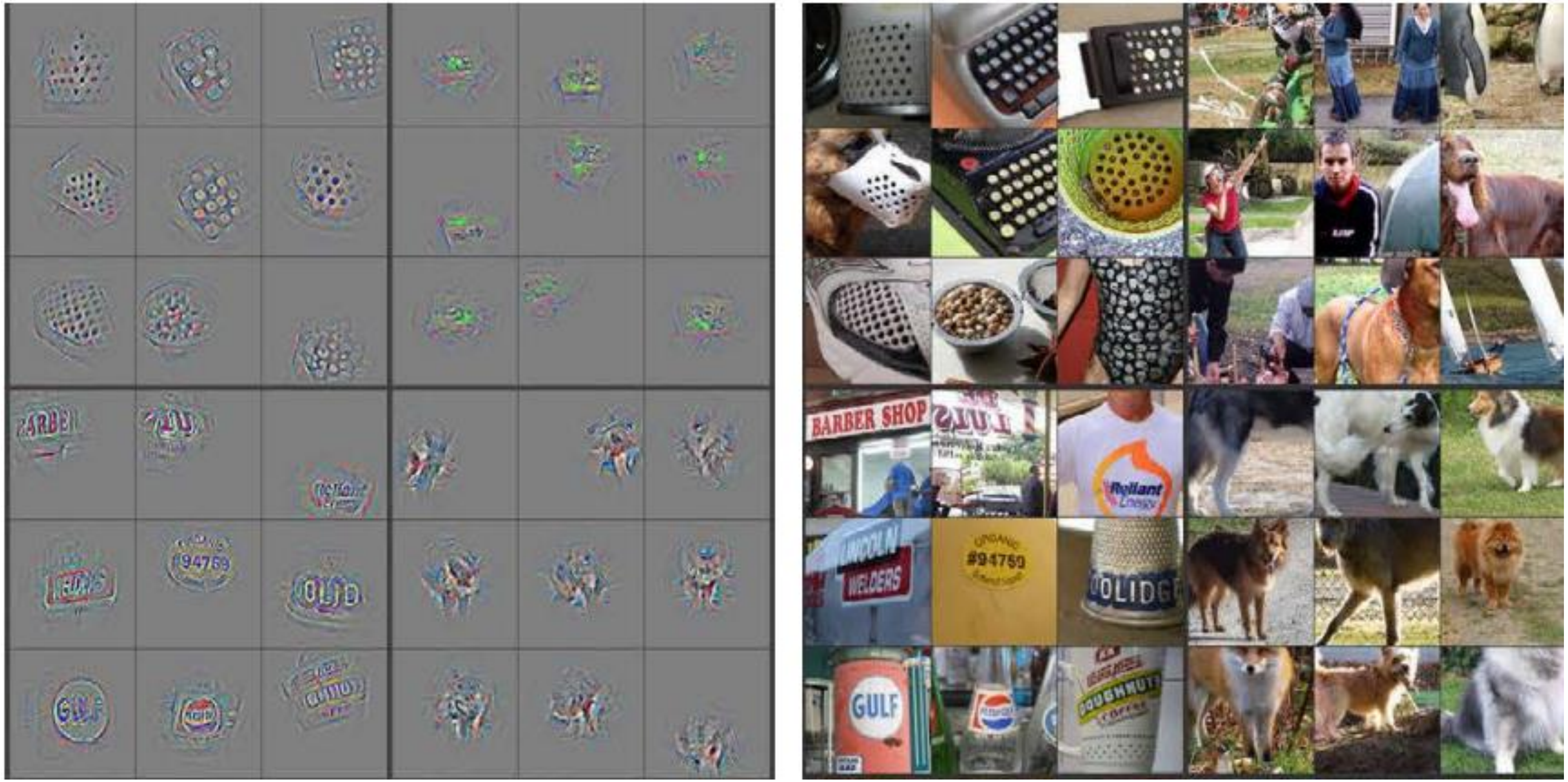


Matthew D. Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks,” *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet

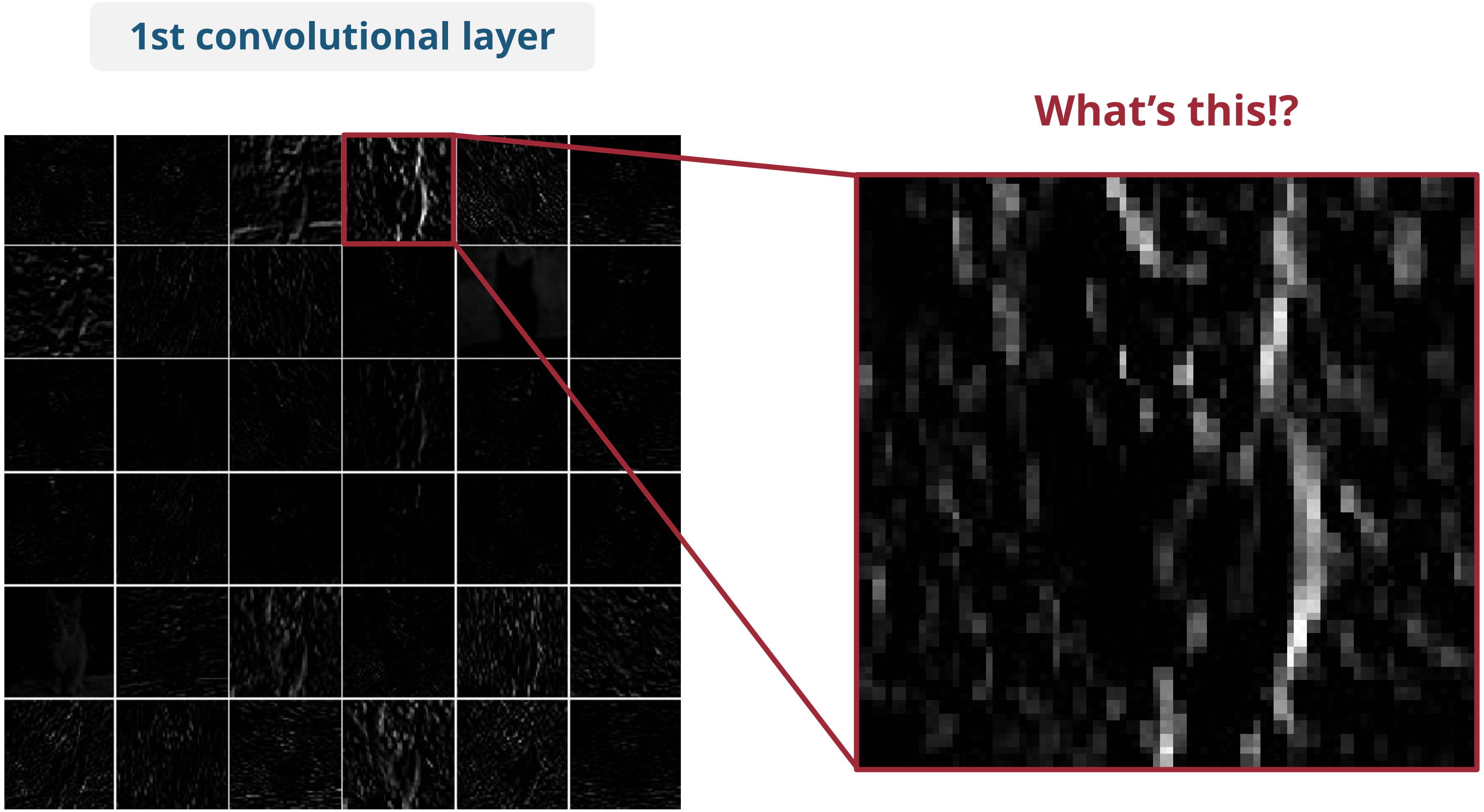
Matthew D. Zeiler and Rob Fergus, "Visualizing and Understanding Convolutional Networks," *ECCV*, 2014.

Learned CNN Kernels in a Trained AlexNet



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

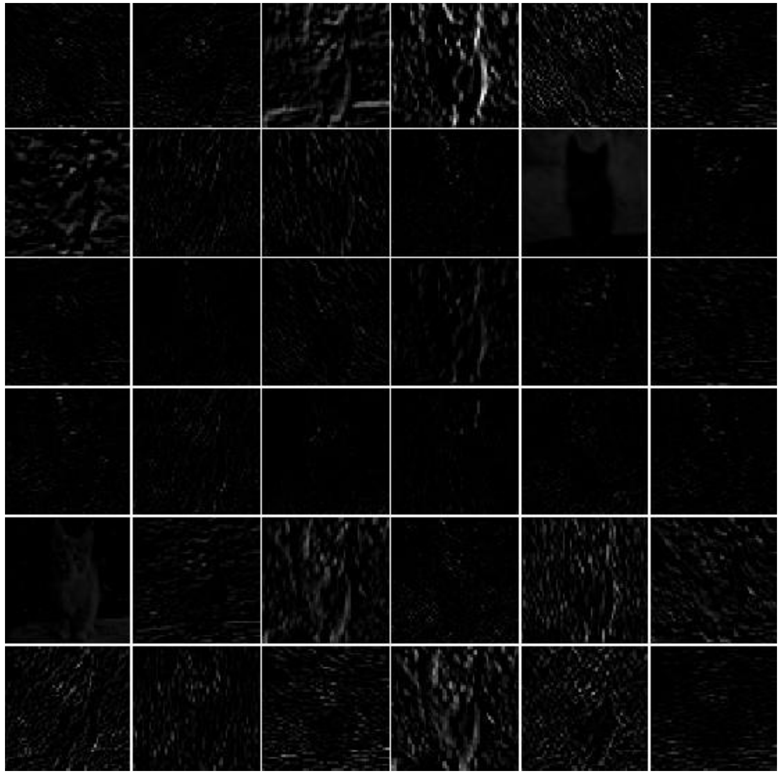
Activations in a Trained AlexNet



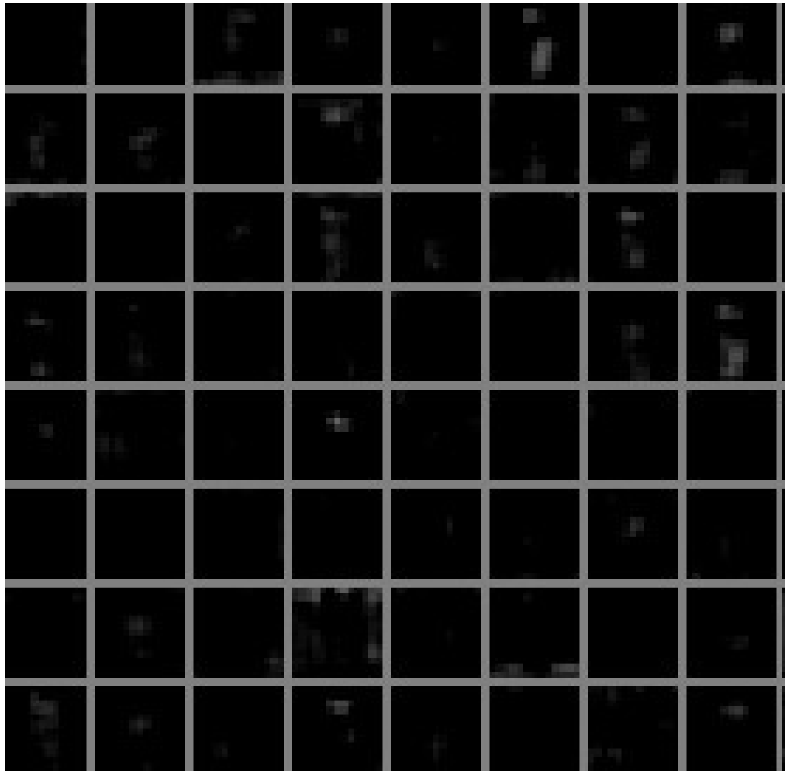
cs231n.github.io/understanding-cnn/

Activations in a Trained AlexNet

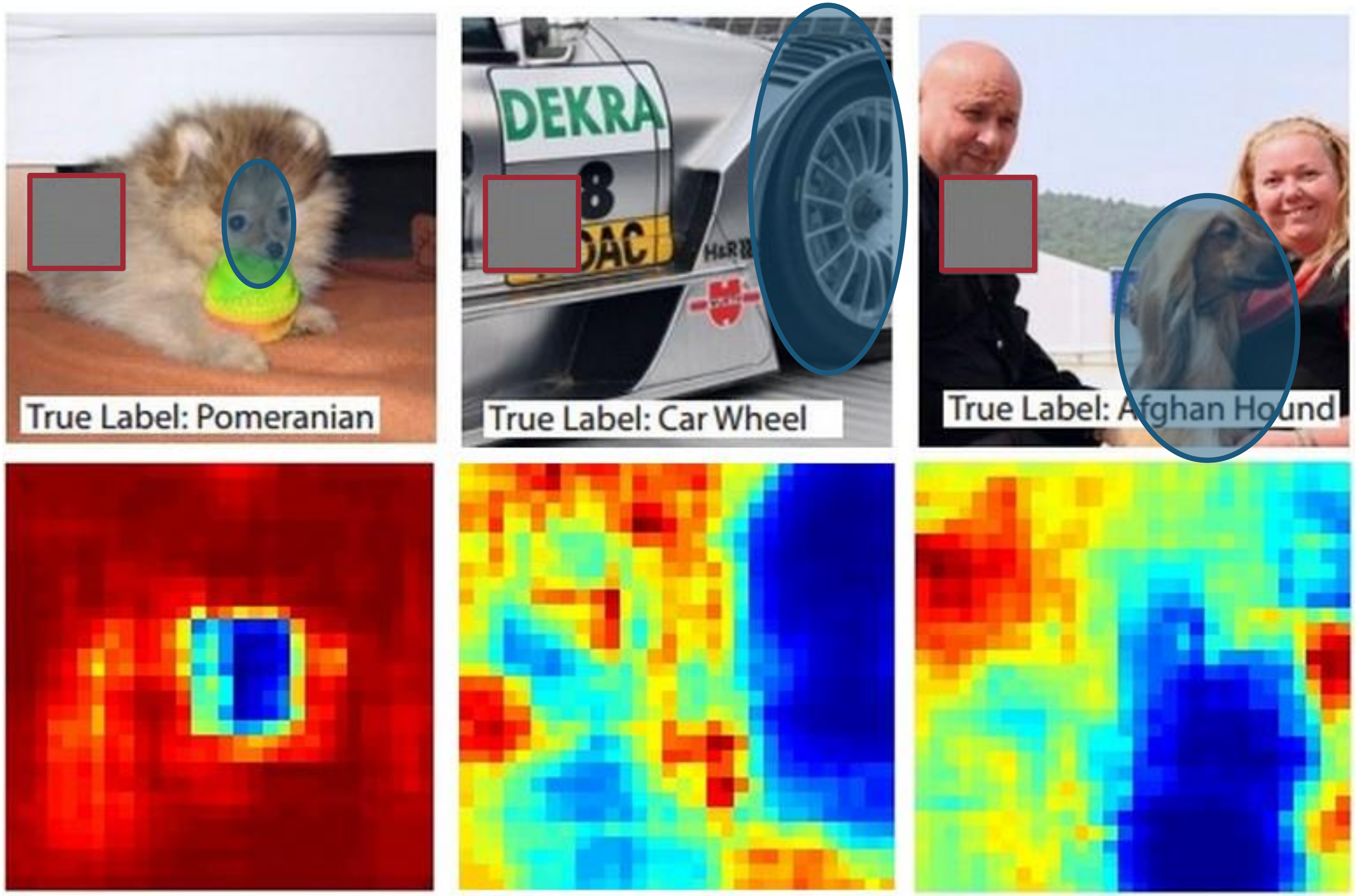
1st convolutional layer



5th convolutional layer



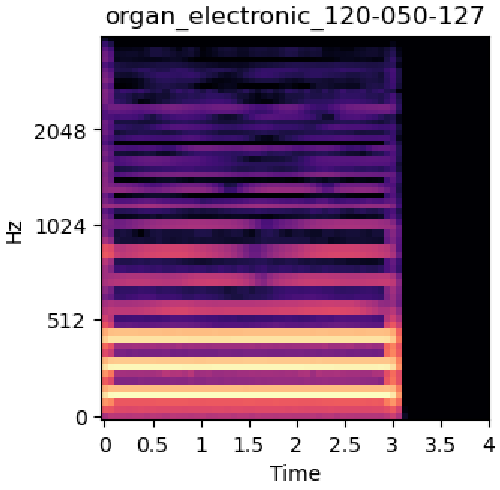
What does a CNN Learn?

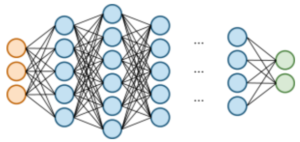


cs231n.github.io/understanding-cnn/

Assignment 2: 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)



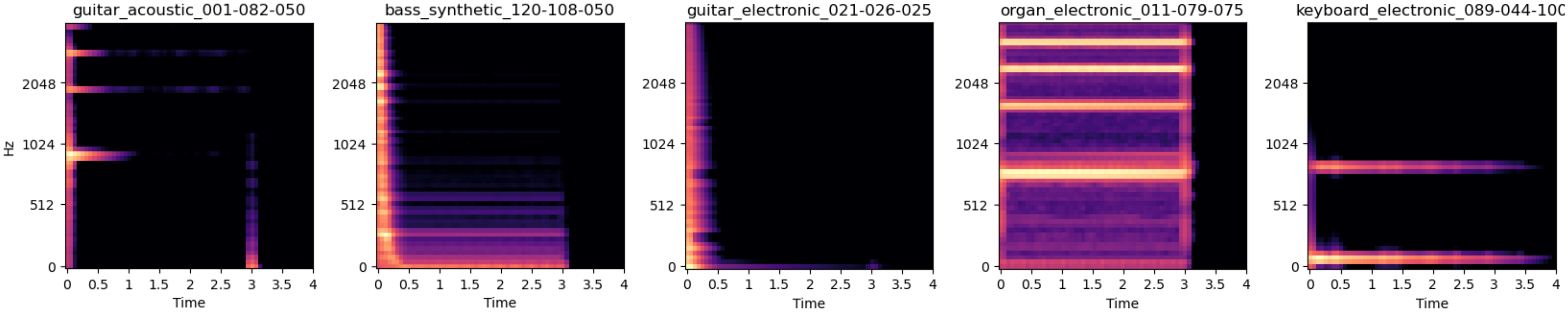


Bass
Brass
Flute
Guitar
Keyboard
Mallet
Organ
Reed
String
Synth Lead
Vocal

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.

Assignment 2: Musical Note Classification using CNNs

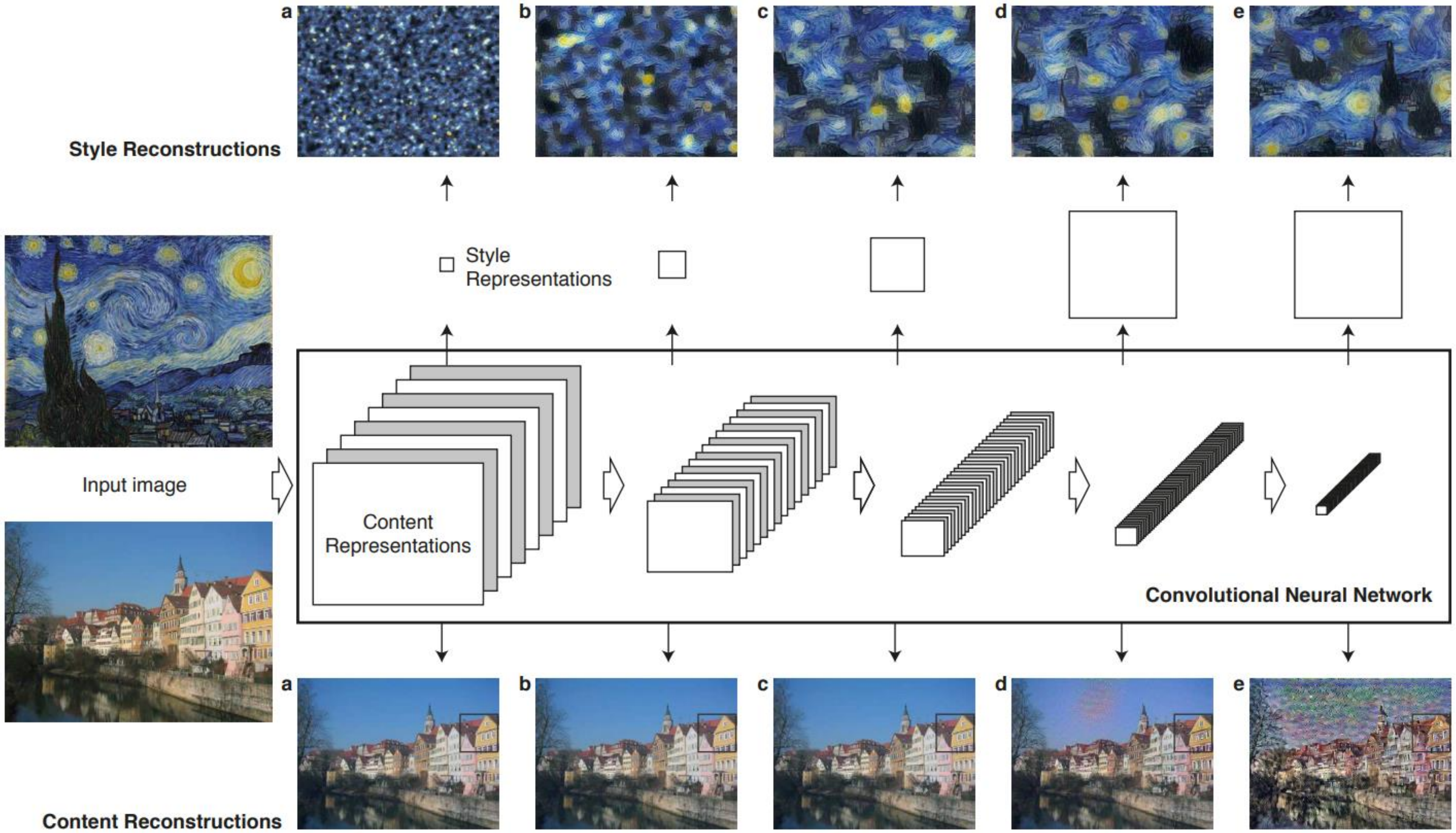
- Instructions will be released on Gradescope
- Due at **11:59pm ET** on **October 7**
- Late submissions: **3 point deducted per day**

Great Lakes

- **Great Lakes** is a high-performance computing cluster at U-M
- You will be provided **3000 CPU hours** (**~400 GPU hours**)
- Before you access Great Lakes, you'll need to first **create an HPC login**!
- **U-M VPN** is required to access the web portal off-campus

Neural Style Transfer

Neural Style Transfer



Leon A. Gatys, Alexander S. Ecker and Matthias Bethge, "Image Style Transfer Using Convolutional Neural Networks," *CVPR*, 2016

Neural Style Transfer – Examples

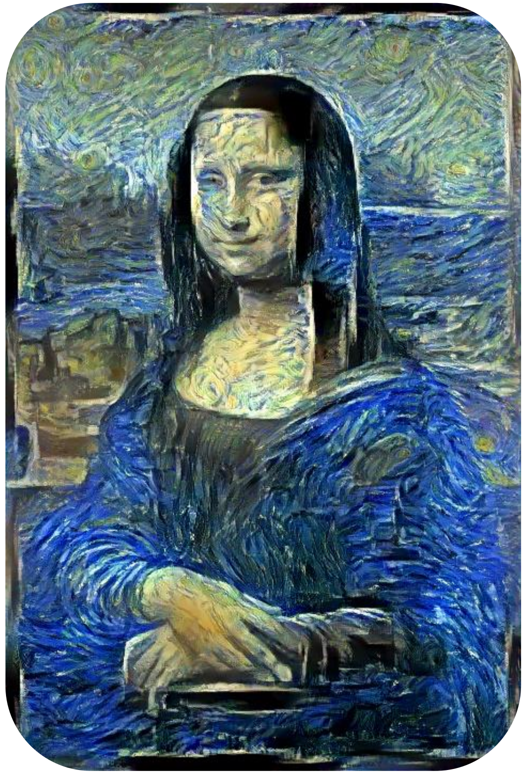
Content

+

Style

→

Output



github.com/PJ-Finlay/Neural-Style-Transfer-Images

Neural Style Transfer – Examples

Content

+

Style

→

Output

github.com/PJ-Finlay/Neural-Style-Transfer-Images

Neural Style Transfer – Examples

Content

+

Style

→

Output

github.com/PJ-Finlay/Neural-Style-Transfer-Images

Deep Dream

Deep Dream

- **Adjust the input image** so that it **maximizes the activation of a certain neuron**

github.com/PJ-Finlay/pytorch-deepdream

Deep Dream – Examples

research.google/blog/inceptionism-going-deeper-into-neural-networks/