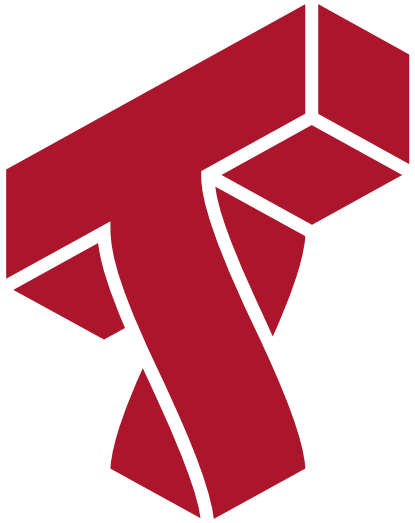


Lecture 17: Deep Learning

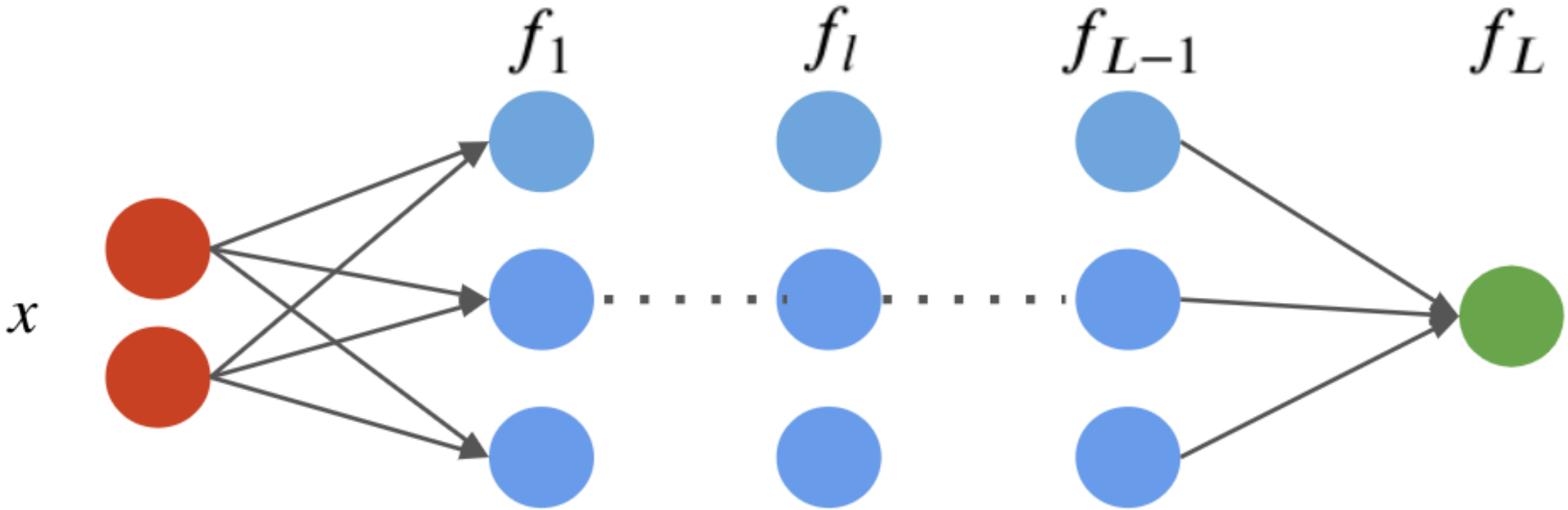
Part 1: What is Deep Learning?

Deep learning is a subfield of machine learning closely tied to neural networks.

Let's find out what deep learning is, and then we will see some deep learning algorithms.

Review: Neural Networks

A (fully connected) neural network is a model $f : \mathbb{R} \to \mathbb{R}$ that consists of a composition of L neural network layers:



What is Deep Learning?

Deep learning is a modern evolution of the field of artificial neural networks that is defined by three key features:

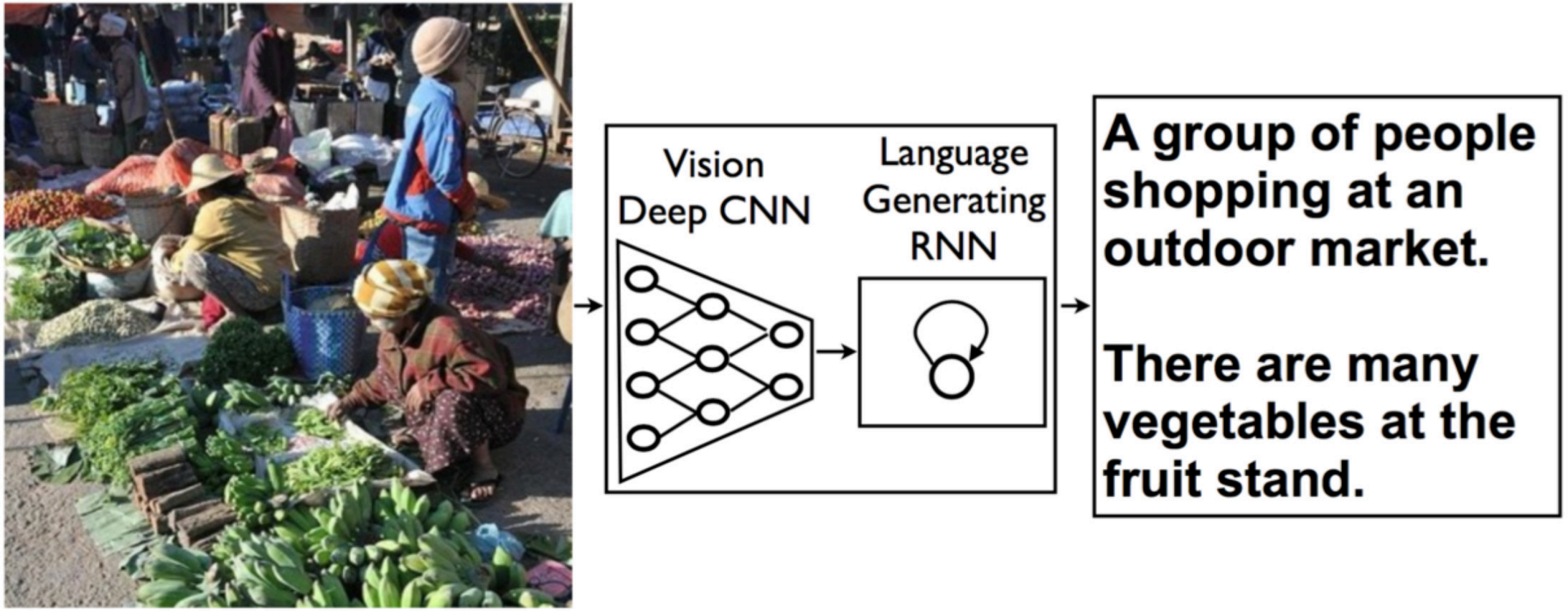
- Neural networks with a very large depth (up to 100's of layers)
- Large unstructured datasets, especially images, text, and audio
- The use of modern computational resources, like GPUs

Expressivity of Deep Models

Deep neural networks are very powerful models in part because they can represent well complex mappings between x and y.

- Shallow networks can represent any function, but need very large hidden layers.
- Deep networks can represent very complex $\mathcal{X} \rightarrow \mathcal{Y}$ mappings with fewer parameters.

In practice, deep neural networks can learn very complex mappings such as $\text{image} \rightarrow \text{text description}$ that other algorithms cannot.



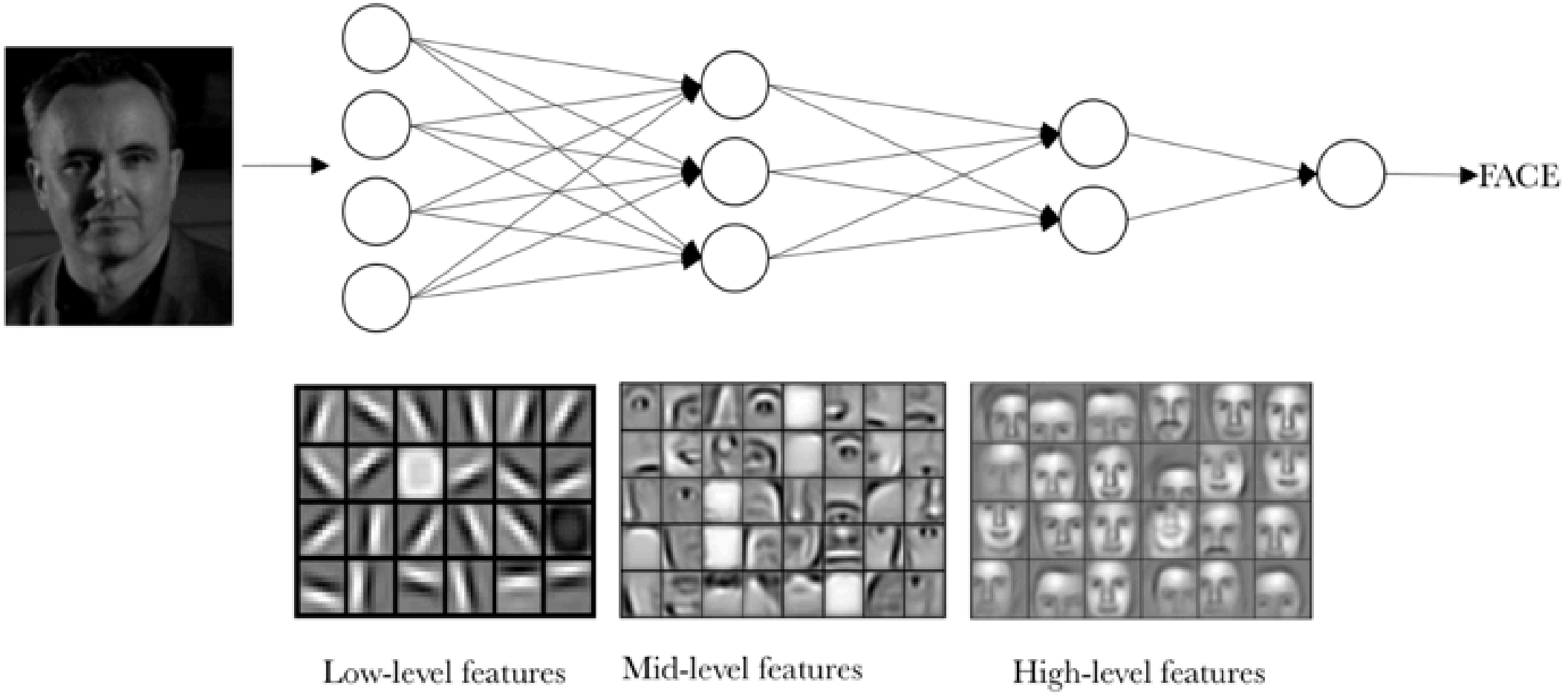
Ref: Learning CNN-LSTM Architectures for Image Caption Generation; https://cs224d.stanford.edu/reports/msoh.pdf

Representation Learning

How does a deep neural network use its representational power?

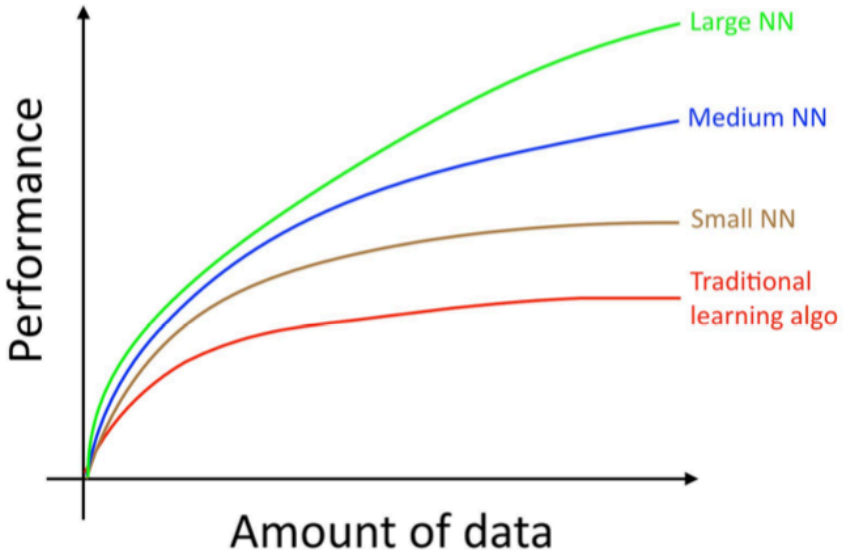
- The sequential layers can learn to represent data at an increasing level of abstraction
- This can also be interpreted as learning: each layer maps input to a more abstract feature space that is *learned* from data.

This is an example of representations that a deep neural network learns from data.



Scaling to Large Datasets

Deep neural nets also scale to very large datasets (figure by Andrew Ng).

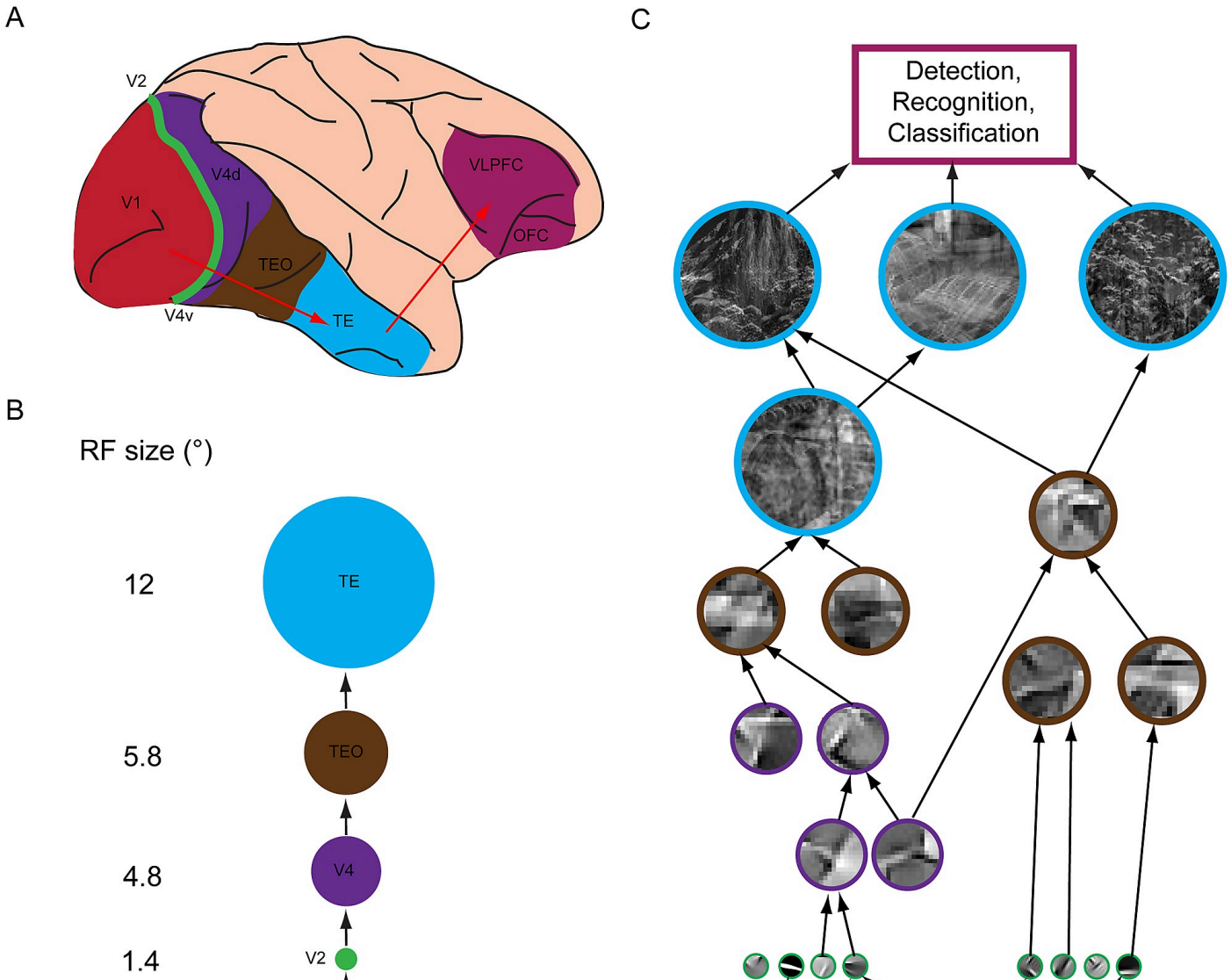


Classical algorithms like linear regression saturate after a certain dataset size. Deep learning models keep improving as we add more data.

Connections to Neuroscience

As artificial neurons are designed to mimic the biological ones, deep neural networks have strong connections with biological neural systems.

There is ample evidence that deep neural networks perform computation very similar to that of the visual cortex (Kravitz el al.).



Successes of Deep Learning

Deep learning has been a major breakthrough in machine learning in the last decade.

- It has dramatically improved common technologies like speech recognition or machine translation.
- It has enabled new technologies that captured the popular imagination.

These images were generated by a deep generative model from a short text prompt:

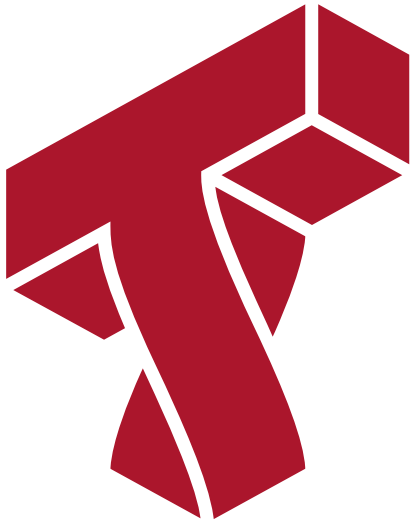
e.g., "*kneeling cat knight, portrait, finely detailed armor, intricate design, silver, silk, cinematic lighting, 4k, --ar 9:16 --beta --upbeta*"

Pros and Cons of Deep Neural Networks

Deep neural nets (DNNs) are among the most powerful ML models.

- DNNs set a new state-of-the-art in many application areas.
- They can be combined with many algorithms (supervised, unsupervised), and significantly improve them.
- Many specialized software and hardware platforms for DNNs.

However, DNNs can be slow and hard to train and require a lot of data.

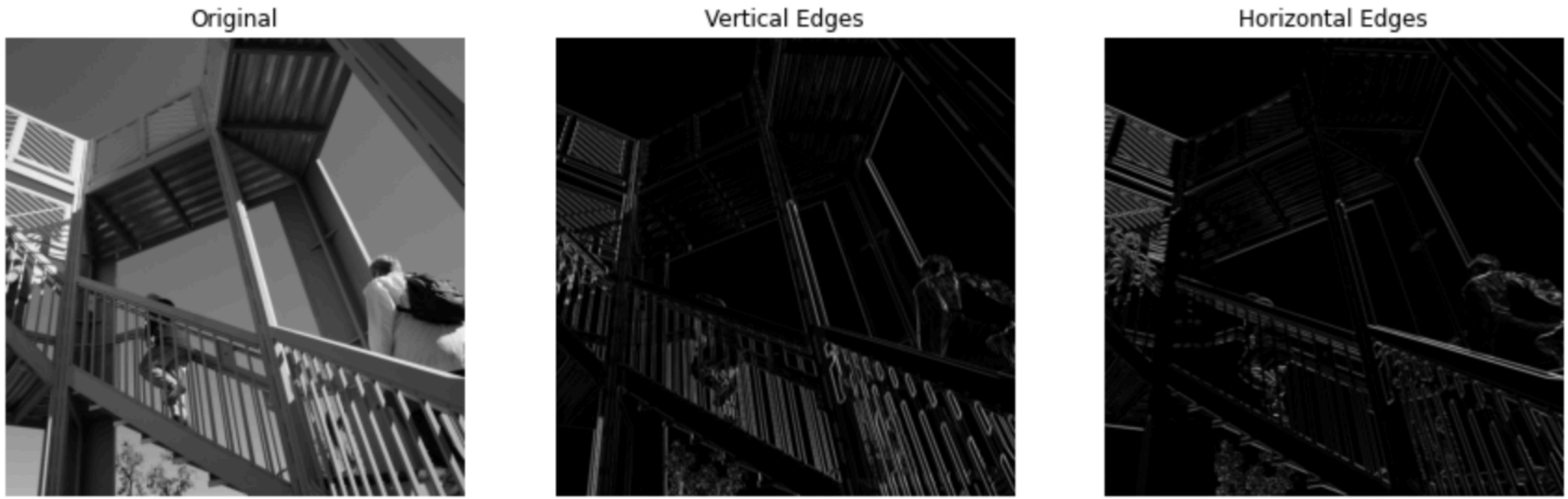


Part 2: Convolutions and Pooling

These basic operations are the building blocks of modern convolutional neural networks.

Motivation: Edge Detection

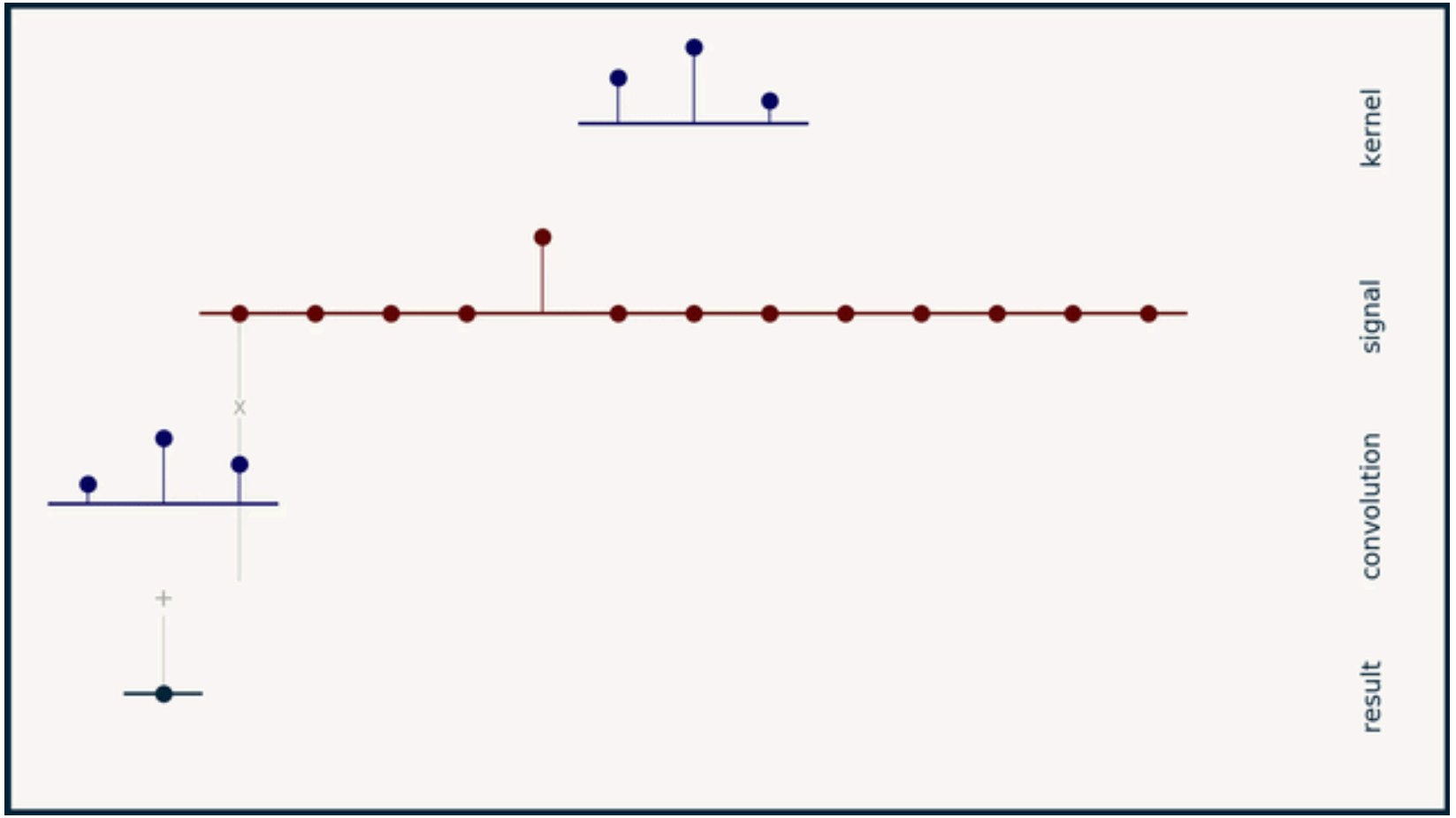
Consider the problem of detecting edges in an image.



This can be useful for detecting or classifying objects in a scene.

Convolution: Intuition

Here, we convolve a 3x3 *filter* with a 5x5 *signal* to obtain an *activation map*.



Convolution: Definition

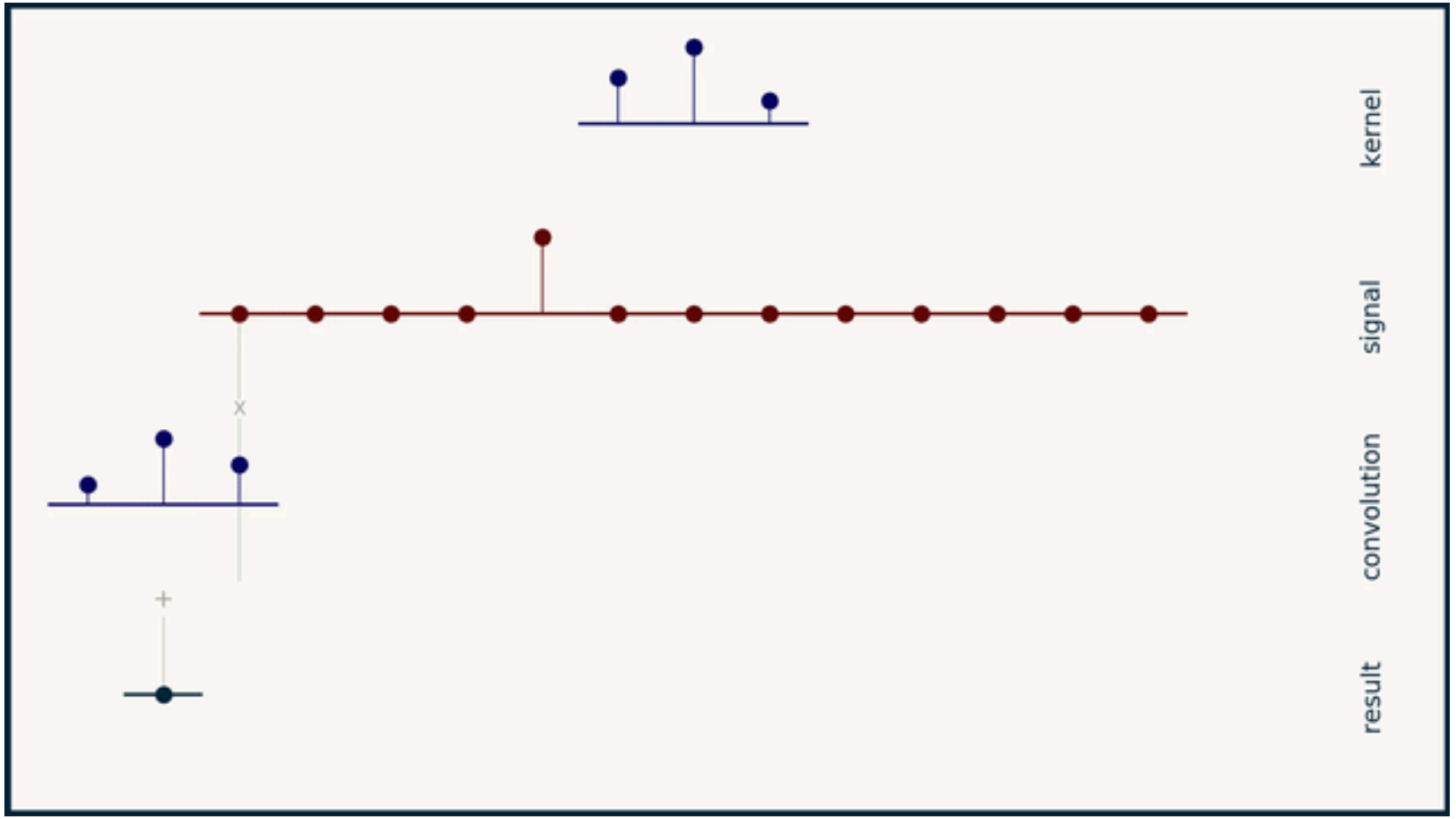
Let $f \in \mathbb{R}^p$ and $x \in \mathbb{R}^d$ be two vectors, called the *filter* (or *kernel*) and the *signal* respectively. Typically, $p < d$.

In deep learning, a convolution $(f * x) \in \mathbb{R}^d$ is typically defined as

$$(f * x)[i] = \underbrace{\sum_{j=1}^{p} f[j]x[i+j]}_{\text{dot product of } f \text{ with part of } g} = x[i:i+p]^\top f \tag{1}$$

where $x[j] = 0$ when $j \leq 0$ or $j > d$.

This is best understood via a picture (credit: Brandon Rohrer)



Notice how the output has large values at locations where the dot product between the kernel and the signal is large.

On a small technical note, what we have defined is called the cross-correlation in mathematics.

The convolution is technically defined as $\sum_{j=1}^{p} f[j]x[i-j]$, but in deep learning both formulas effectively give the same results and the cross-correlation is used in practice (and is called "convolution").

We can implement a convolution in Python as follows.

```
# A naive (*slow*) implementation of convolution with loops, don't use this in practice!
def conv_naive(f, x):
    P = len(f)
    D = len(x)
    result = [0]*D
    for d in range(D):
        for p in range(P):
            if d-p>=0 and d-p<D:
                result[d] += f[p]*x[d-p]
    return result[P-1:] #only return valid results

# numpy also provides convolution function np.convolve(f,g,'valid')
```

Example: Edge Detection

To gain more intuition about convolution, let's look at another an in 1D.

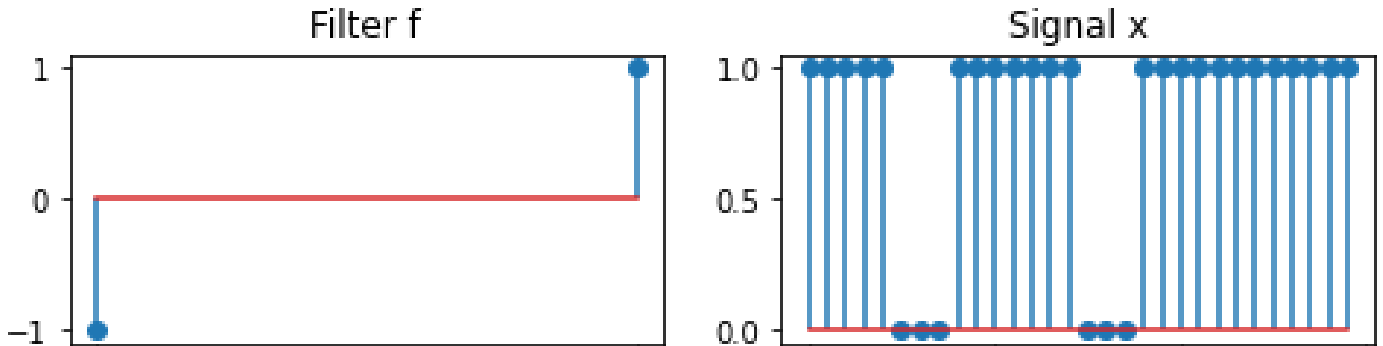
We start by defining a filter and a signal as numpy arrays.

```
import numpy as np

# create filter
f = np.array([-1.,1.])

# create signal
x = np.ones(30)
x[5:8] = 0
x[15:18] = 0

# create convolution
conv_result = conv_naive(f,x)
```

Here, the signal has "jumps" or "edges" between 0 and 1 at a few locations.

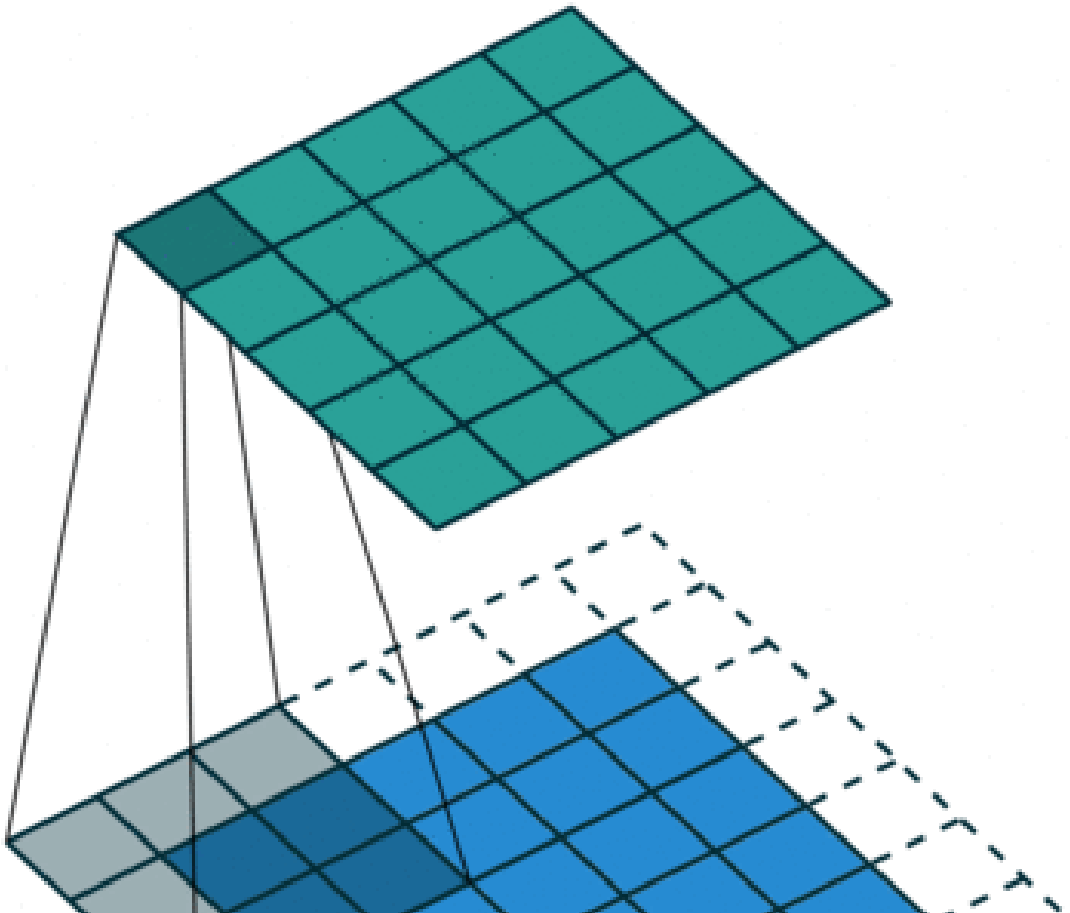
The filter equals ± 1 at these edges, and zero everywhere else. If g was an audio signal, then f would detect boundaries between silence.

```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [8, 4]

ax1 = plt.subplot(221)
ax1.stem(range(len(f)), f, use_line_collection=True)
ax1.set_title('Filter f')
ax1.set_xticks([0,1])
ax2 = plt.subplot(222)
ax2.stem(range(len(x)), x, use_line_collection=True)
ax2.set_title('Signal x')
ax3 = plt.subplot(212)
ax3.stem(range(len(conv_result)), conv_result, use_line_collection=True)
ax3.set_title('Convolution f * x', y=-0.5)

plt.show()
```



Convolutions in 2D

Convolutions can be extended to 2D by expanding dimensions of f and x.



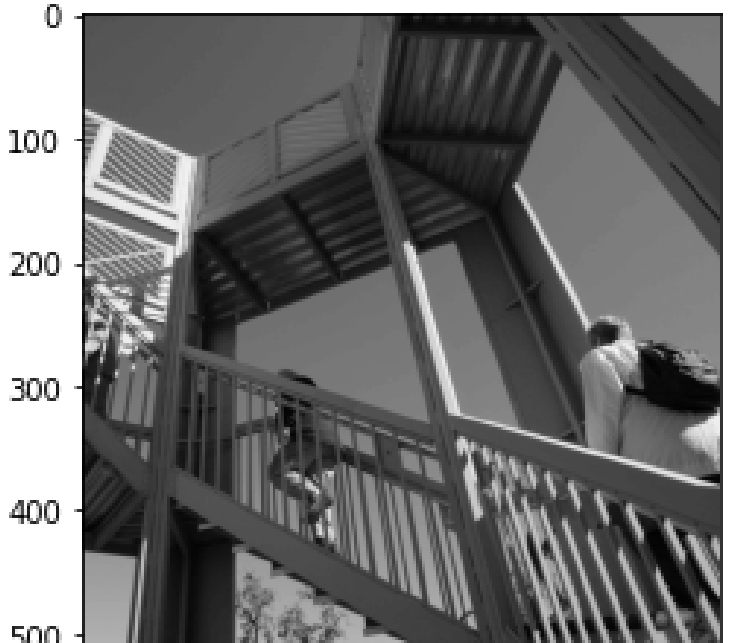
Example: Edge Detection in 2D

Let's revisit our edge detection example, but this time the signal g will be a 2D image.

We will use two filters, each of which is defined below. We will also load an image as the signal g.

```
from scipy import misc

# define the filters
sobel_x = np.array([[1,0,-1],[2,0,-2],[1,0,-1]])
sobel_y = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])

# load and visualize the "signal"
img = misc.ascent()
plt.imshow(img, cmap='gray')
```

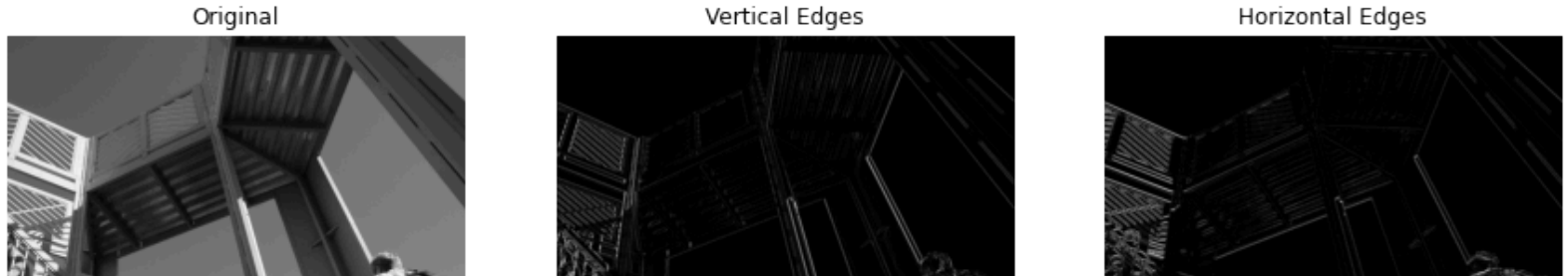
```
<matplotlib.image.AxesImage at 0x1222c62b0>
```



We can now convolve and visualize the filters with the image.

The result is an activation map with high activation around horizontal and vertical edges -- parts of the image where it changes from light to dark.

```
from scipy import signal

# create the convolution with each filter
grad_x = signal.convolve2d(img, sobel_x, boundary='symm', mode='same')
grad_y = signal.convolve2d(img, sobel_y, boundary='symm', mode='same')

# visualize the convolution
fig, (ax_orig, ax_x, ax_y) = plt.subplots(1, 3, figsize=(15, 15))
ax_orig.imshow(img, cmap='gray')
ax_orig.set_title('Original')
ax_orig.set_axis_off()
ax_x.imshow(np.absolute(grad_x), cmap='gray')
ax_x.set_title('Vertical Edges')
ax_x.set_axis_off()
ax_y.imshow(np.absolute(grad_y), cmap='gray')
ax_y.set_title('Horizontal Edges')
ax_y.set_axis_off()
```



Convolutional Layers

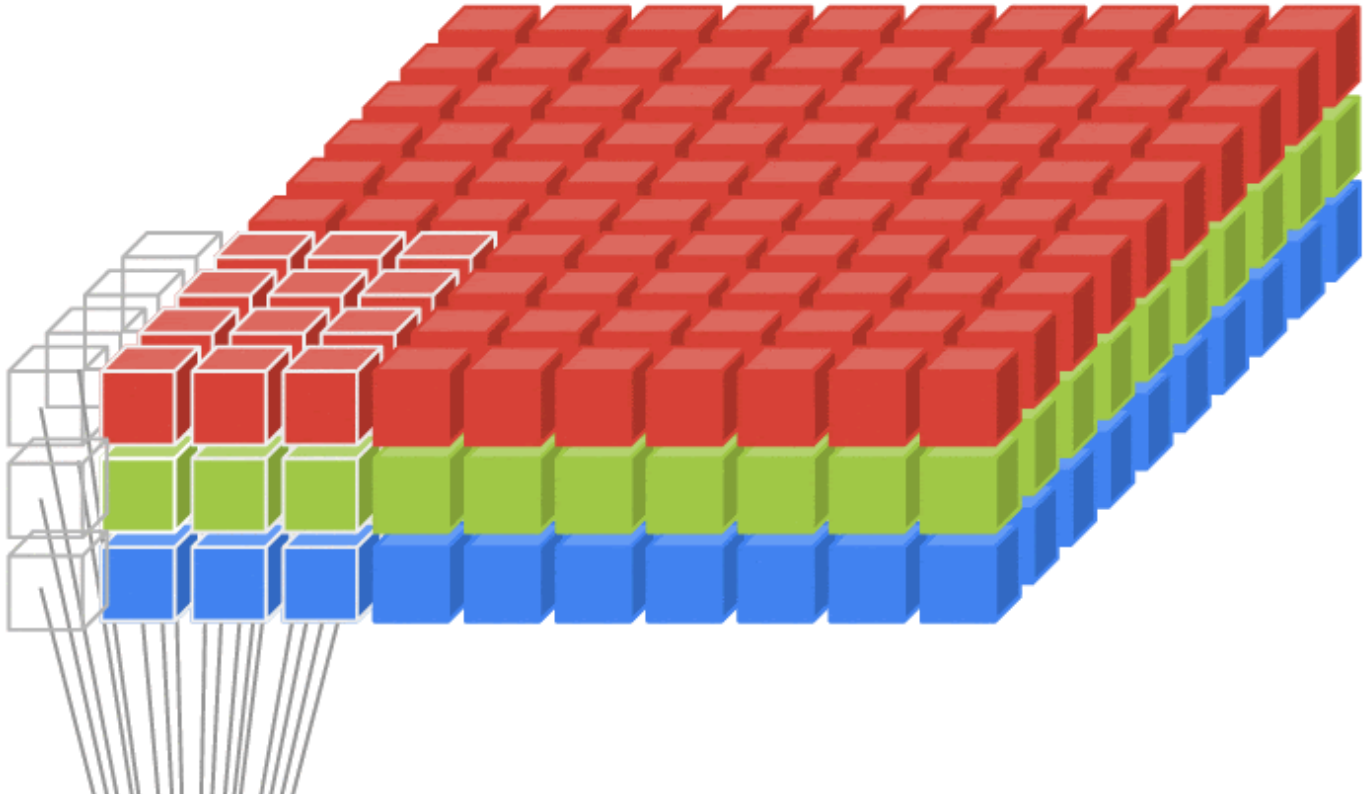
A convolution layer is a model $f : \mathbb{R}^d \to \mathbb{R}^p$ that applies p convolutions in parallel to an input x.

$$f(x) = \text{conv}(W, x) = \begin{bmatrix} \text{conv}(w_1, x) \\ \text{conv}(w_2, x) \\ \vdots \\ \text{conv}(w_p, x) \end{bmatrix},$$

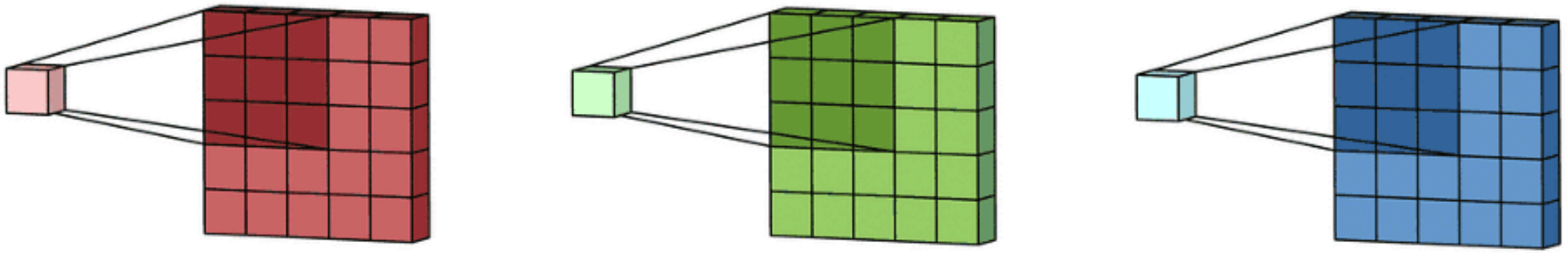
where each weight w_k is the **filter** of the k-th convolution.

Understanding 2D Convolutional Layers

Inputs to convolutional 2d layer have a "depth" dimension (e.g., color channels in an image or stacked activation maps; image by Google).

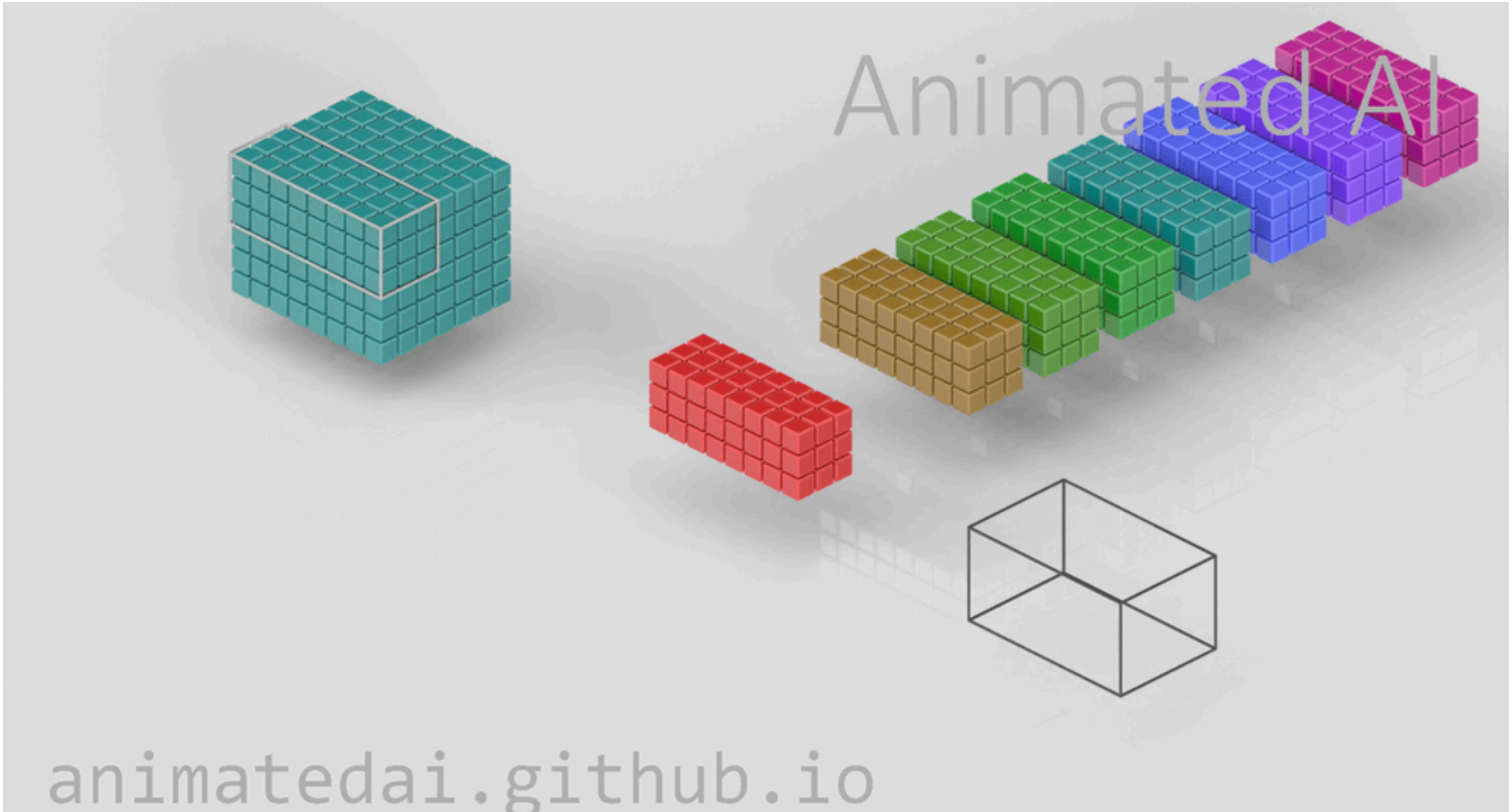


A convolutional layer outputs an activation map for each of its filters.

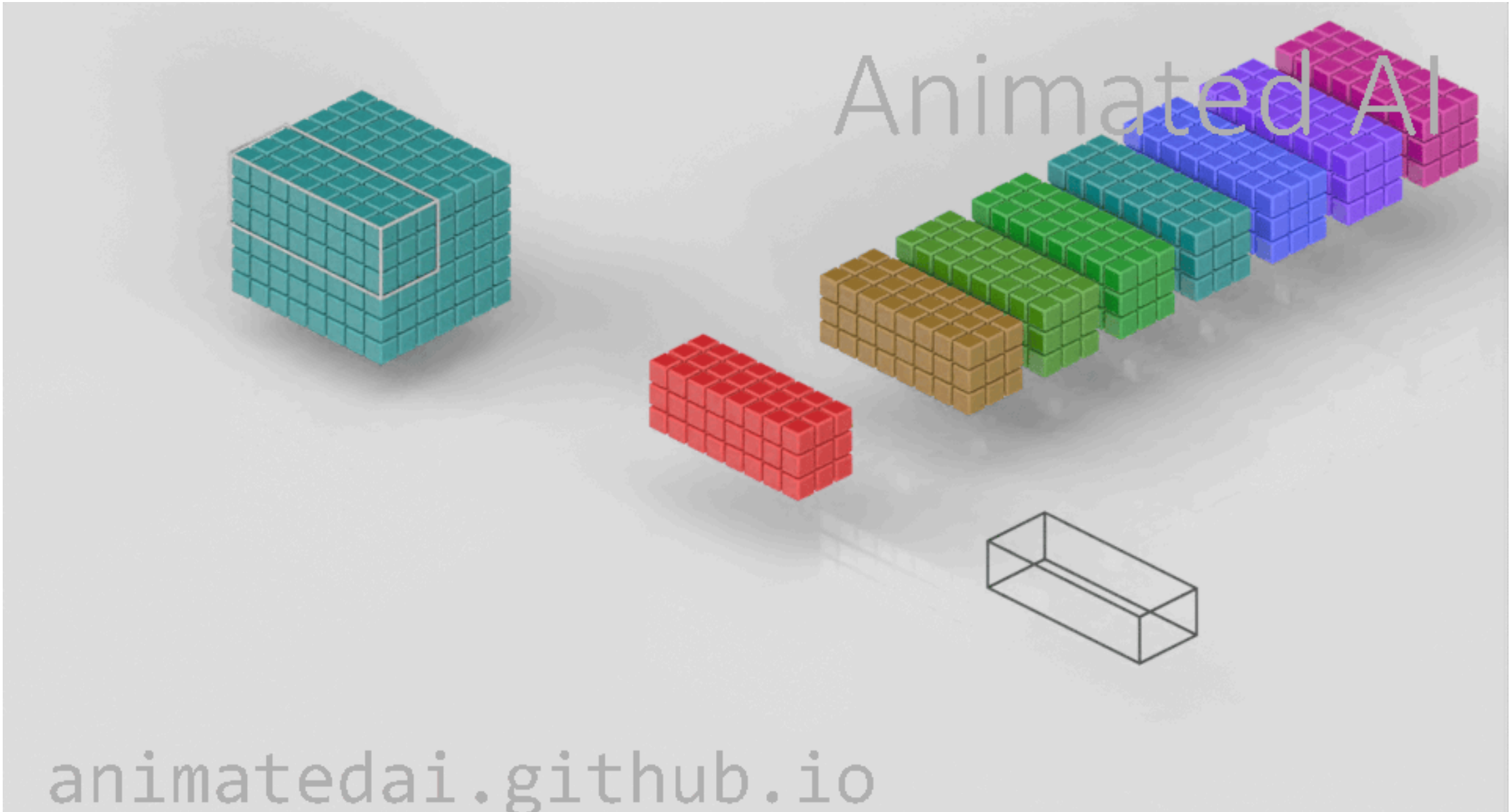


The activation maps are typically stacked along the depth dimension.

This image illustrates a full convolutional layer.



A convolutional layer can also have a **stride**.



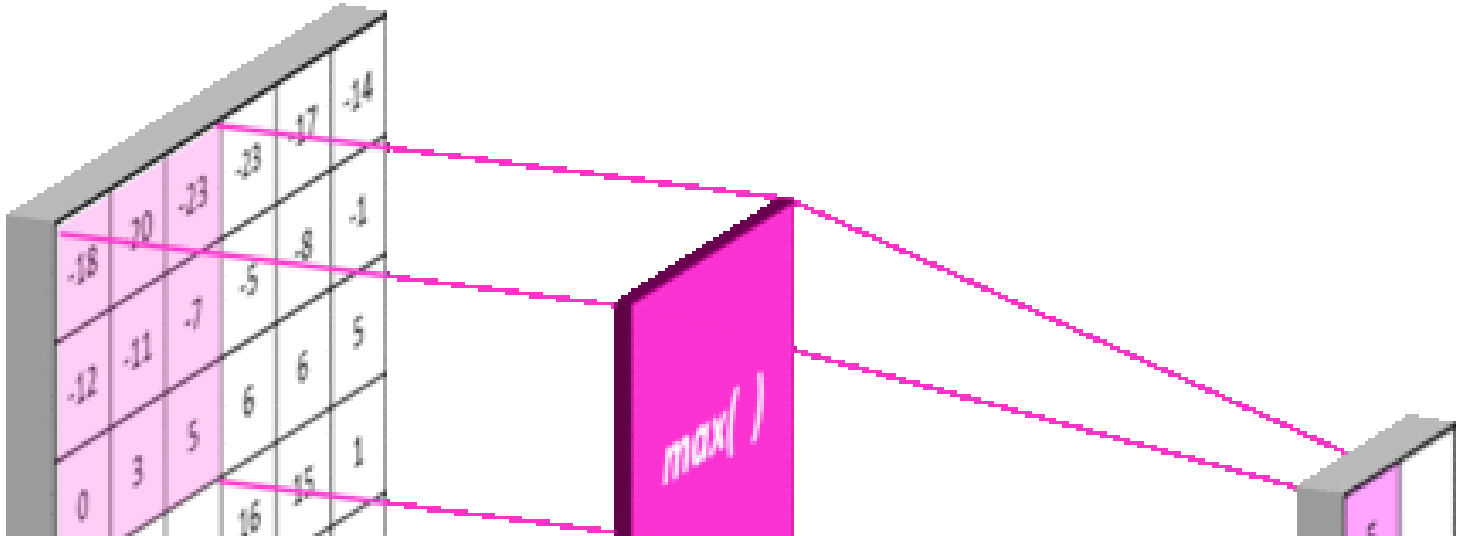
When to Use Convolutions?

Convolutions work best on inputs containing "attributes" that are interesting, but whose location in the input is not important.

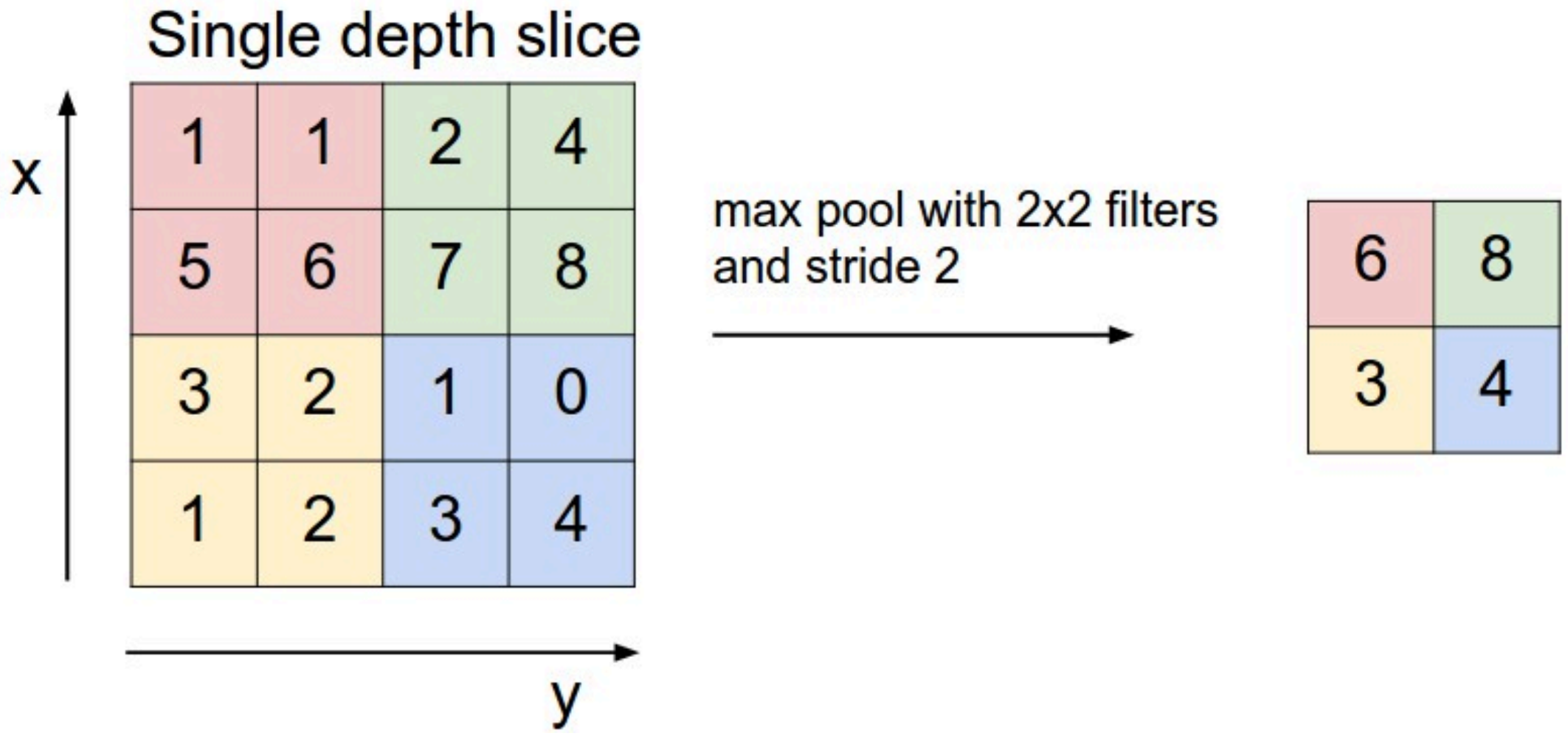
- Images
- Audio

Pooling

Pooling is an operation often added after a convolution layer. It is applied to each activation map separately and reduces the spatial size of its input.

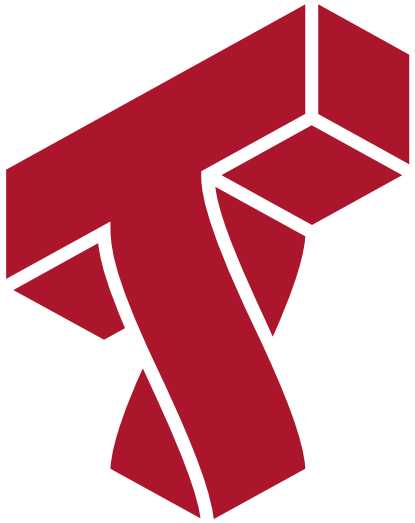


Max Pooling: for each region in the input, return the max value. This is the most common type.



Purpose of Pooling

- Simplifies the pooling window: the exact location of the relevant feature is not critical. This improves feature detection.
- When used together with convolutions, pooling reduces spatial size of feature maps to capture more abstract representations.

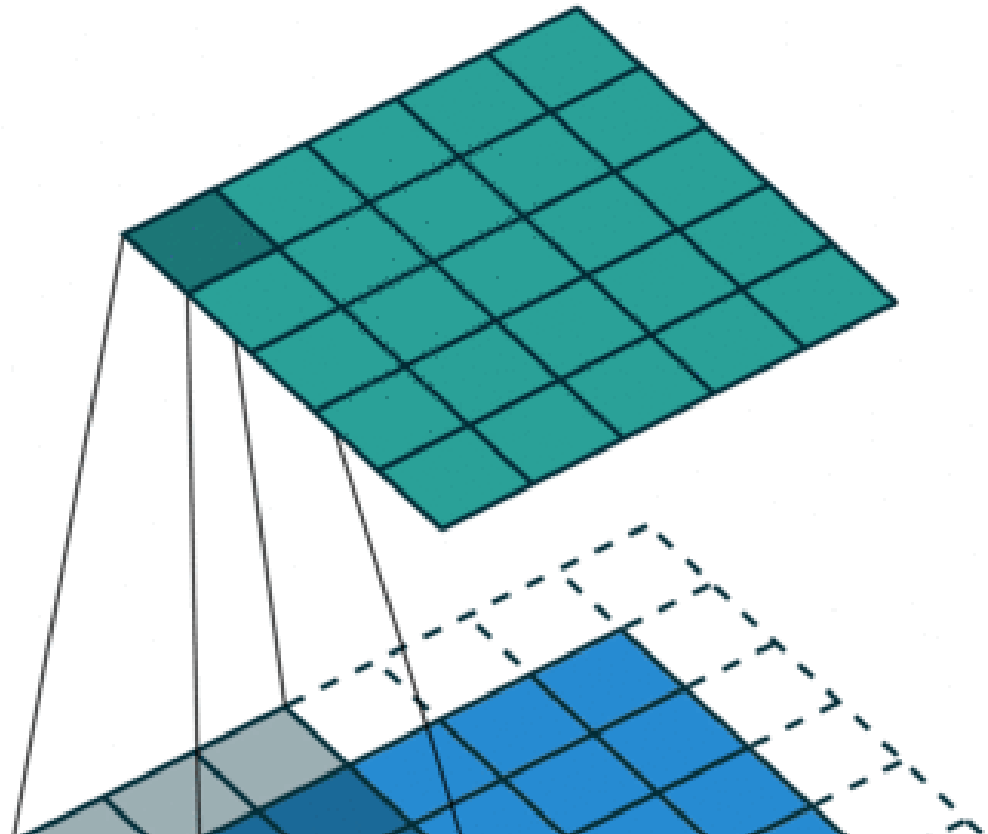


Part 3: Convolutional Neural Networks

Convolutional neural networks use convolutional layers to process signals such as images, audio, and even text.

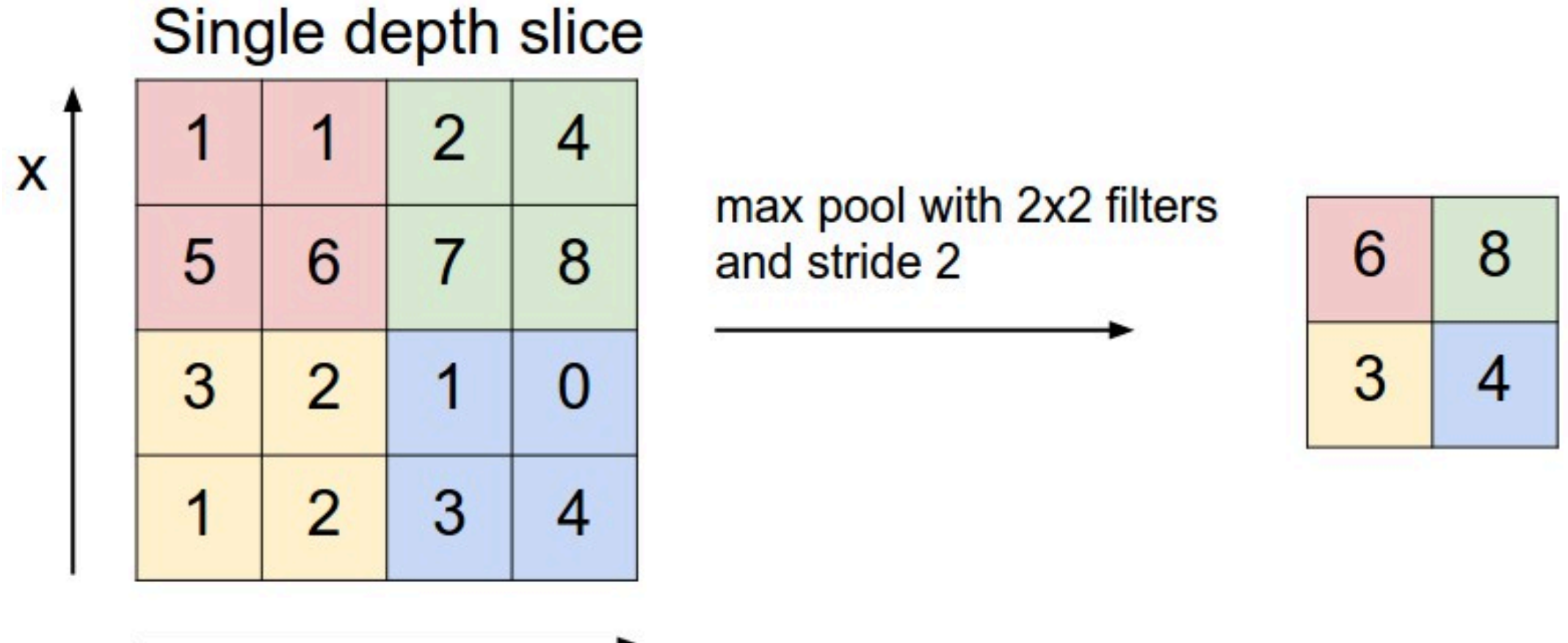
Review: Convolution

Convolutions apply a filter f (gray) over a signal g (blue). The output is an activation map (green).



Review: Pooling

Pooling is a common operation to be added after a convolution layer. It is applied to each activation map separately and reduces the spatial size of its input.



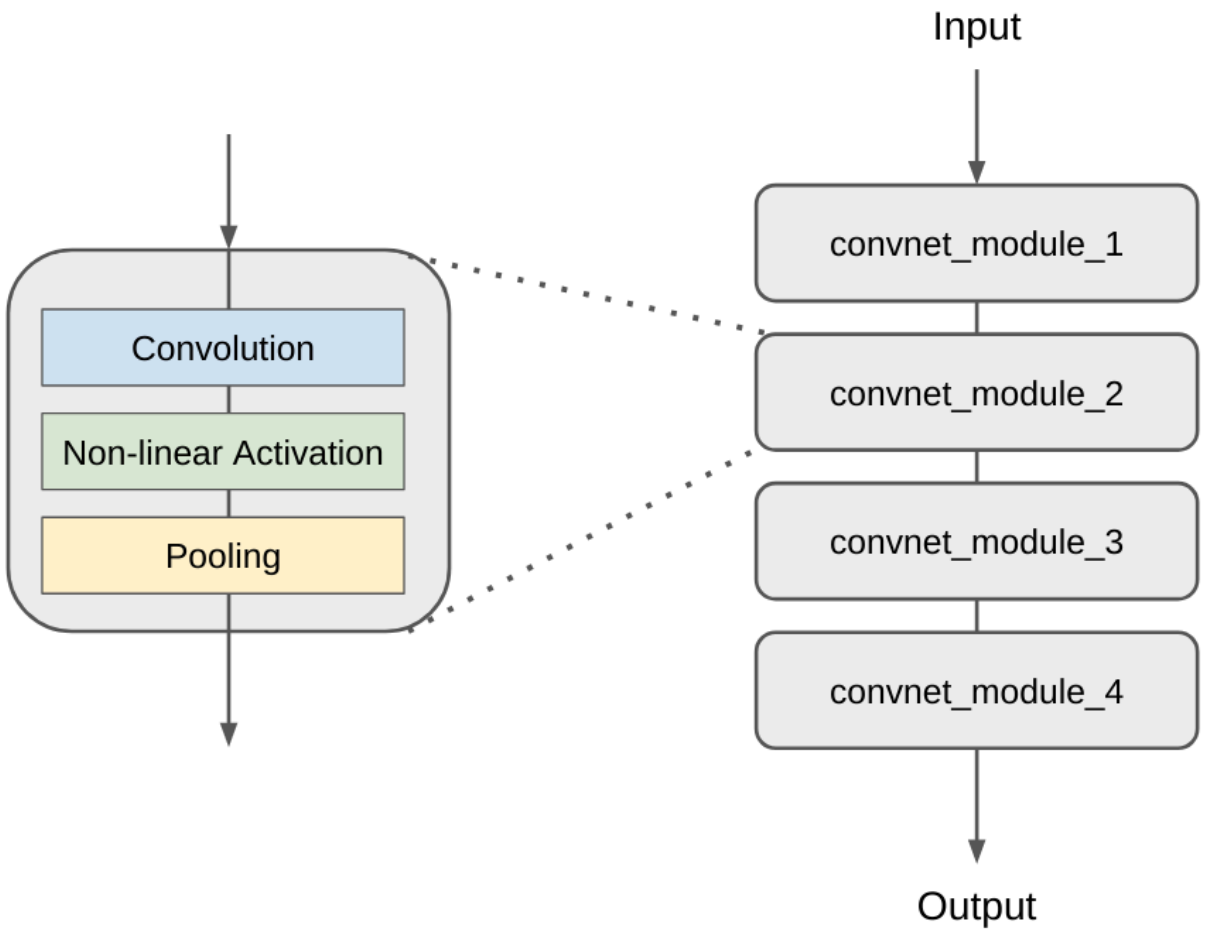
Convolutional Neural Networks

A convolutional neural network (CNN) is a model $f : \mathbb{R} \to \mathbb{R}$ that consists of a composition of L neural network layers that contain convolutions:

$$f(x) = f_L \circ f_{L-1} \circ \dots f_1(x).$$

The final f_L is often a fully connected output later of size one.

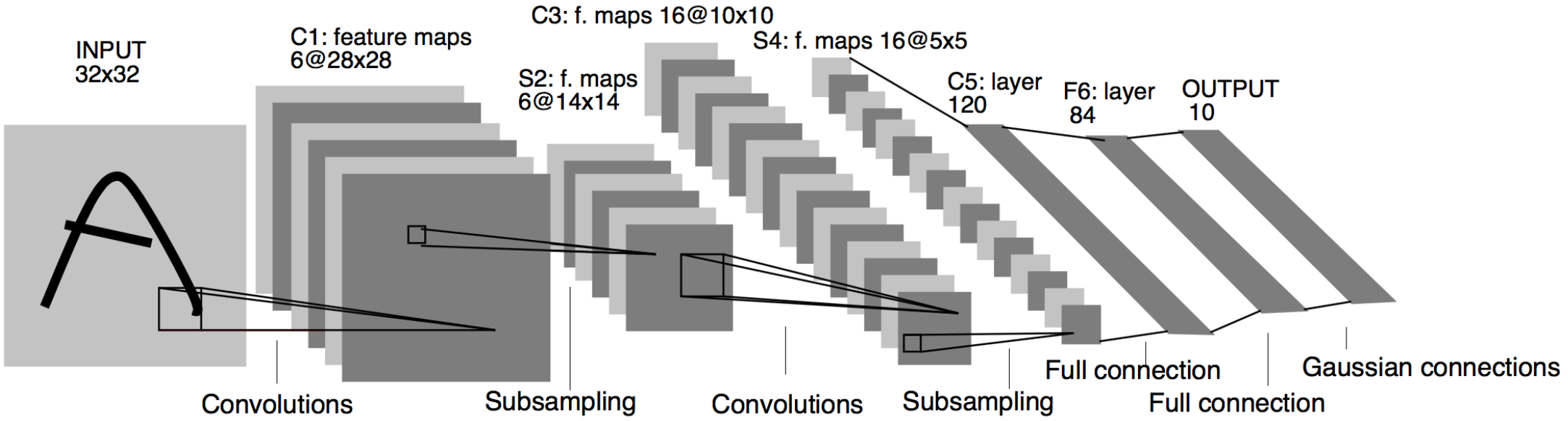
Typically, CNNs are made of consecutive convolution + activation + pooling layers that form into blocks.



Next we are going to see a few famous examples.

LeNet for MNIST Digits Recognition

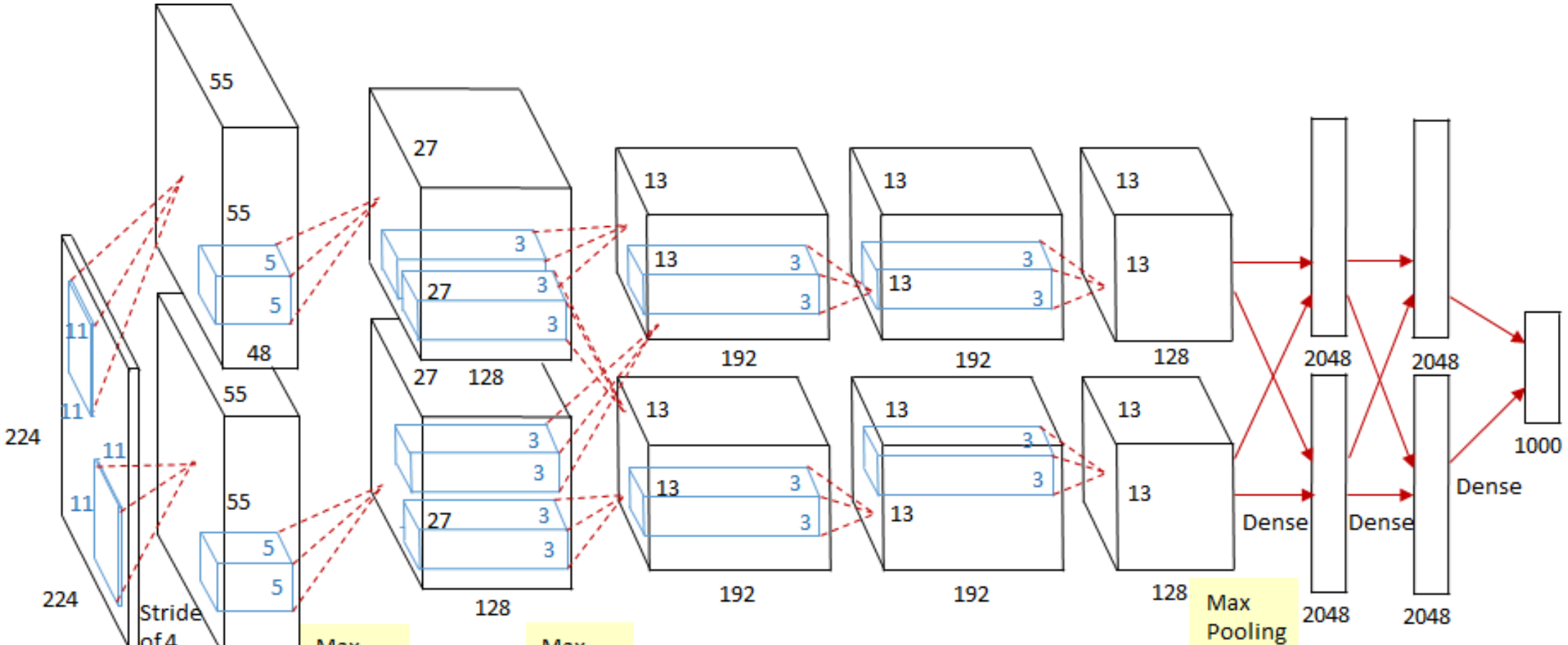
LeNet successfully used CNNs for digit recognition [LeCun, Bottou, Bengio, Haffner 1998].



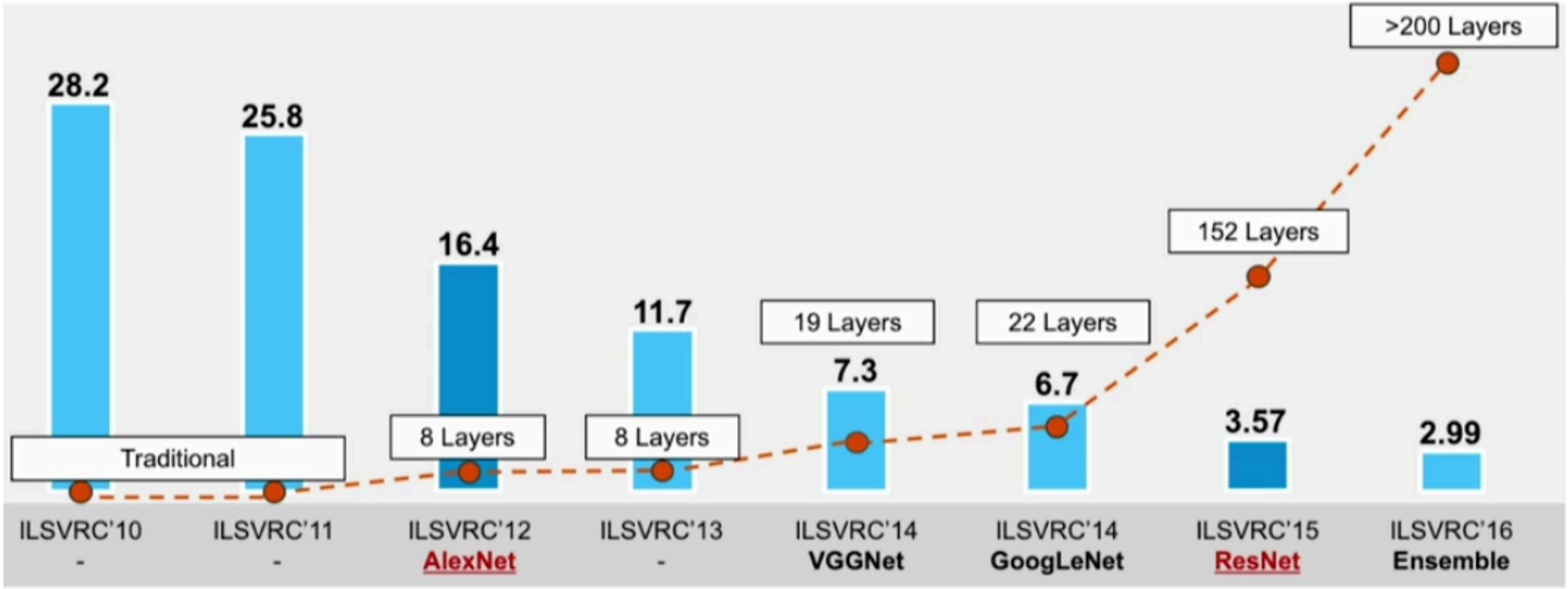
- LeNet contains multiple convolutional and max-pooling (called subsampling in the original paper) layers and a fully connected MLP in the end.
- Given an image of a hand-written digit, LeNet predicts it to be one of the 10 classes (0-9).
- This work inspired many subsequent CNN architectures.

AlexNet and ImageNet

ImageNet classification with deep convolutional neural networks [Krizhevsky, Sutskever, Hinton 2012]



Starting from AlexNet, the performance of image classification on ImageNet has entered a new era:



http://sqlml.azurewebsites.net/2017/09/12/convolutional-neural-network/

Convolutional neural networks with deeper layers (and other tricks) have already

Algorithm: Convolutional Neural Network

- **Type**: Supervised learning (regression and classification).
- **Model family**: Compositions of convolutional, pooling and other types of layers.
- **Objective function**: Any differentiable objective.
- **Optimizer**: Gradient descent.

Visualizing CNN Internals

CNNs can be seen as extracting features at low-, mid-, and high-levels of abstraction that are relevant to corresponding visual concepts.

Below, we reproduce feature visualization from the paper Visualizing and Understanding Convolutional Networks [Zeiler, Fergus 2013].

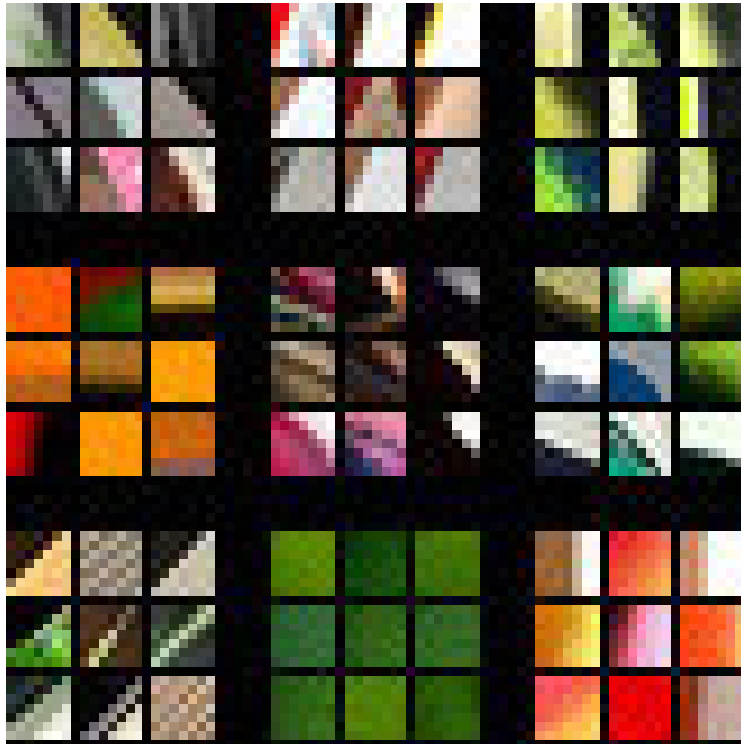
For a convolutional network with

$$\text{Input} \to \text{Conv Layer1} \to \text{Conv Layer2} \to \cdots \to \text{Conv Layer5} \to \cdots \to \text{Output},$$

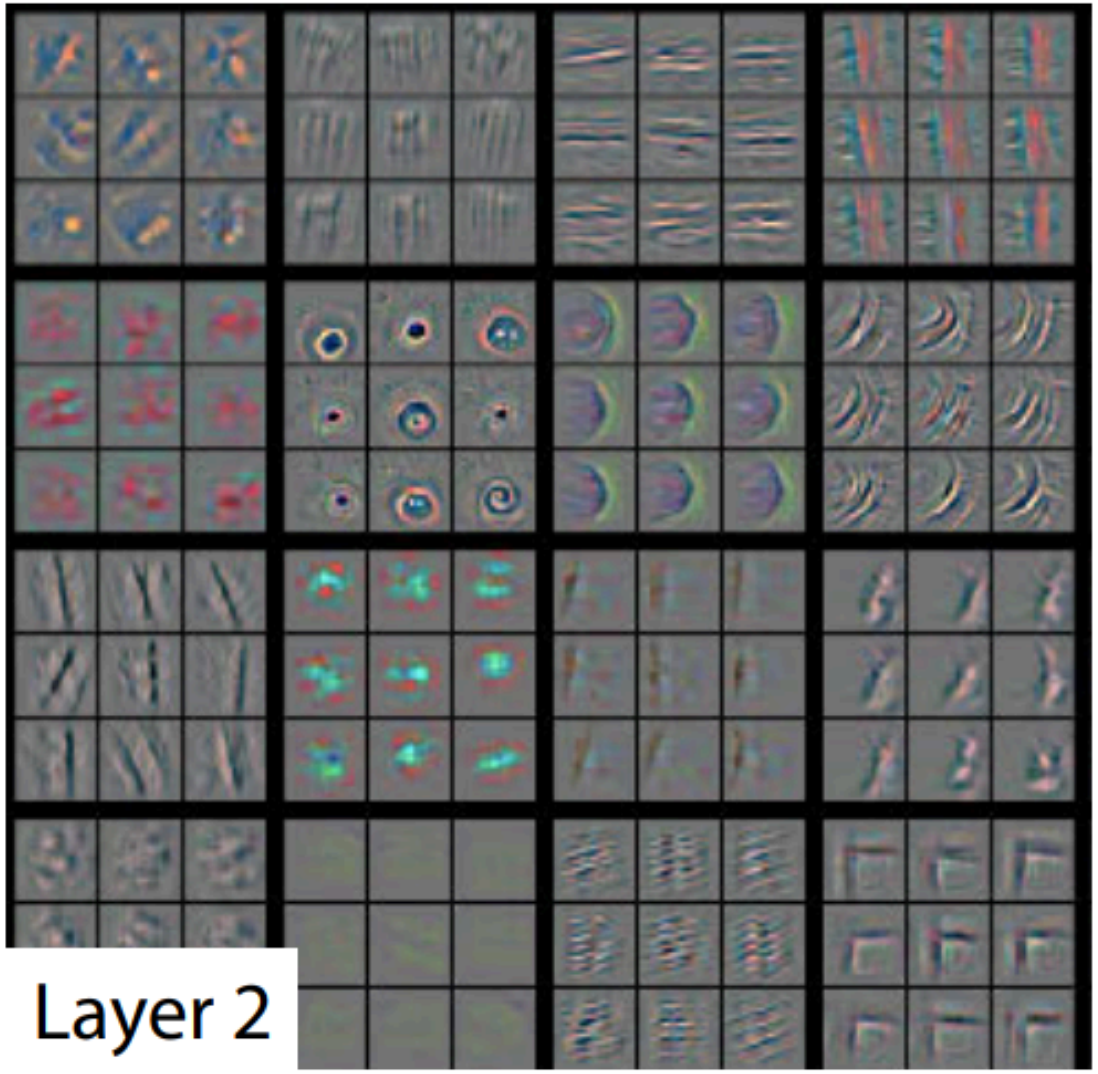
we can visualize what each layer is doing by finding the image patch with highest activation responses.

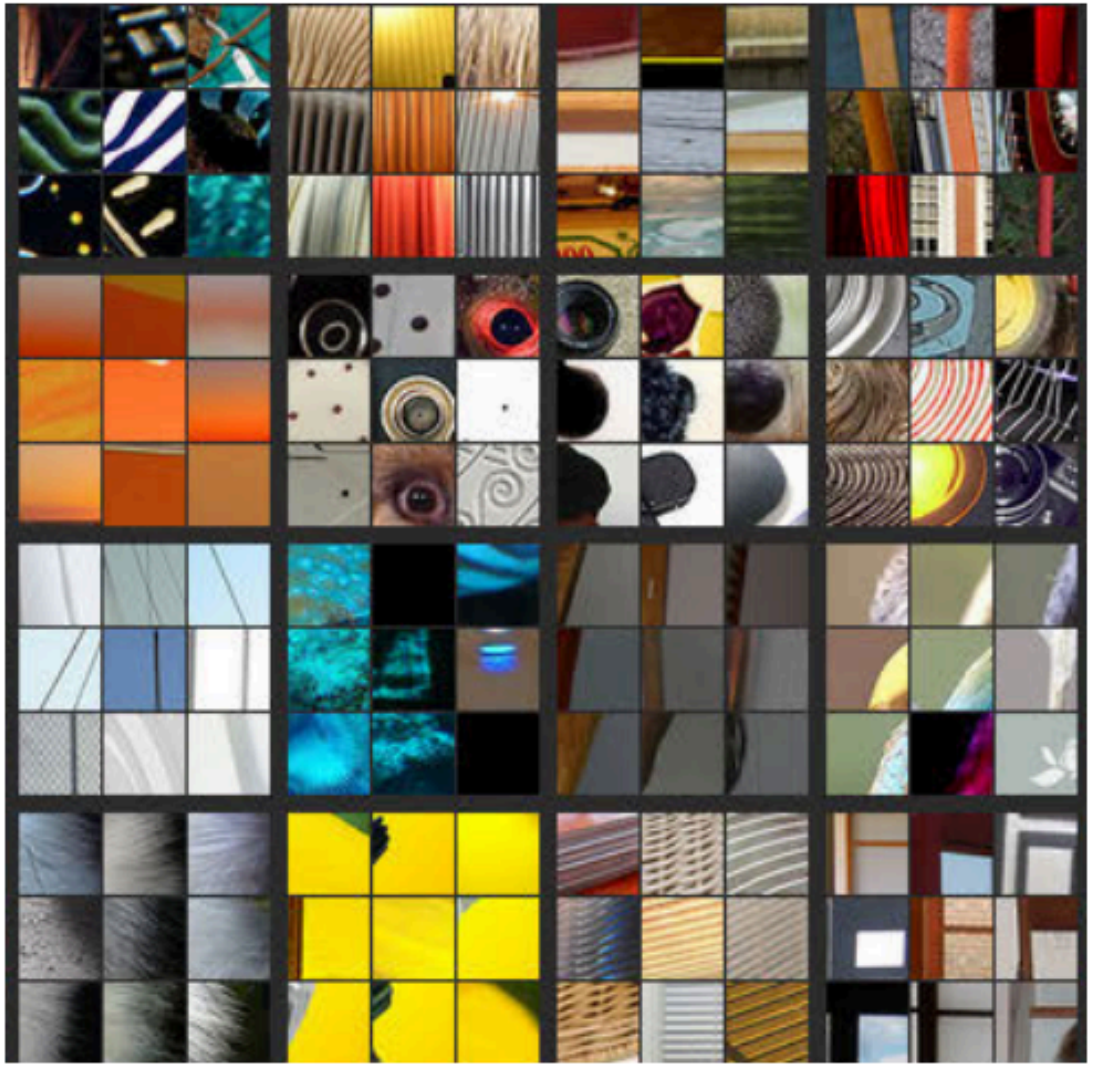
Layer 1 focuses on colored edges and blobs.

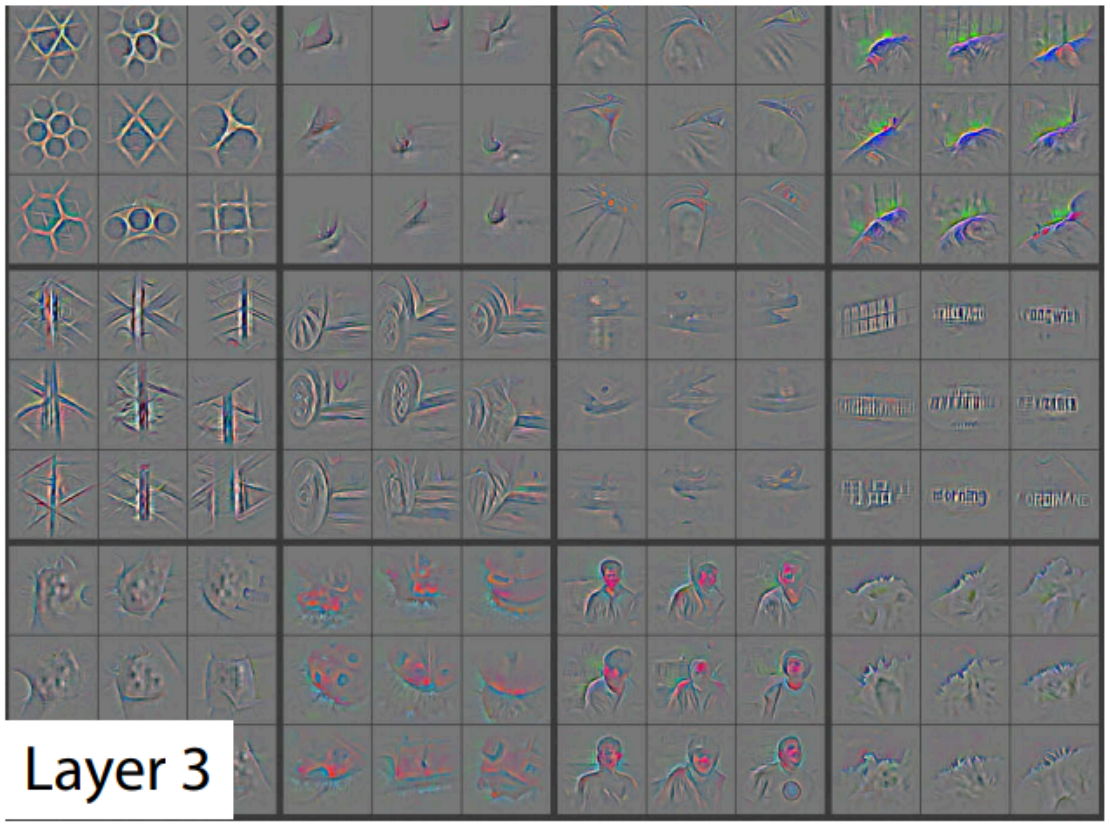
Layer 1



Layers 2-3 focus on object parts, such as the wheels of a car, or the beak of a bird.

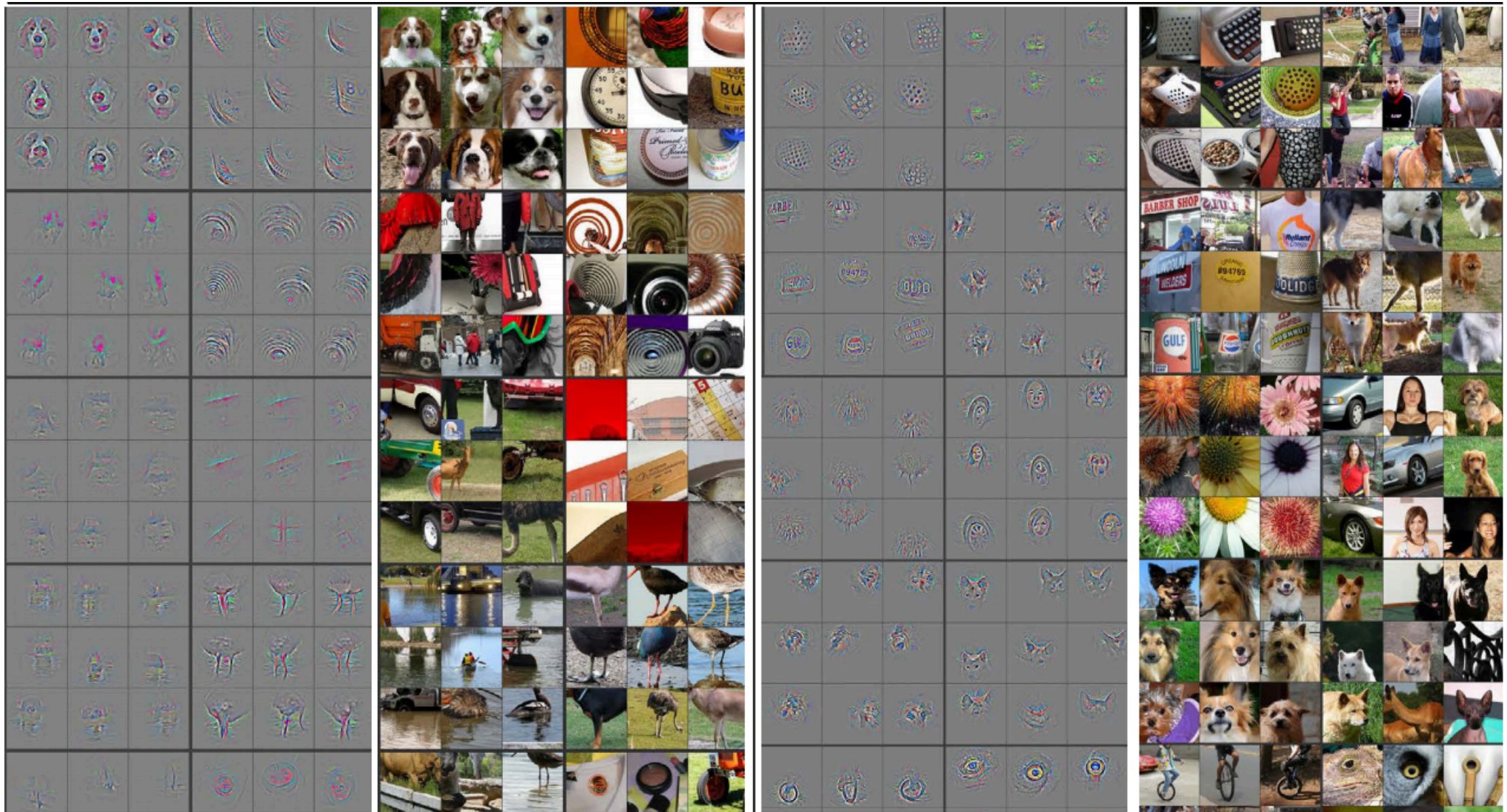






Layer 3

Layers 4-5 focus on object parts and even entire objects.



Pros and Cons of CNNs:

CNNs are powerful tools because they encode visual information efficiently.

- CNNs are now used as universal visual feature extractors. It is common to use ImageNet pretrained CNNs.
- They serve as building blocks in ML pipelines and can work with other types of networks such as a recurrent network for complex tasks.

Their main drawbacks are computational and data requirements.