Convolutional Neural Networks

CMPUT 261: Introduction to Artificial Intelligence

GBC §9.0-9.4

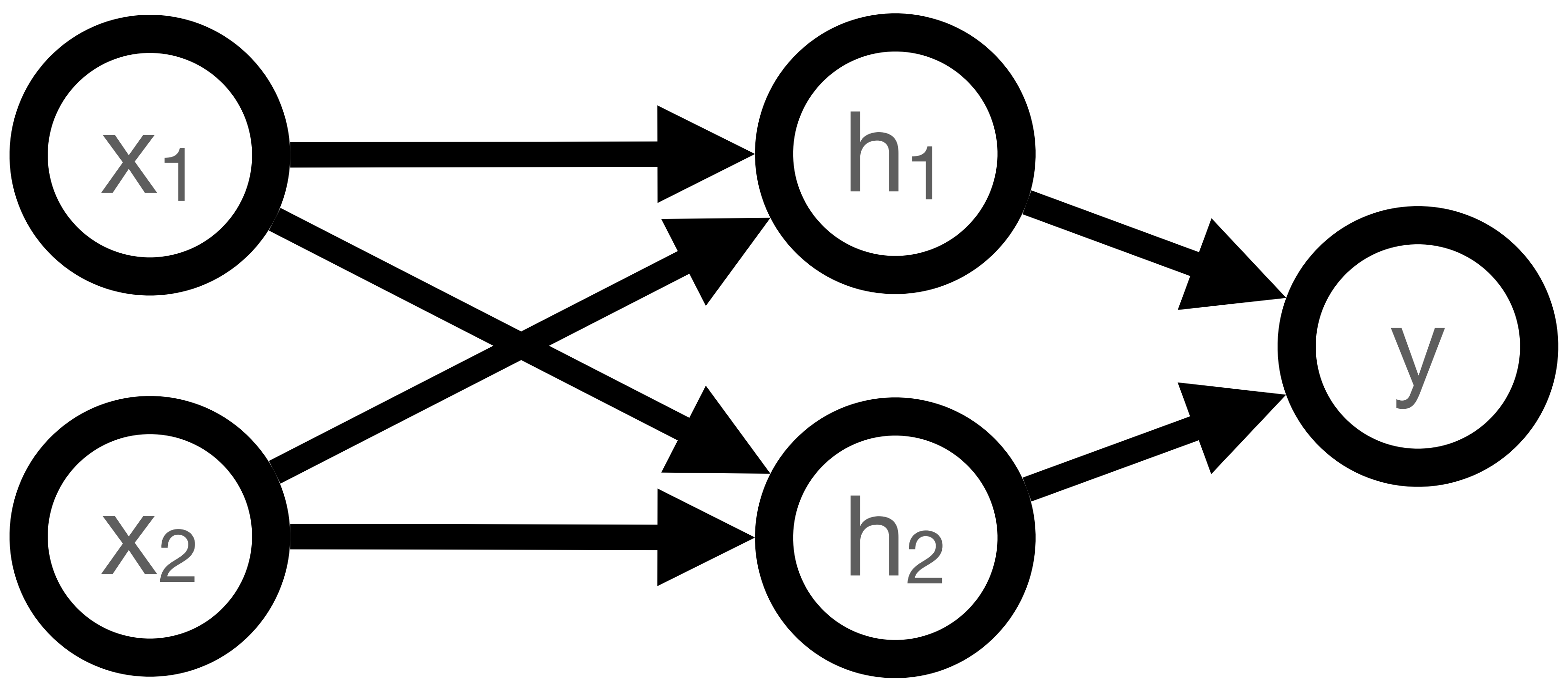
Lecture Outline

1. Recap & Logistics
2. Neural Networks for Image Recognition
3. Convolutional Neural Networks

After this lecture, you should be able to:

- explain why convolutional neural networks are more efficient to train on image data than dense feedforward networks
- define sparse interactions and parameter sharing
- define the convolution operation and demonstrate it on an example input
- define the pooling operation and demonstrate it on an example input

Recap:
Feedforward Neural Network



$$h_1(\mathbf{x}; \mathbf{w}^{(1)}, b^{(1)}) = g\left(b^{(1)} + \sum_{i=1}^{n} w_i^{(1)} x_i\right)$$

$$y(\mathbf{x}; \mathbf{w}, \mathbf{b}) = g\left(b^{(y)} + \sum_{i=1}^{n} w_i^{(y)} h_i(\mathbf{x}_i; \mathbf{w}^{(i)}, b^{(i)})\right)$$

$$= g\left(b^{(y)} + \sum_{i=1}^{n} w_i^{(y)} g\left(b^{(i)} + \sum_{j=1}^{n} w_j^{(i)} x_j\right)\right)$$

- A **neural network** is many **units** **composed** together
- **Feedforward neural network:** Units arranged into **layers**
  - Each layer takes outputs of **previous layer** as its **inputs**

Recap: Training Neural Networks

- Specify a **loss** L and a set of **training examples:**

$$E = (\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})$$

- Training by **gradient descent**:

Loss function
(e.g., squared error)

1. Compute **loss** on training data: $L(\mathbf{W}, \mathbf{b}) = \sum_i \ell \left(\underbrace{f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b})}_{\text{Prediction}}, \underbrace{y^{(i)}}_{\text{Target}} \right)$

2. Compute **gradient** of loss: $\nabla L(\mathbf{W}, \mathbf{b})$

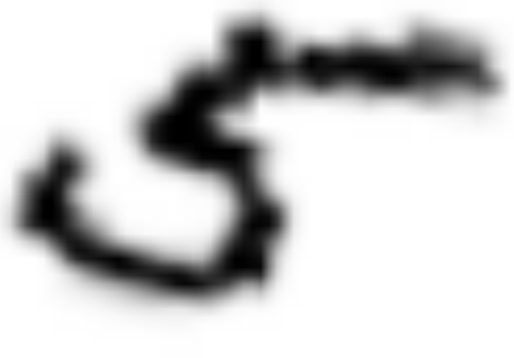
3. **Update parameters** to make loss smaller:

$$\begin{bmatrix} \mathbf{W}^{new} \\ \mathbf{b}^{new} \end{bmatrix} = \begin{bmatrix} \mathbf{W}^{old} \\ \mathbf{b}^{old} \end{bmatrix} - \eta \nabla L(\mathbf{W}^{old}, \mathbf{b}^{old})$$

Recap: Automatic Differentiation

- **Forward mode** sweeps through the graph, computing $s_i' = \dfrac{\partial s_i}{\partial s_1}$ for each s_i
    - The **numerator varies**, and the **denominator is fixed**
    - At the end, we have computed $s_n' = \dfrac{\partial s_n}{\partial x_i}$ for a **single** input x_i
- **Backward mode** does the opposite:
    - For each s_i, computes the **local gradient** $\overline{s_i} = \dfrac{\partial s_n}{\partial s_i}$
    - The **numerator is fixed**, and the **denominator varies**
    - At the end, we have computed $\overline{x_i} = \dfrac{\partial s_n}{\partial x_i}$ for each input x_i
- **Key point:** The intermediate results are computed **numerically** at **each step**

Image Classification



FIVE

Problem: Recognize the handwritten digit from an image

- What are the **inputs**?
- What are the **outputs**?
- What is the **loss**?

Image Classification with Neural Networks

How can we use a **neural network** to solve this problem?

- How to represent the **inputs**?
- How to represent the **outputs**?
- What are the **parameters**?
- What is the **loss**?

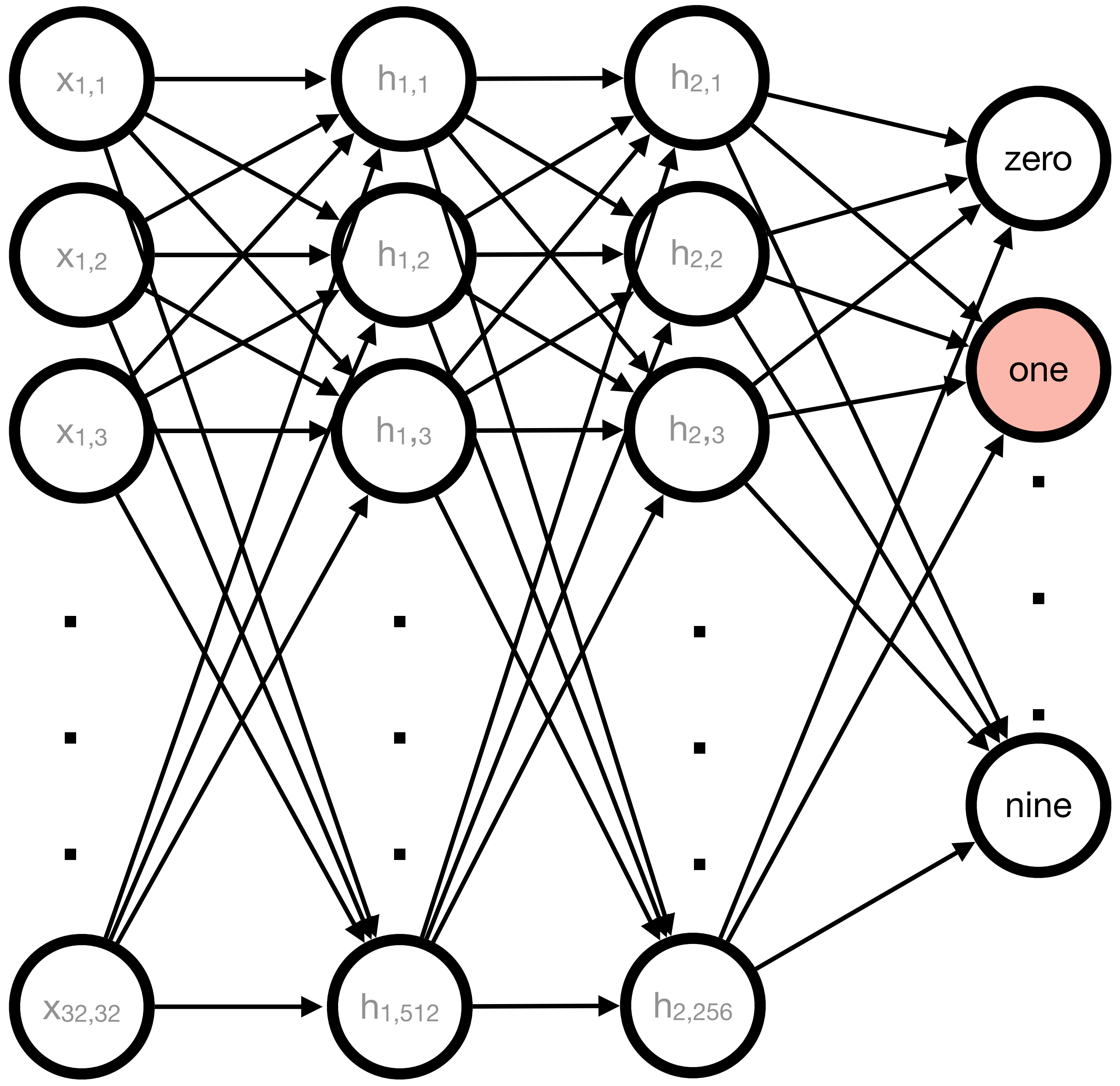
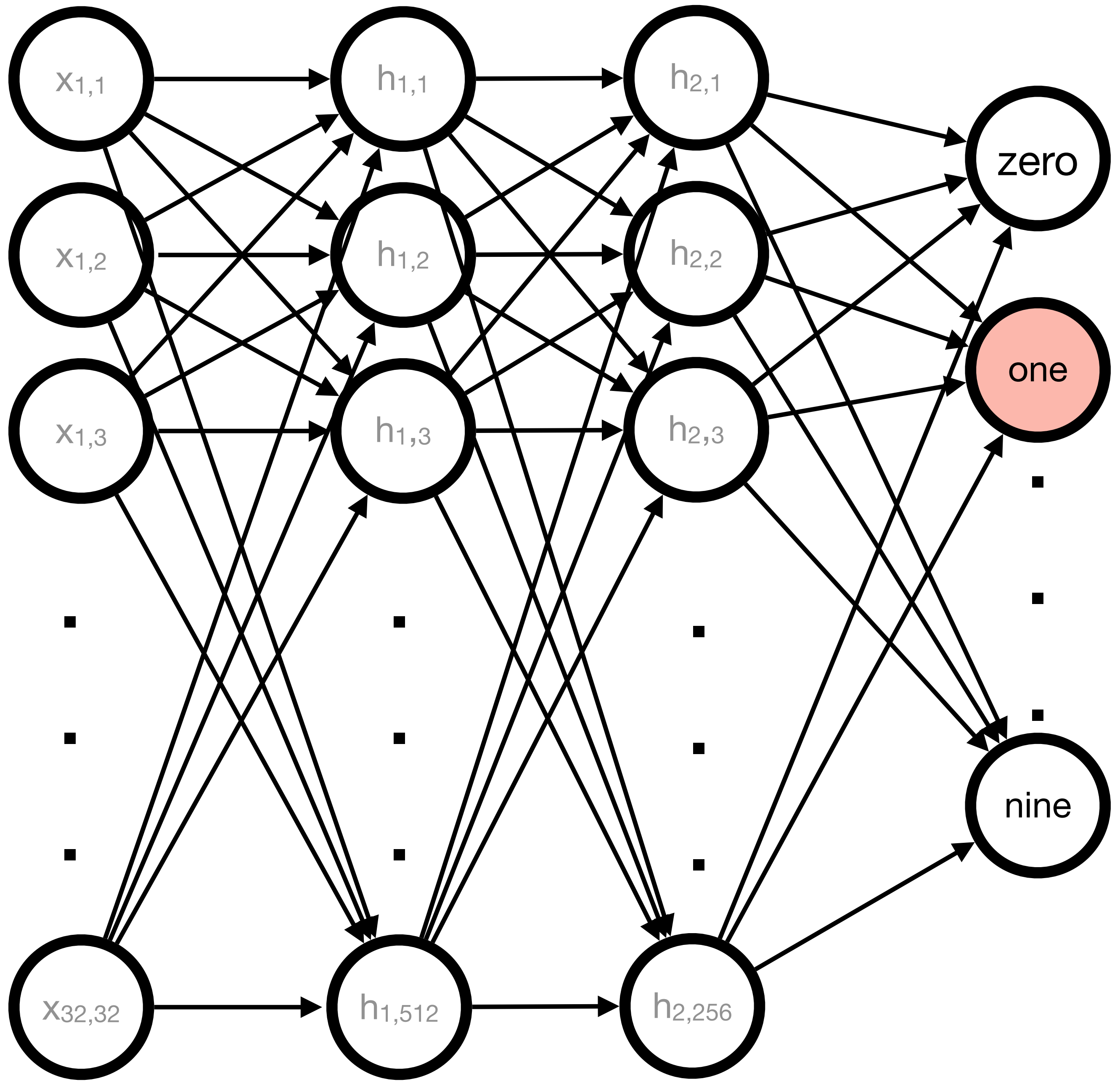


Image Recognition Issues

- For a large image, the number of parameters will be **very large**
  - For 32x32 greyscale image, hidden layer of 512 units hidden layer of 256 units, 1024×512 + 512×256 + 256×10 = **657,920 weights** (and 1,802 offsets)
  - Needs **lots of data** to train
- Want to **generalize** over **transformations** of the input

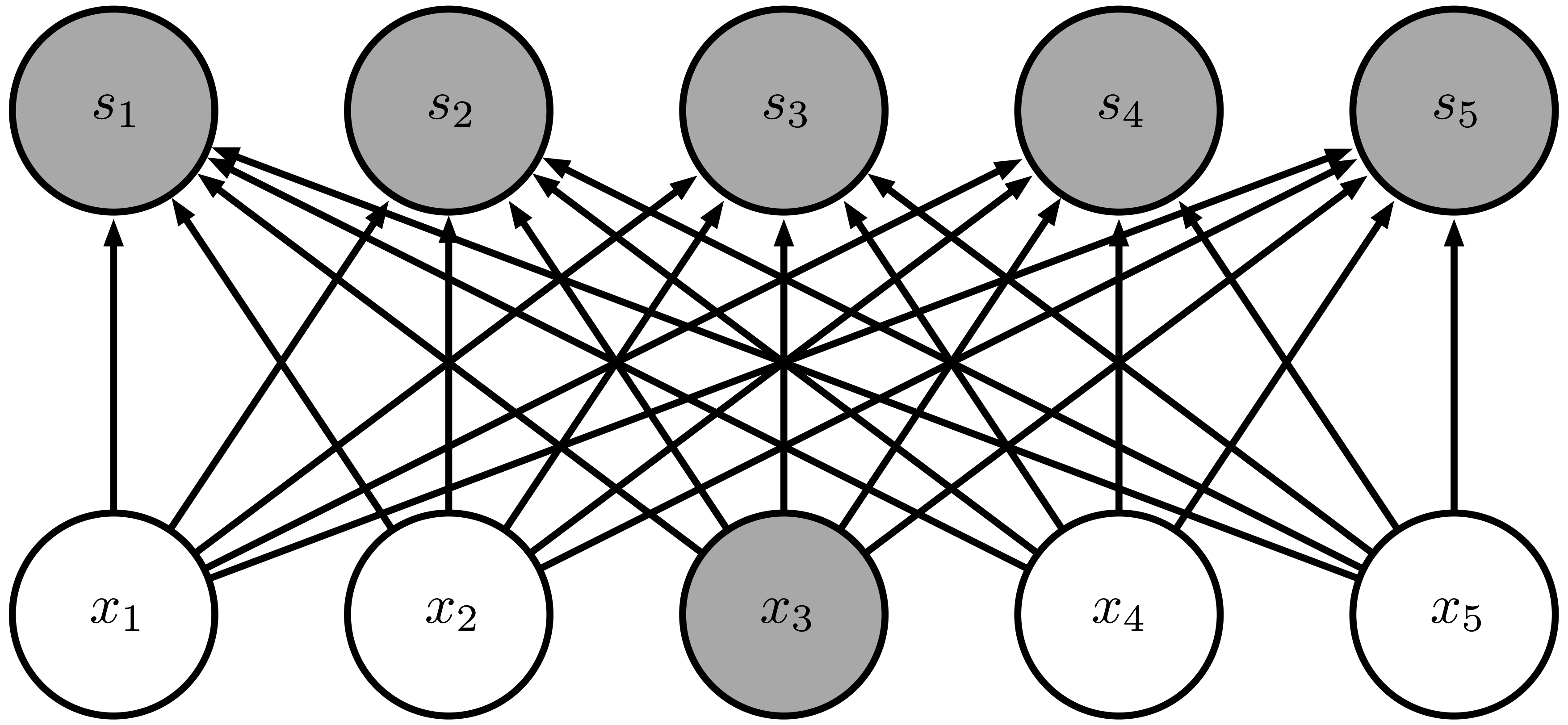


Convolutional Neural Networks

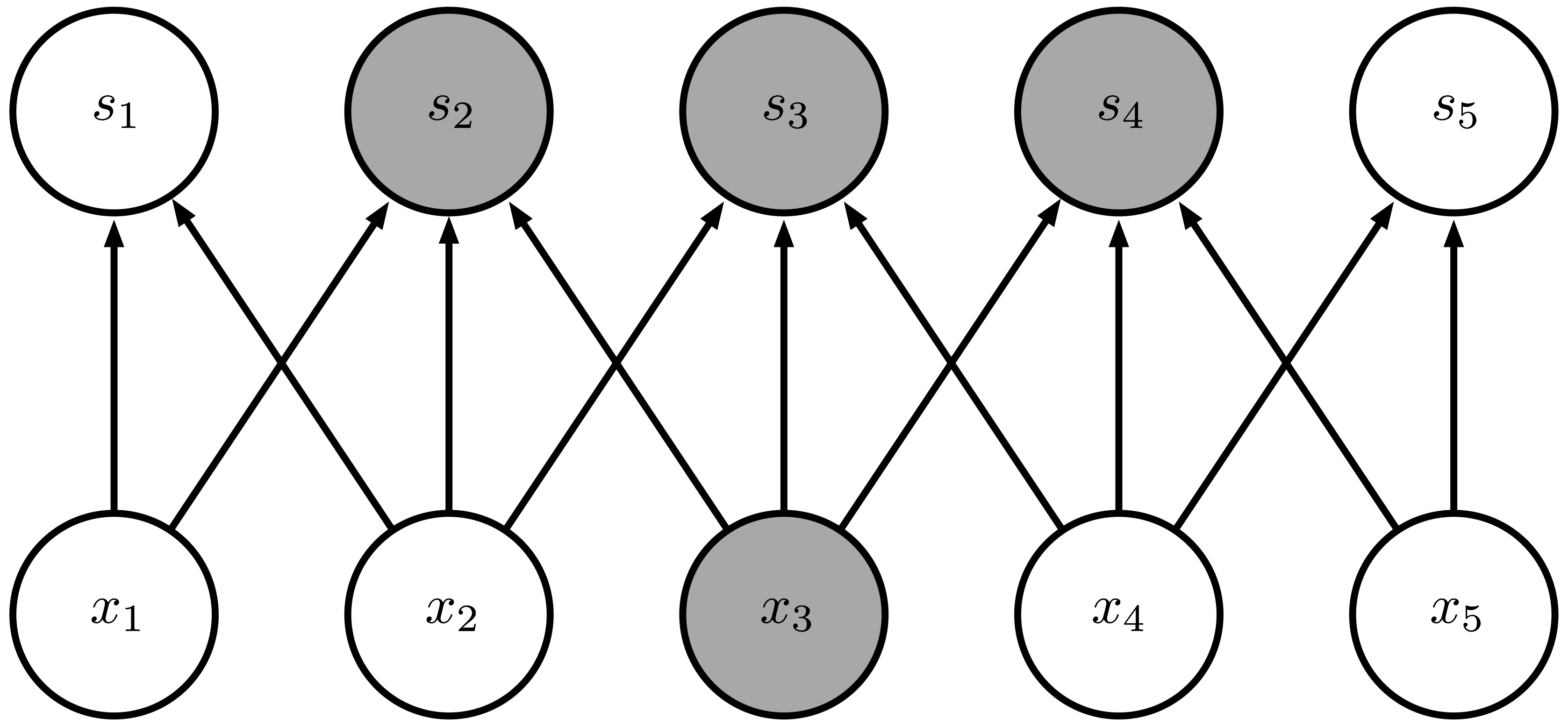
- **Convolutional neural networks:** a **specialized architecture** for image recognition
- Introduce two **new operations**:
  1. Convolutions
  2. Pooling
- Efficient **learning** via:
  1. Sparse interactions
  2. Parameter sharing
  3. Equivariant representations

1. Sparse Interactions

Dense connections



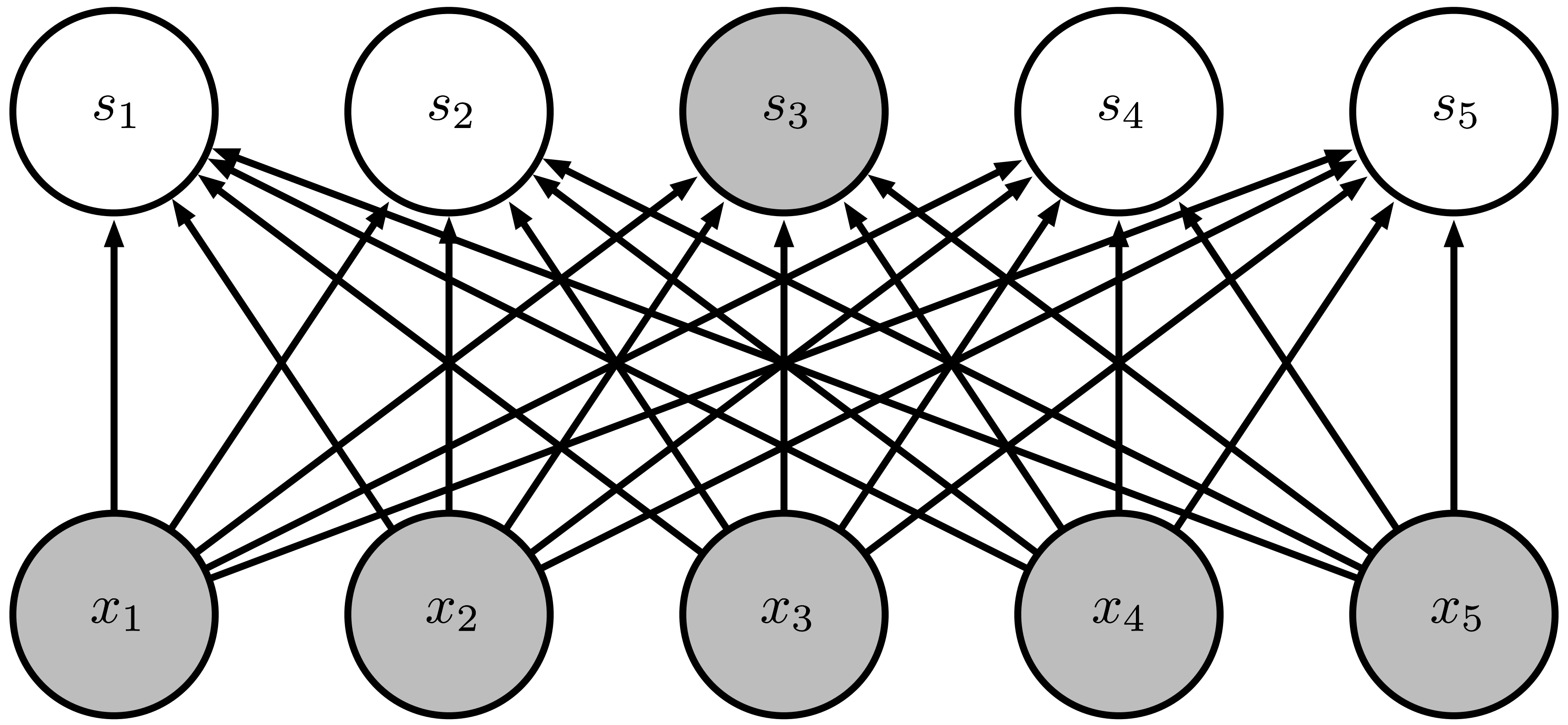
Sparse connections



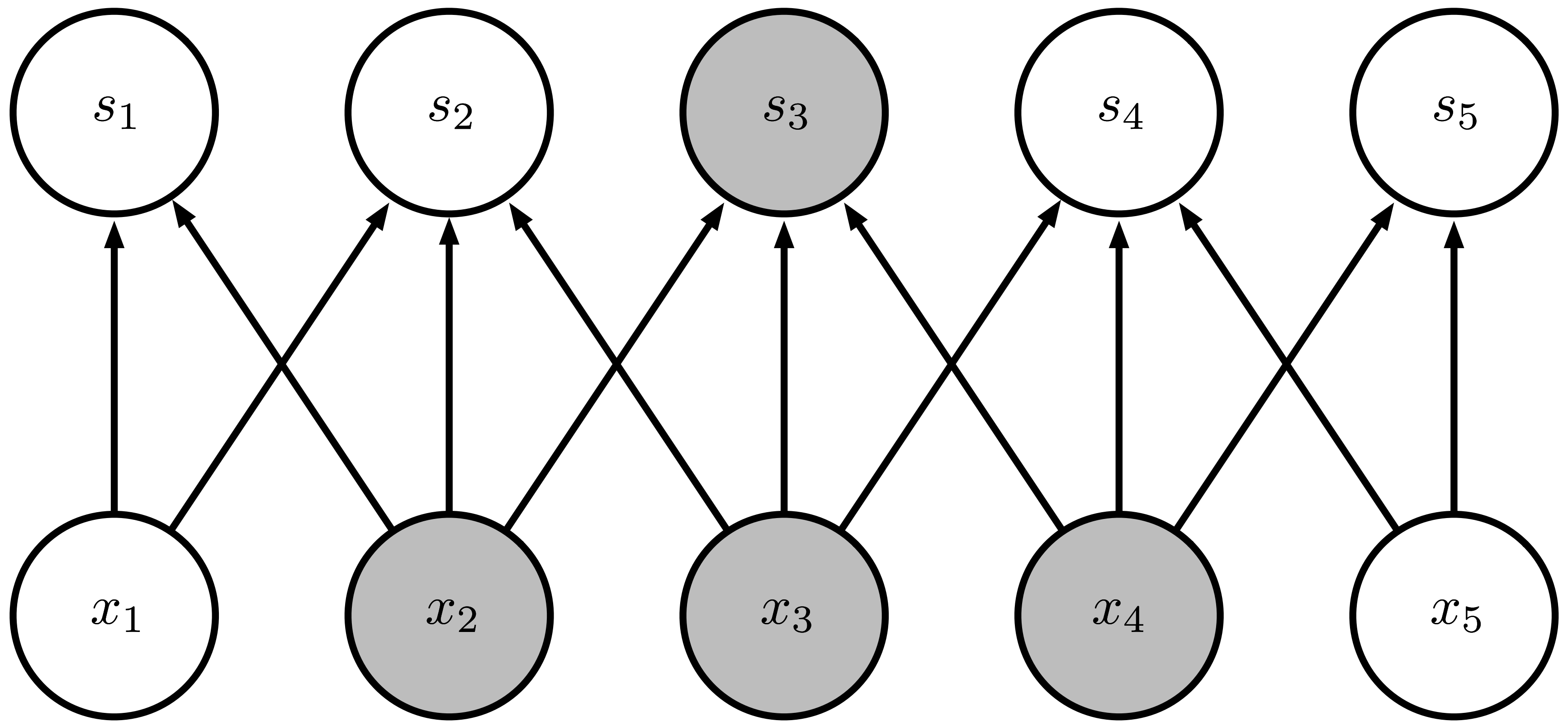
(Images: Goodfellow 2016)

1. Sparse Interactions

Dense connections



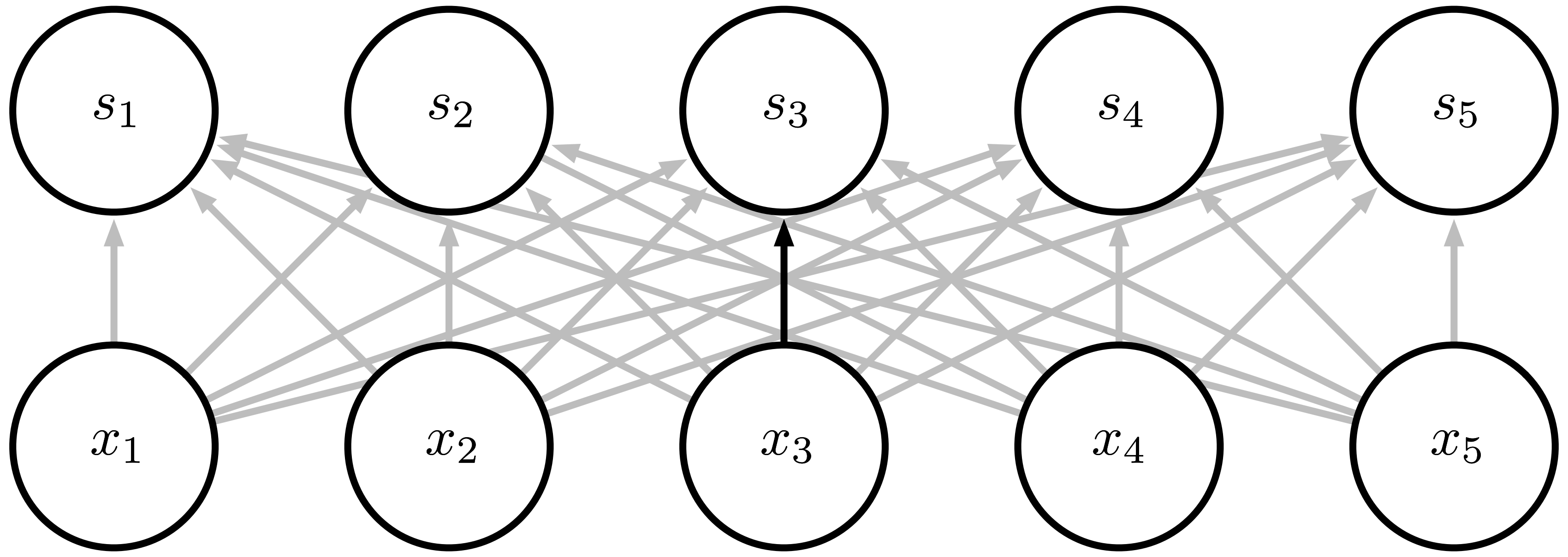
Sparse connections



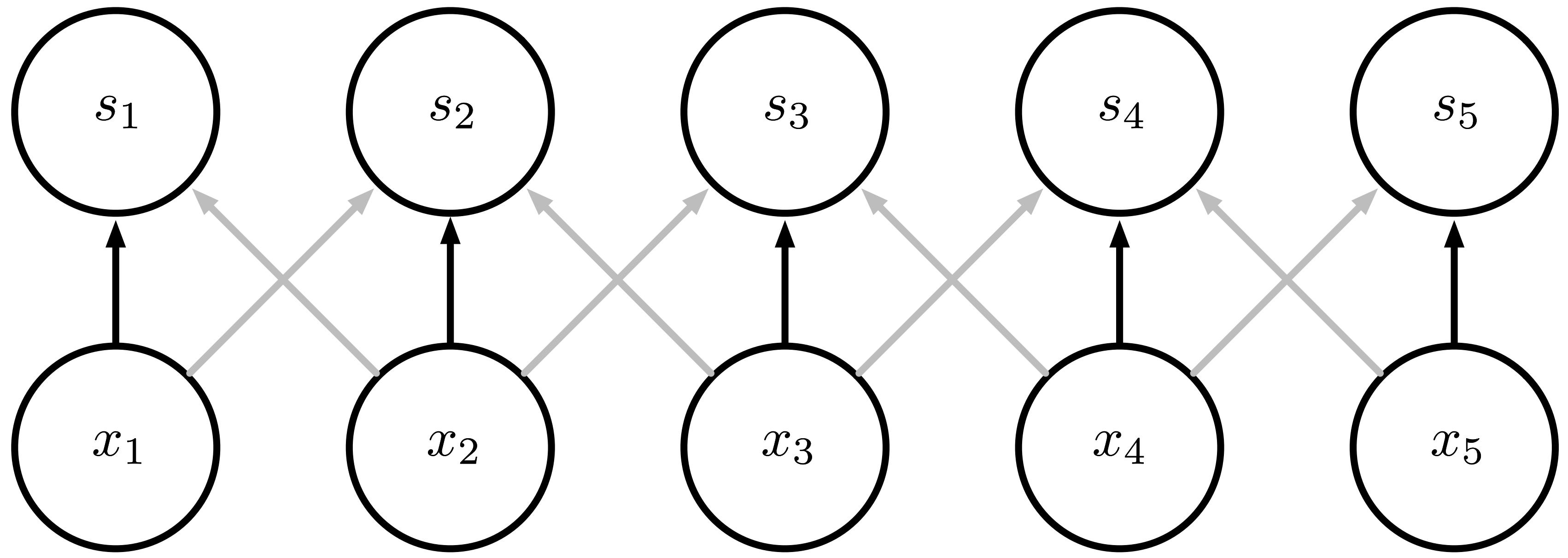
(Images: Goodfellow 2016)

2. Parameter Sharing

Traditional neural nets learn a **unique value** for **each connection**



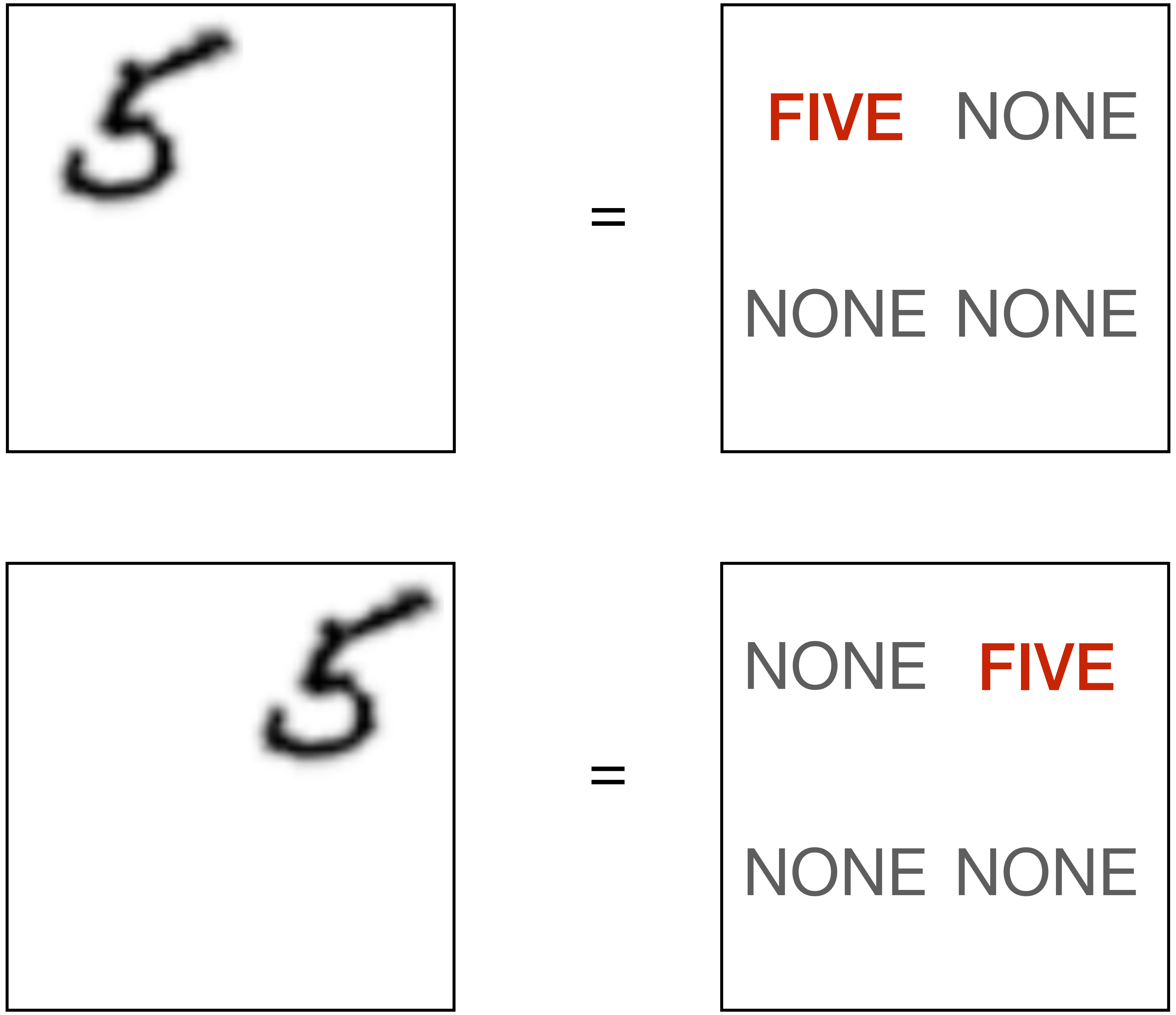
Convolutional neural nets **constrain** multiple parameters to be **equal**



(Images: Goodfellow 2016)

3. Equivariant Representations

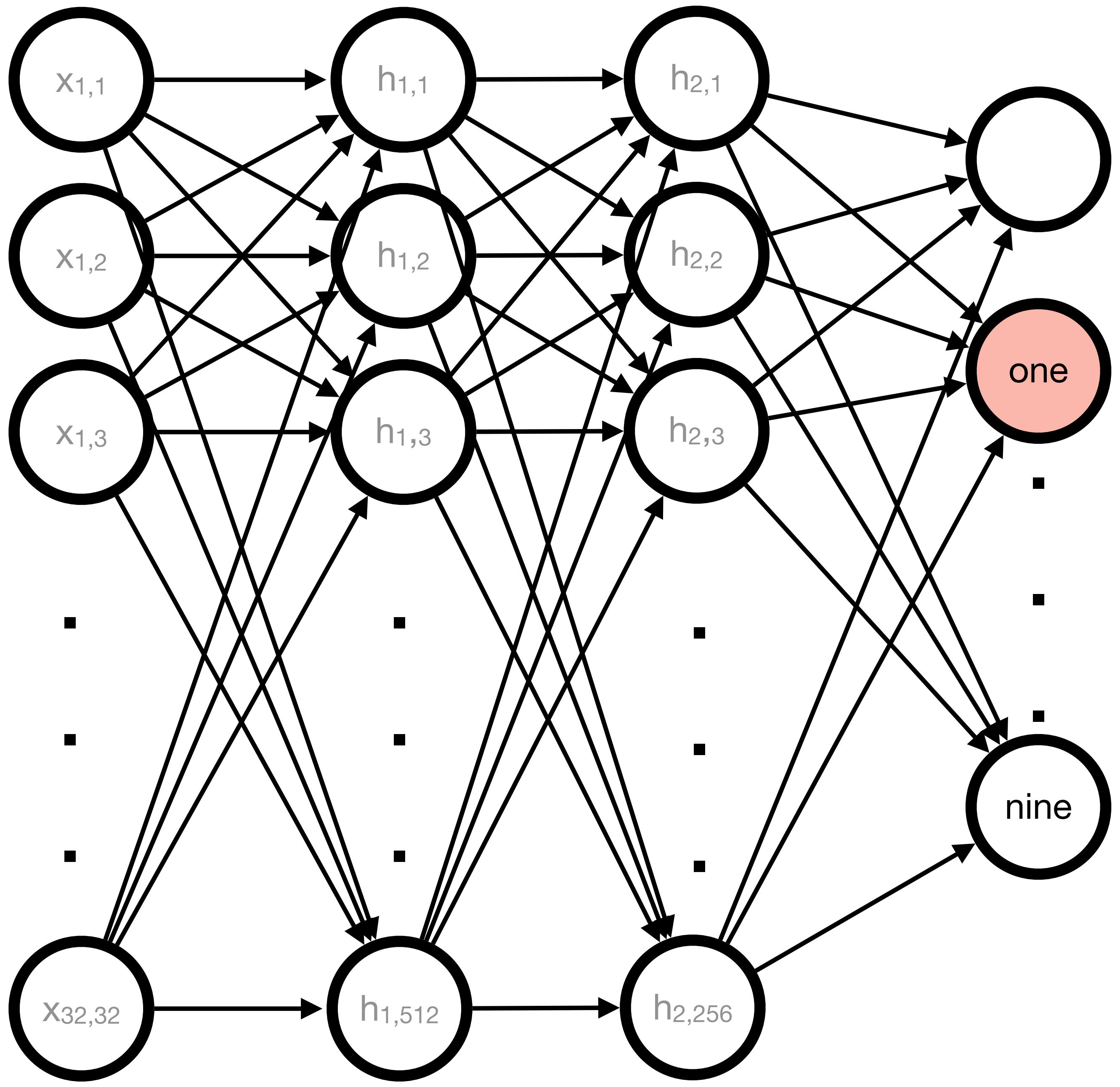
- We want to be able to recognize transformed versions of inputs we have seen before
  - e.g., translation
- Without having been **trained** on all transformed versions
- **Equivariance:** Changes in the input induce the **same changes** in the output



Operation: Matrix Product

Recall that we can represent the **activations** in a densely connected neural network by a **matrix product**

$$W^{(1)}\mathbf{x} = \begin{bmatrix} w^{(1)}_{x_1 \to h_1} & w^{(1)}_{x_2 \to h_1} & \cdots w^{(1)}_{x_n \to h_1} \\ w^{(1)}_{x_1 \to h_2} & \ddots & \\ \vdots & & \\ w^{(1)}_{x_1 \to h_m} & & w^{(1)}_{x_n \to h_m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



$$\mathbf{h_1} = g_h\left(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

$$\mathbf{h_2} = g_h\left(W^{(2)}\mathbf{h_1} + \mathbf{b}^{(2)}\right)$$

$$\mathbf{y} = g_y\left(W^{(3)}\mathbf{h_2} + \mathbf{b}^{(3)}\right)$$

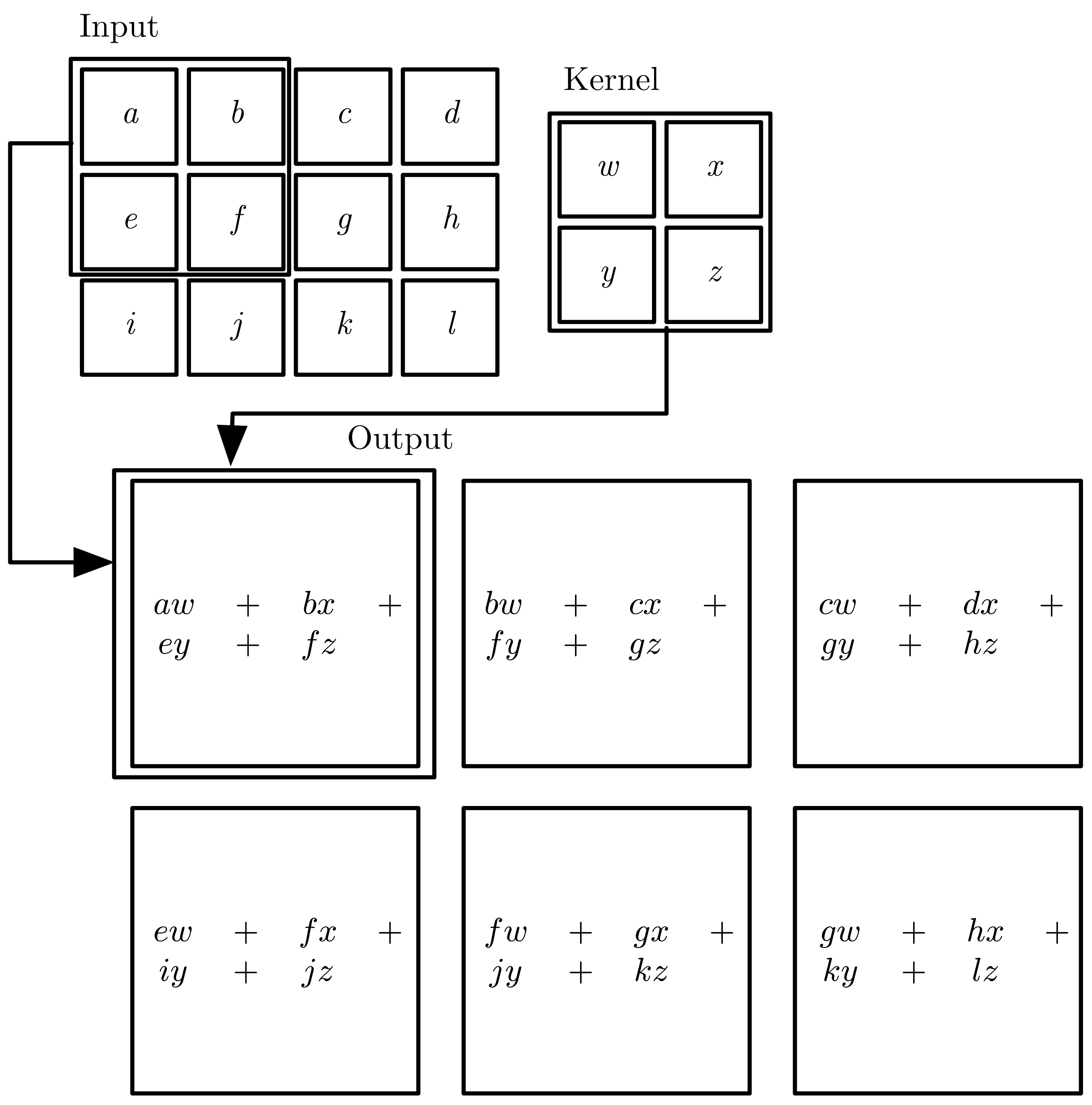
(Image: Goodfellow 2016)

Operation: 2D Convolution

Convolution scans a small block of weights (called the **kernel**) over the elements of the inputs, taking **weighted averages**

- Note that input and output dimensions **need not match**
- **Same weights** used for very many combinations

- The number of elements skipped by each "slide" is called the **stride**
  - This example has a stride of 1



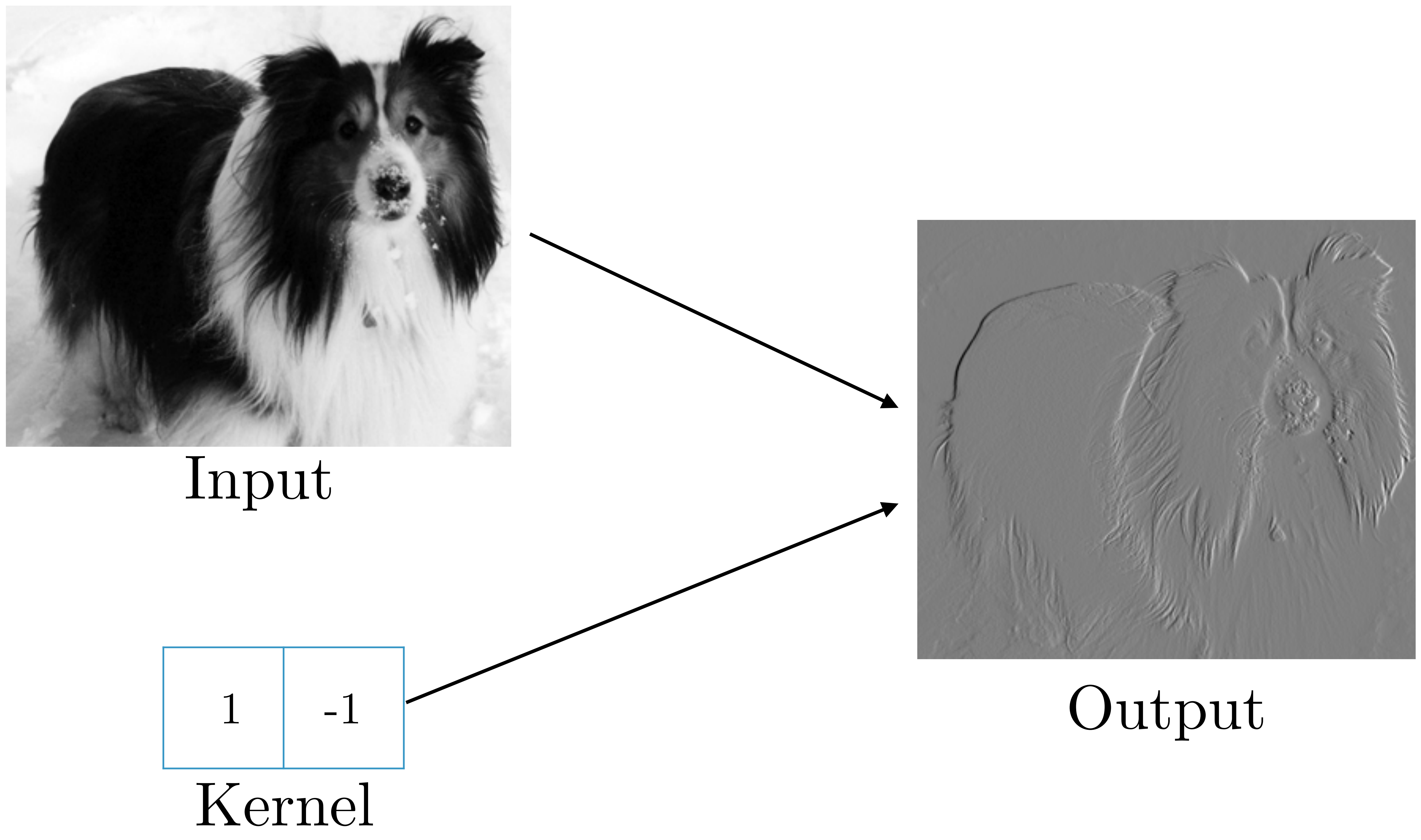
(Image: Goodfellow 2016)

Replace Matrix Multiplication by Convolution

Main idea: Replace matrix multiplications with convolutions

- **Sparsity:** Inputs only combined with **neighbours**
- **Parameter sharing:** Same kernel used for entire input

Example: Edge Detection



(Image: Goodfellow 2016)

Efficiency of Convolution

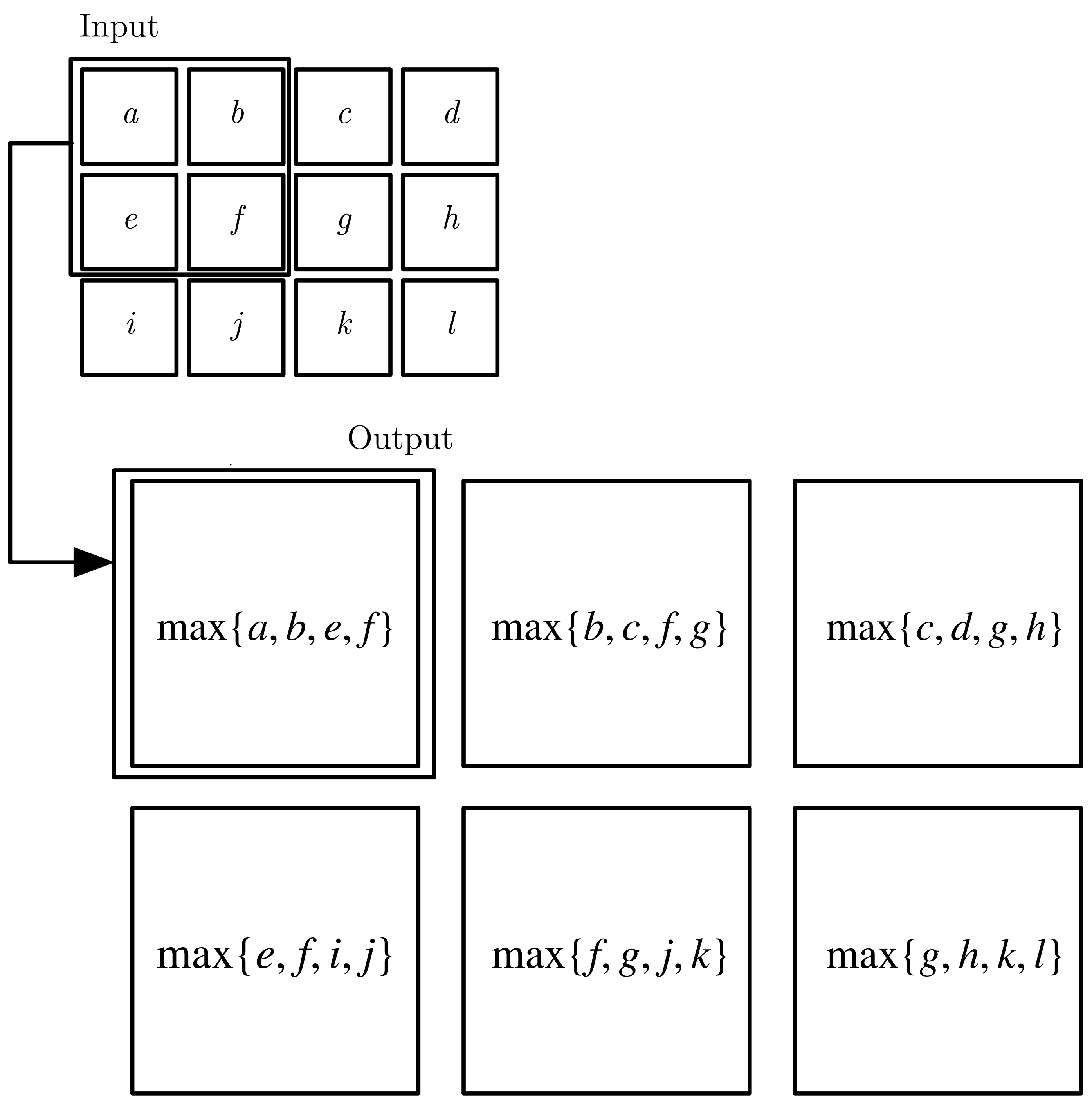
Input size: 320 by 280
Kernel size: 2 by 1
Output size: 319 by 280

	Dense matrix	Sparse matrix	Convolution
Stored floats	319*280*320*280 > 8e9	2*319*280 = 178,640	2
Float muls or adds	> 16e9	Same as convolution (267,960)	319*280*3 = 267,960

(Goodfellow 2016)

Operation: 2D Pooling

- Pooling **summarizes** its inputs into a single value, e.g.,
  - max
  - average
- Max-pooling is **parameter-free** (no bias or edge weights to learn)
  - This example has **stride** of 1



(Image: adapted from Goodfellow 2016)

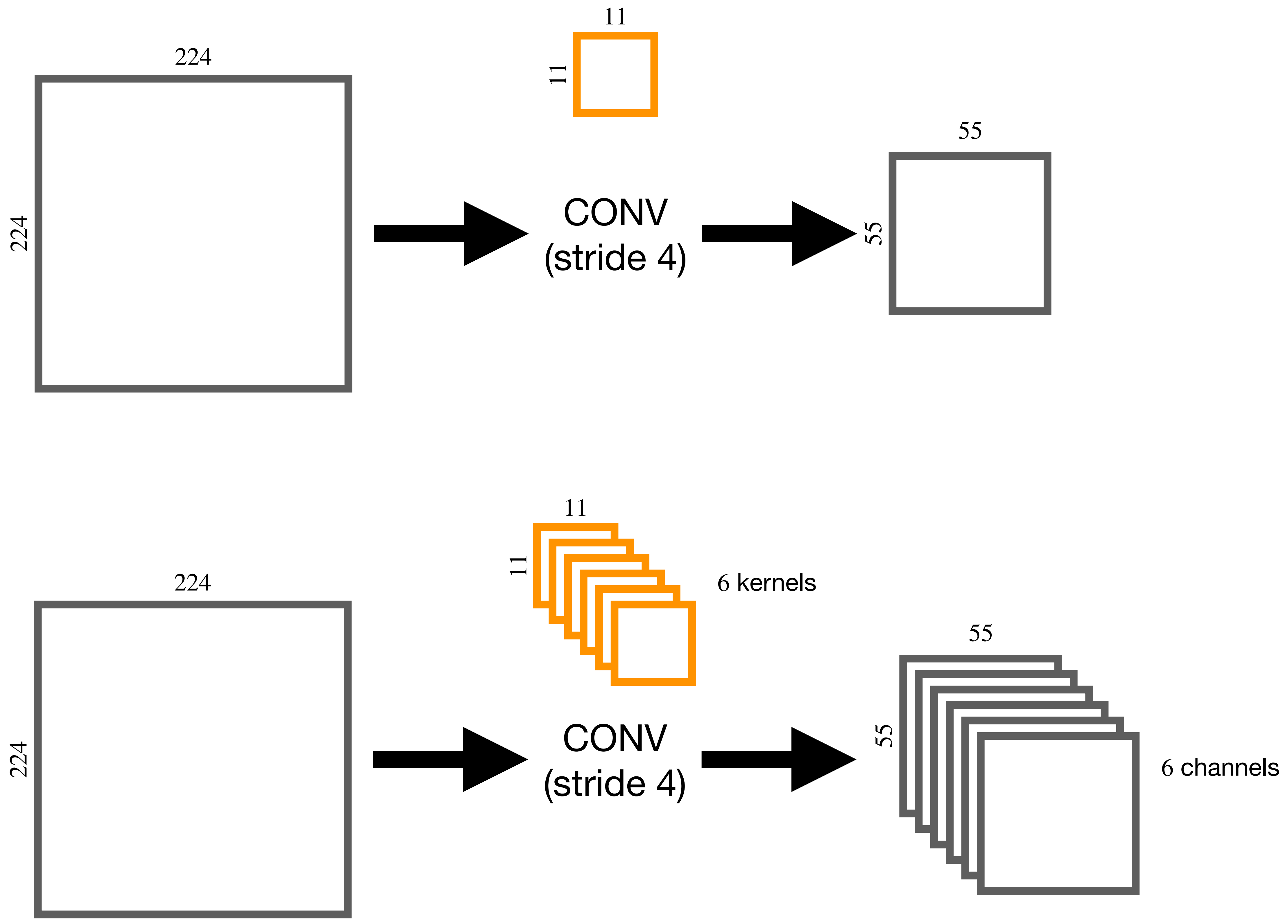
Operation: 1D Softmax

- **Softmax** converts a vector of real values into a vector of **probabilities**
- Often used as the **final operation** in a classifier

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \rightarrow \begin{bmatrix} \dfrac{\exp a}{\exp a + \exp b + \exp c} \\ \dfrac{\exp b}{\exp a + \exp b + \exp c} \\ \dfrac{\exp c}{\exp a + \exp b + \exp c} \end{bmatrix}$$

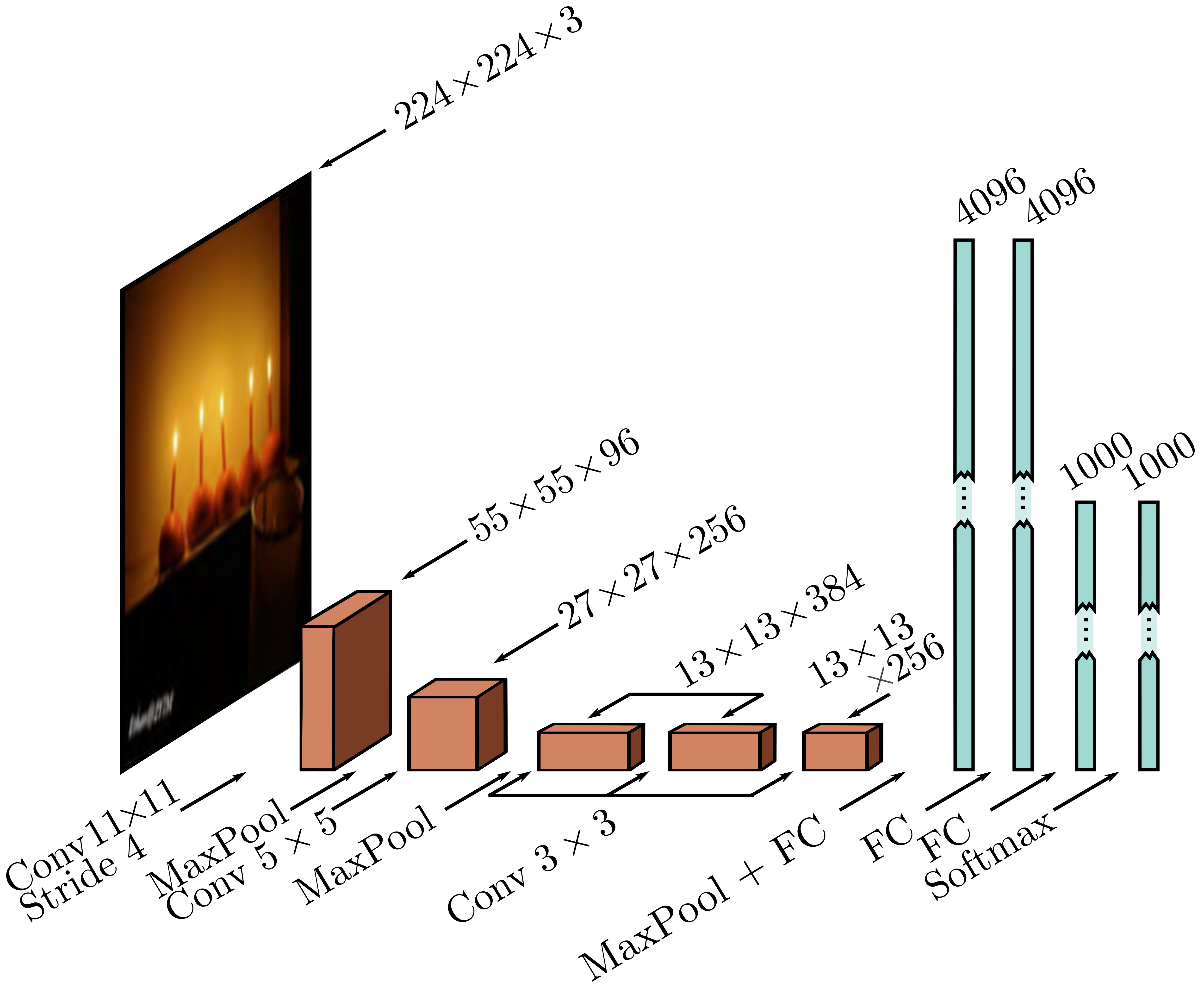
Channels & Kernels

- Convolution of a 224×224 image with an 11×11 kernel with a stride of 4 yields a **single** 55×55 output
- But we might want to learn more than one kernel!
- If we apply 6 **different** kernels to the input image, we will get 6 **different** 55×55 outputs
    - Each output is called a **channel**
    - Convolution with a single kernel yields a single **channel**



Example Architecture: AlexNet

[Krizhevsky et al. 2012]



Question:

1. How many **weights** are needed to convert the **43,264** vector after the final convolution layer into the **4096** vector of the next hidden layer?

2. How many **biases**?

(Image: Prince 2023)

Summary

- Classifying images with a standard feedforward network requires vast quantities of **parameters** (and hence **data**)
- Convolutional networks add **pooling** and **convolution**
  - Sparse connectivity
  - Parameter sharing
  - Translation equivariance
- Fewer parameters means far more **efficient to train**