Convolutional Neural Networks

CMPUT 261: Introduction to Artificial Intelligence

P §10.1-10.4.1, §10.5

Lecture Outline

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

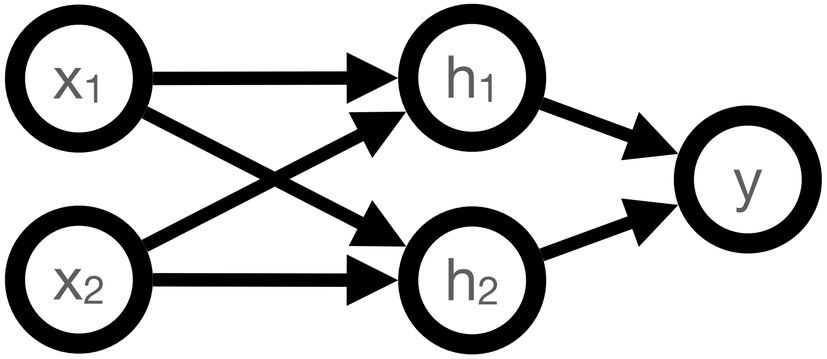
After this lecture, you should be able to:

- 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

Logistics

- **Assignment #3** is available
  - Due **Thursday, Nov 20**
  - Submit via Canvas (deadline is firm)
  - Blank submissions **will not be regraded**, nor granted EA

Recap: Feedforward Neural Network



A graph representing a feedforward neural network. There are two nodes labelled x1 and x2, two additional nodes labelled h1 and h2, and a final node labelled y. h1 has an incoming edge from x1 and an incoming edge from x2. h2 has an incoming edge from x1 and an incoming edge from x2. y has incoming edges from h1 and h2.

- 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**

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

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

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

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: $L(\mathbf{W}, \mathbf{b}) = \sum_i \ell\left(f(\mathbf{x}^{(i)}; \mathbf{W}, \mathbf{b}), y^{(i)}\right)$

Prediction Target

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

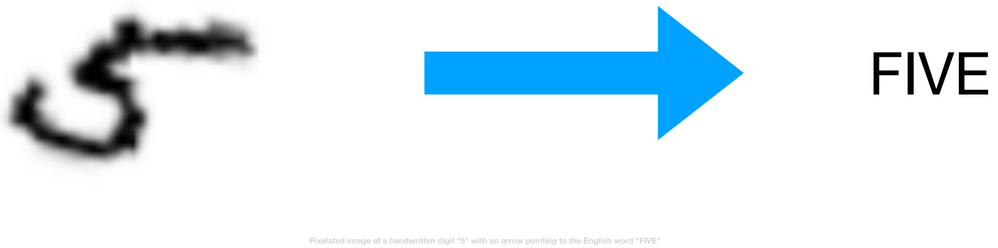
3. **Update parameters** to make loss smaller:

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

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



Pixelated image of a handwritten digit "5" with an arrow pointing to the English word "FIVE"

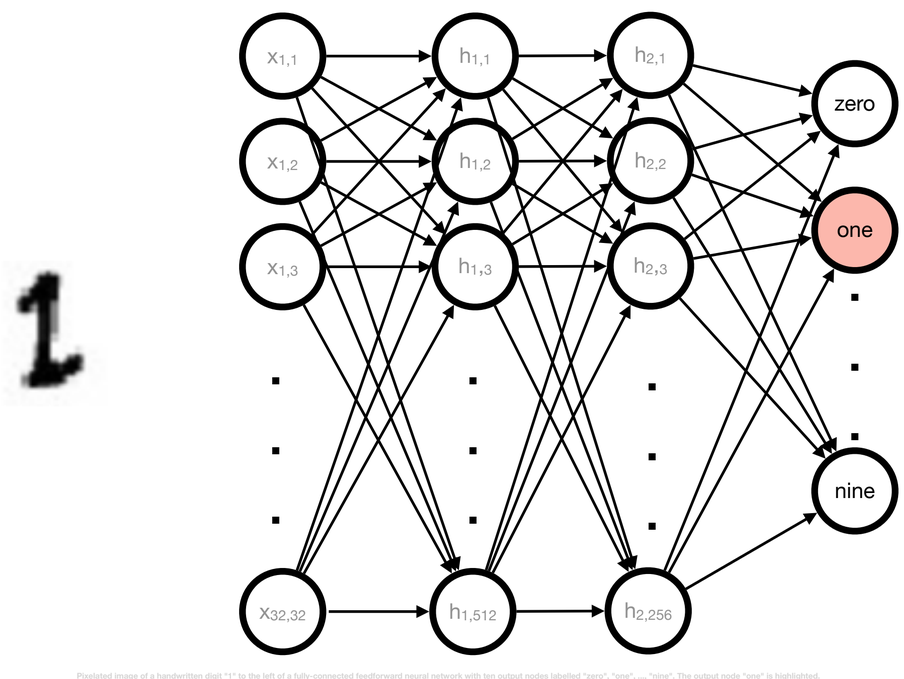
Problem: Recognize the handwritten digit from an image

- 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?

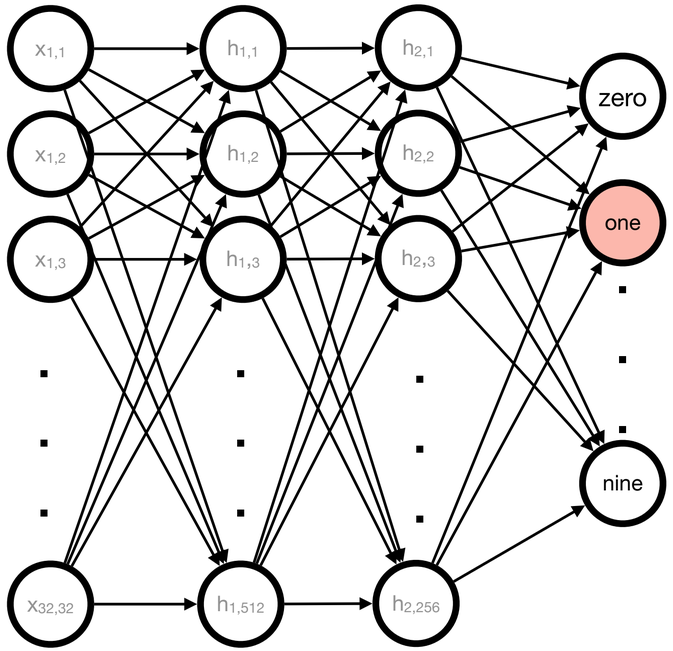
- How to represent the **inputs**?
- How to represent the **outputs**?
- What are the **parameters**?
- What is the **loss**?



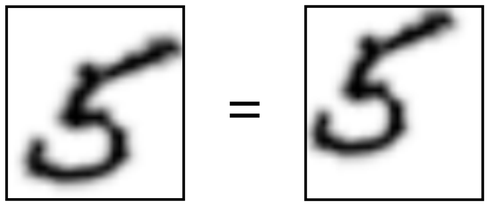
Pixelated image of a handwritten digit "1" to the left of a fully-connected feedforward neural network with ten output nodes labelled "zero", "one", ..., "nine". The output node "one" is highlighted.

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



A fully-connected feedforward neural network with ten output nodes labelled "zero", "one", ..., "nine". The output node "one" is highlighted.



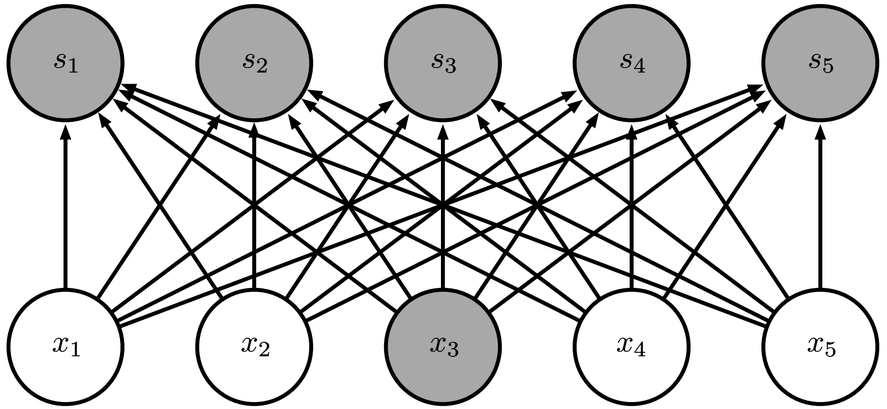
A box containing a pixelated handwritten digit "5" in the bottom right followed by an equal sign followed by another box containing an identical "5" in the top left .

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

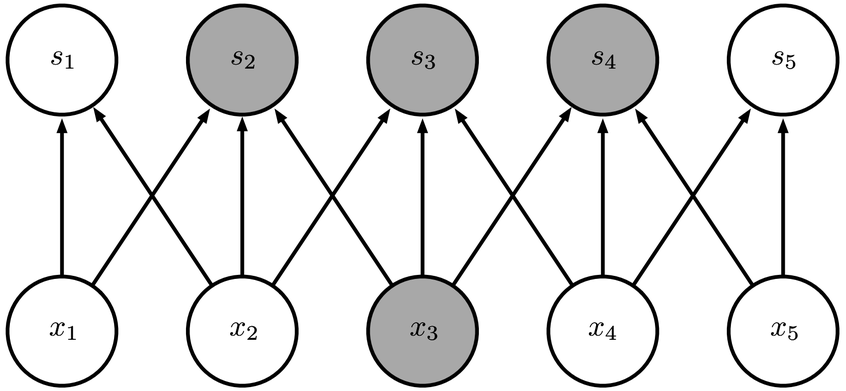
1. Sparse Interactions

Dense connections



A two-layer neural network, with five input units x1...x5 and five output units s1..s5. Each input unit has an edge outgoing to all five output units. Input unit x3 is highlighted, and so is every output unit.

Sparse connections

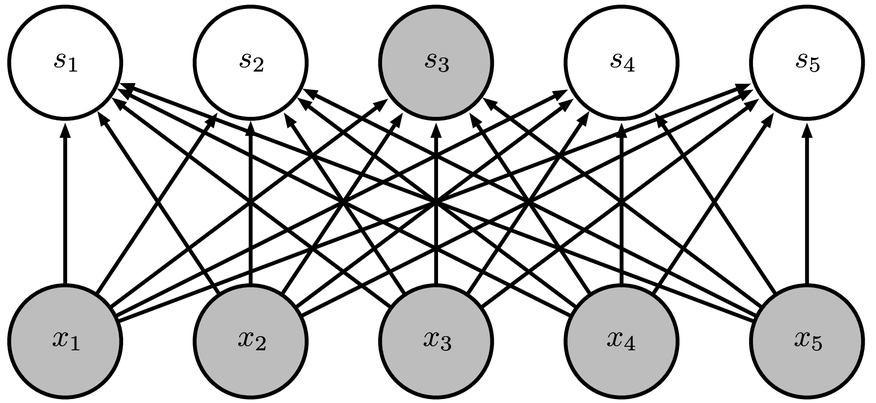


A two-layer neural network, with five input units x1...x5 and five output units s1..s5. Each input unit points to the output unit with the same index, one to the left (if present) and one to the right (if present). Input unit x3 is highlighted, and so are output units s2, s3, and s4 (the units to which x3 has outgoing edges).

(Images: Goodfellow 2016)

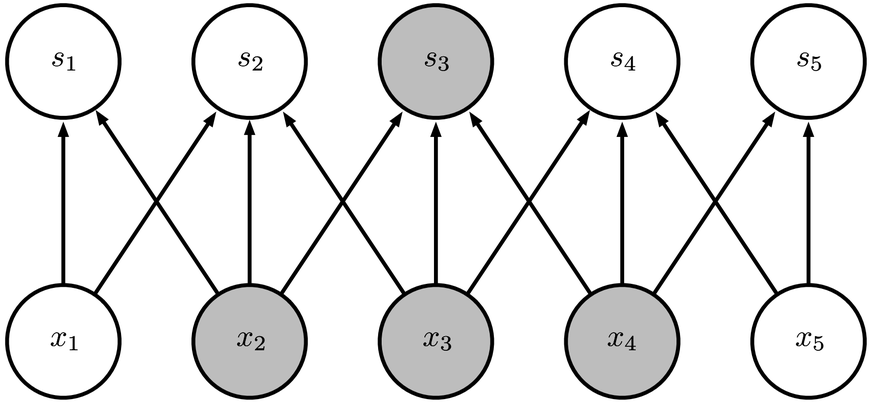
1. Sparse Interactions

Dense connections



A two-layer neural network, with five input units x1...x5 and five output units s1..s5. Each input unit has an edge outgoing to all five output units. Output s3 is highlighted, and so are all input units.

Sparse connections

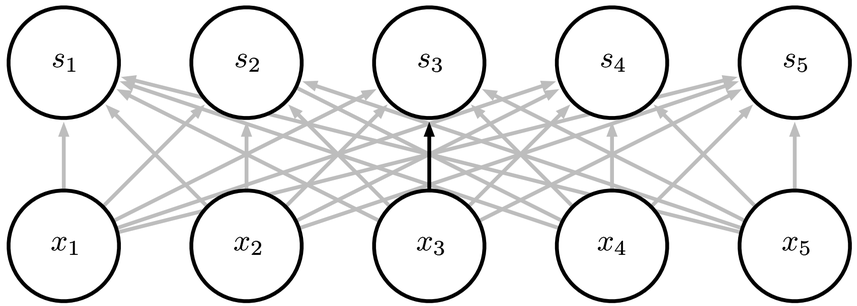


A two-layer neural network, with five input units x1...x5 and five output units s1..s5. Each input unit points to the output unit with the same index, one to the left (if present) and one to the right (if present). Output unit s3 is highlighted, and so are input units x2, x3, and x4 (the units which have outgoing edges to s3).

(Images: Goodfellow 2016)

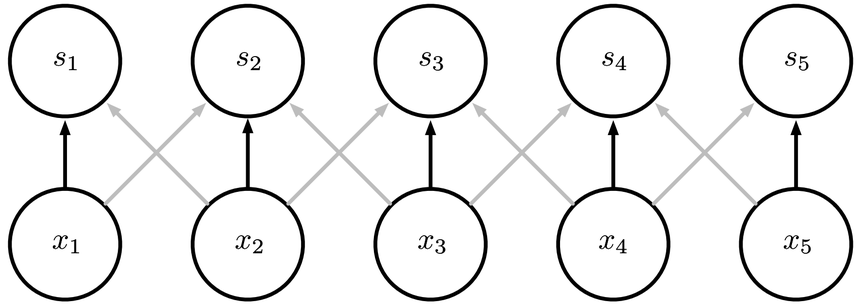
2. Parameter Sharing

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



A two-layer neural network, with five input units x1...x5 and five output units s1..s5. Each input unit has an edge outgoing to all five output units. The edge from x3 to s3 is highlighted.

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

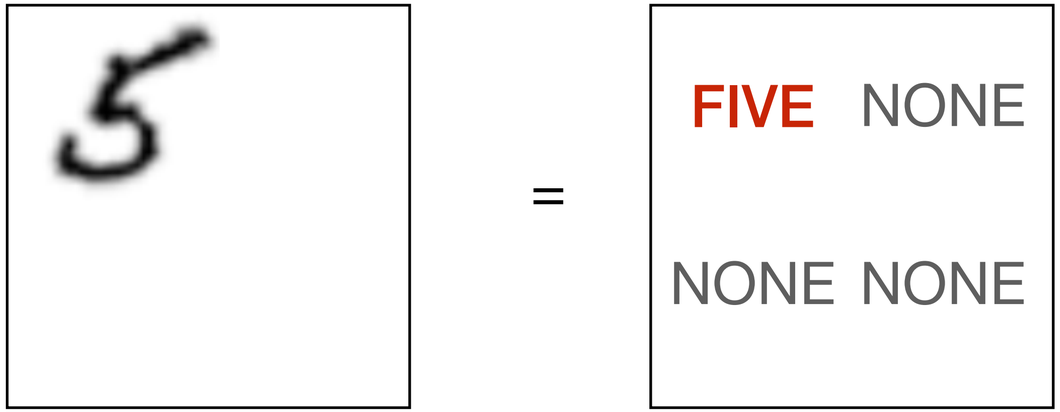


A two-layer neural network, with five input units x1...x5 and five output units s1..s5. Each input unit points to the output unit with the same index, one to the left (if present) and one to the right (if present). The edges from x1 to s1, x2 to s2, x3 to s3, x4 to s4, and x5 to s5 are highlighted.

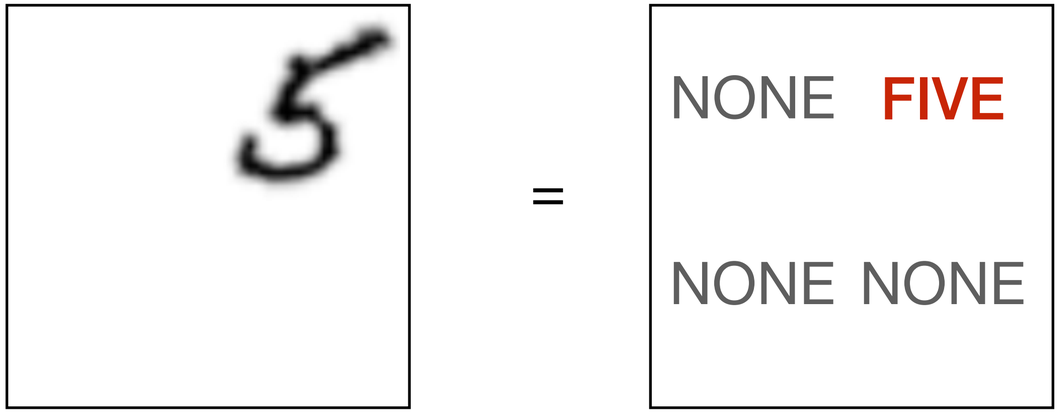
(Images: Goodfellow 2016)

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



A box with a pixellated handwritten "5" in the top left corner, followed by an equal sign, followed by a box that has "FIVE" in the top left corner and "NONE" in the other three corners.

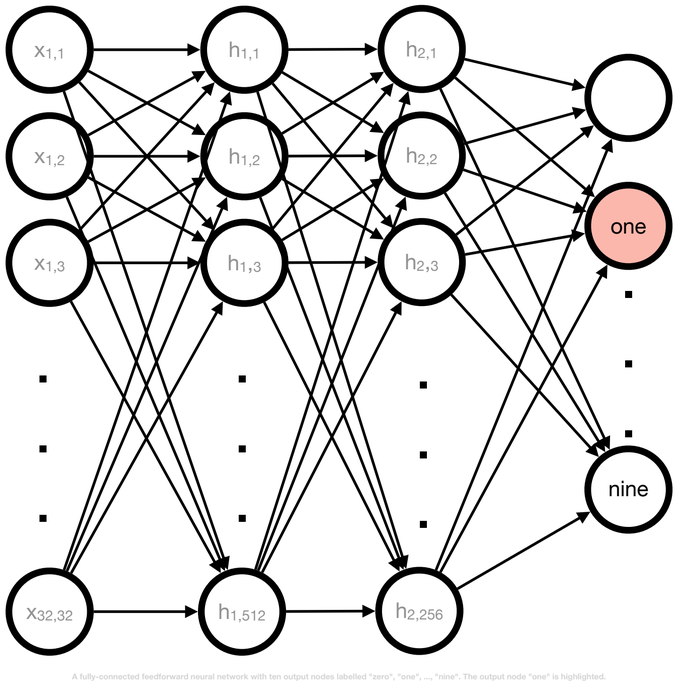


A box with a pixellated handwritten "5" in the top right corner, followed by an equal sign, followed by a box that has "FIVE" in the top right corner and "NONE" in the other three corners.

Operation: Matrix Product

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

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



A fully-connected feedforward neural network with ten output nodes labelled "zero", "one", ..., "nine". The output node "one" is highlighted.

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

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

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

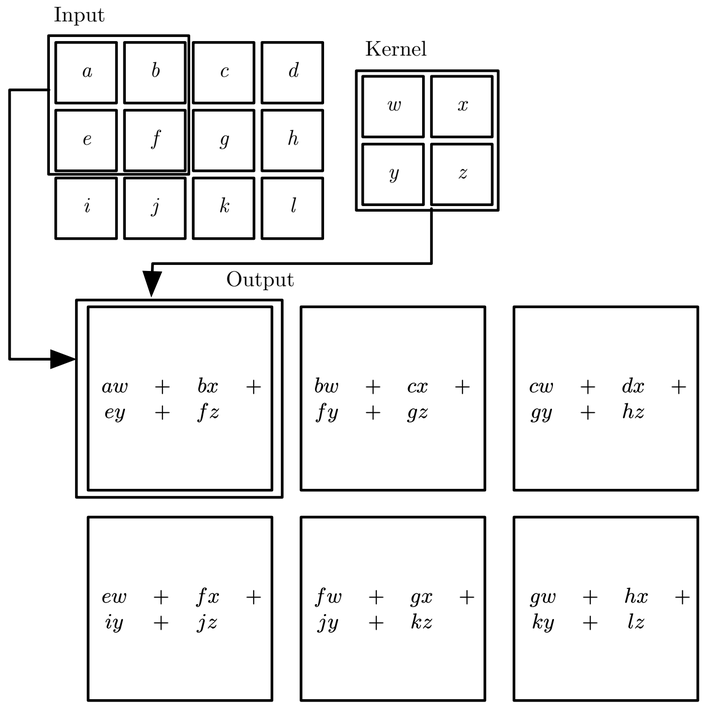
(Image: Goodfellow 2016)

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



Squares respectively containing "a,b,c,...,l" arranged in 3 rows and 4 columns, with the top-left four squares (a,b,e,f) highlighted and labelled "input". Next to this, four squares respectively containing "w,x,y,z" arranged in 2 rows and columns, highlighted and labelled "kernel". Below, large squares arranged in 2 rows and 3 columns. The "input" and "Kernel" squares have arrows pointing to the top-left large square, labelled "Output". The top-left large square contains "aw + bx + ey + fz". The top-center square contains "bw + cx + fy + gz". The top-right large square contains "cw + dx + gy + hz". The bottom-left large square contains "ew + fx + iy + jz". The bottom-center large square contains "fw + gx + jy + kz". The bottom-right large square contains "gw + hx + ky + lz".

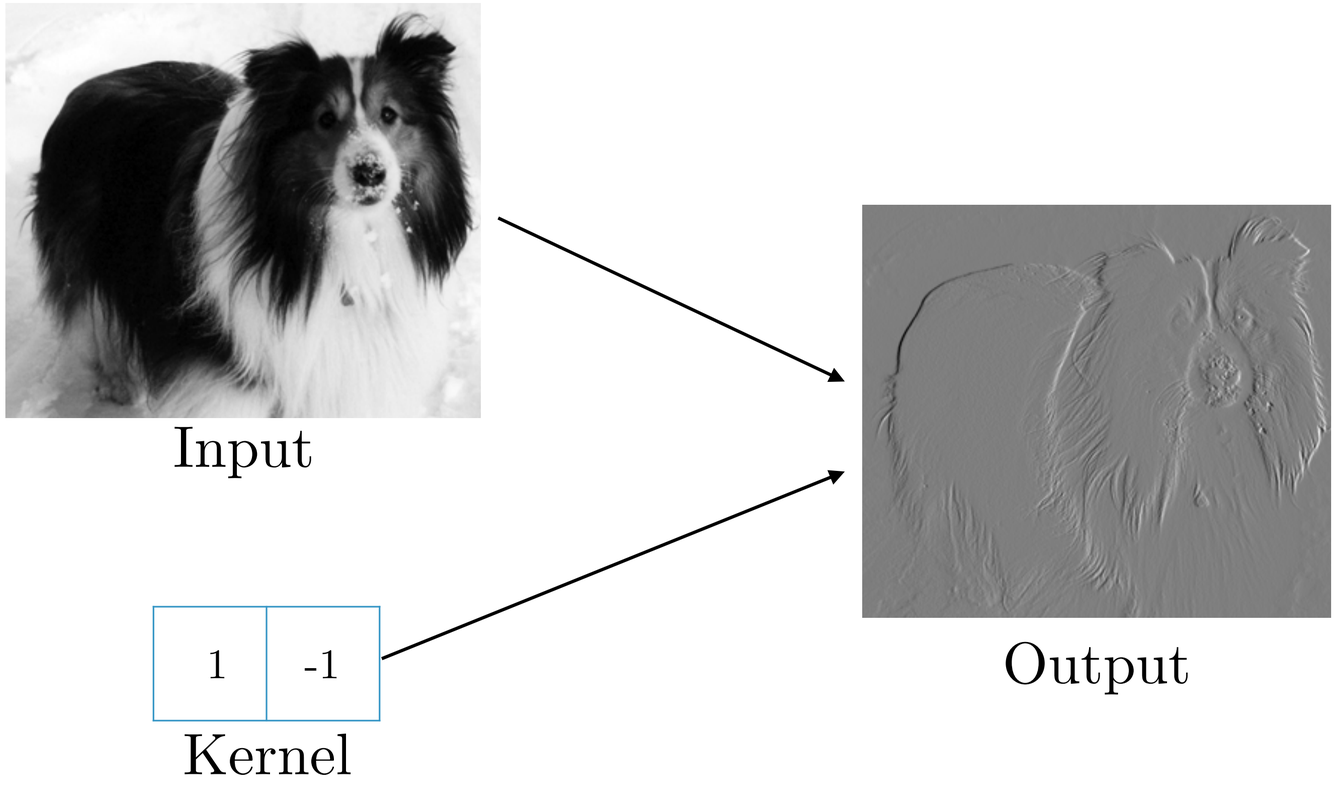
(Image: Goodfellow 2016)

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



An image of a dog labeled "input" and two boxes arranged horizontally. with 1 in the left box and -1 in the right box labeled "kernel". An arrow from each to a greyscale image labelled "output" where edges of the dog are either black or white.

(Image: Goodfellow 2016)

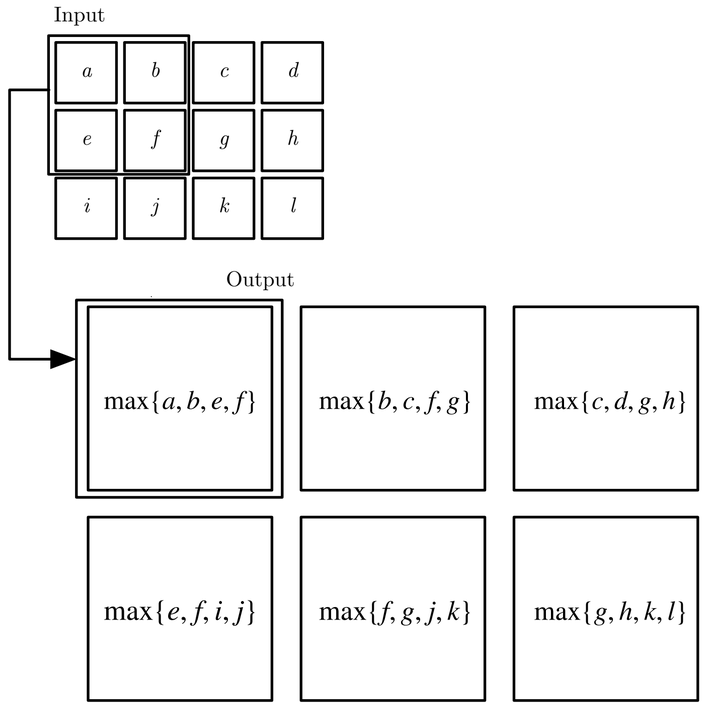
Efficiency of Convolution

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

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

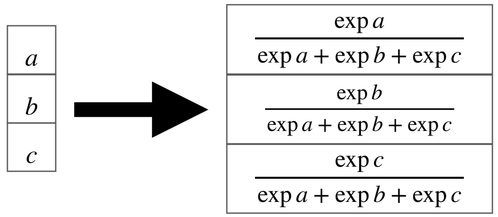


Squares respectively containing "a,b,e,...,l" arranged in 3 rows and 4 columns, with the top-left four squares (a,b,e,f) highlighted and labelled "input". Below, large squares arranged in 2 rows and 3 columns. The "input" square has an arrow pointing to the top-left large square, labelled "Output". The top-left large square contains "max{a,b,e,f}". The top-center large square contains "max{b,c,f,g}". The top-right large square contains "max{c,d,g,h}". The bottom-left large square contains "max{e,f,i,j}". The bottom-center large square contains "max{f,g,j,k}". The bottom-right large square contains "max{g,h,k,l}".

(Image: adapted from Goodfellow 2016)

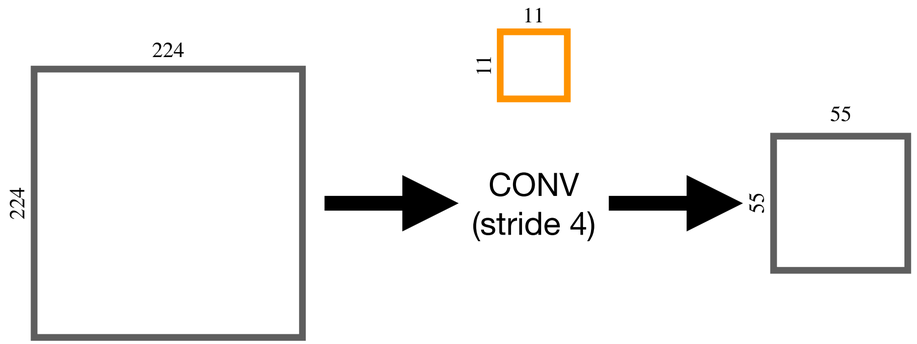
Operation: 1D Softmax

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

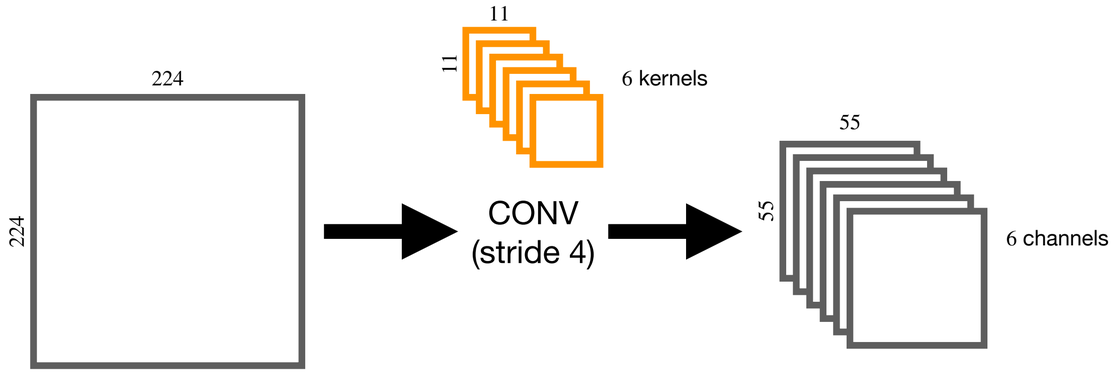


Channels & Kernels

- Convolution of a 224×224 image with an 11×11 kernel with a stride of 4 (and 0-padding of 2 on the right and bottom) 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**

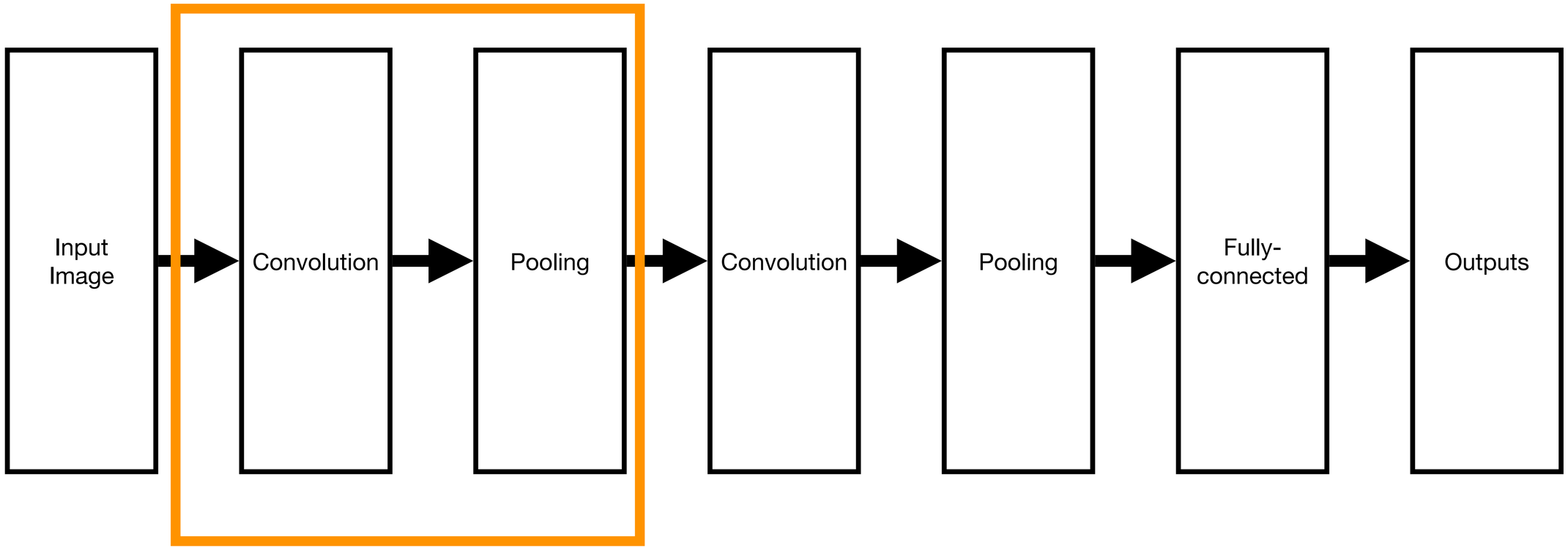


A 224 by 224 box, with an arrow pointing to the words "CONV (stride 4)", which has another arrow pointing to a 55 by 55 box. Above "CONV (stride 4)" is an orange 11 by 11 box.



A 224 by 224 box, with an arrow pointing to the words "CONV (stride 4)", which has another arrow pointing to 6 55 by 55 boxes, labeled "6 channels". Above "CONV (stride 4)" are 6 orange 11 by 11 boxes, labelled "6 kernels".

Typical Architecture

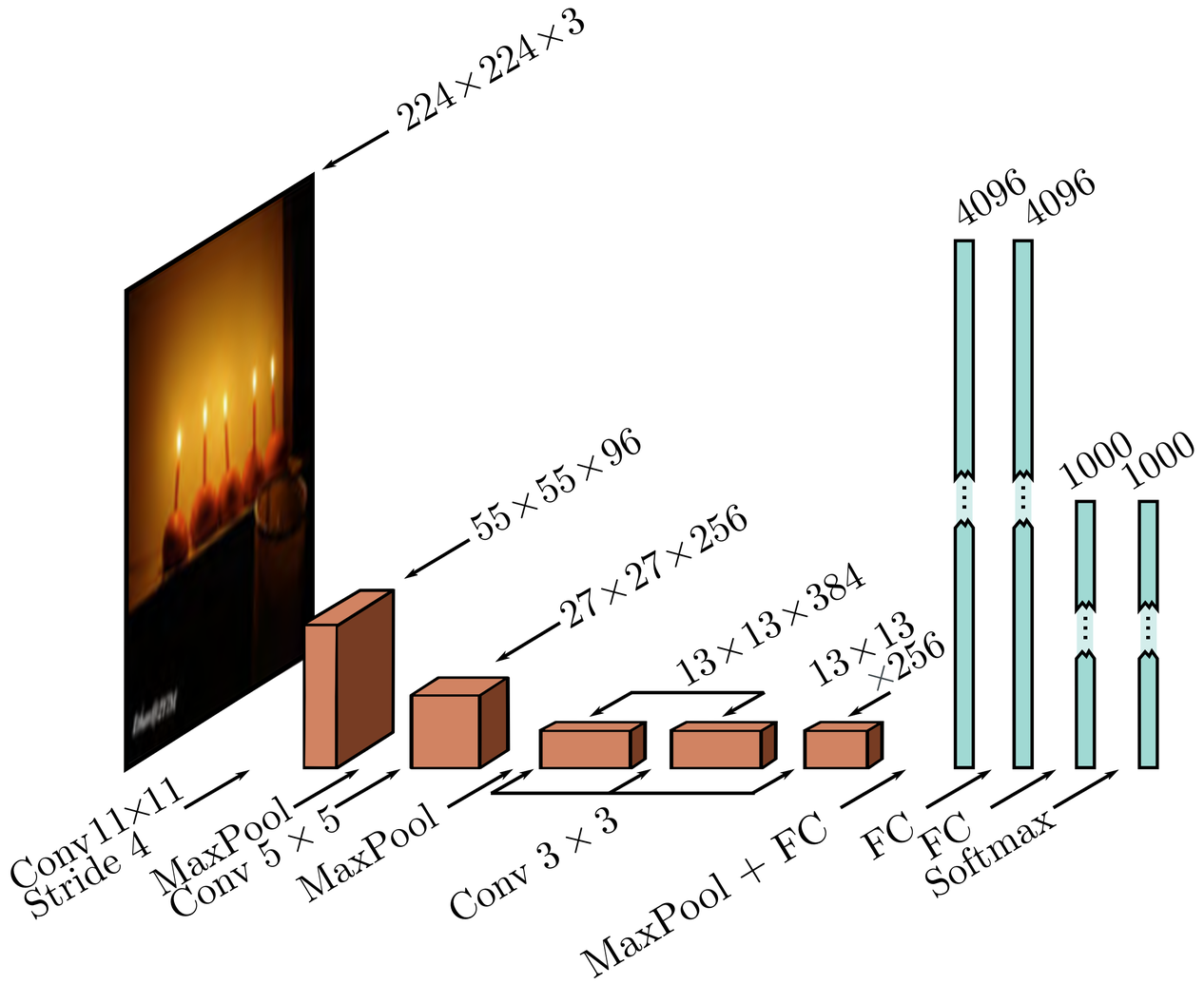


Often convolution-then-pooling is collectively referred to as a "**convolution layer**"

A rectangle labelled "input image" with an arrow pointing to a rectangle labelled "pooling" with an arrow pointing to a rectangle labelled "convolution" with an arrow pointing to a rectangle labelled "pooling" with an arrow pointing to a rectangle labelled "fully connected" with an arrow pointing to a rectangle labelled "outputs". The first two rectangles labelled "convolution" and "pooling" are in an orange outline.

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)

A color image of candles, labelled "224x224x3", with an arrow labelled "Conv 11x11 Stride 4" pointing to a flat slab labelled "55x55x96", with an arrow labelled "MaxPool" followed by an arrow labelled "Conv 5x5" pointing to a cube labelled "27x27x256" with an arrow labelled "MaxPool" followed by an arrow labelled "Conv 3x3" pointing to a long slab labelled "13x13x384" with an arrow also labelled "Conv 3x3" pointing to another long slab also labelled "13x13x384" followed by an arrow labelled "Conv 3x3" pointing to a slab labelled "13x13x256" with an arrow labelled "MaxPool + FC" pointing to a long rectangle labelled "4096" with an arrow labelled "FC" pointing to a long rectangle also labelled "4096" with an arrow labelled "FC" pointing to a long rectangle labelled "1000" with an arrow labelled "Softmax" pointing to a long rectangle also labelled "1000".

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**