Deep Convolutional Neural Nets

COMPSCI 371D — Machine Learning

Outline

Why Neural Networks?

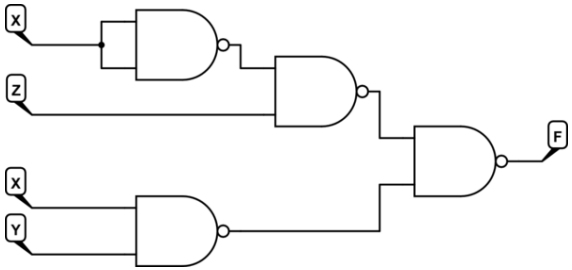
- Neural networks are very *expressive* (large $\mathcal{H}$)
- Can approximate any well-behaved function from a hypercube in $\mathbb{R}^d$ to an interval in $\mathbb{R}$ within any $\epsilon > 0$
- *Universal approximators*
- However
  - Complexity grows exponentially with $d = \dim(X)$
  - L_T is not convex (not even close)
  - Large $\mathcal{H}$ $\Rightarrow$ overfitting $\Rightarrow$ *lots* of data!
- Amazon's Mechanical Turk made deep neural networks possible
- Even so, we cannot keep up with the curse of dimensionality!

Why Neural Networks?

- Neural networks are data hungry
- Availability of lots of data is not a sufficient explanation
- There must be deeper reasons
- Special structure of image space (or audio space, or language)?
- Specialized network architectures?
- Regularization tricks and techniques?
- We don't really know. Stay tuned...
- Be prepared for some hand-waving and empirical statements

Circuits

- Describe implementation of $h : X \to Y$ on a computer
- Algorithm: A finite sequence of steps
- *Circuit*: Many *gates* of few types, wired together



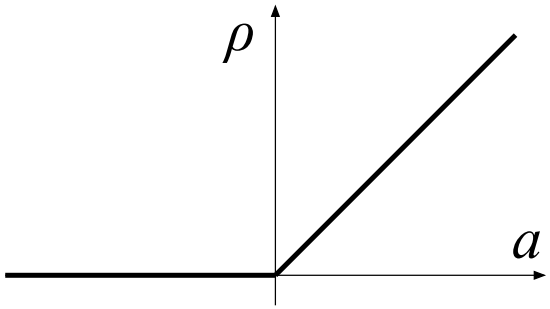
- These are NAND gates. We'll use *neurons*
- Algorithms and circuits are equivalent
- Algorithm can simulate a circuit
- Computer is a circuit that runs algorithms!
- Computer really only computes Boolean functions...

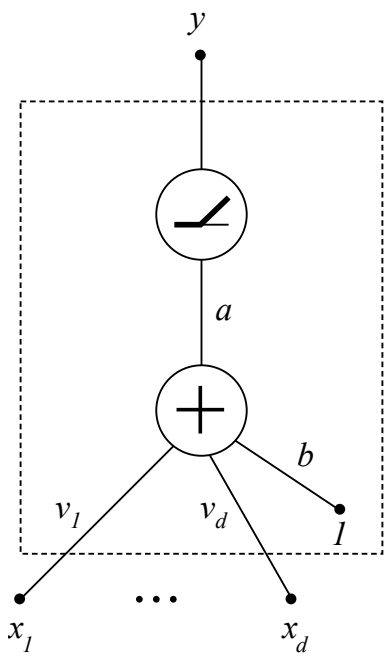
Deep Neural Networks as Circuits

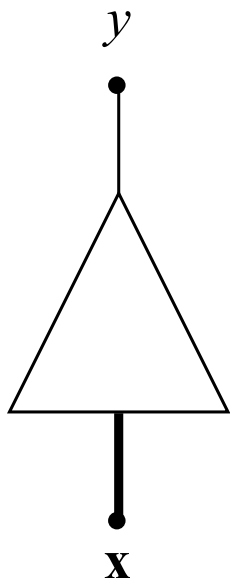
- Neural networks are typically described as circuits
- Nearly always implemented as algorithms
- One gate type, the *neuron*
- Many neurons that receive the same input form a *layer*
- A cascade of layers is a *network*
- A *deep* network has many layers
- Layers with a special constraint are called *convolutional*

The Neuron

- $y = \rho(a(\mathbf{x}))$ where $a = \mathbf{v}^T\mathbf{x} + b$
  $\mathbf{x} \in \mathbb{R}^d,\ y \in \mathbb{R}$
- $\mathbf{v}$ are the *gains*, $\mathbf{w} = \begin{bmatrix} \mathbf{v} \\ b \end{bmatrix}$ are the *weights*
- $\rho(a) = \max(0, a)$ (ReLU, Rectified Linear Unit)

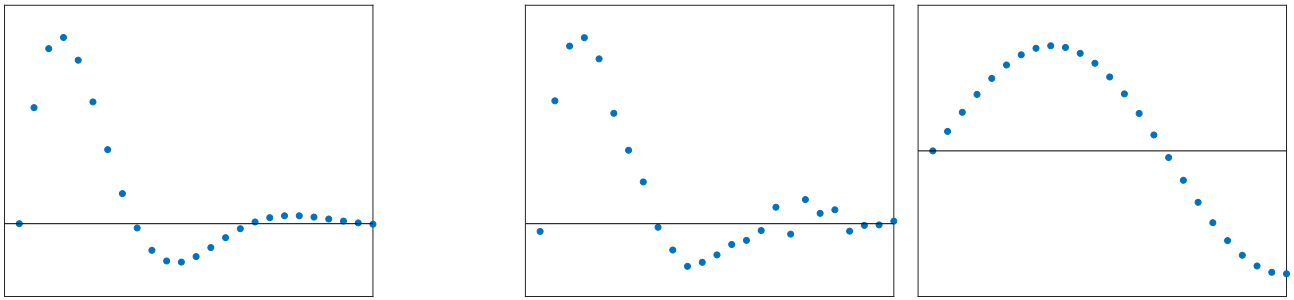






The Neuron as a Pattern Matcher (Almost)

- Left pattern is a drumbeat **g** (a pattern template):



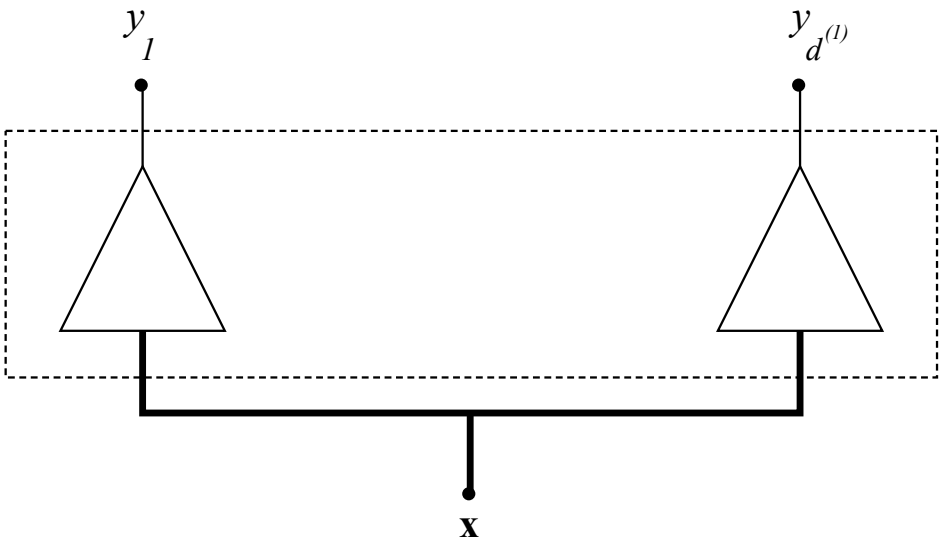
- Which of the other two patterns **x** is a drumbeat?
- Normalize both **g** and **x** so that $\|\mathbf{g}\| = \|\mathbf{x}\| = 1$
- Then $\mathbf{g}^T\mathbf{x}$ is the cosine of the angle between the patterns
- If $\mathbf{g}^T\mathbf{x} \geq -b$ for some threshold $-b$, output $a = \mathbf{g}^T\mathbf{x} + b$ (amount by which the cosine exceeds the threshold) otherwise, output 0
- $y = \rho(\mathbf{g}^T\mathbf{x} + b)$

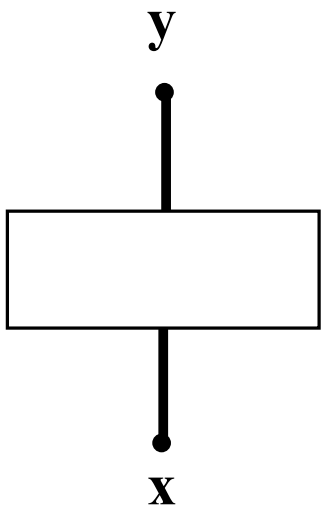
The Neuron as a Pattern Matcher (Almost)

- $y = \rho(\mathbf{g}^T\mathbf{x} + b)$
- A neuron is a pattern matcher, except for normalization and with a partial decision function
- In neural networks, normalization may happen in later or earlier layers
- This interpretation is not necessary to understand neural networks
- Nice to have a mental model, though
- Many neurons wired together can approximate any function we want: A neural network is a *universal function approximator*

Layers and Networks

- A *layer* is a set of neurons that share the same input





- A *neural network* is a cascade of layers
- A neural network is *deep* if it has many layers
- *Two* layers can make a universal approximator
- If neurons did not have nonlinearities, any cascade of layers would collapse to a single layer

Convolutional Layers

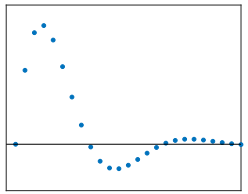
- A layer with input $\mathbf{x} \in \mathbb{R}^d$ and output $\mathbf{y} \in \mathbb{R}^e$ has e neurons, each with d gains and one bias
- Total of $(d+1)e$ weights to be trained in a single layer
- For images, d, e are in the order of hundreds of thousands or even millions
- Too many parameters
- *Convolutional layers* are layers restricted in a special way
- Many fewer parameters to train
- Also good justification in terms of basic principles

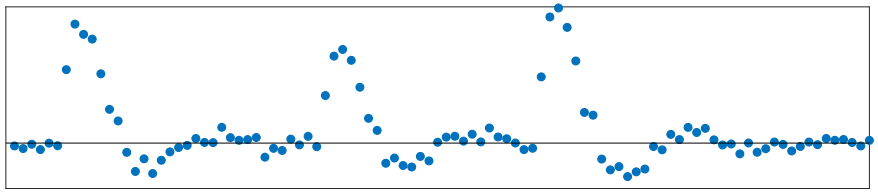
Hierarchy, Locality, Reuse

- To find a person, look for a face, a torso, limbs,...
- To find a face, look for eyes, nose, ears, mouth, hair,...
- To find an eye look for a circle, some corners, some curved edges,...
- A *hierarchical* image model is less sensitive to viewpoint, body configuration, ...
- Hierarchy leads to a *cascade* of layers
- Low-level features are *local*: A neuron doesn't need to see the entire image
- Circles are circles, regardless of where they show up: A single neuron can be *reused* to look for circles anywhere in the image

Correlation, Locality, and Reuse

- Does the drumbeat on the left show up in the clip on the right?





- Drumbeat **g** has 25 samples, clip **x** has 100
- Make $100 - 25 + 1 = 76$ neurons that look for **g** in every possible position
- $y_i = \rho(\mathbf{v}_i^T \mathbf{x} + b_i)$ where $\mathbf{v}_i^T = [\underbrace{0, \ldots, 0}_{i-1}, \underbrace{g_0, \ldots, g_{24}}_{\mathbf{g}}, \underbrace{0, \ldots 0}_{76-i}]$
- Layer gain matrix $V =$

$$\begin{bmatrix} g_0 & \cdots & g_{24} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & \cdots & g_{24} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & g_0 & \cdots & g_{24} \end{bmatrix}$$

Compact Computation

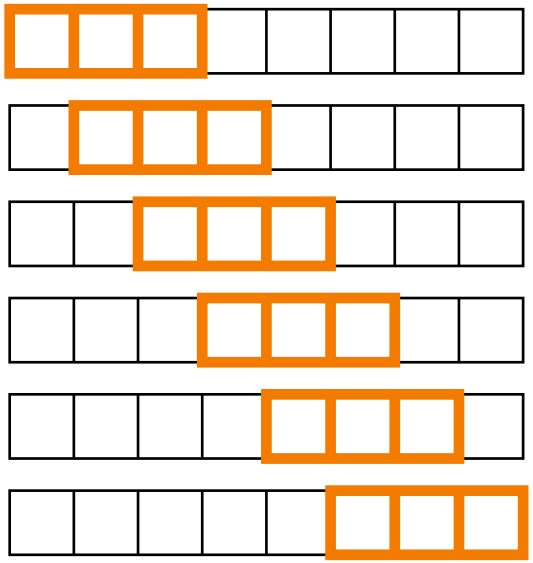
- Gain matrix $V = \begin{bmatrix} g_0 & \cdots & g_{24} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & \cdots & g_{24} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & g_0 & \cdots & g_{24} \end{bmatrix}$
- Rows of $V\mathbf{x}$: $z_i = \mathbf{v}_i^T \mathbf{x} = \sum_{a=0}^{24} g_a x_{i+a}$ for $i = 0, \ldots, 75$
- In general,

$$z_i = \sum_{a=0}^{k-1} g_a x_{i+a} \quad \text{for} \quad i = 0, \ldots, e-1 = 0, \ldots, d-k$$

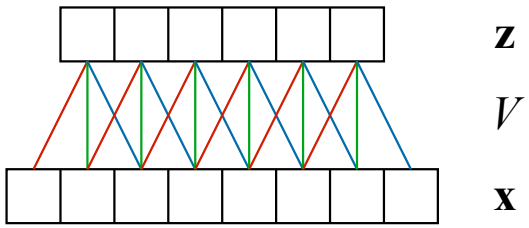
- *(One-dimensional) correlation*
- **g** is the *kernel*

A Small Example

$$z_i = \sum_{a=0}^{2} g_a x_{i+a} \quad \text{for} \quad i = 0, \ldots, 5$$



$$\mathbf{z} = V\mathbf{x} = \begin{bmatrix} g_0 & g_1 & g_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & g_0 & g_1 & g_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & g_0 & g_1 & g_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 \end{bmatrix} \mathbf{x}$$

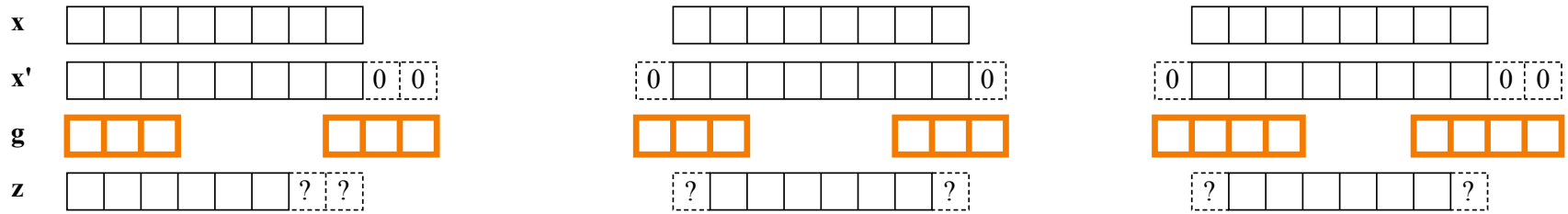


Correlation and Convolution

- The correlation of $\mathbf{x}$ with $\mathbf{g} = [g_0, \ldots, g_{k-1}]$ is the *convolution* of $\mathbf{x}$ with $\mathbf{r} = [r_0, \ldots, r_{k-1}] = [g_{k-1}, \ldots, g_0]$
- There are deep reasons why mathematicians prefer convolution to correlation
- We do not need to get into these, but see notes
- A layer whose gain matrix V is a correlation matrix is called a *convolutional layer*
- Also includes bias b

Input Padding

- If the input has d entries and the kernel has k, then the output has $e = d - k + 1$ entries
- This shrinkage is inconvenient when cascading several layers
- Pad input with $k - 1$ zeros to make the output have d entries
- Padding is typically asymmetric when index is time, symmetric when index is position in space



- *Shape-preserving* or *'same'* correlation or convolution

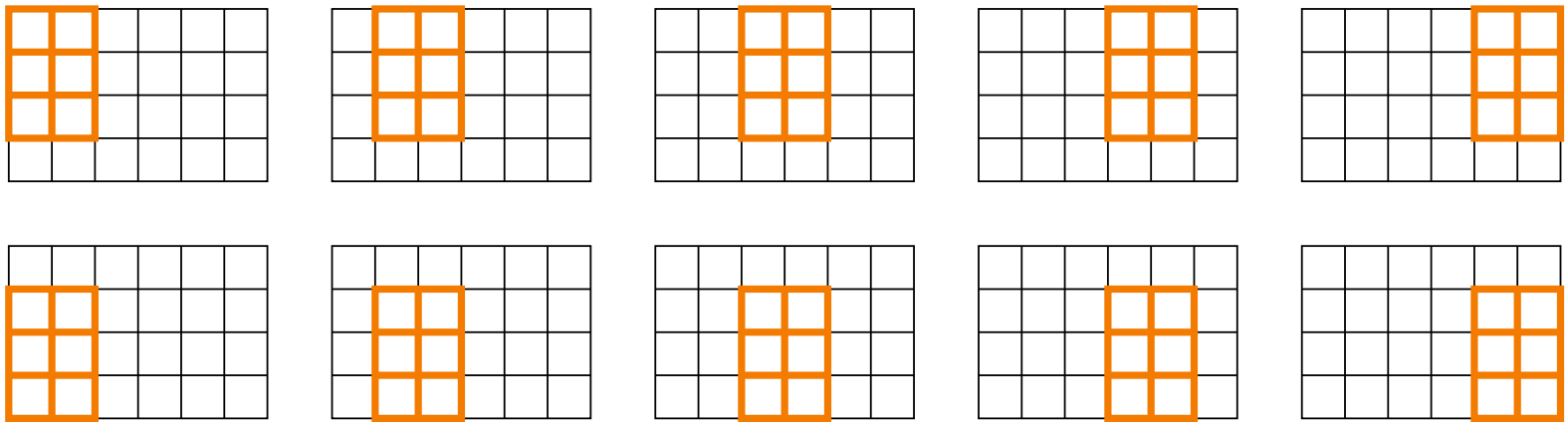
2D Correlation

- Generalize in a straightforward way for 2D images:

$$z_{ij} = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} g_{ab}\, x_{i+a,j+b}$$

for $i = 0, \ldots, e_1 - 1 = 0, \ldots, d_1 - k_1$

and $j = 0, \ldots, e_2 - 1 = 0, \ldots, d_2 - k_2$



Stride

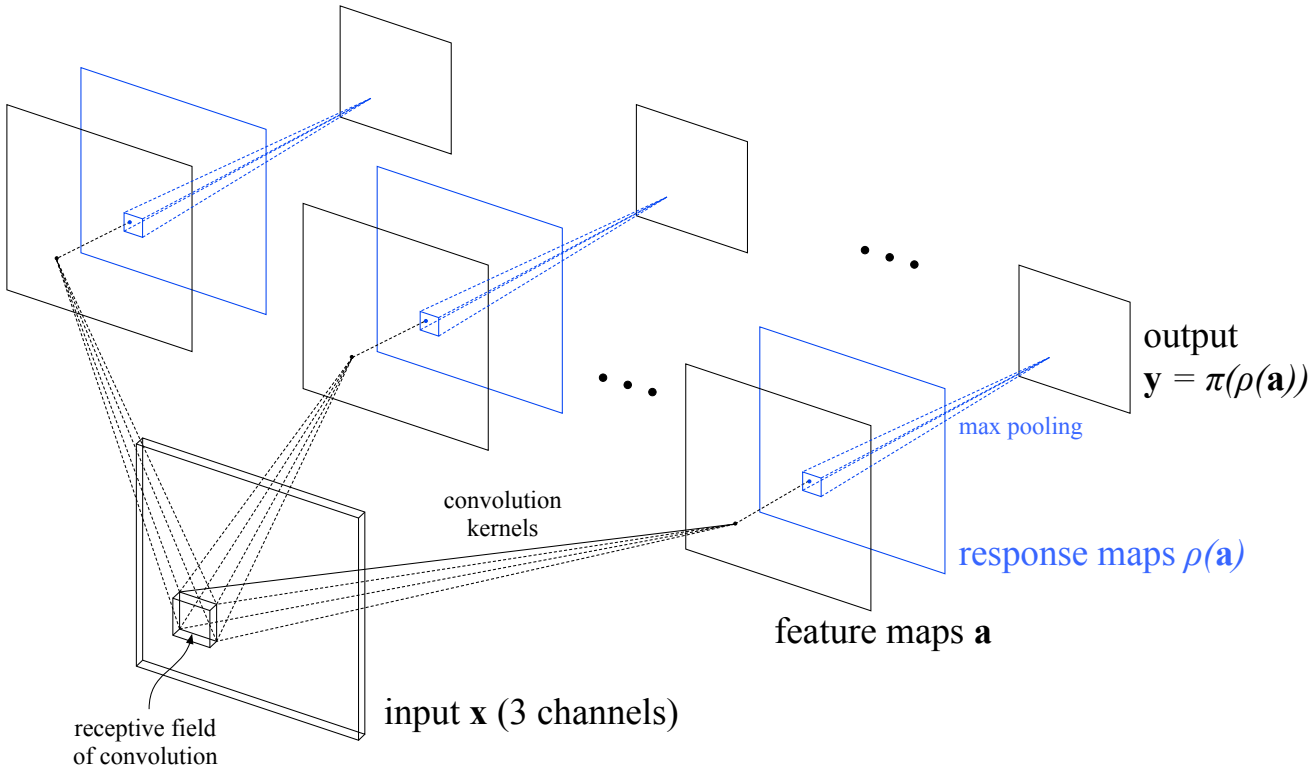
- Images often vary slowly over space
- Output z_{ij} is often similar to $z_{i,j+1}$ and $z_{i+1,j}$
- Reduce the redundancy in the output by computing correlations with a *stride* s_m greater than one
- Only compute every s_m output values in dimension $m \in \{1, 2\}$
- Output size shrinks from $d_1 \times d_2$ to about $d_1/s_1 \times d_2/s_2$

Max Pooling

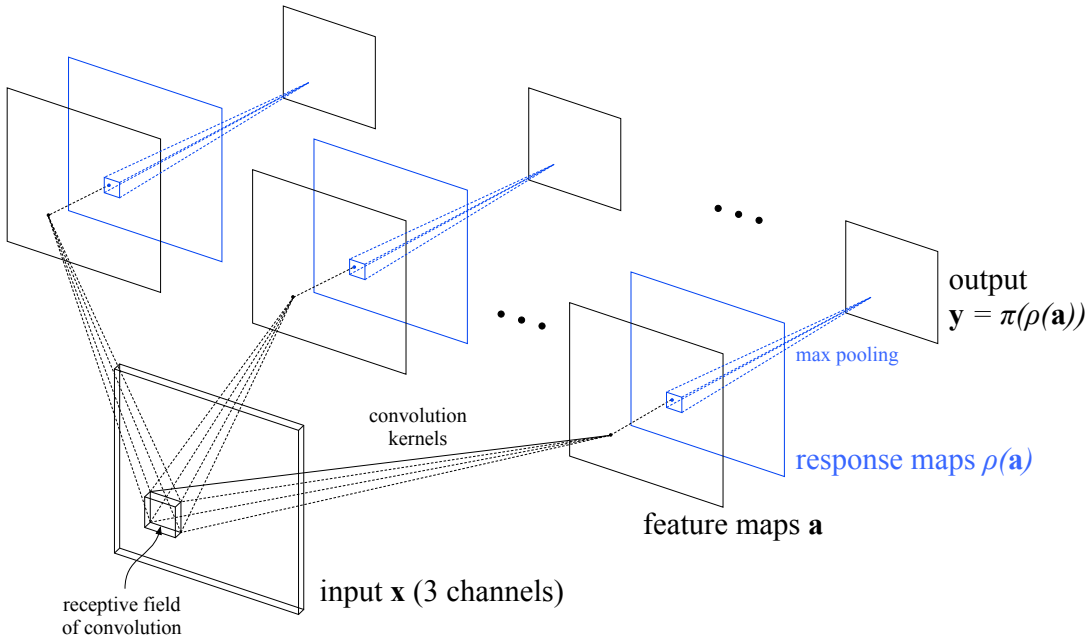
- Another way to reduce output resolution is *max pooling*
- This is a layer of its own, separate from correlation
- Consider $k \times k$ windows with stride s
- Often $s = k$ (adjacent, non-overlapping windows)
- For each window, output the maximum value
- Output is about $d_1/s \times d_2/s$
- Returns highest response in window, rather than the response in a fixed position

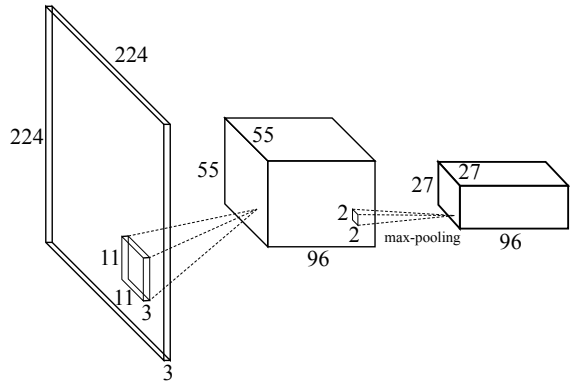
The Input Layer of AlexNet

- AlexNet *circa* 2012, classifies color images into one of 1000 categories
- Trained on ImageNet, a large database with millions of labeled images

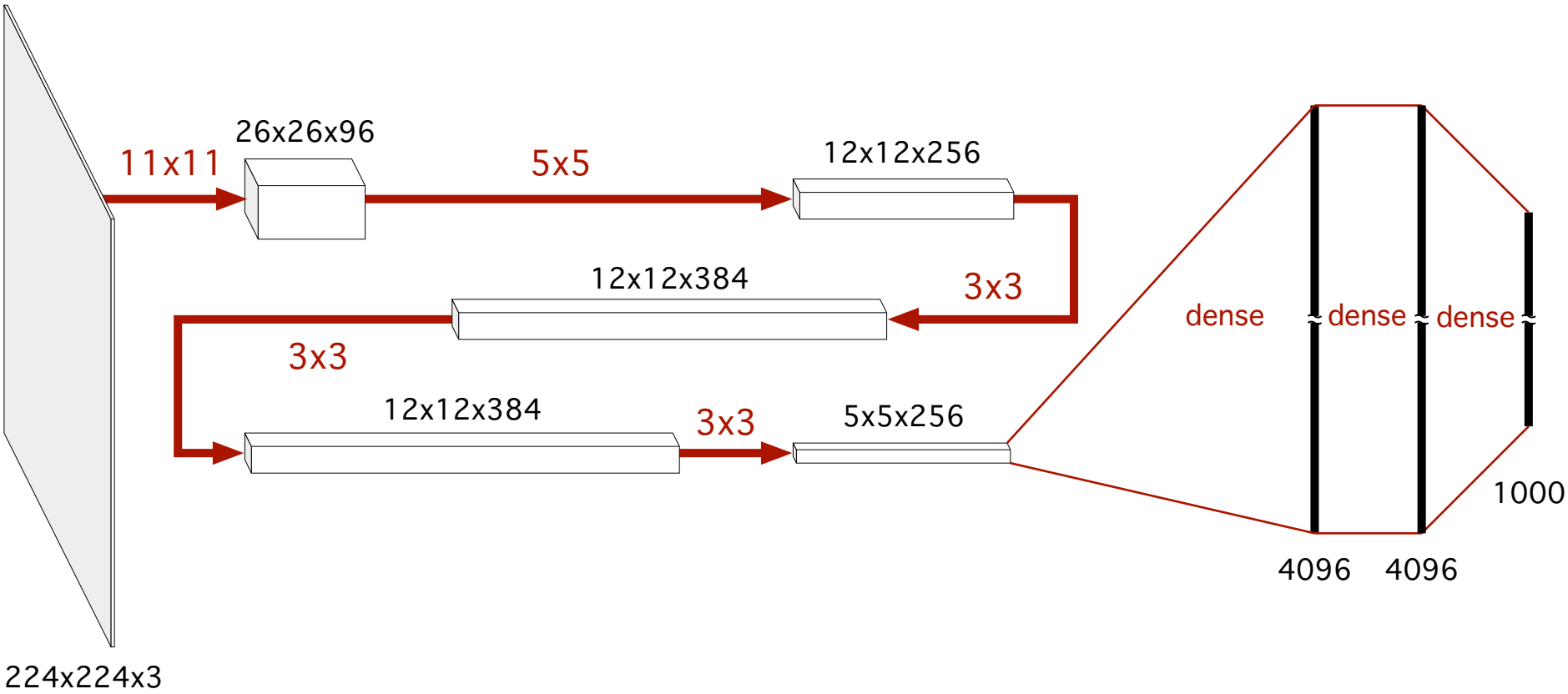


A more Compact Drawing





AlexNet


11x11
26x26x96
5x5
12x12x256
3x3
12x12x384
3x3
12x12x384
3x3
5x5x256
dense
dense
dense
1000
4096
4096
224x224x3

Output

- The last layer of a neural net used for classification is a soft-max layer

  $\mathbf{p} = \sigma(\mathbf{y}) = \frac{\exp(\mathbf{y})}{\mathbf{1}^T \exp(\mathbf{y})}$
- The function from $\mathbf{x}$ to $\mathbf{p}$ is (nearly) differentiable
- Use cross-entropy loss on $\mathbf{p}$ to train
- After training, replace loss function with $\arg\max \mathbf{p}$
- Or even with $\arg\max \mathbf{y}$ since the soft-max is monotonic

AlexNet Numbers

- Input is $224 \times 224 \times 3$ (color image)
- First layer has 96 feature maps of size 26×26
- A fully-connected layer would have about $224 \times 224 \times 3 \times 26 \times 26 \times 96 \approx 10^{10}$ gains
- With convolutional kernels of size 11×11, there are only $96 \times 11^2 = 10^4$ gains
- That's a big deal! Locality and reuse
- Most of the complexity is in the last few, fully-connected layers, which still have millions of parameters
- More recent neural nets have much lighter final layers, but many more layers