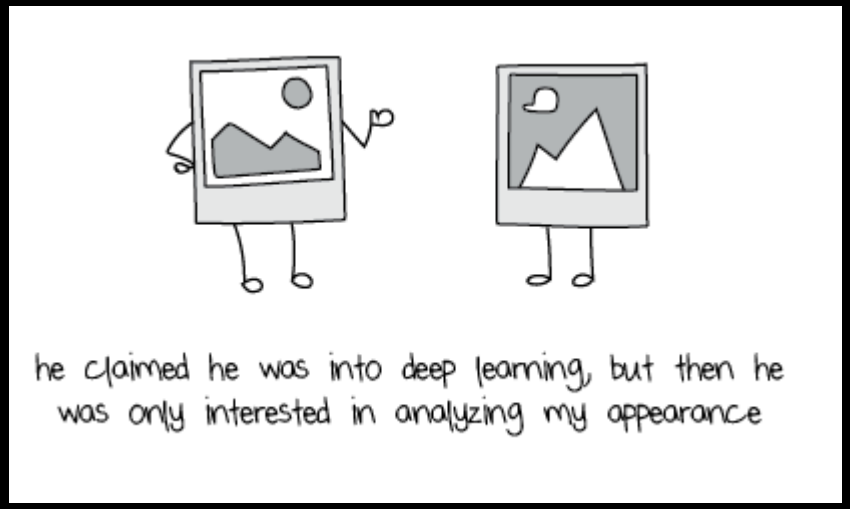
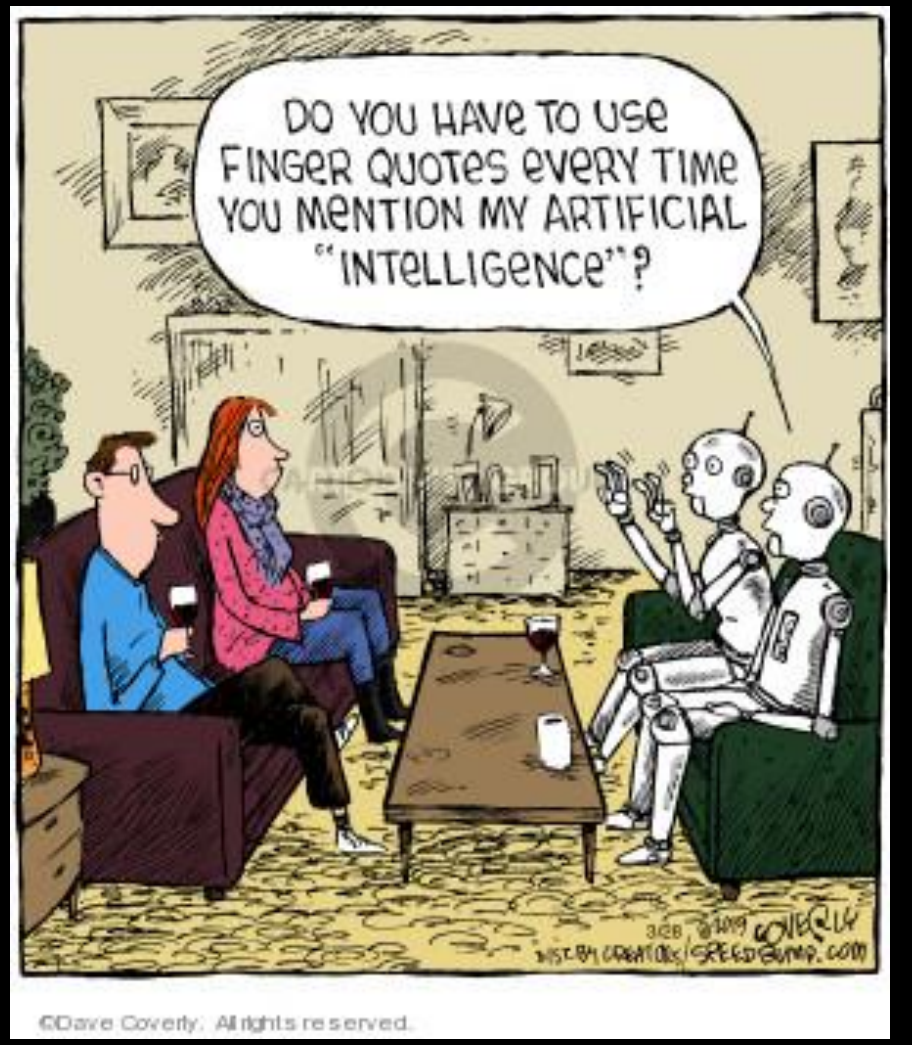




INFO 251: Applied Machine Learning

Deep Learning

Announcements

- PS5 posted, due April 3

Course Outline

- Causal Inference and Research Design
  - Experimental methods
  - Non-experiment methods
- **Machine Learning**
  - Design of Machine Learning Experiments
  - Linear Models and Gradient Descent
  - Non-linear models
  - Fairness and Bias in ML
  - Neural models
  - **Deep Learning**
  - Practicalities
  - Unsupervised Learning
- Special topics

Key Concepts (last lecture)

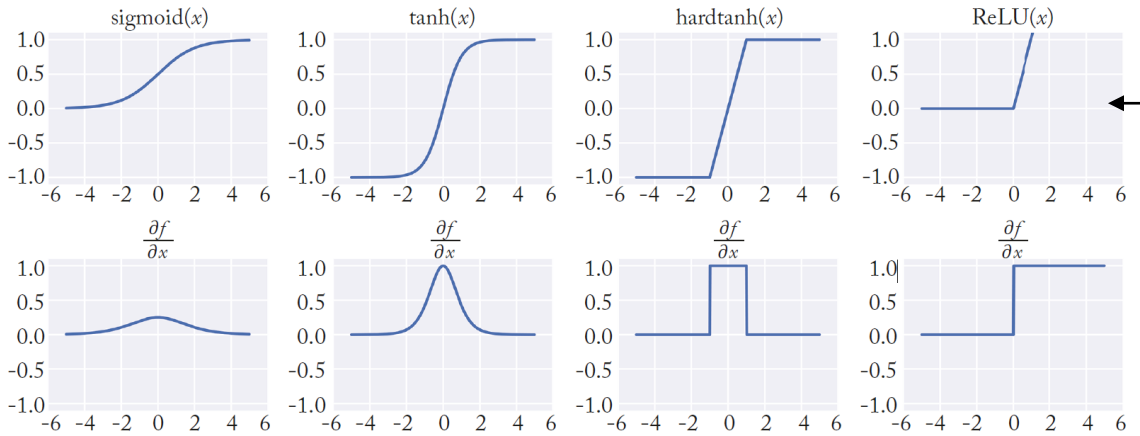
- Multilayer networks
- Activation and non-convexity
- Why GD doesn't work on MLP's
- Backpropagation
- Practical considerations
- Key NN hyperparameters
- Tuning networks

Today's outline

- Neural networks: Loose Ends
- Deep Learning
- Building blocks
- Autoencoders
- Convolutions
- Pooling
- Convolutional Neural Networks
- Modern CNNs

Neural networks: Loose Ends

- About that activation function…
  - We've mainly focused on the sigmoid function (in context of logistic regression)
    - However, sigmoid has issue of "vanishing gradient" (product of lots of small numbers --> zero)
  - In practice, there are a few options for neural networks:



ReLU = "rectified linear unit"
ReLU(x) = max(0,x)

Figure 4.3: Activation functions (top) and their derivatives (bottom).

  - Modern systems: ReLU and tanh in intermediate layers; sigmoid on final layer

Neural networks: Generalization

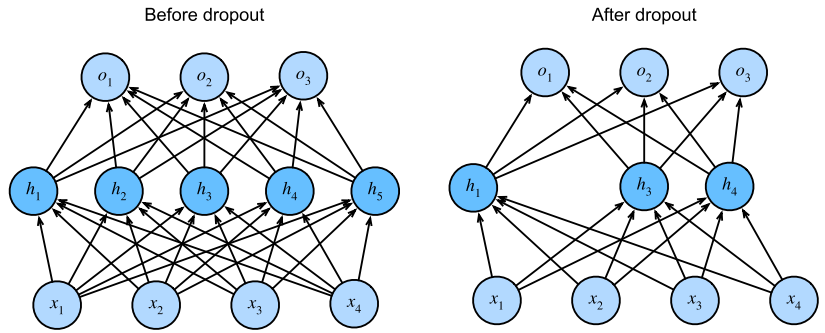
- Network architecture is complex!
  - How many layers? Units per layer? How to initialize? What learning rate? When to stop training? How to regularize?
- Regularization and overfitting
  - Often, many different architectures can perfectly fit training data
  - With ANN's, reducing model complexity doesn't always help with overfitting; the relationship can be non-monotonic (e.g., Nakkiran et al. 2021)

Neural networks: Generalization

- Two popular techniques for improving generalization
  1. "Early stopping" of training
     - Set "patience criterion" – minimum reduction in error that must occur with each epoch/iteration
     - Helps with generalization, also saves you time!
     - But the criterion is difficult to determine ex ante

Neural networks: Generalization

- Two popular techniques for improving generalization
  2. "Dropout"
     - Idea: Adding small amount of noise to inputs shouldn't dramatically change output
     - Bishop (1994): Input noise is equivalent to regularization
     - Dropout (Srivastava et al. 2014):
       - A fraction *p* of nodes in each layer are set to zero
       - Other nodes are debiased (i.e., output is scaled up by *p*)
       - Dropout is used during training
     - Like a form of ensemble learning
       - In each training iteration, a different subnetwork is trained
       - At test time, these subnetworks are "merged" by averaging their weights

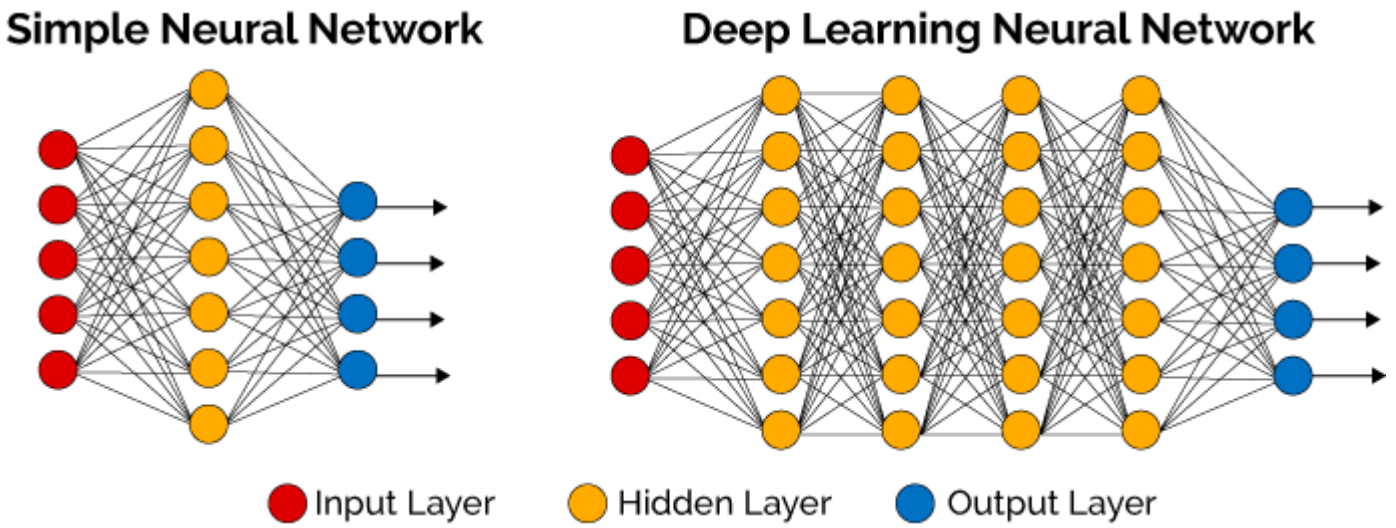


Today's outline

- Neural networks: Remarks
- **Deep Learning**
- Building blocks
- Autoencoders
- Convolutions
- Pooling
- Convolutional Neural Networks
- Modern CNNs

Deep Learning

- What is "deep" learning?



Deep Learning: Overview

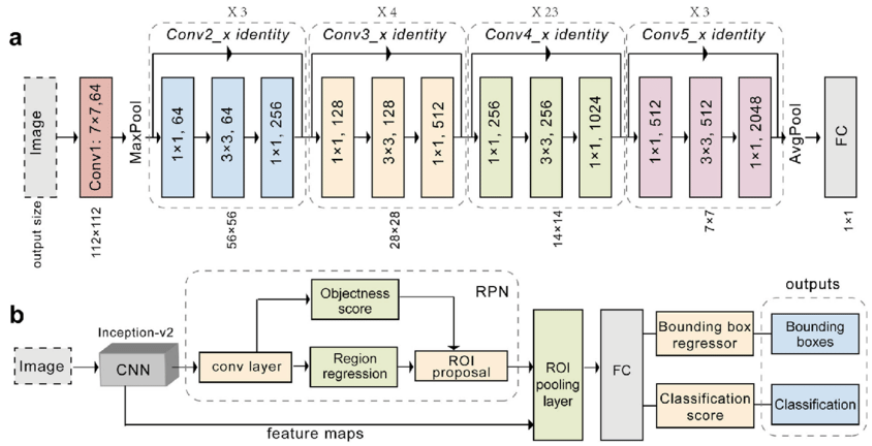
- 10,000-foot view of deep learning
  - Both theory and practice of DL is evolving fast
    - Often, theory is playing catch-up with technical innovations
    - Current theory is not comprehensive; many practical techniques are not fully understood
    - Example: Explicit regularization ("weight decay", very similar to ridge) is very commonly used, even though theoretical results show it doesn't actually prevent the network from overfitting (e.g., Zhang et al. 2021)
  - Classical theory cannot typically tell us how to improve model generalization

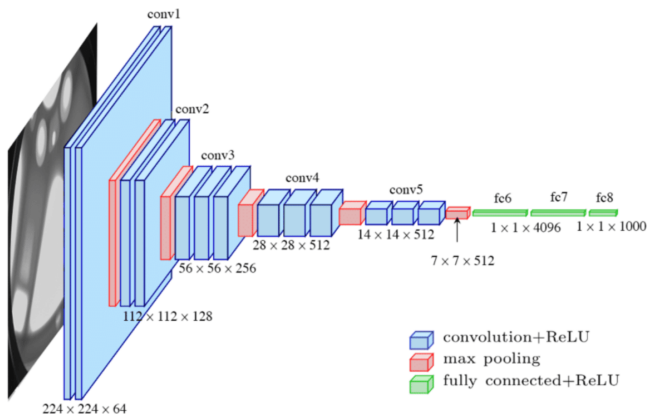
Today's outline

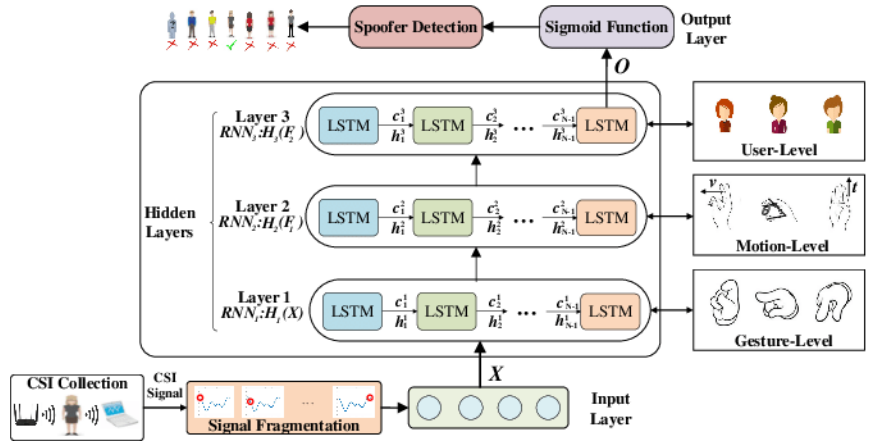
- Neural networks: Remarks
- Deep Learning
- **Building blocks**
- Autoencoders
- Convolutions
- Pooling
- Convolutional Neural Networks
- Modern CNNs

Deep Learning: Building blocks

- So, how do we actually build deep neural nets?
  - The most basic unit of abstraction is a neuron
    - It takes in an input, applies a function, and generates an output
  - But it is very tedious to build big and complex networks from scratch
    - Many networks have hundreds of very complex layers
  - This motivates higher levels of abstraction, i.e., modules





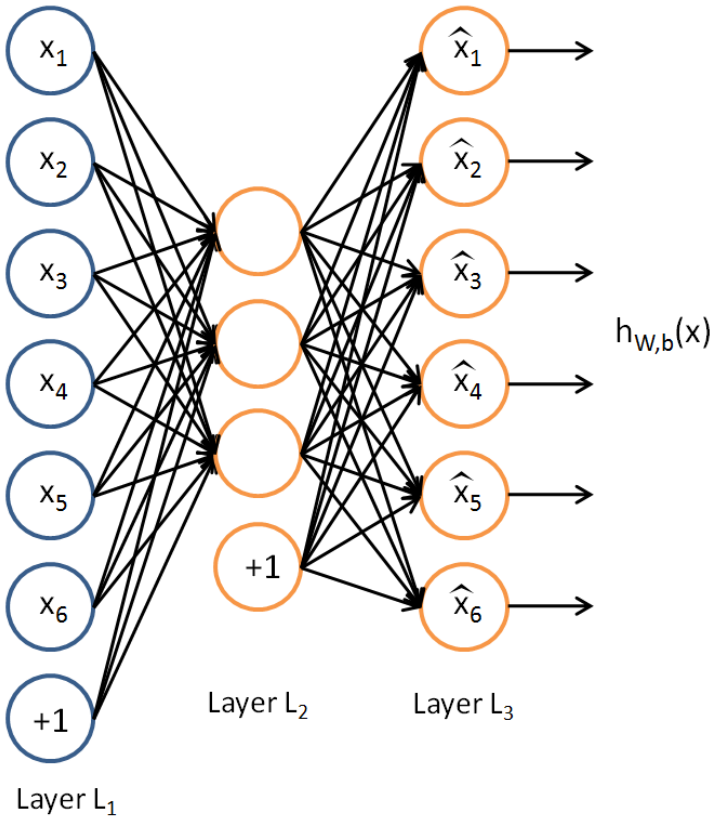


Deep Learning: Building blocks

- We'll review a few key modules
  - Autoencoders, convolutions, pooling, LSTM, etc…
- But you can also create your own modules! This requires:
  - Take input data, as arguments to forward propagation
  - Generate output (this can change dimensionality of data)
  - Calculate gradient of output with respect to input (can be done automagically)
  - Store and expose parameters to allow for optimization
- It's not that hard!
  - See DDL chapter 6 for examples in PyTorch, TensorFlow, MXNet, JAX

Example NN building block: Autoencoders

- **Auto-encoders:** map inputs to inputs (i.e., it approximates the identity function)
  - $\hat{y}_i(x_i) = x_i$
- Why bother?
  - Data compression
  - Data abstraction
- Example
  - 96 x 96 pixel image: 9216 features
  - This NN compresses this to *h* features, where *h* is the number of hidden nodes in L_2

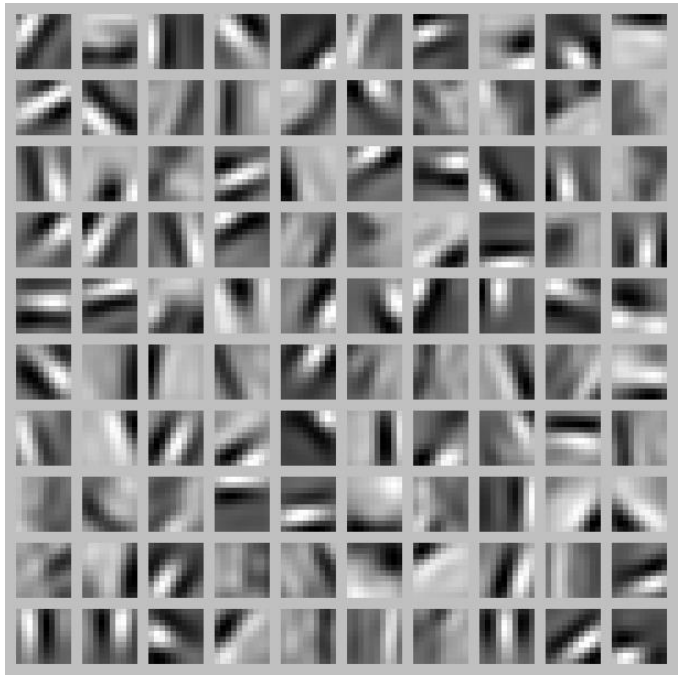


Sparse Autoencoders

- "Sparsity"
  - Even if number of hidden units h is large, can impose sparsity constraints on those hidden units
  - Achieve sparsity by
    - Random dropout
    - Choosing the *k* highest activations and setting the rest to zero (see Makhzani et al., 2013)
    - Constraining avg. activation of each hidden neuron to be small (e.g., <0.05)
      - see Ng's lecture notes on bCourses for details of this approach

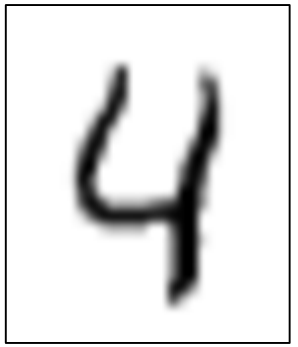
Sparse Autoencoders

- Take-away
  - Often, autoencoder features are not competitive with hand-crafted features
  - But, features themselves are often useful



Sparse Autoencoders

- *k*-Sparse Autoencoders Example
  - MNIST data (28x28 images)
  - Hidden layer retains only the *k* largest elements

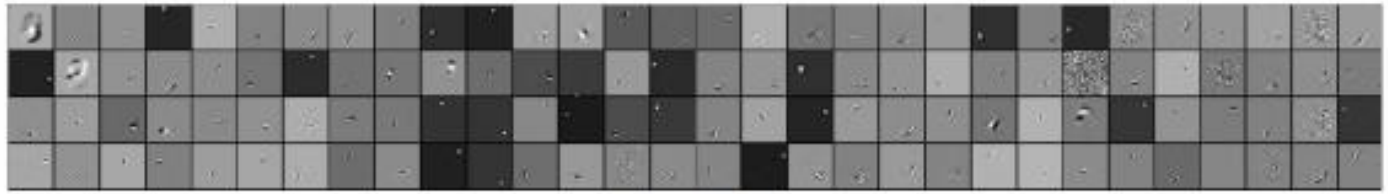


---

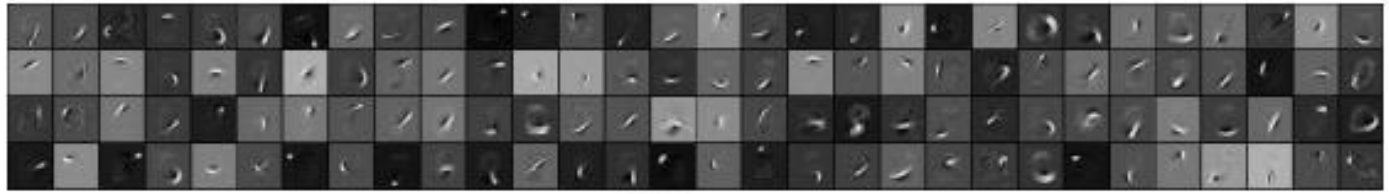
k-Sparse Autoencoders

Alireza Makhzani MAKHZANI@PSI.UTORONTO.CA
Brendan Frey FREY@PSI.UTORONTO.CA
University of Toronto, 10 King's College Rd. Toronto, Ontario M5S 3G4, Canada

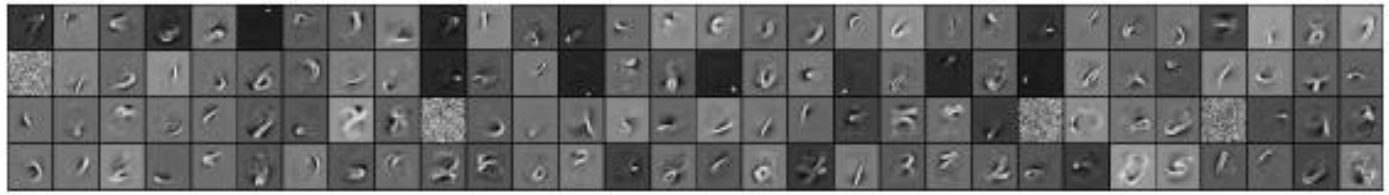
Sparse Autoencoders



(a) $k = 70$



(b) $k = 40$



(c) $k = 25$

(d) $k = 10$

Figure 1. Filters of the k-sparse autoencoder for different sparsity levels k, learnt from MNIST with 1000 hidden units.

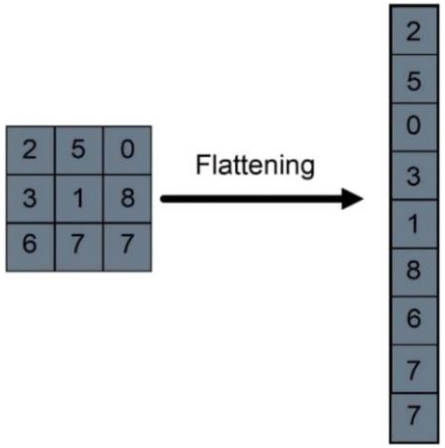
Today's outline

- Neural networks: Remarks
- Deep Learning
- Building blocks
- Autoencoders
- **Convolutional Neural Networks: Big picture**
- Convolutions
- Pooling
- Convolutional Neural Networks
- Modern CNNs

Convolutional Neural Nets: Motivation

- Fully connected networks have lots of parameters
  - 96x96 pixels => 9,216 pixels
  - "Flattened" feature vector has length 9,216
  - Example: If internal layer has 500 nodes
    - FC network has 4.6 Million weights/parameters!
  - Real world systems: would require billions of parameters

- Structure of input data is not preserved
  - Pixels far from each other are treated the same as nearby pixels
  - Random permutation of order of pixels (for all images) would have no effect

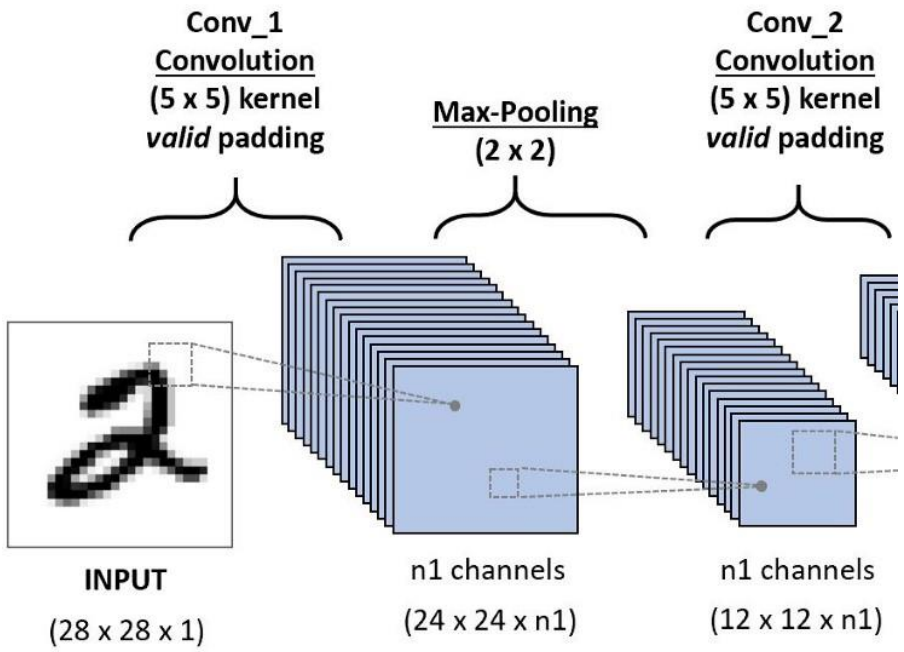


Convolutional Neural Nets: Motivation

- Convolutional neural nets (aka CNNs; aka ConvNets)
  - Idea: take advantage of local structure in input data
  - Why? Lots of data has local structure (images, video, audio)
  - In practice
    - Can produce very accurate classifiers/predictors
    - Even on sequential data without 2D structure (e.g., time series, text)
  - Computational advantages
    - Fewer parameters than fully connected networks
    - Can be easily parallelized

Convolutional Neural Nets: Intuition

- Convolutional neural nets (aka CNNs; aka ConvNets)
  - "Locally connected", "locality principle"
    - Early layers focus on local regions
    - (Each hidden unit connects to a subset of input units)
    - This is sort of how the visual cortex works
    - For images, these are often contiguous
    - For other data, depends on how "contiguous" is defined
  - "Translation invariance"
    - In early layers, shouldn't matter *where* that local patch is
    - E.g., a bird in the top corner == a bird in the bottom corner



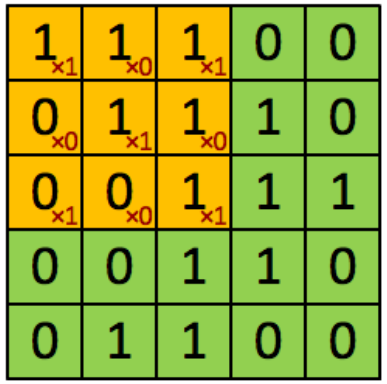
Today's outline

- Neural networks: Remarks
- Deep Learning
- Building blocks
- Autoencoders
- Convolutional Neural Networks: Big picture
- **Convolutions**
- Pooling
- Convolutional Neural Networks
- Modern CNNs

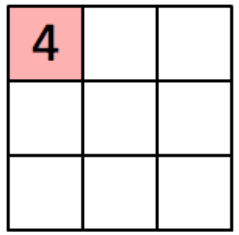
Convolutions

http://ufldl.stanford.edu/tutorial/

- "Convolutions" apply filter/kernel to input (element-wise multiplication)
  - Filter: common set of "shared" weights applied to all subregions
    - Element-wise multiplication (not matrix multiplication); often referred to as "dot product"
  - E.g., instead of 96 x 96 weights connecting to each internal node, 3 x 3 weights/node
    - Number of parameters/node: 9,216 => 9
  - Much more efficient than fully connected network!



Image



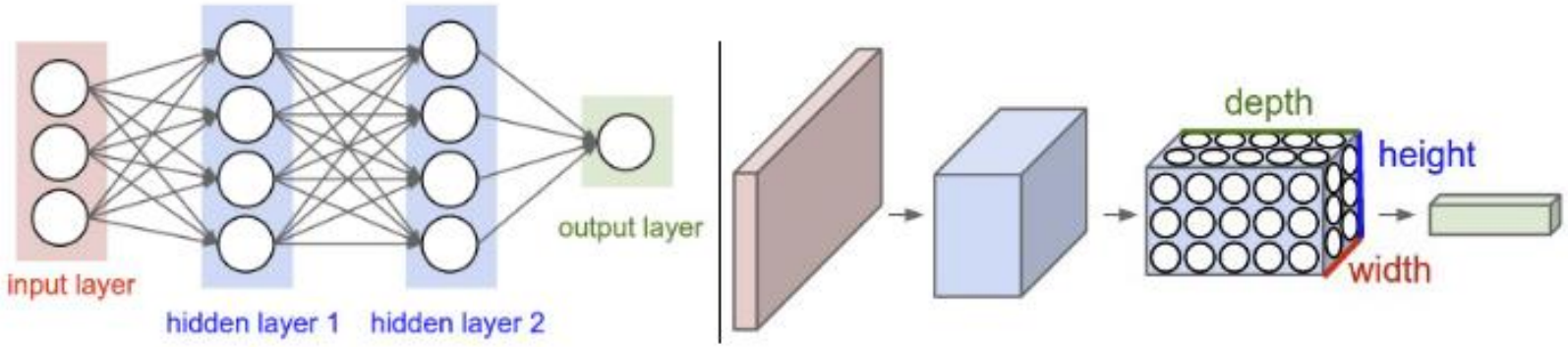
Convolved Feature

← Animation shows a single convolutional filter:

```
F = [1  0  1
     0  1  0
     1  0  1]
```

Convolutions and Features

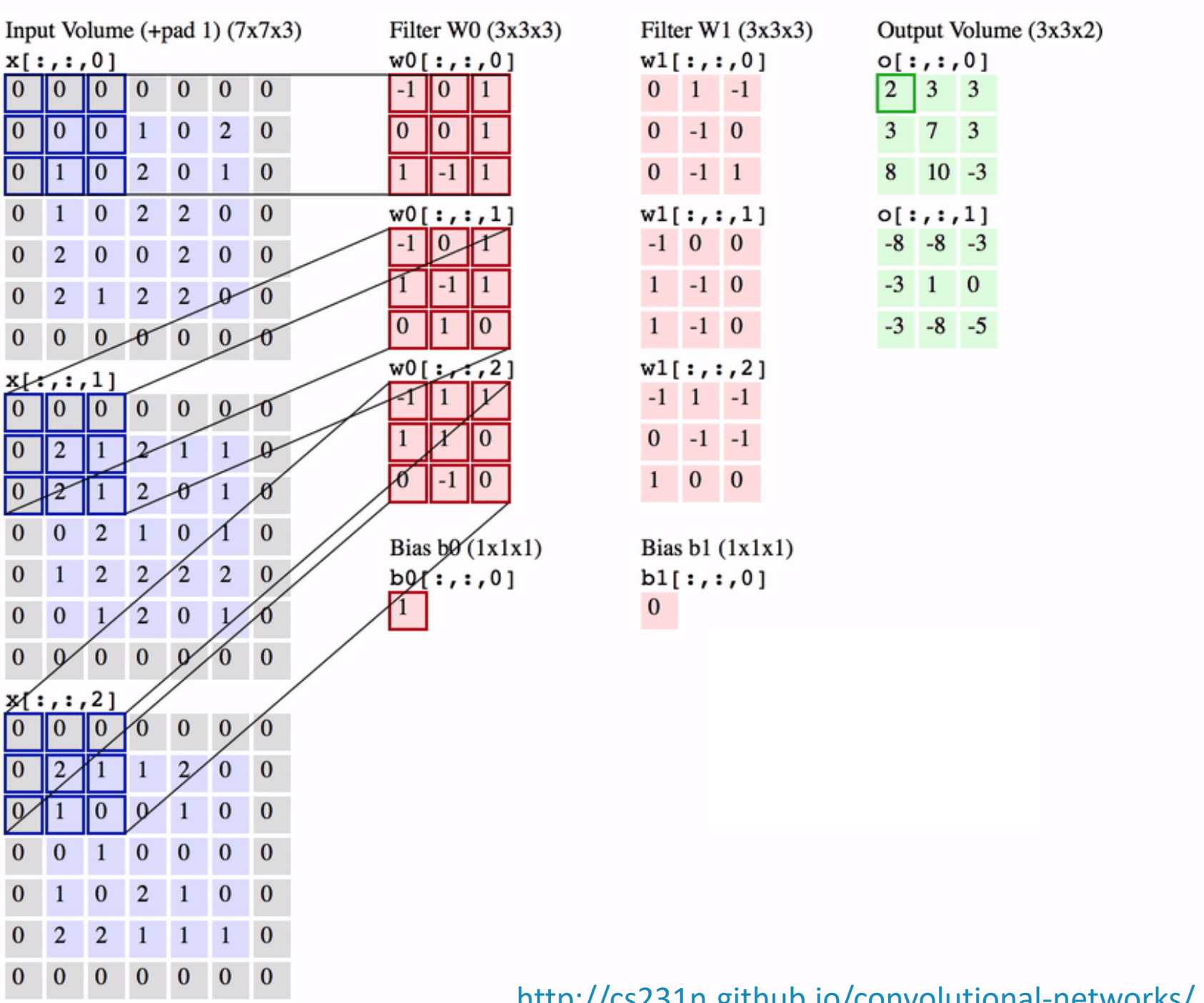
- Each layer of a ConvNet transforms an input 3D volume into an output 3D volume
  - "3D" because of depth, e.g., RGB has depth 3
  - Hidden layer might have depth = number of filters



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Example

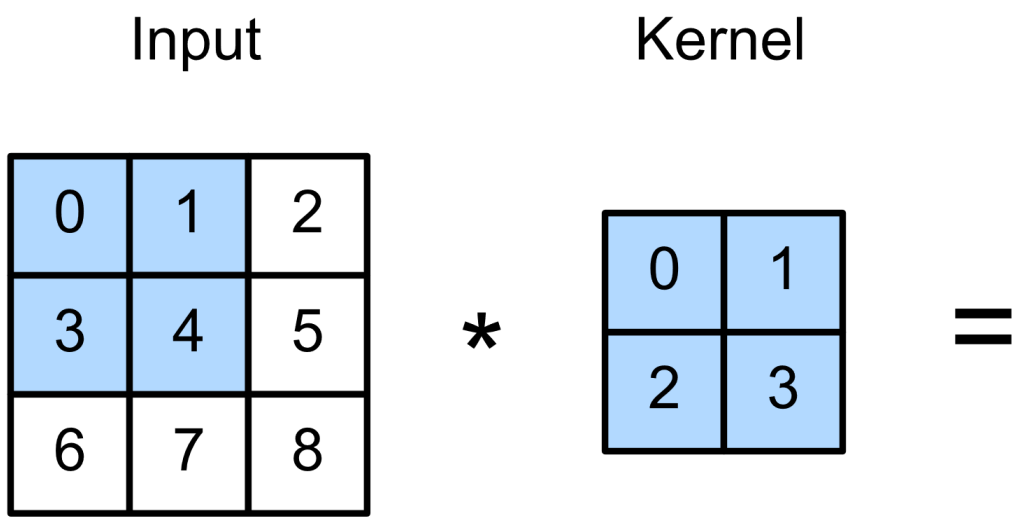
- Input is 5 x 5 x 3
- Output is 3 x 3 x 2
- 2 filters (w0 and w1)
- Stride = 2
- Input padding = 1



http://cs231n.github.io/convolutional-networks/

Intuition check

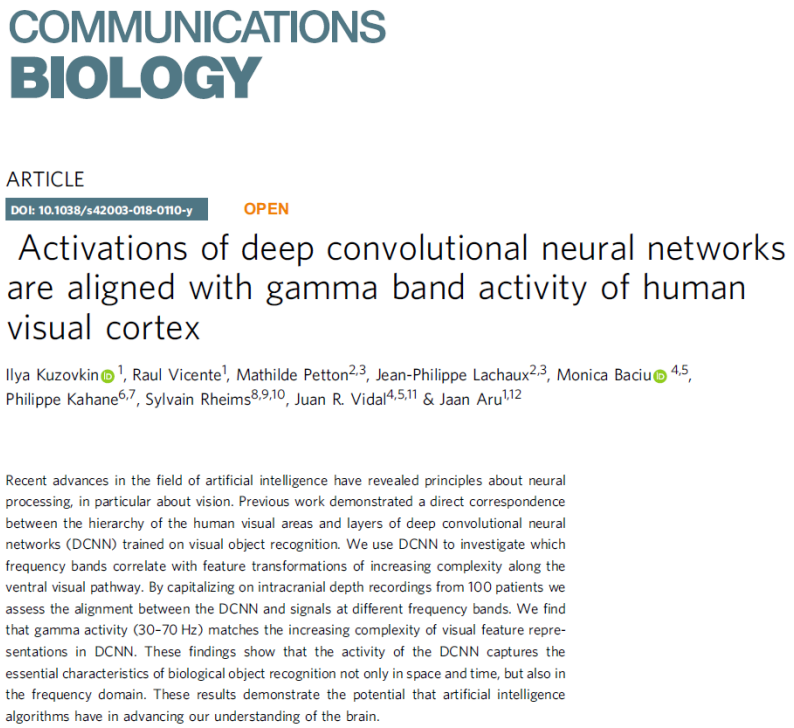
- What would be the output if we apply the following 2x2 filter (kernel) to the 3x3 input volume (stride 1)



Convolutions

- What do the convolution kernels represent?
  - "Feature maps": learned representation of inputs to next layer
  - "Receptive field": area that influences output
  - Strong analogy to visual cortex

> "These findings show that the activity of the DCNN captures the essential characteristics of biological object recognition not only in space and time, but also in the frequency domain. These results demonstrate the potential that artificial intelligence algorithms have in advancing our understanding of the brain."



COMMUNICATIONS BIOLOGY

ARTICLE

DOI: 10.1038/s42003-018-0110-y OPEN

Activations of deep convolutional neural networks are aligned with gamma band activity of human visual cortex

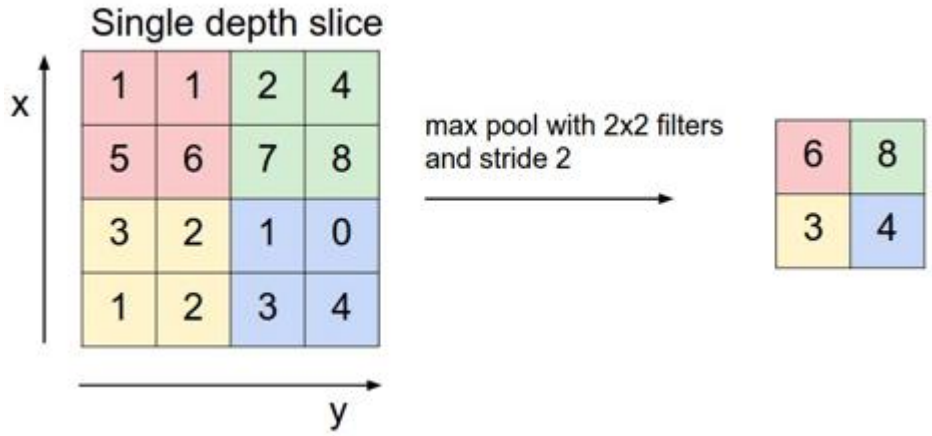
Ilya Kuzovkin[1], Raul Vicente[1], Mathilde Petton[2,3], Jean-Philippe Lachaux[2,3], Monica Baciu[4,5], Philippe Kahane[6,7], Sylvain Rheims[8,9,10], Juan R. Vidal[4,5,11] & Jaan Aru[1,12]

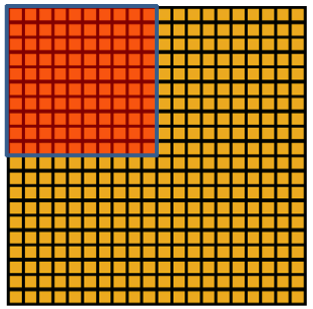
Recent advances in the field of artificial intelligence have revealed principles about neural processing, in particular about vision. Previous work demonstrated a direct correspondence between the hierarchy of the human visual areas and layers of deep convolutional neural networks (DCNN) trained on visual object recognition. We use DCNN to investigate which frequency bands correlate with feature transformations of increasing complexity along the ventral visual pathway. By capitalizing on intracranial depth recordings from 100 patients we assess the alignment between the DCNN and signals at different frequency bands. We find that gamma activity (30–70 Hz) matches the increasing complexity of visual feature representations in DCNN. These findings show that the activity of the DCNN captures the essential characteristics of biological object recognition not only in space and time, but also in the frequency domain. These results demonstrate the potential that artificial intelligence algorithms have in advancing our understanding of the brain.

- In practice:
  - Detect edges, lines, blur/sharpen images
  - These filters are *learned from data*

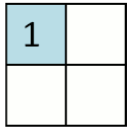
"Pooling"

- Even after convolutions, can have lots of features
  - (but a much smaller set of shared weights!)
- "Pooling" aggregates regions of a convolved layer
  - Typically: mean or max feature activation in a region
  - Pooling progressively shrinks dimensionality of network
  - Often used to reduce overfitting





Convolved feature



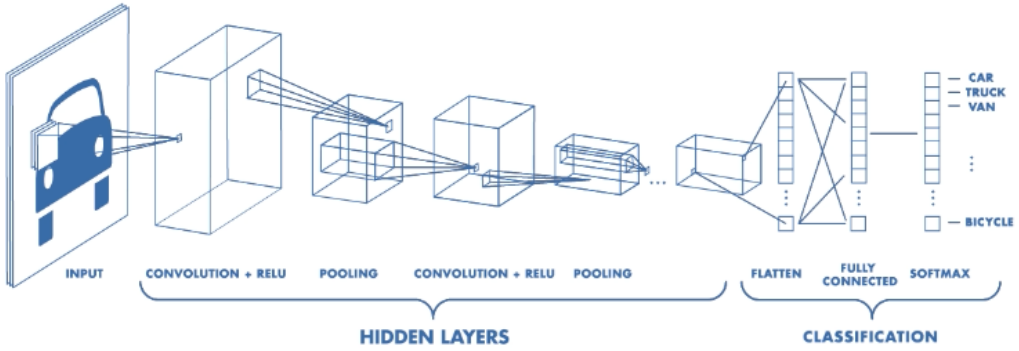
Pooled feature

Today's outline

- Neural networks: Remarks
- Deep Learning
- Building blocks
- Autoencoders
- Convolutions
- Pooling
- **Convolutional Neural Networks**
- Modern CNNs

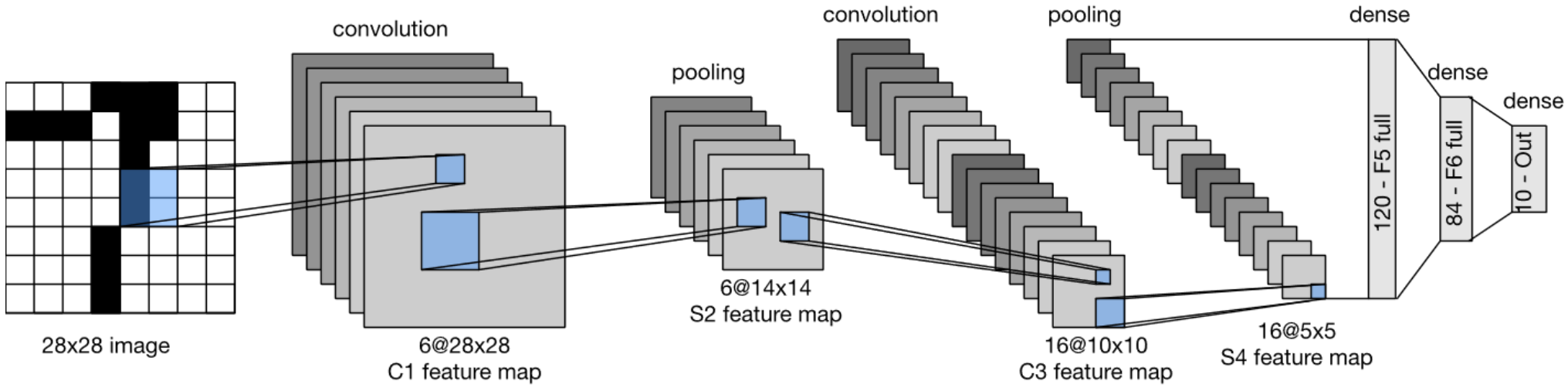
Putting it all together into a CNN

- ConvNets typically includes many different types of layer
  - Input
    - e.g, for a 32-pixel image with RGB channels: 32 x 32 x 3
  - Convolutions
    - e.g., after applying 12 filters: 8 x 8 x 12
  - ReLU (Rectified Linear Units) activation
    - Removes negatives, introduces nonlinearities
    - leaves volume unchanged: 8 x 8 x 12
  - Pooling
    - e.g., 3 x 3 x 12
  - Fully connected layers
    - e.g., final 'softmax' classification layer: 1 x 1 x 10 (e.g., output vector indicating class probabilities)



Example CNN (LeNet)

- LeNet: Yann LeCun's first CNN (1998), for digit classification
  - Used sigmoid activation and average pooling
    - (Modern networks tend to use ReLU + max pooling)
  - First layer: 6 filters, 5x5 kernel, then 2x2 pooling (stride 2)
  - Last conv layer flattens, then has two fully connected layers
  - Final output is 10-unit sigmoid



LeNet in code

PYTORCH MXNET JAX TENSORFLOW

```
def init_cnn(module):  #@save
    """Initialize weights for CNNs."""
    if type(module) == nn.Linear or type(module) == nn.Conv2d:
        nn.init.xavier_uniform_(module.weight)

class LeNet(d2l.Classifier):  #@save
    """The LeNet-5 model."""
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(6, kernel_size=5, padding=2), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.LazyConv2d(16, kernel_size=5), nn.Sigmoid(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Flatten(),
            nn.LazyLinear(120), nn.Sigmoid(),
            nn.LazyLinear(84), nn.Sigmoid(),
            nn.LazyLinear(num_classes))
```

Convolutional Neural Networks

- Distinguishing features of CNN's
  - 3D volumes of neurons: different types of locally or fully connected layers are stacked to form CNN
  - Local connectivity: small regions of one layer connected to next layer. Early layers learn local features; later layers learn larger areas
  - Shared weights: dramatically reduces number of free parameters

Today's outline

- Neural networks: Remarks
- Deep Learning
- Building blocks
- Autoencoders
- Convolutions
- Pooling
- Convolutional Neural Networks
- **Modern CNNs**