ECG782: MULTIDIMENSIONAL DIGITAL SIGNAL PROCESSING

DEEP COMPUTER VISION USING CNNS

Géron Chapter 14

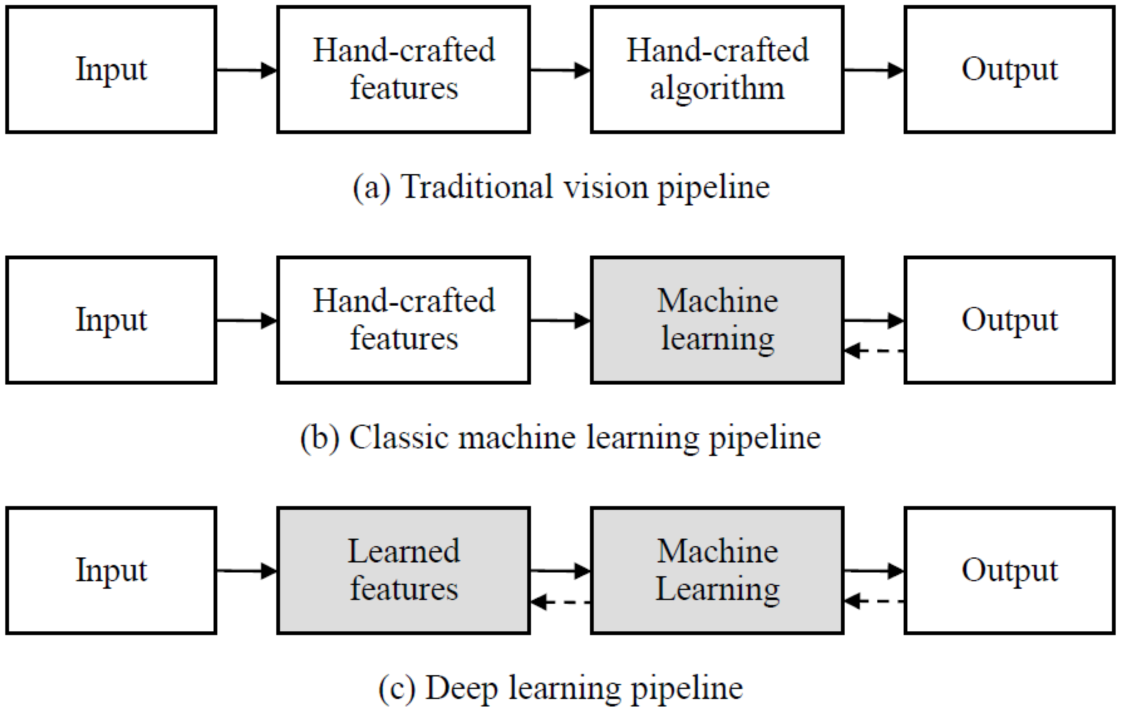
http://www.ee.unlv.edu/~b1morris/ecg782

OUTLINE

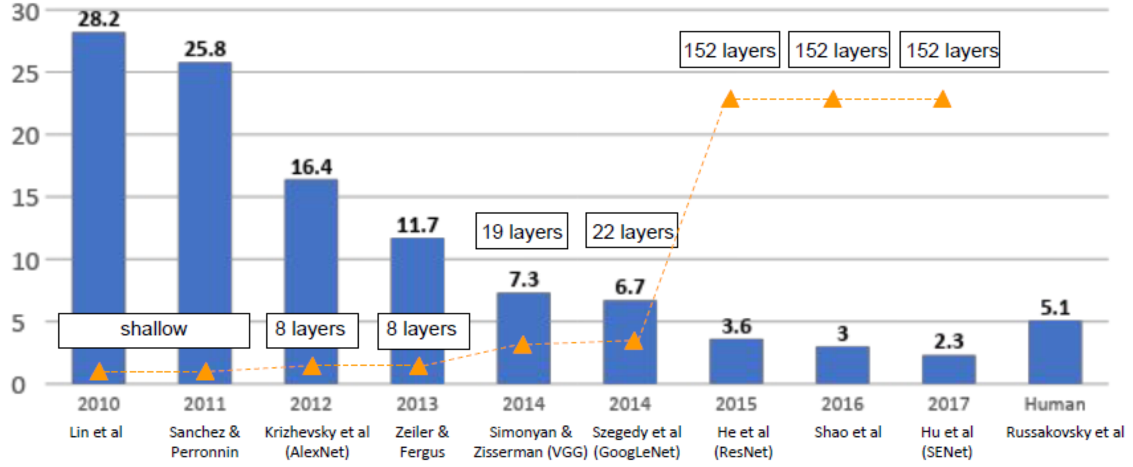
- Biological Inspiration
- Convolutional Layers
- Pooling Layers
- CNN Architectures
- Object Detection
- Semantic Segmentation

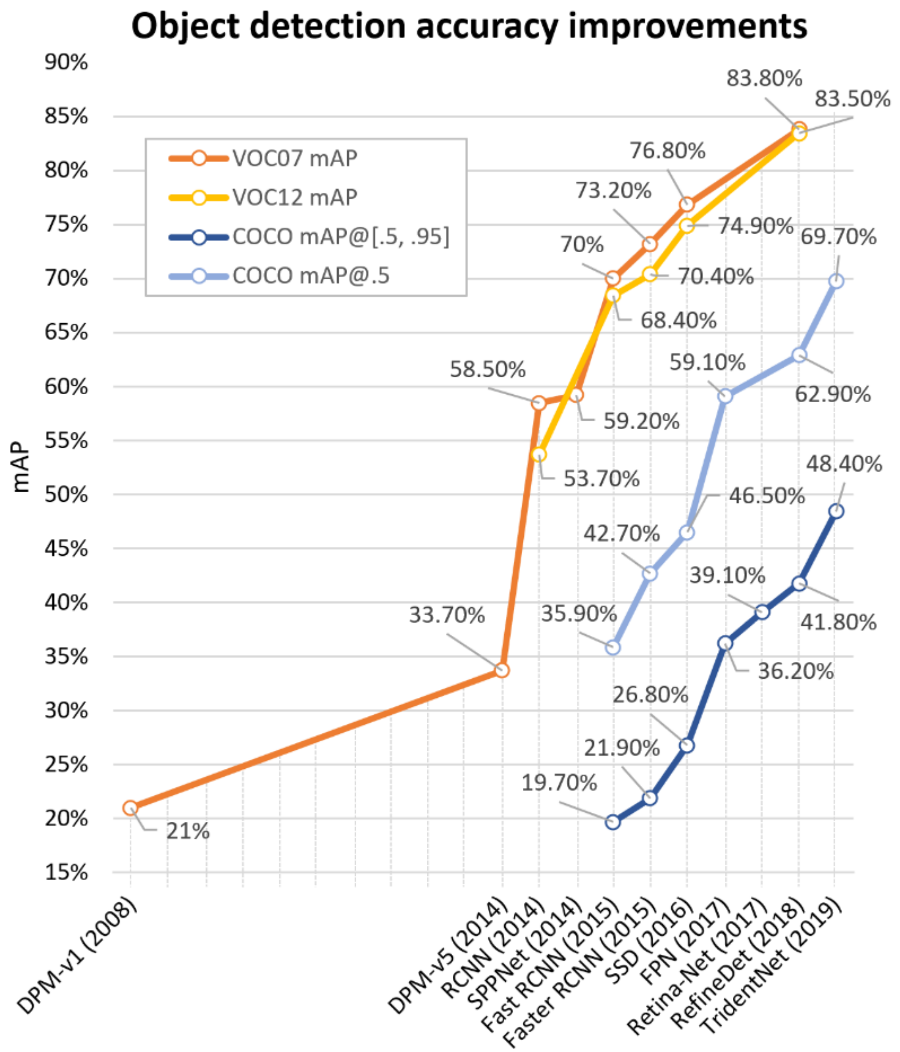
EVOLUTION OF COMPUTER VISION

- Classical vision
  - Hand-crafted features and algorithm based on expert knowledge
- Classical machine learning
  - Hand-crafted features (pre-processing) but ML for classification
- Deep learning
  - Both features and classification are learned
  - End-to-end training (from pixels to output)



DEEP CNN DOMINANCE IN CV



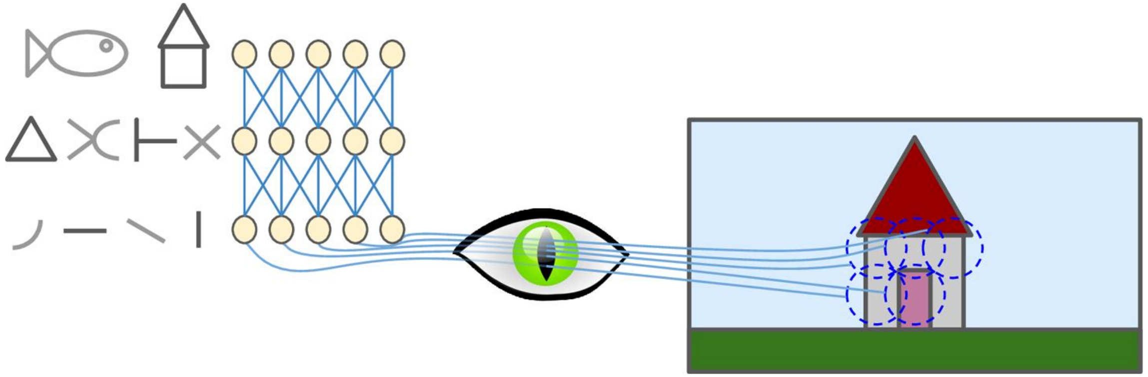


Li, Johnson, and Yeung, 2019

Zou et al., “Object Detection in 20 Years: A survey, 2019

ARCHITECTURE OF THE VISUAL CORTEX

- Modern CV is inspired by human vision (sensory modules)
- Hubel and Wiesel showed that neurons in the visual cortex had a small local receptive field
  - Only reacted to stimuli in a limited region of visual field (blue dashed circles)
- Lower-level neurons with simple pattern response (e.g. lines of specific orientation)
- Higher-level neurons with larger receptive field and combination of lower-level patterns
  - Neurons at higher-levels only connected to few at lower-level

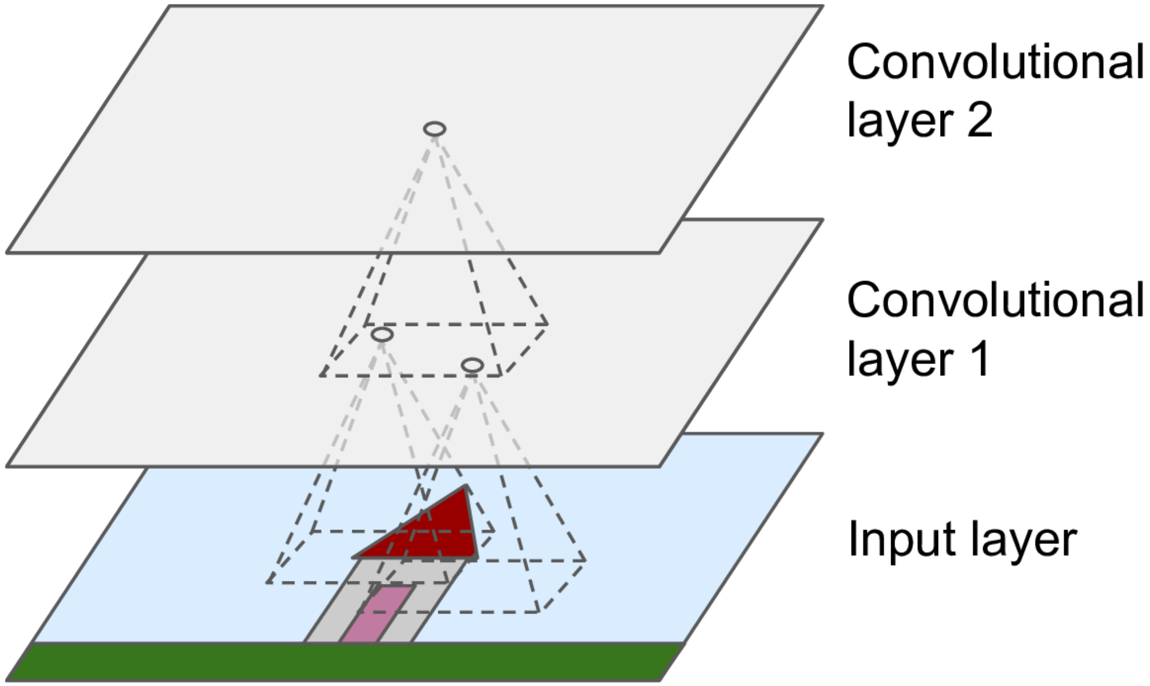


CONVOLUTIONAL NEURAL NETWORK

- Stacked neuron architecture enables detection of complex patterns in any area of the visual field → convolutional neural networks (CNNs)
- Led to LeNet-5 architecture by Yann LeCunn for handwritten number recognition (MNIST)
  - Fully connected layers and sigmoid activations
  - Convolutional layers and pooling layers
- Why not fully connected layers for images?
  - Even small images have large number of pixels resulting in huge networks
  - CNNs solve this with partial connected layers and weight sharing

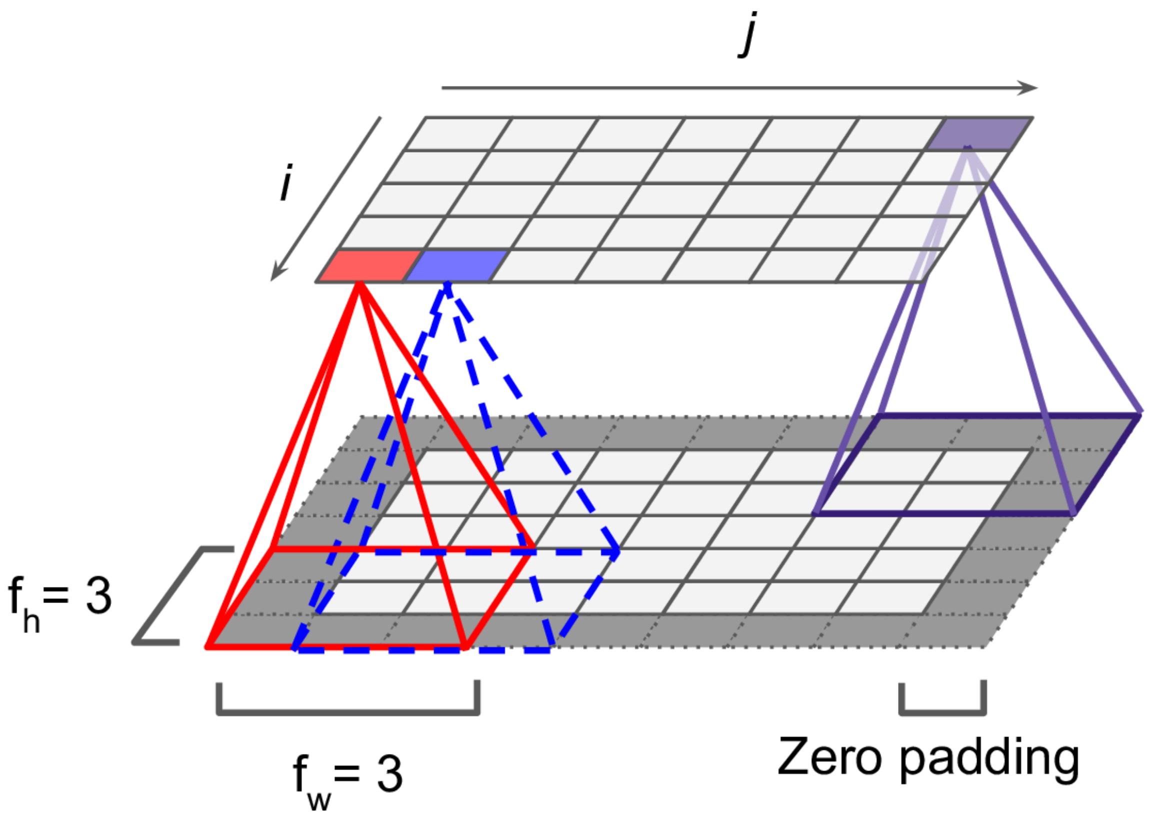
CONVOLUTIONAL LAYERS

- Neurons in the first layer are not connected to every single pixel in input image
  - Connected to receptive field
  - Stacked receptive field approach
- Hierarchical structure
  - First layer – small low-level features
  - Higher-levels – assemble lower-level features into higher-level features
  - Structure is common in real-world images



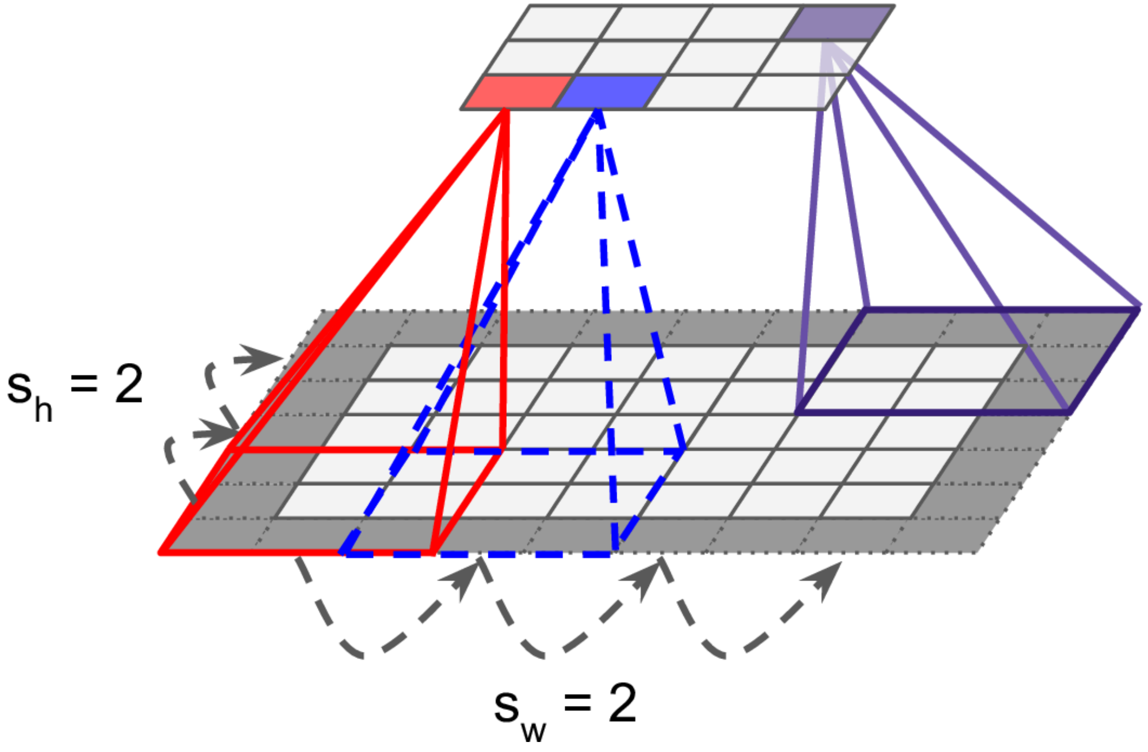
CONVOLUTIONAL LAYER CONNECTIONS

- Note: the actual operation performed is cross-correlation (no-flipping)
- Neuron (row, column) (i, j) is connected to neurons in previous layer within receptive field
  - Row $[i, i + f_h - 1]$
    - f_h - height of receptive field
  - Column $[j, j + f_w - 1]$
    - f_w - width of receptive field
  - Note: this is a causal filter though shown as symmetric
- Zero padding used to keep output/input layers of same size



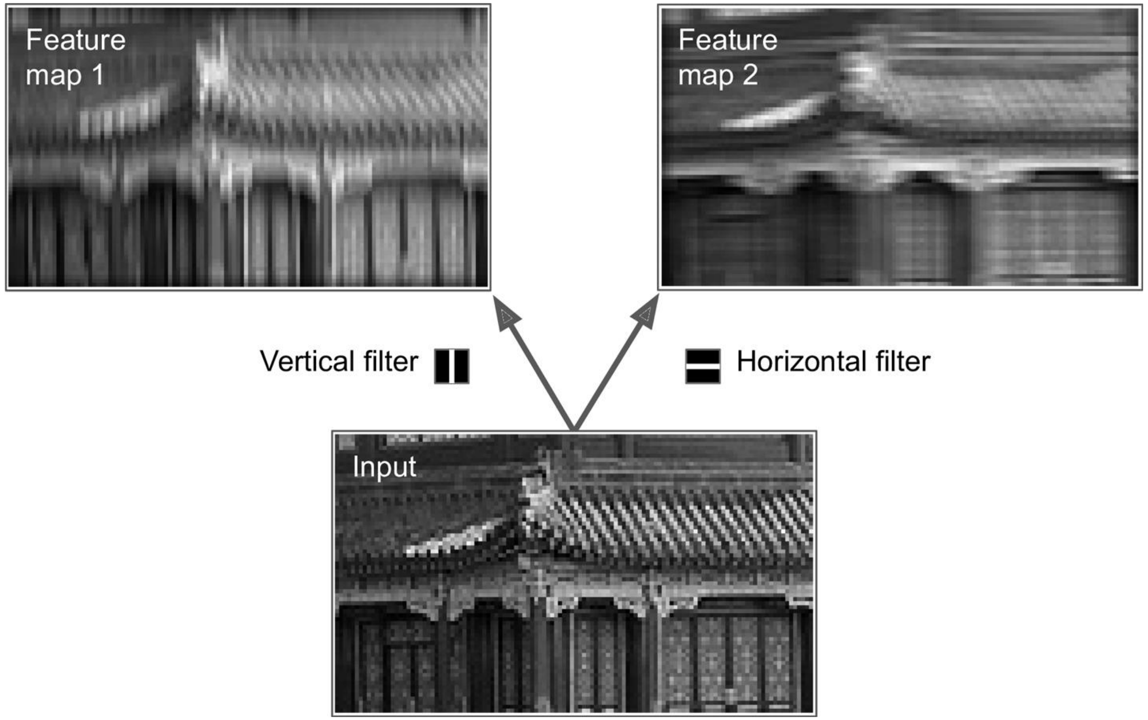
CONVOLUTIONAL LAYERS STRIDE

- Stride can be used to connect a large input layer to smaller output layer
- Change the spacing the of the receptive field
- Dramatically reduce model computational complexity (squared)
  - Height and width stride can be different

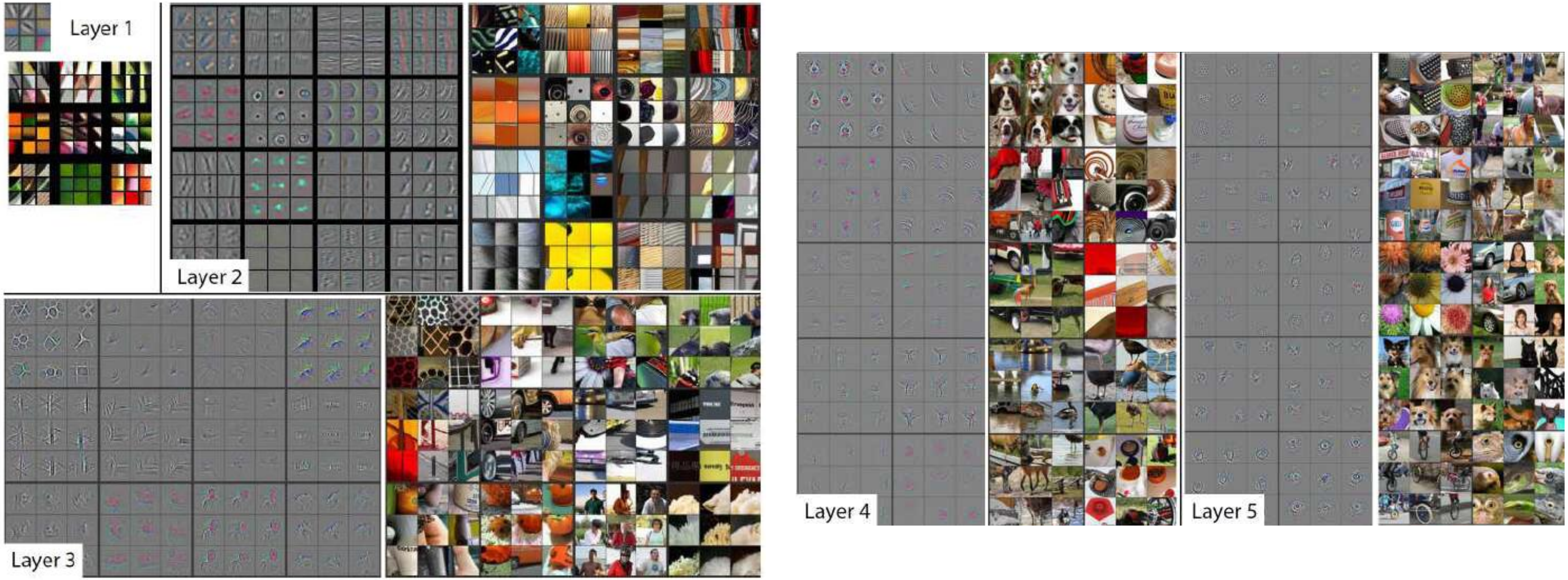


FILTERS

- Filters = convolutional kernels
- Size of the kernel is the receptive field for the neuron
- Feature map – output of the "convolution" operation
  - Highlights areas in an image that activate the filter most
- For CNNs, the filters are not defined manually!
  - Learn most useful filters for a task
  - Higher layers will learn to combine into more complex patterns



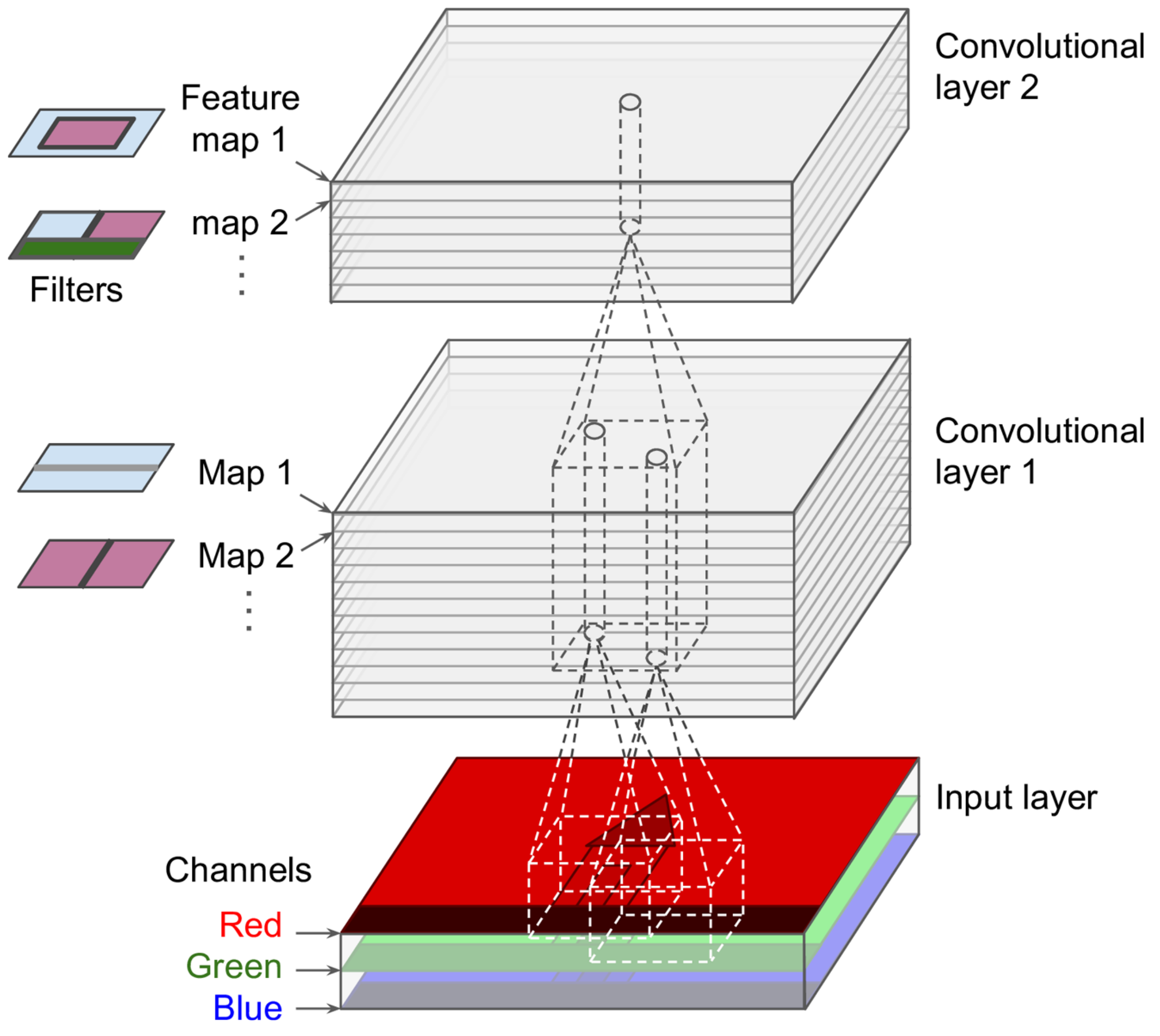
VISUALIZING WEIGHTS AND FEATURES



See Szeliski 2e, Ch 5.4.5

STACKING MULTIPLE FEATURE MAPS I

- Each convolution layer has multiple filters
  - Stacked 3D output (1 feature map for each filter)
- Each neuron in a feature map shares the same parameters (weights and bias)
- Neurons in different feature maps use different parameters
- Neuron's receptive field applies to all feature maps of previous layer
- Note input images often have multiple sublayers (channels)



STACKING MULTIPLE FEATURE MAPS II

- Output of a neuron in a convolutional layer

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k=0}^{f_{n'}-1} x_{i',j',k'} \times w_{u,v,k',k}$$

$$\begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

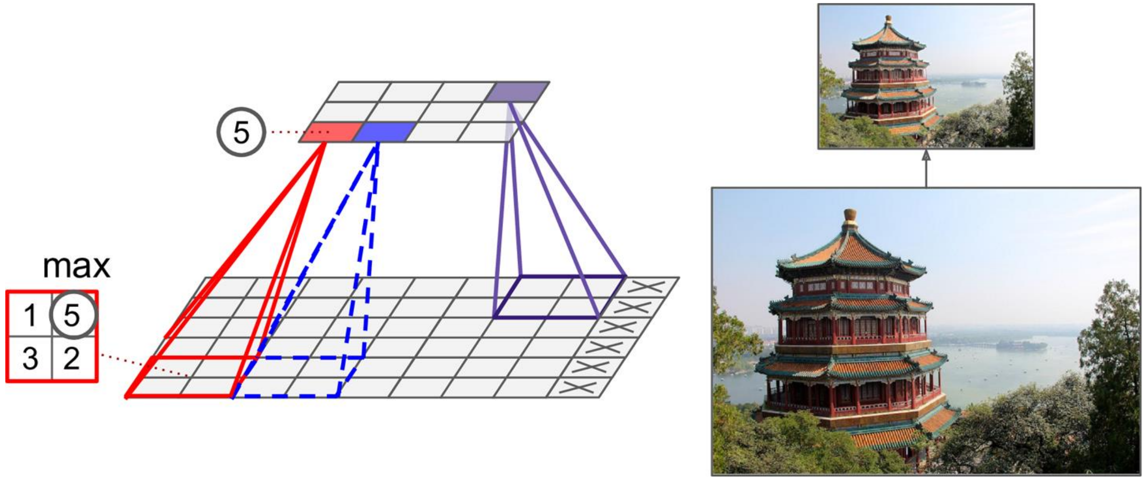
- $z_{i,j,k}$ - output of neuron in row i, column, j, in feature map k of the convolutional layer l
- b_k - bias term for feature map k (in layer l)
  - Tweaks overall brightness of feature map k
- s_h, s_w - vertical and horizontal strides
- f_h, f_w - height and width of receptive field (kernel)
- $f_{n'}$ - number of feature maps in previous (lower layer)
- $x_{i',j',k'}$ - output of neuron located in layer $l-1$, row i', column j', feature map k
- $w_{u,v,k',k}$ - connection weight between any neuron in feature map k of the layer l and its input located at row u, column v (relative to the neuron's receptive field), and feature map k'

MEMORY REQUIREMENTS

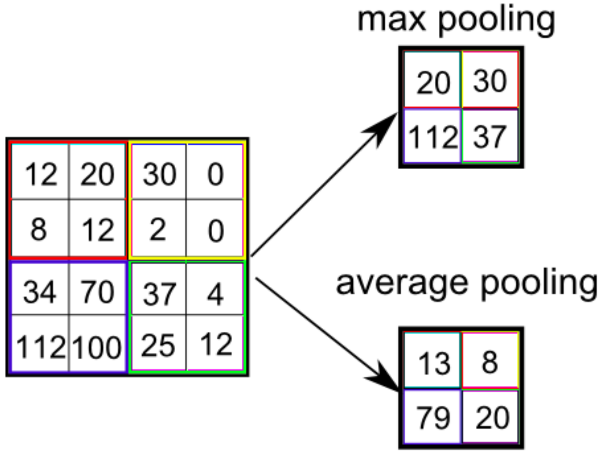
- Though much smaller the fully connected networks, CNNs still use significant amount of RAM
- During training, the reverse pass of backpropagation requires all the intermediate values computed during the forward pass
  - Need to have enough for all layers in the network
  - Forward pass can release memory after each layer is computed (only two consecutive layers required)
- Out-of-memory error
  - Reduce mini-batch size, increase stride, remove layers, change precision (16-bit vs 32-bit floats or use int), or distribute the CNN across devices

POOLING LAYERS

- Subsample input in order to reduce computational load, memory usage, and number of parameters (reduce risk of overfitting)
- Aggregate over the receptive field
  - Aggregate functions such as max (most popular) or mean
  - Max tends to work better by preserving only the strongest feature → cleaner signal, more invariance, less compute
- Stride gives downsampling
- Pooling kernel size can be even



Max pooling layers (2x2 kernel, stride=2, no padding)



POOLING LAYERS INVARIANCE

- Introduces some level of invariance to small translations
  - Small image shifts result in same response
  - Additionally small invariance to rotation and scale with max pool
- Max pool every few CNN layers for invariance at larger scale
  - Useful when task should be invariant (e.g. image classification)
- Drawbacks
  - Destructive – 2x2, stride 2 drops 75% of input values
  - Invariance not always desirable (e.g. semantic segmentation should have equivariance)