Deep Convolutional Neural Nets
Part II

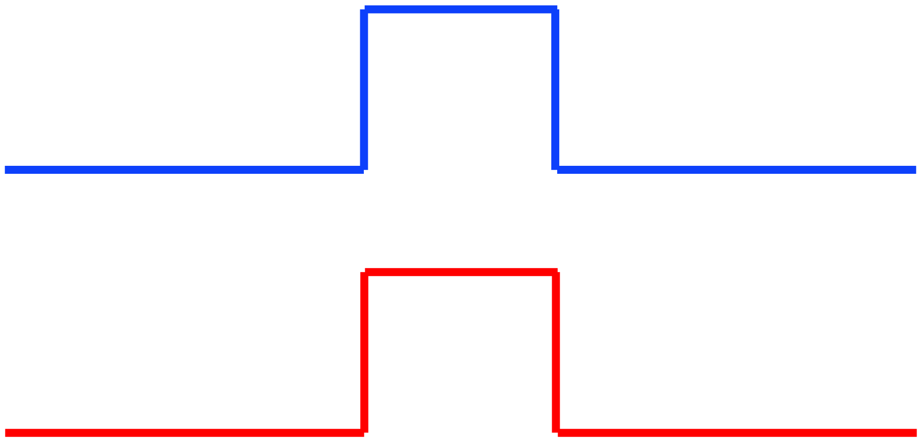
Tim Dunn

Duke MLSS 2018

Intro to the Convolution (1D)

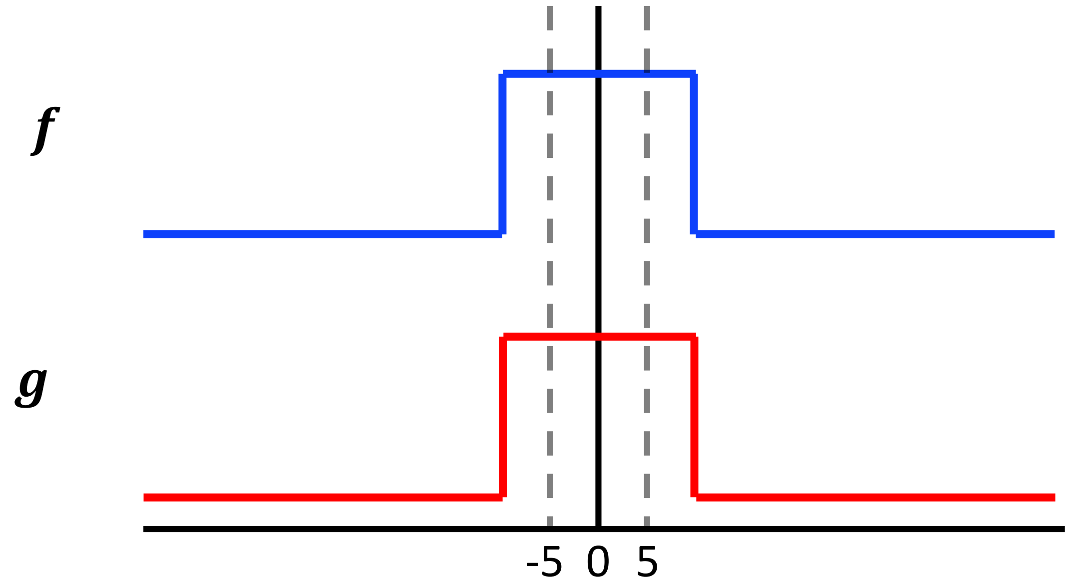
f

g



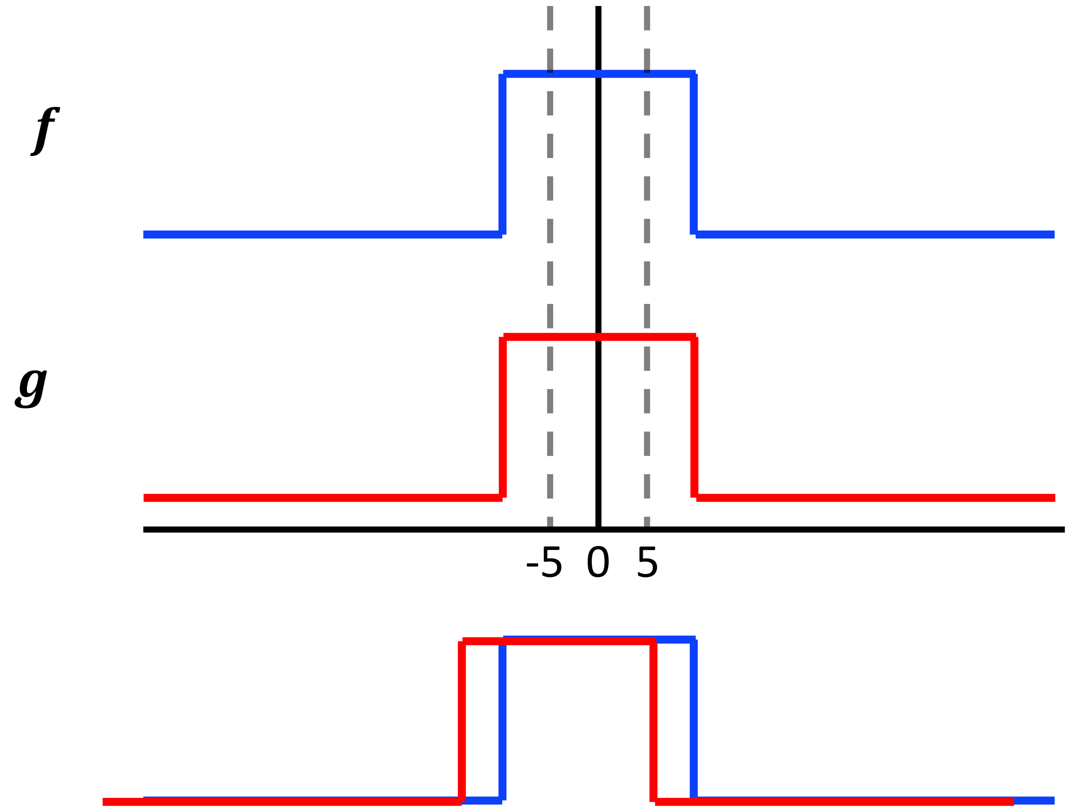
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

Intro to the Convolution (1D)



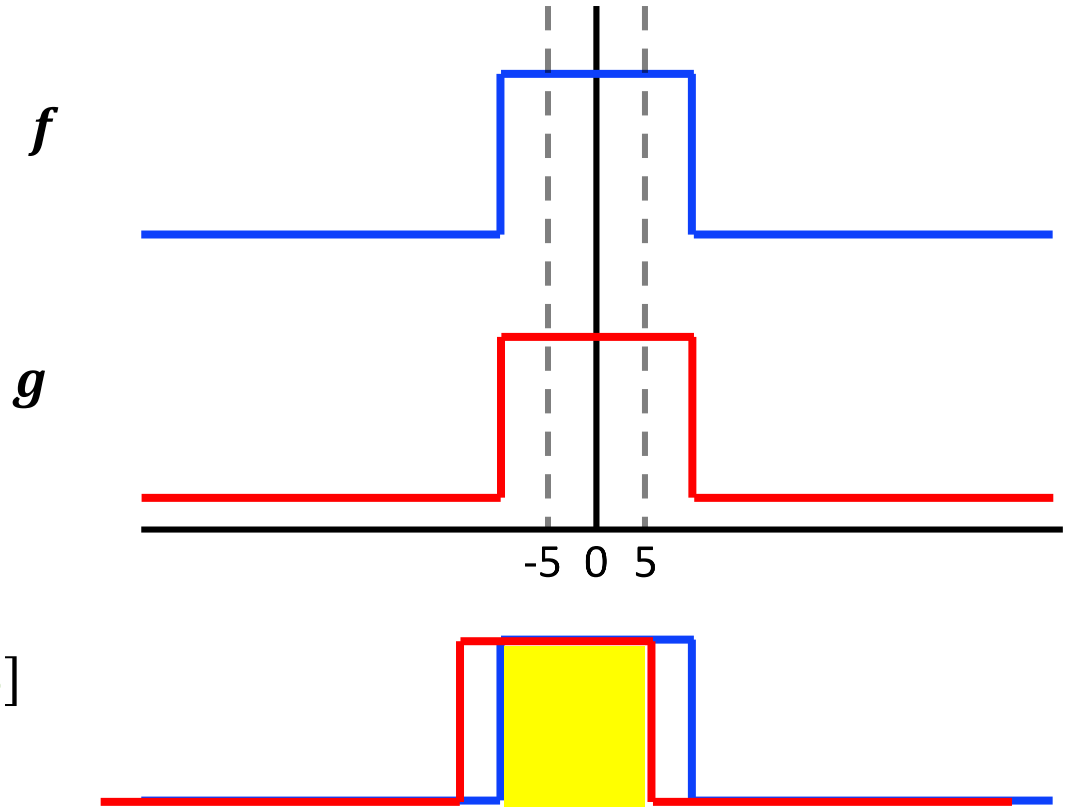
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

Intro to the Convolution (1D)



$$(f * g)[-5] = \sum_{m=-\infty}^{\infty} f[m]g[-(5+m)]$$

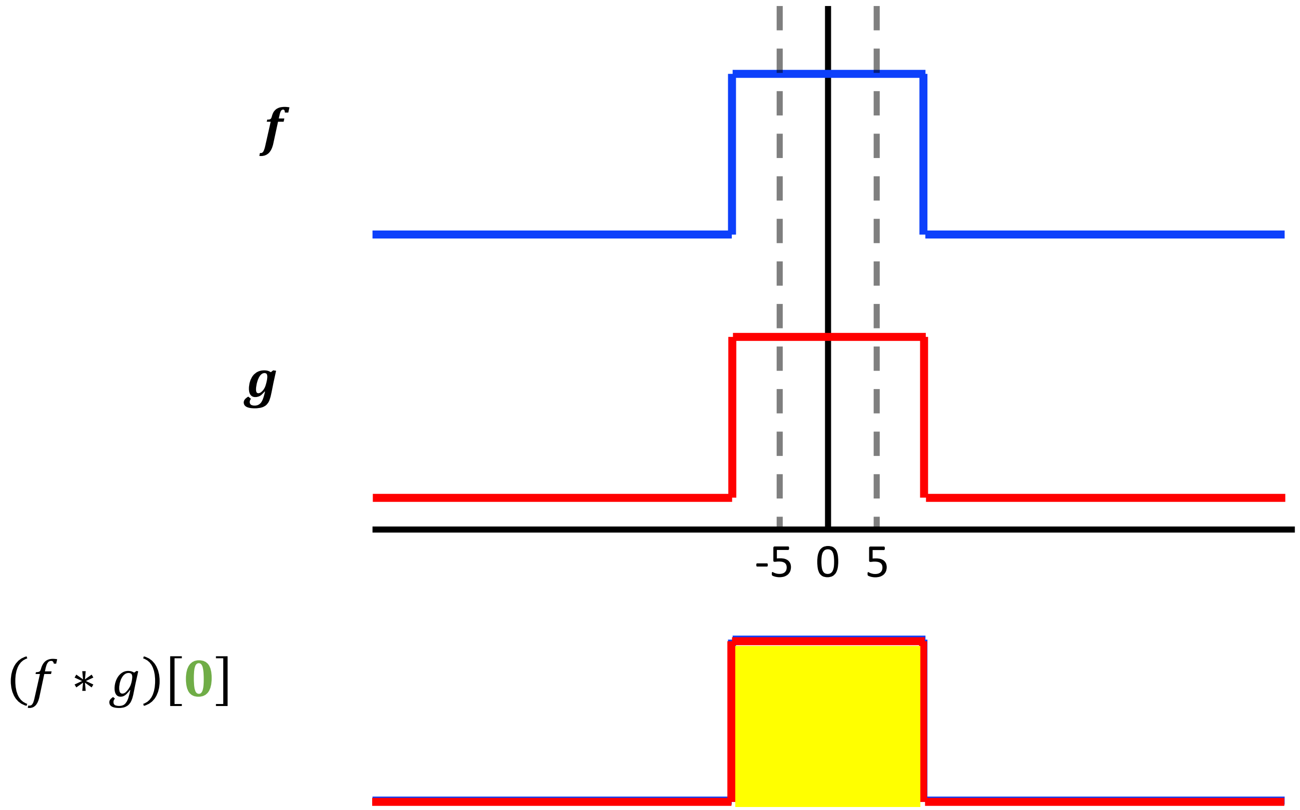
Intro to the Convolution (1D)



$(f * g)[-5]$

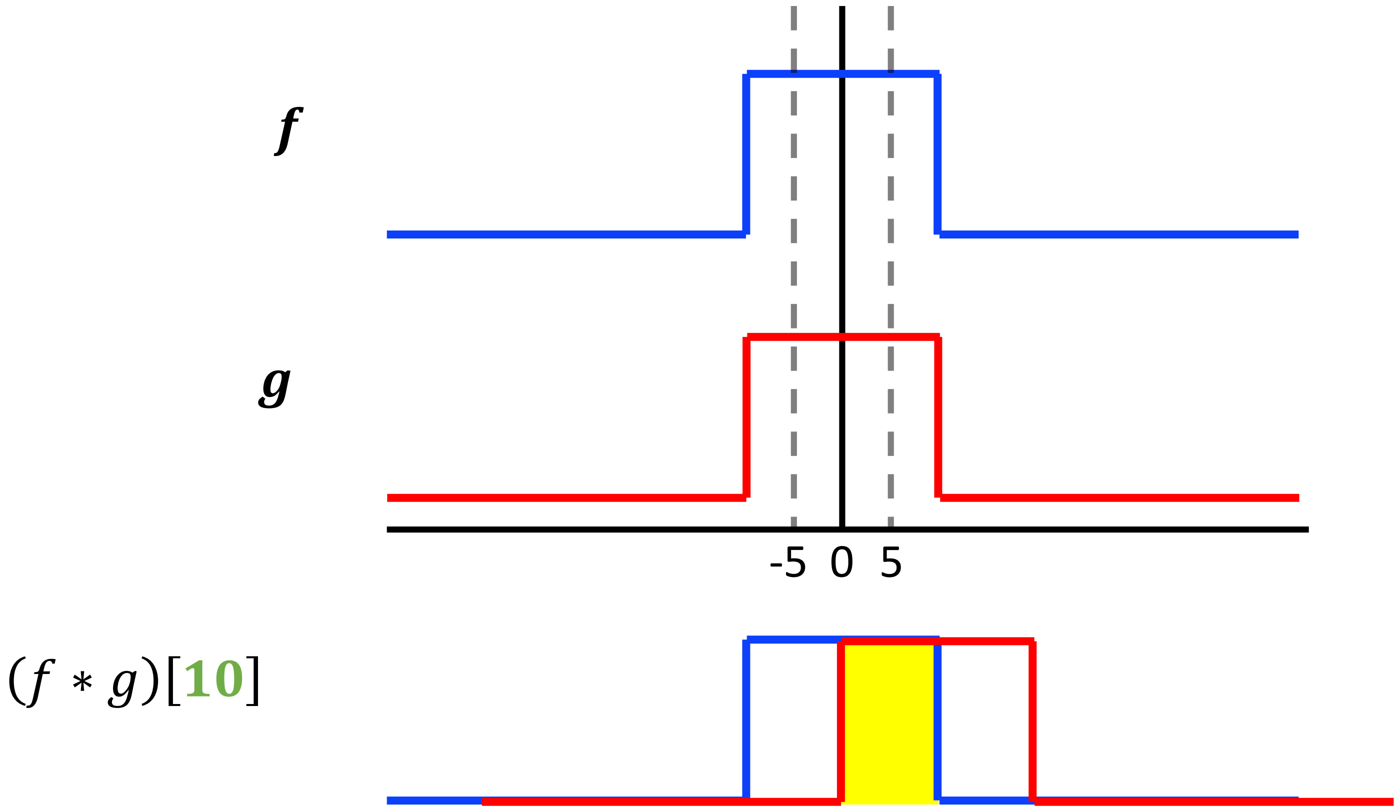
$$(f * g)[-5] = \sum_{m=-\infty}^{\infty} f[m]g[-(5+m)] = 15$$

Intro to the Convolution (1D)



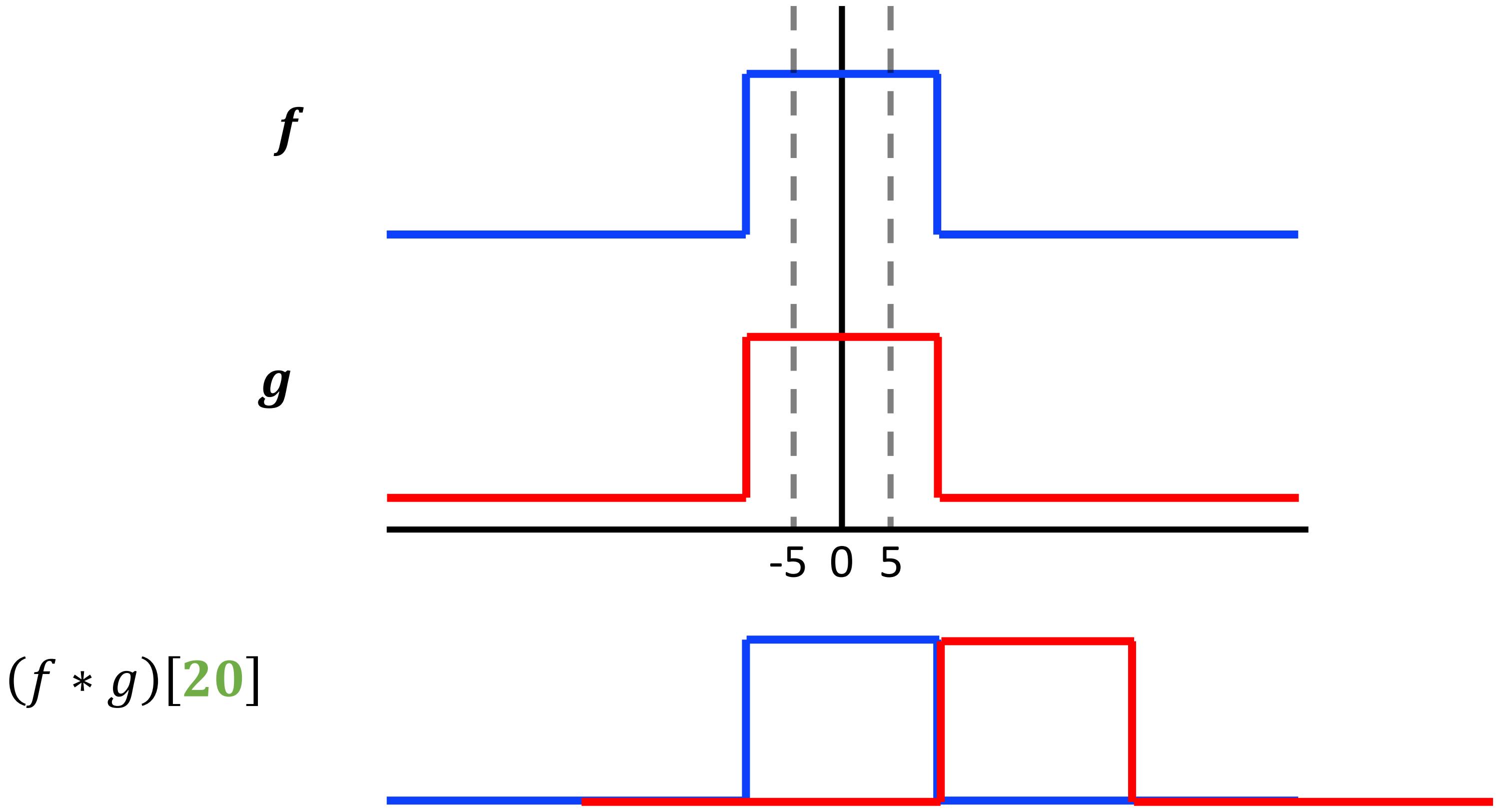
$$(f * g)[\mathbf{0}] = \sum_{m=-\infty}^{\infty} f[m]g[-(m)] = 20$$

Intro to the Convolution (1D)



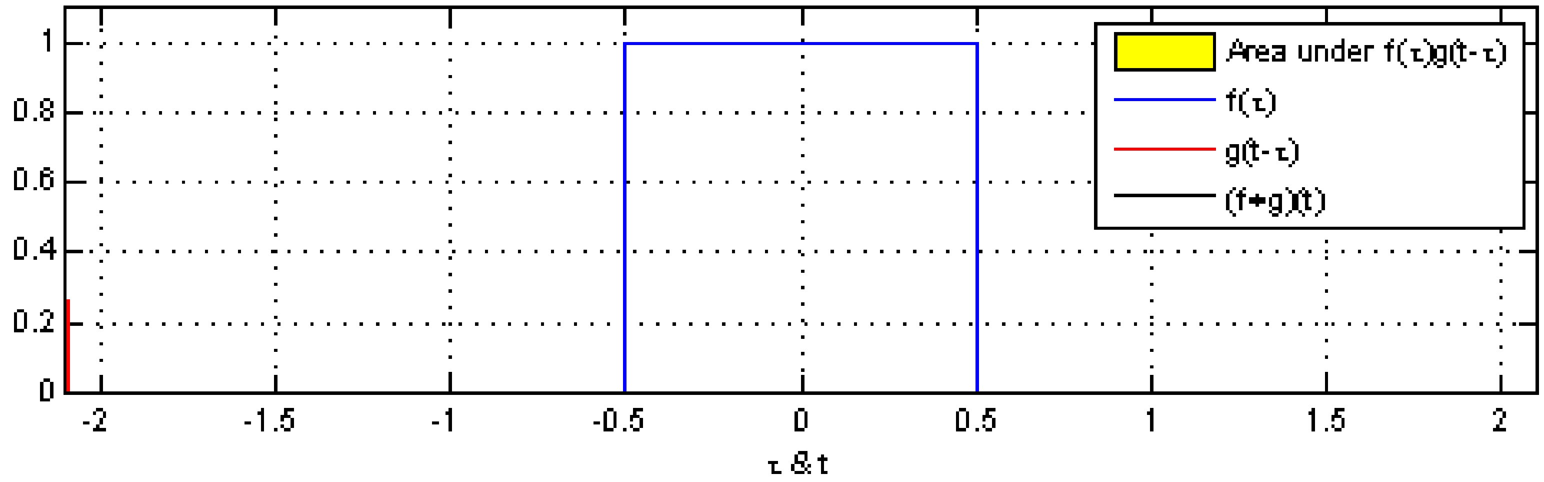
$$(f * g)[10] = \sum_{m=-\infty}^{\infty} f[m]g[-(-10 + m)] = 10$$

Intro to the Convolution (1D)



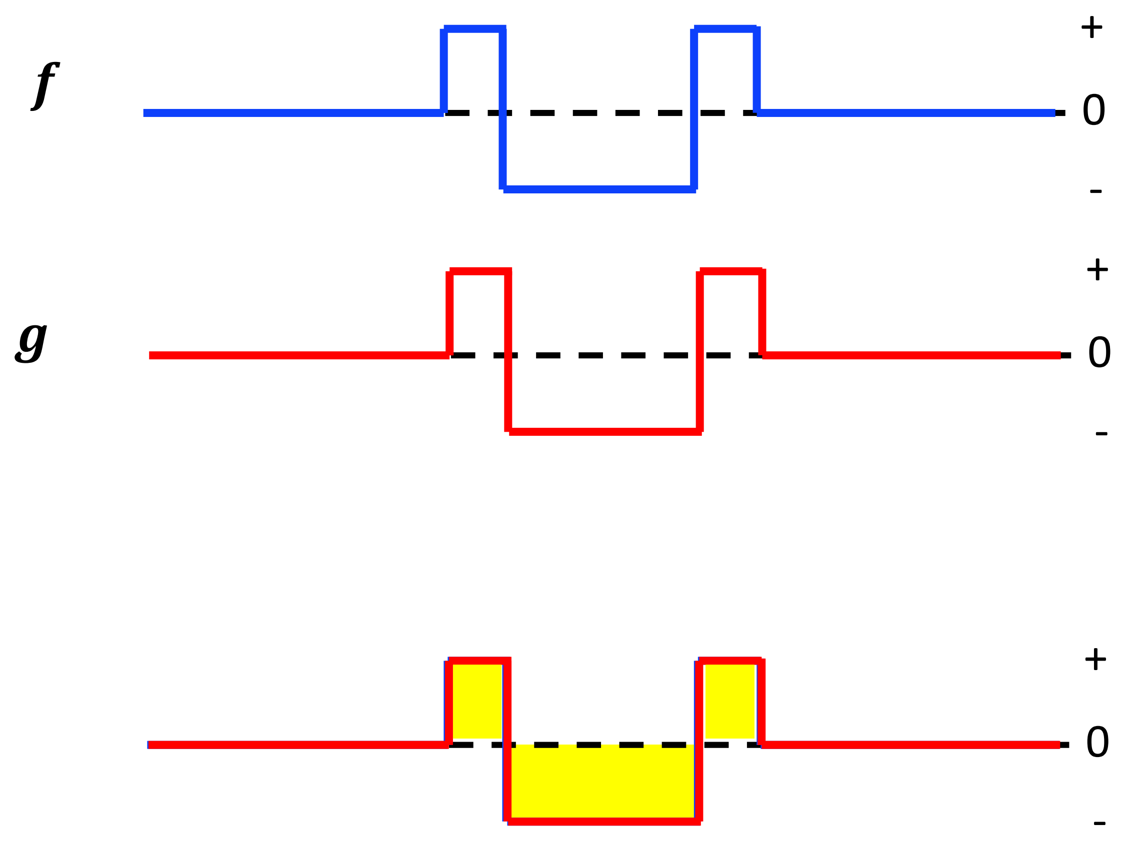
$$(f * g)[20] = \sum_{m=-\infty}^{\infty} f[m]g[-(-20 + m)] = 0$$

Intro to the Convolution (1D)

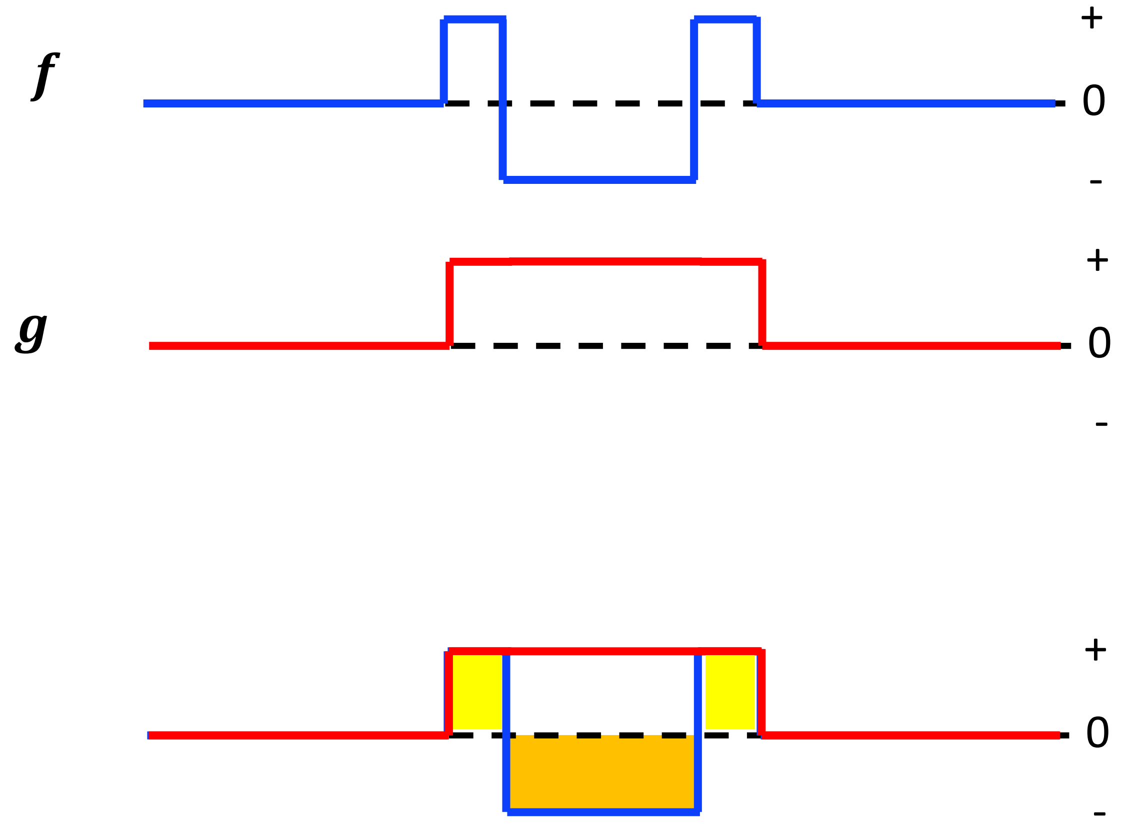


Wikipedia

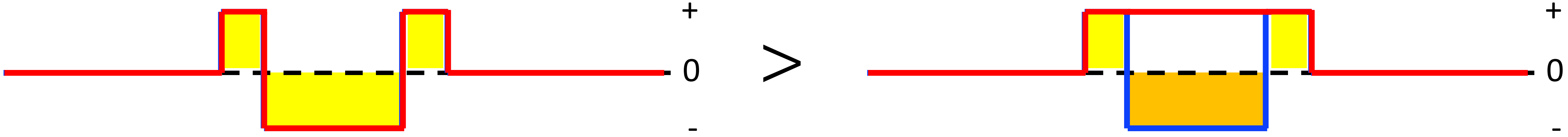
Convolutional Filters Are Feature Detectors



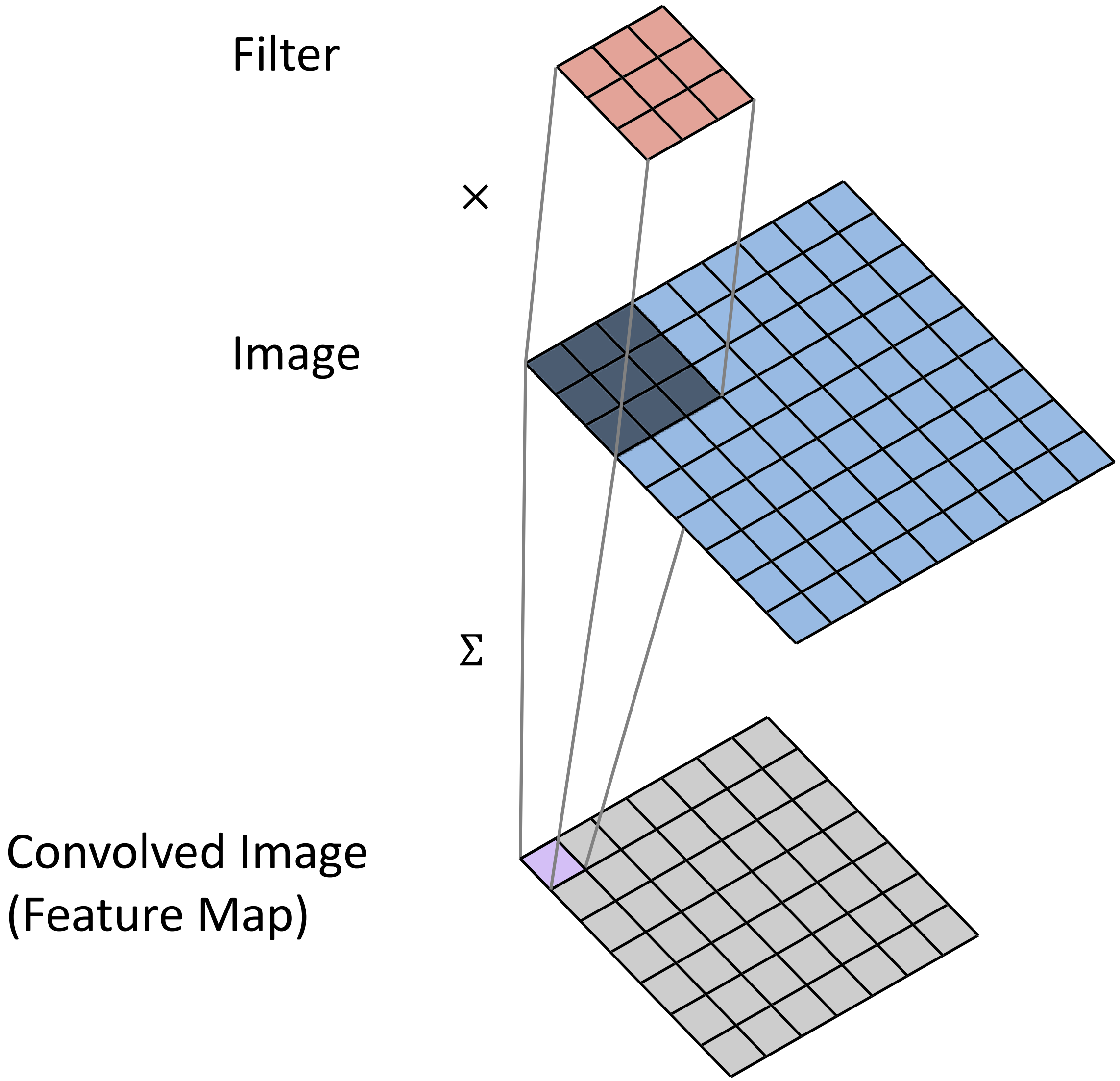
Convolutional Filters Are Feature Detectors



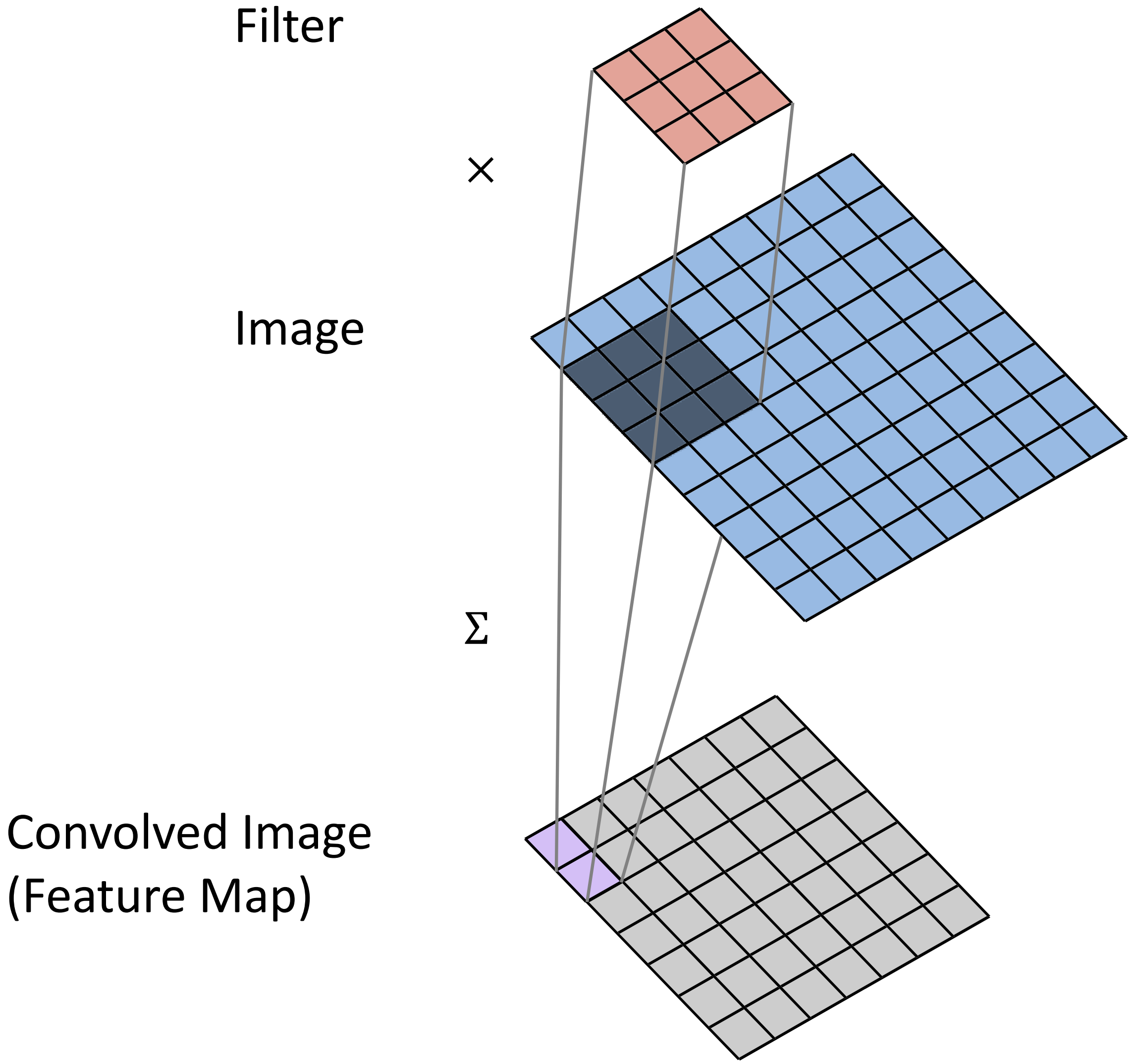
Convolutional Filters Are Feature Detectors



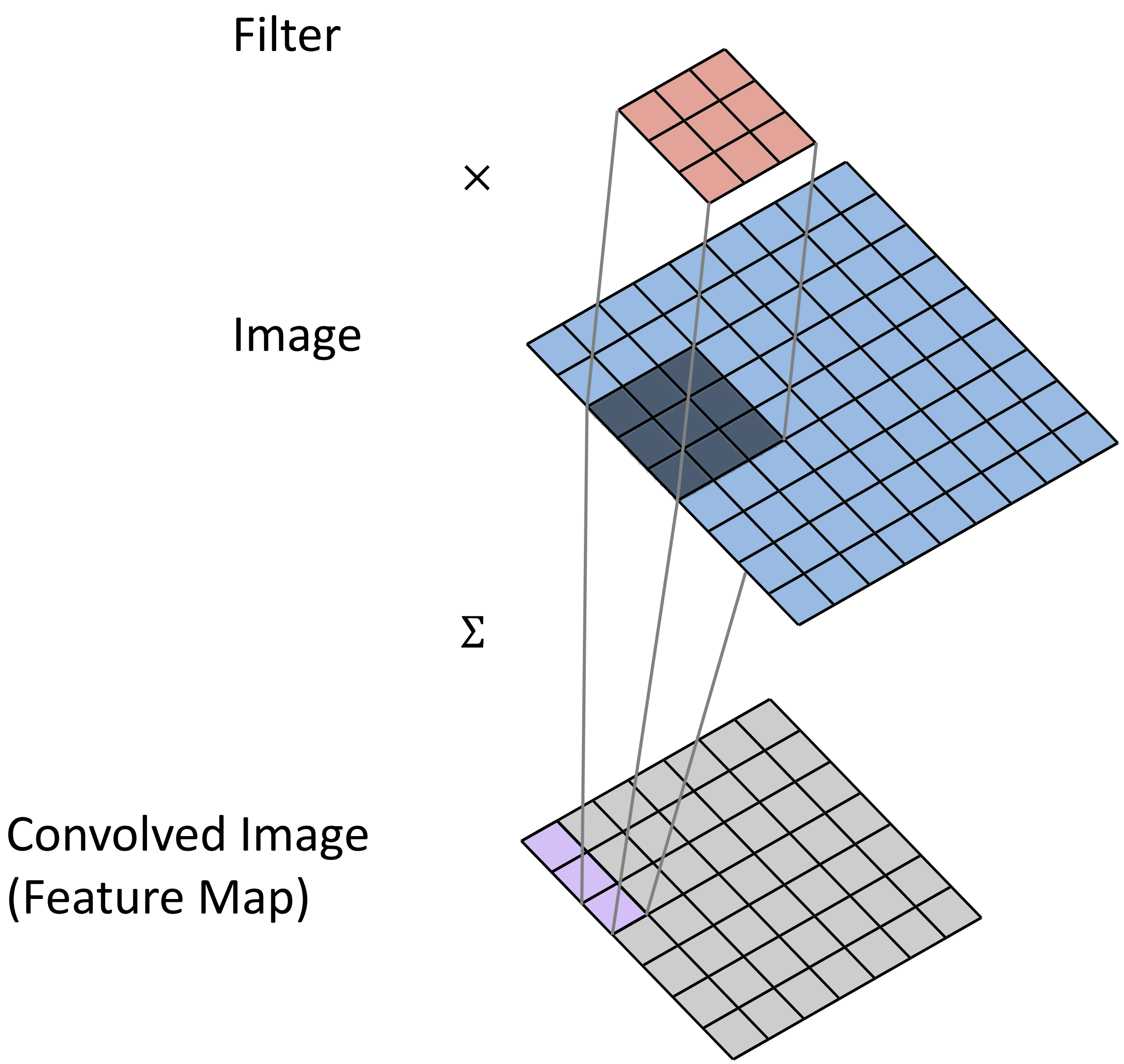
2D Spatial Convolution



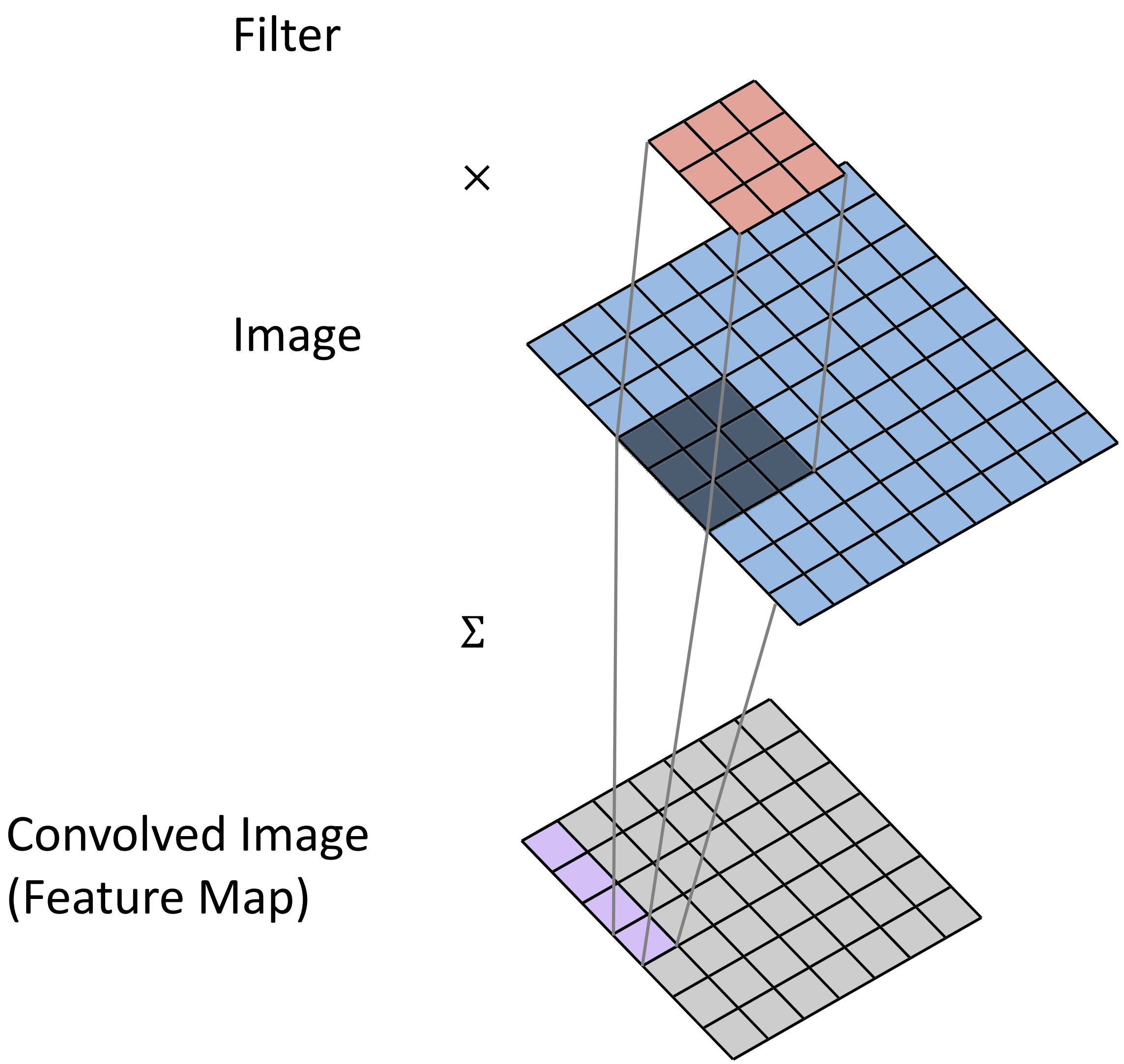
2D Spatial Convolution



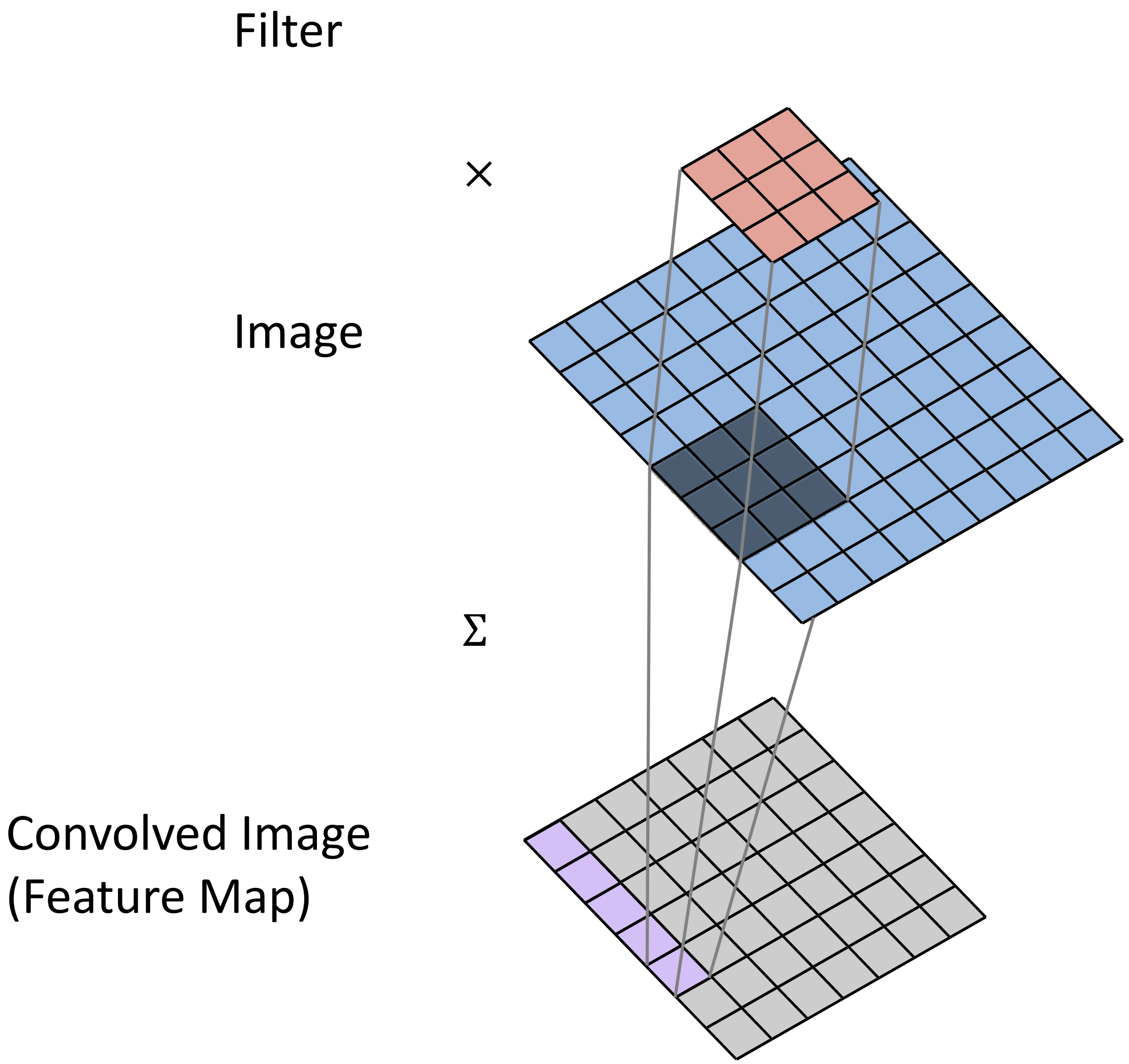
2D Spatial Convolution



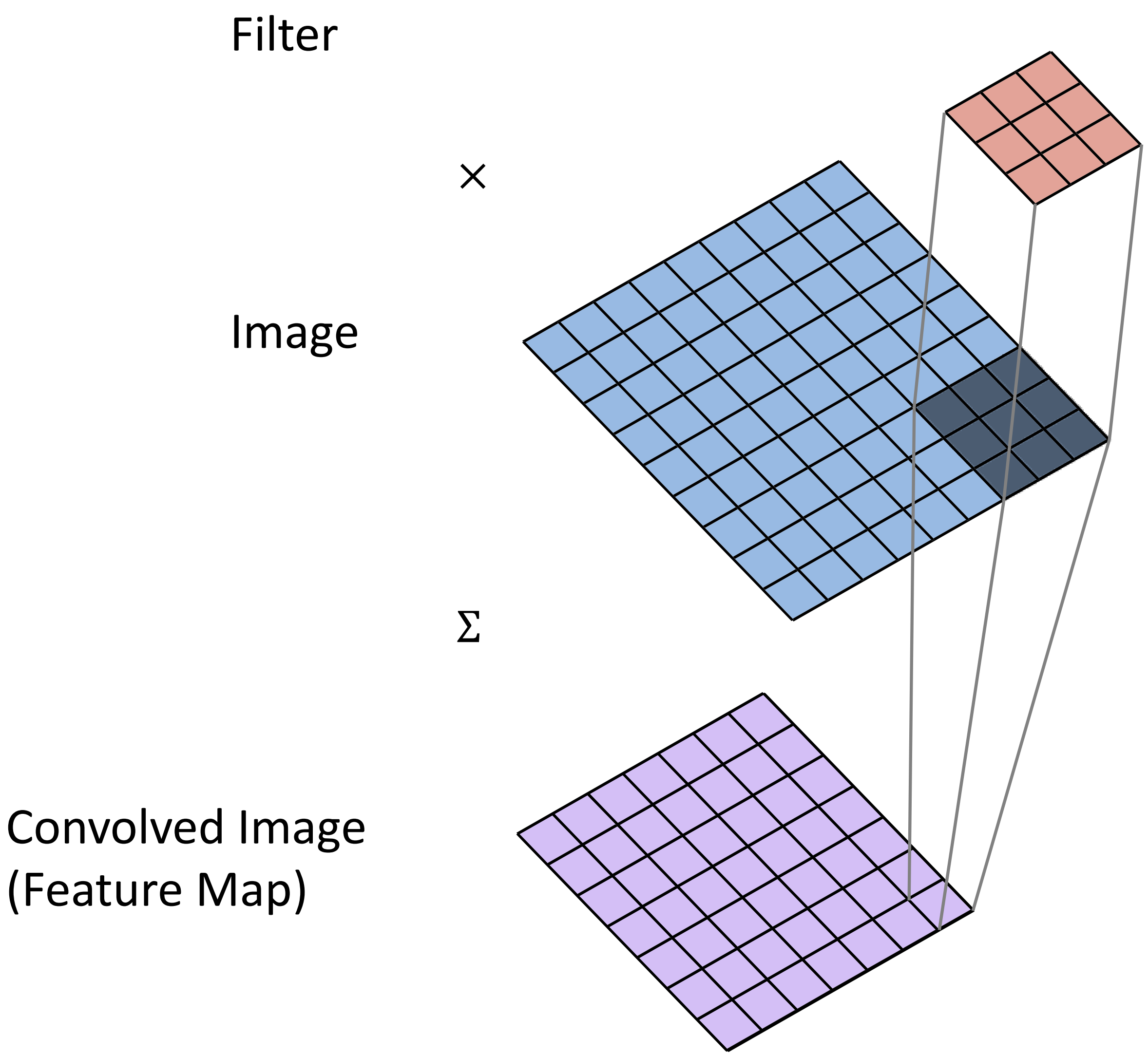
2D Spatial Convolution



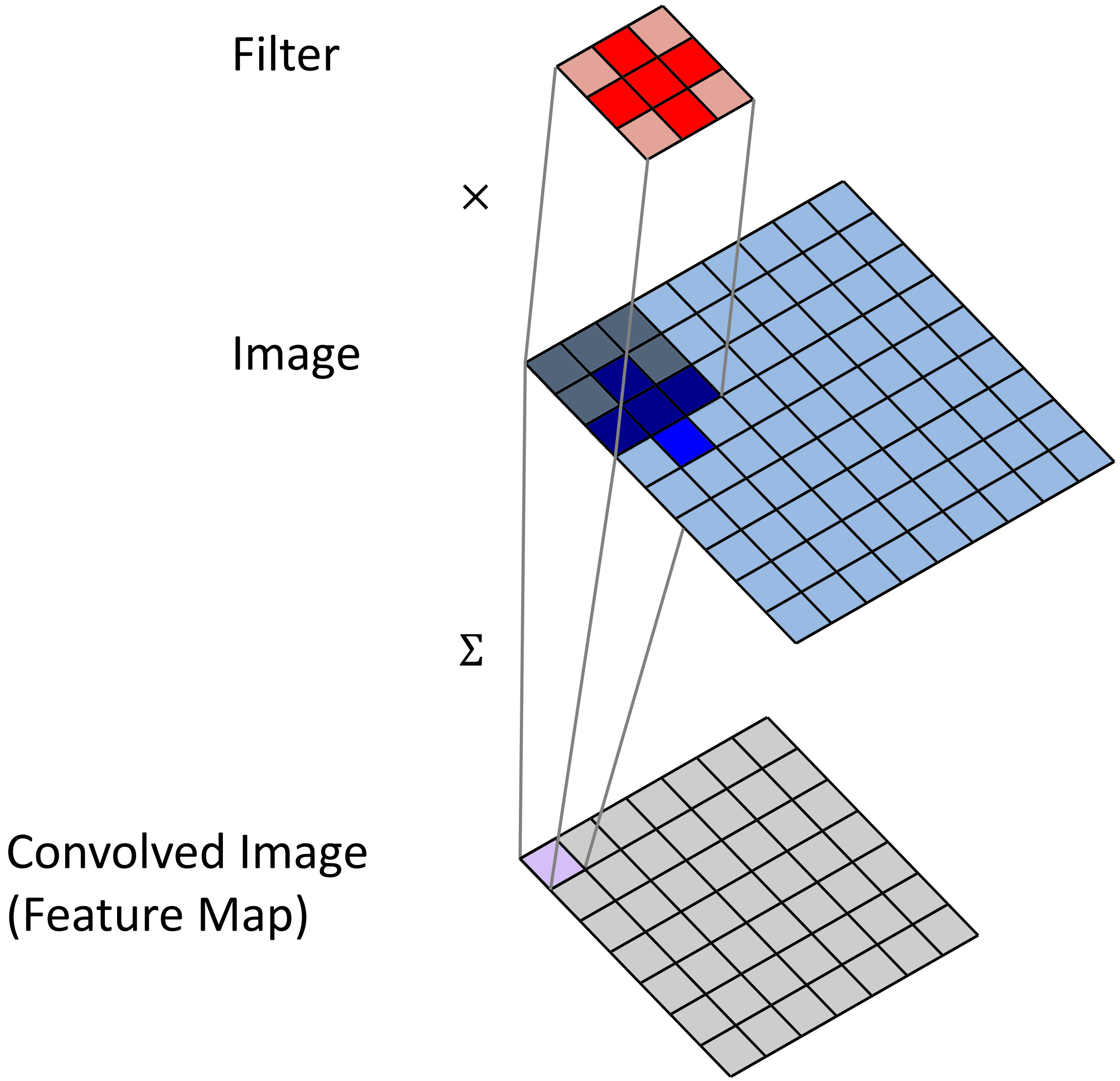
2D Spatial Convolution



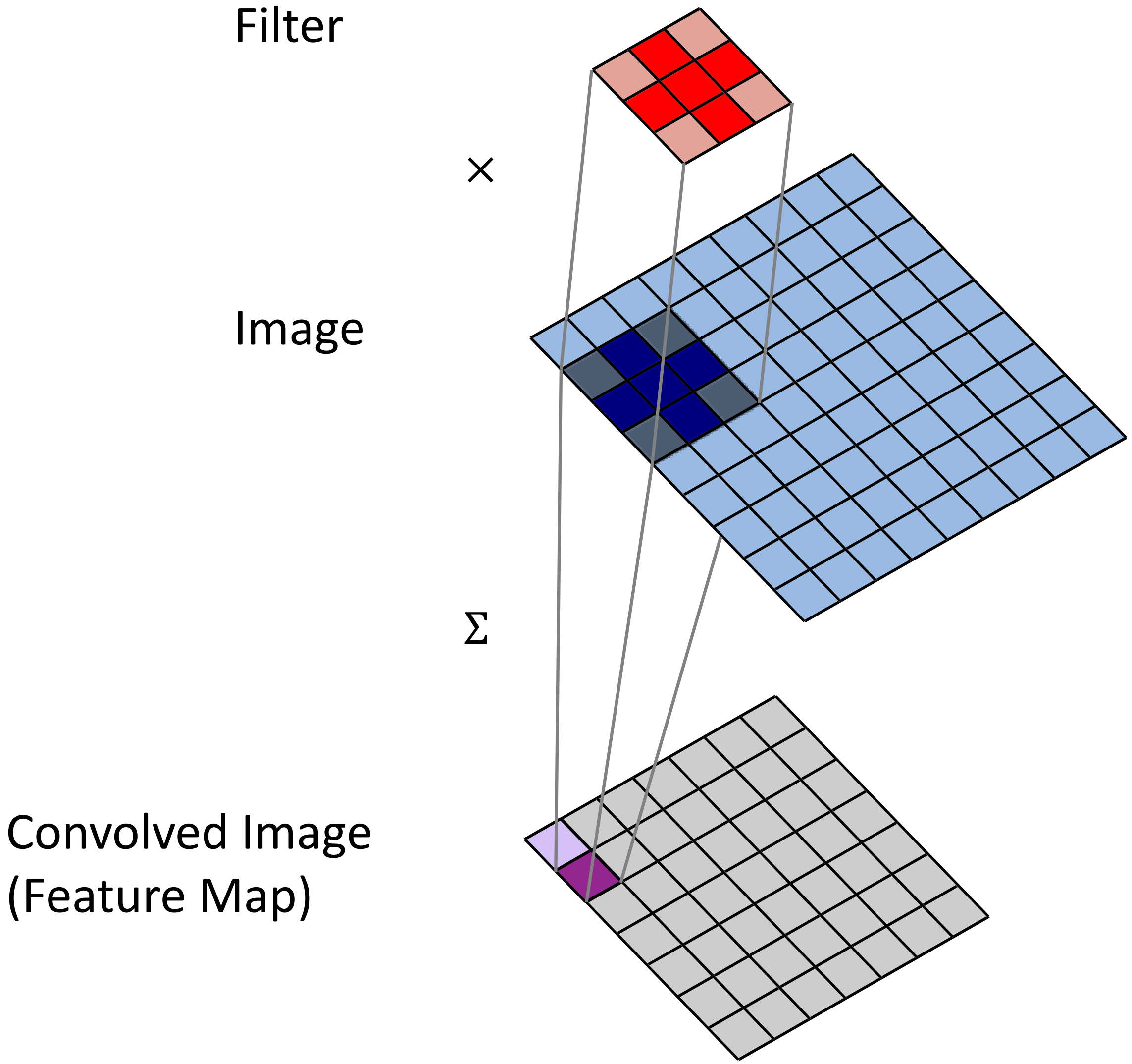
2D Spatial Convolution



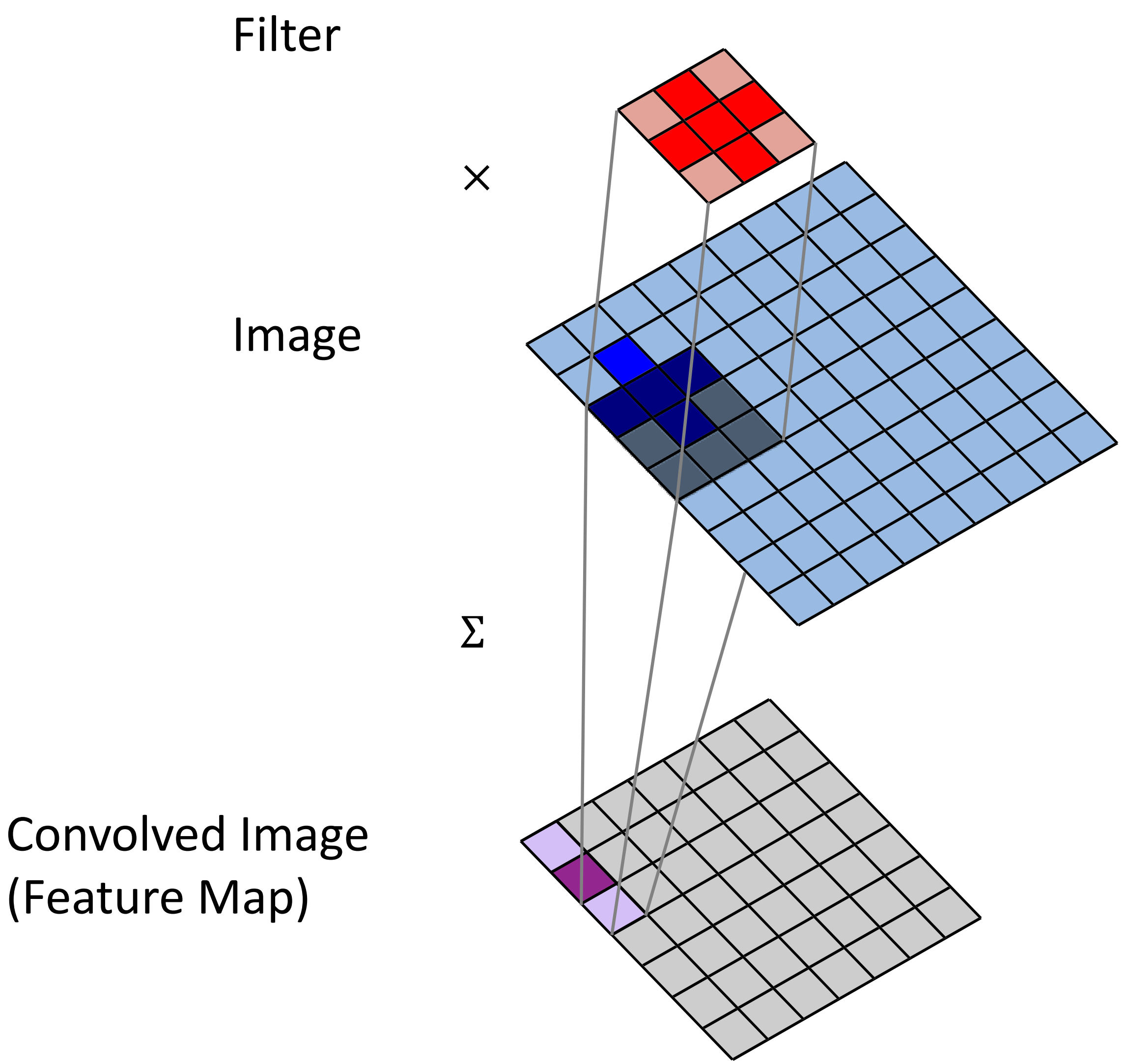
2D Spatial Convolution



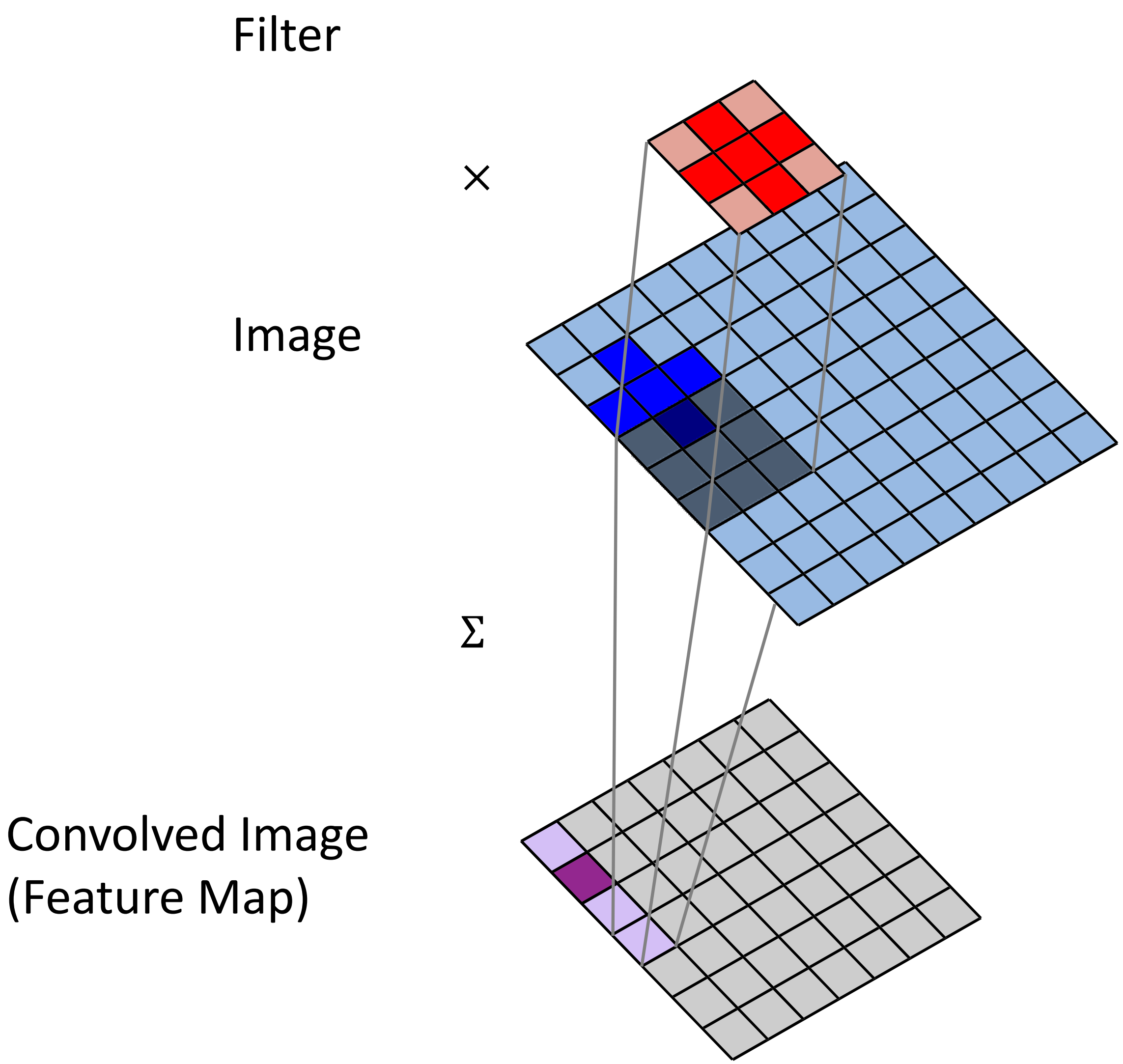
2D Spatial Convolution



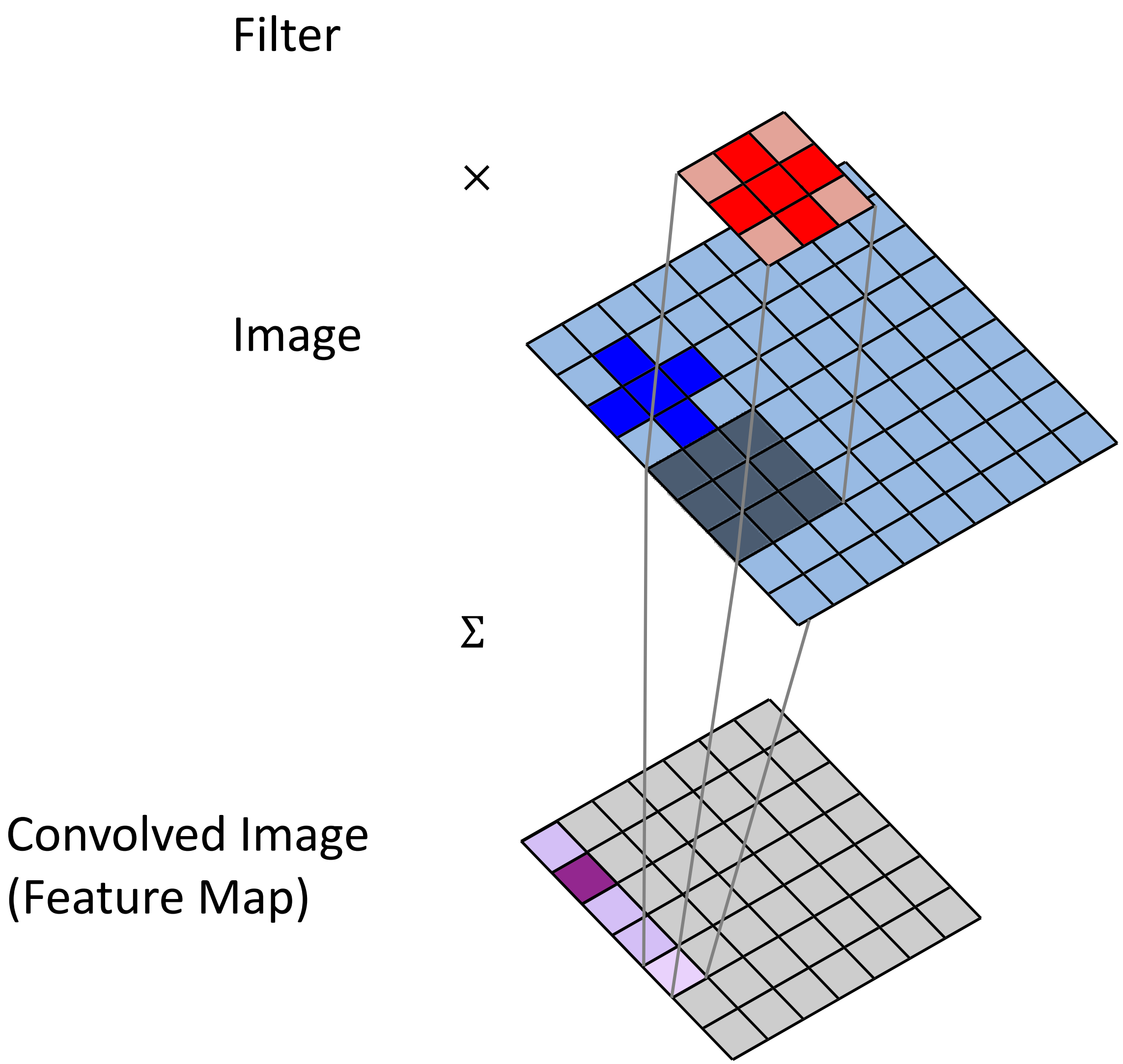
2D Spatial Convolution



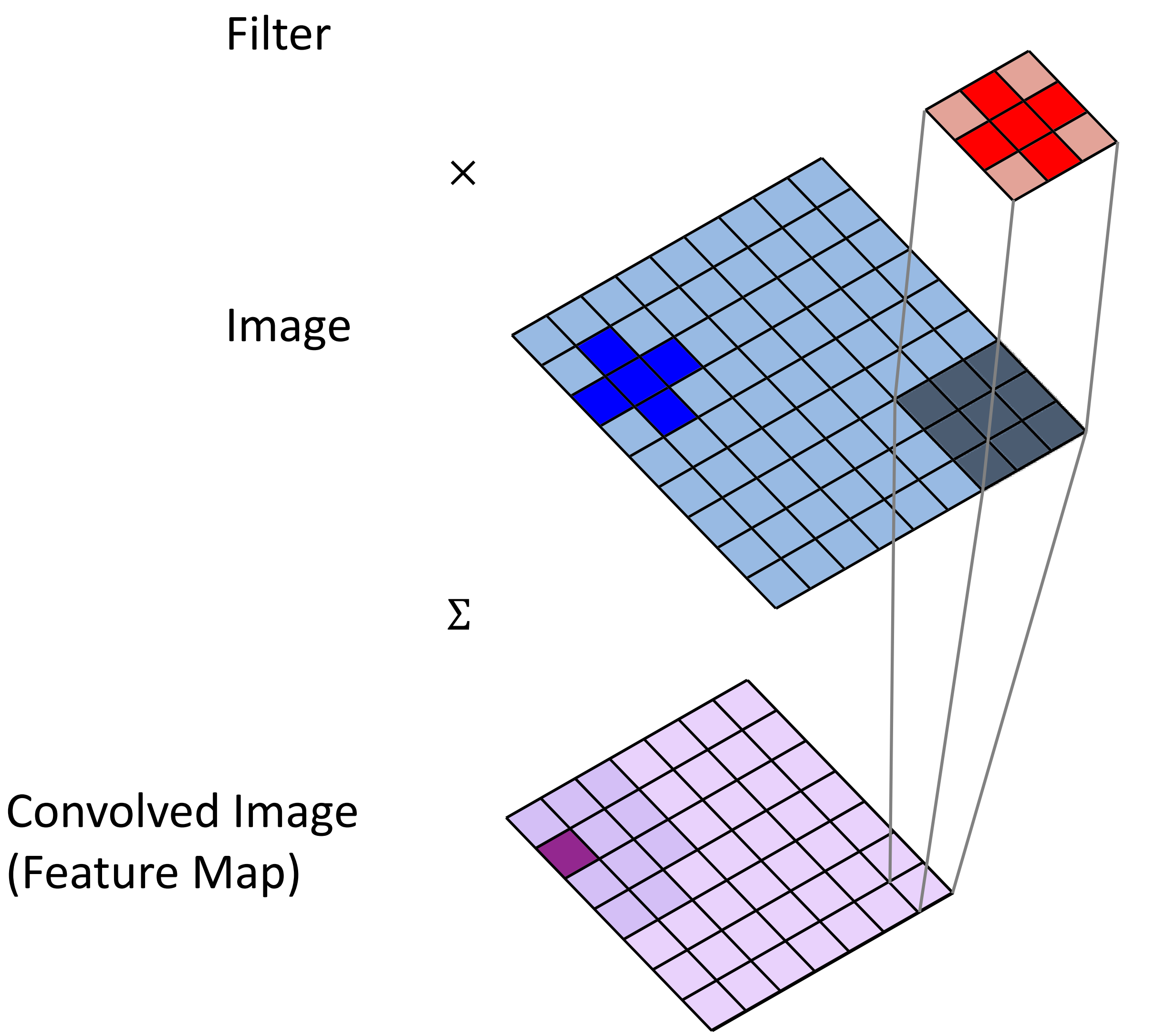
2D Spatial Convolution



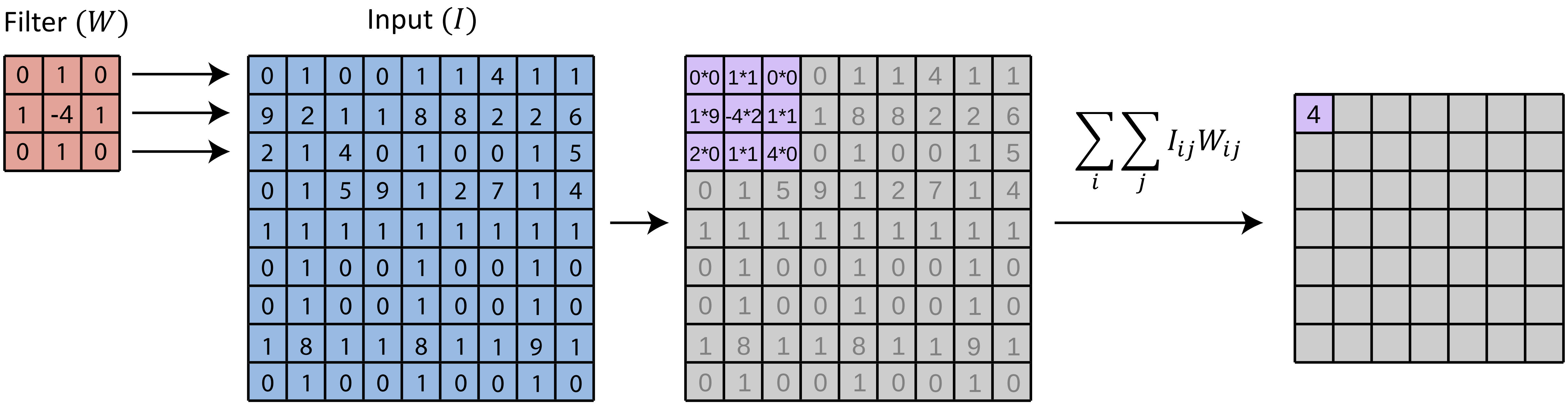
2D Spatial Convolution



2D Spatial Convolution



2D Spatial Convolution



2D Spatial Convolution

Filter (W)

0	1	0
1	-4	1
0	1	0

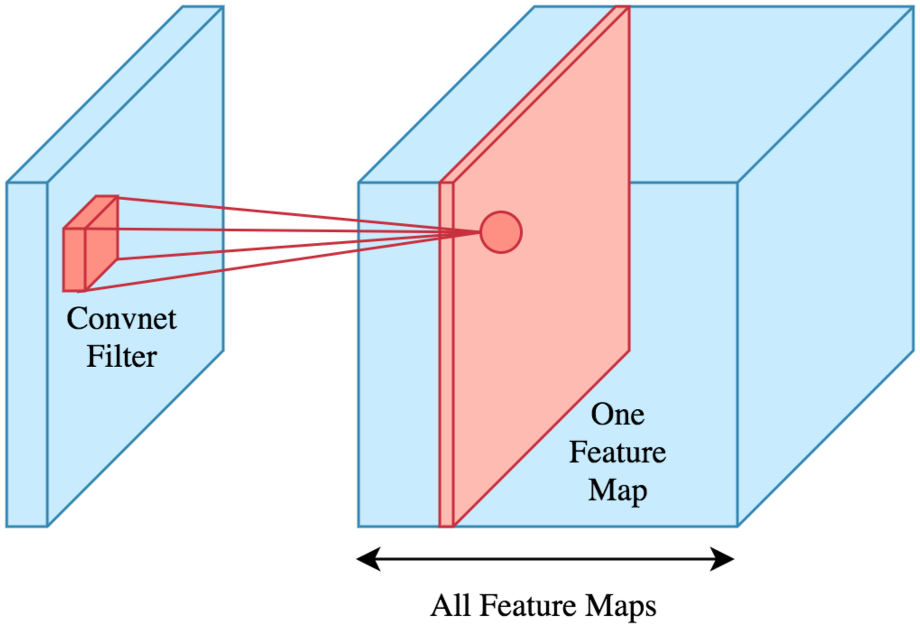
Input (I)

Convolutional Filters Are Feature Detectors

Core Elements of the Convolutional Neural Network

- **Convolutional Layers**
- **Activation Functions**
- **Pooling Layers**
- **Fully Connected Layers**

Convolutional Layer

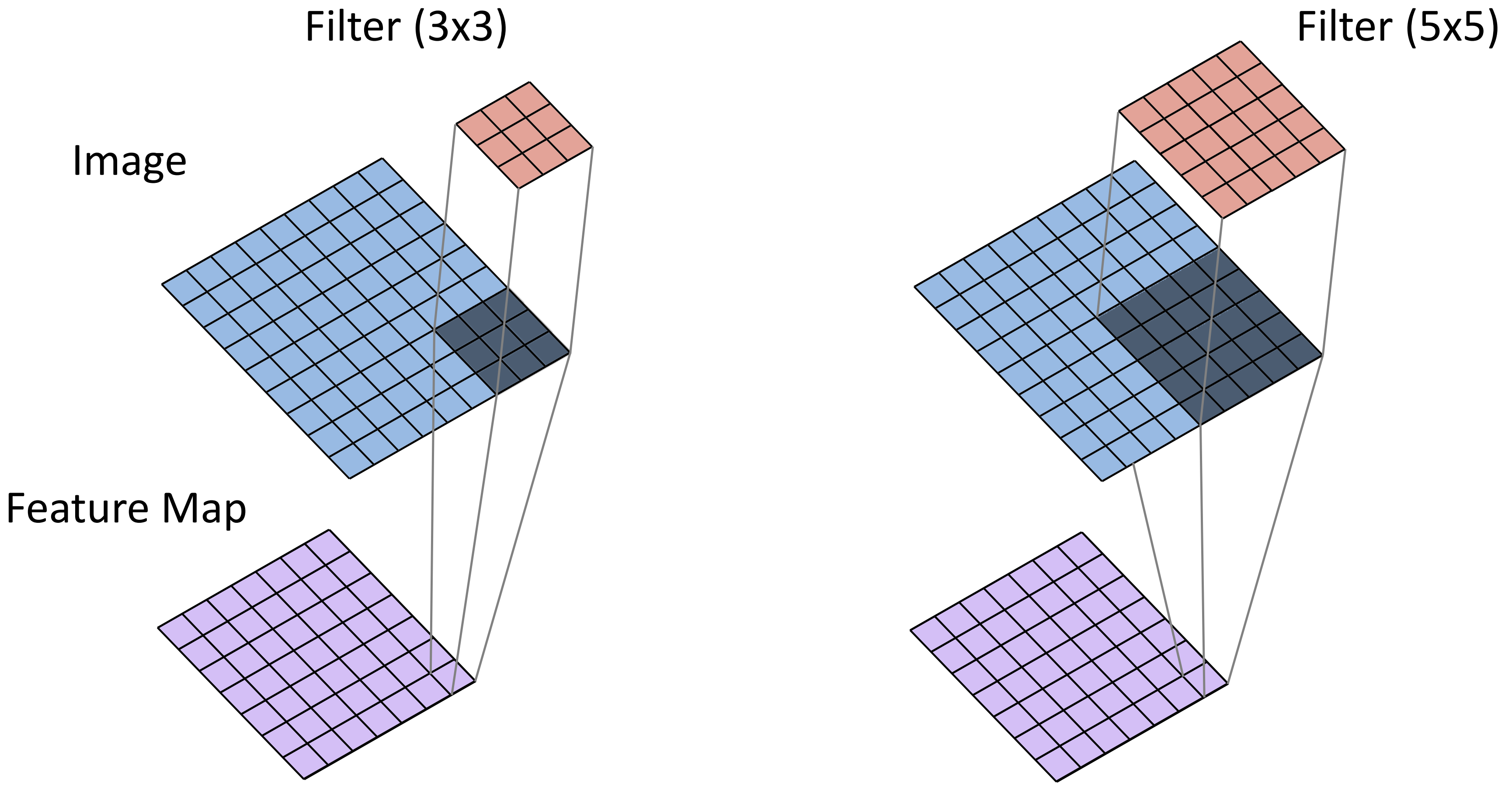


Elements of a convolutional layer:

Filter Size
Filter Stride
Filter (Feature) Number

Convolutional Layer

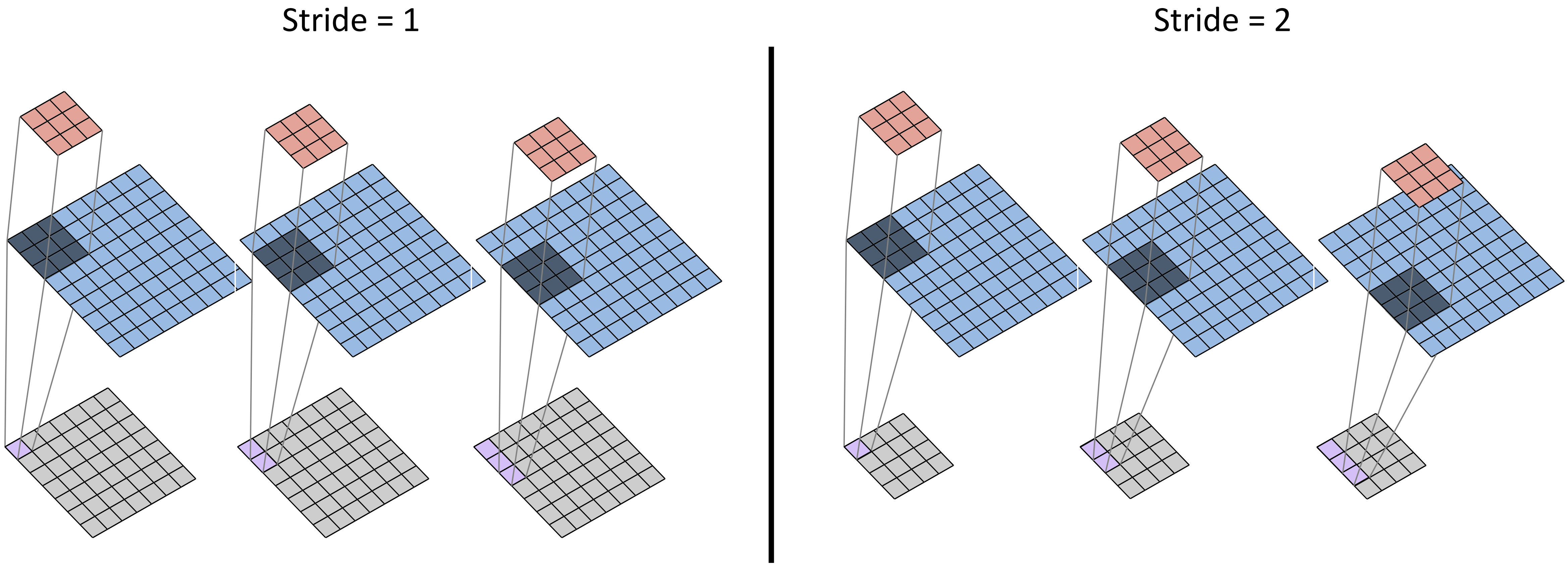
Filter Size



Filters should be just large enough to capture small local features (e.g. edges) in space

Convolutional Layer

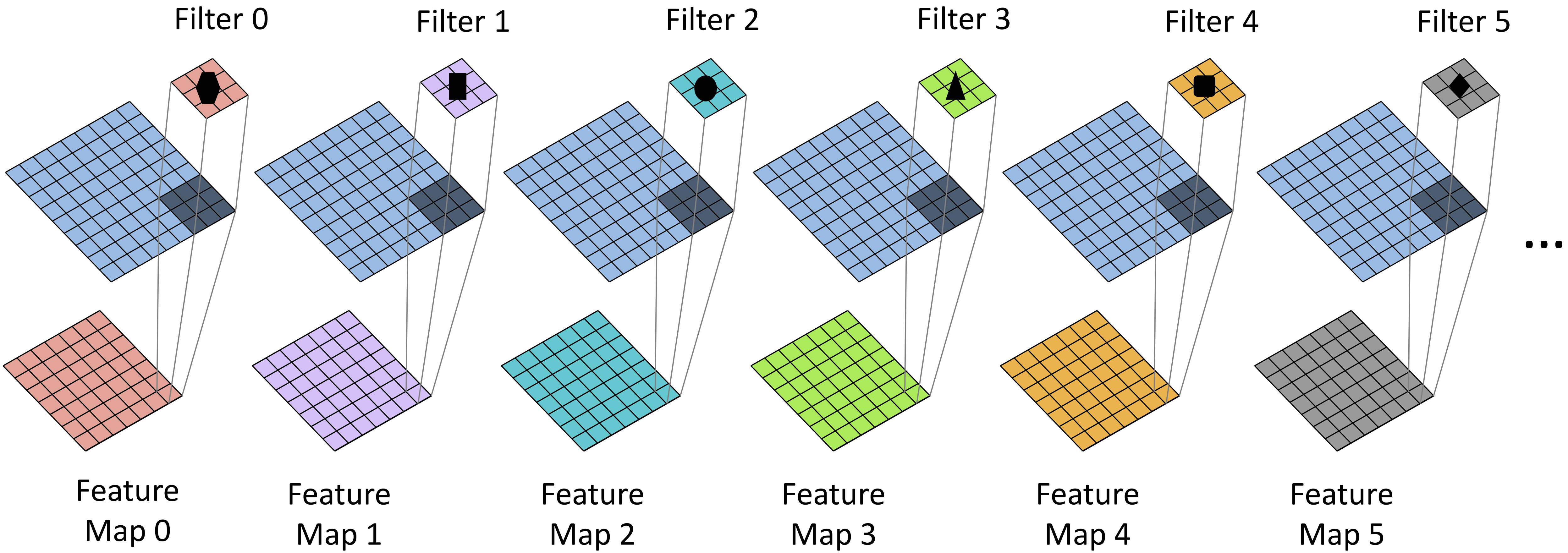
Filter Stride



Filter stride > 1 reduces computational load by downsampling the input

Convolutional Layer

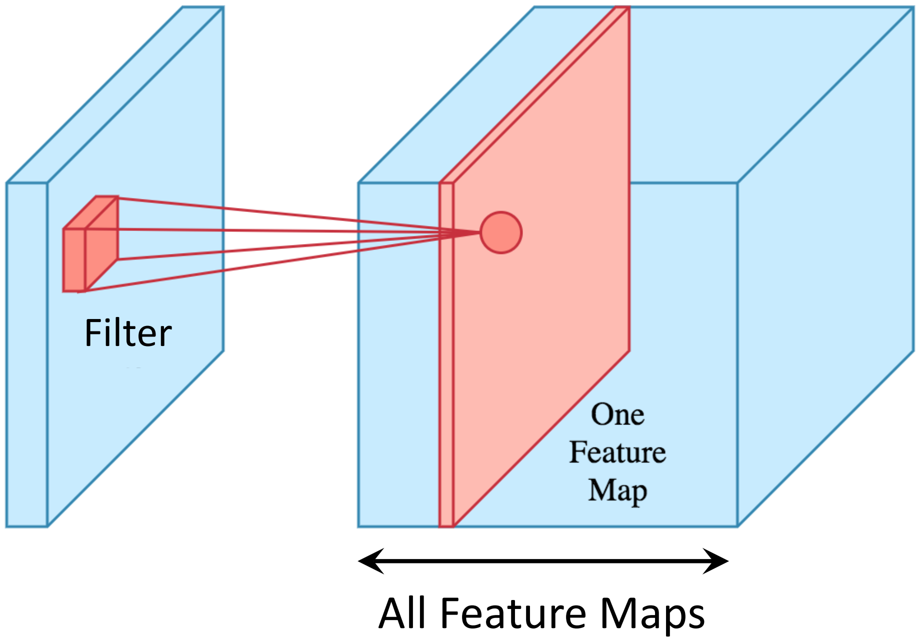
Filter (Feature) Number



Filter number determines the number of unique feature detectors that operate on inputs

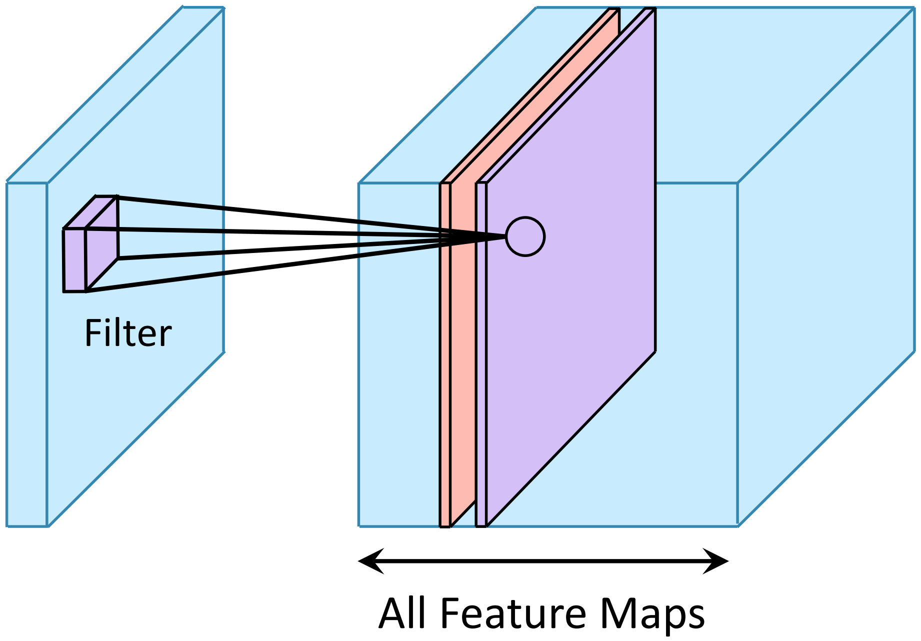
Convolutional Layer

Filter (Feature) Number



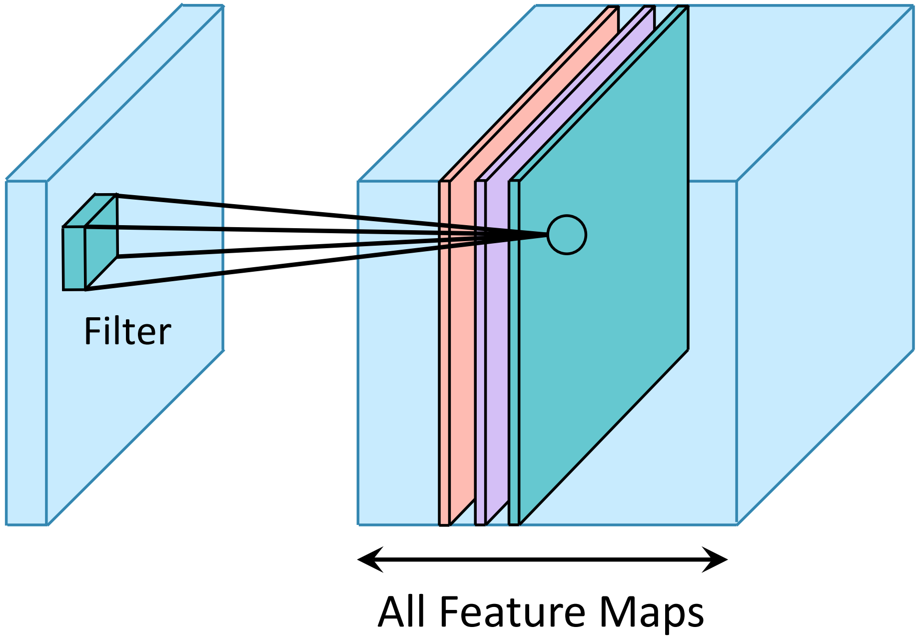
Convolutional Layer

Filter (Feature) Number



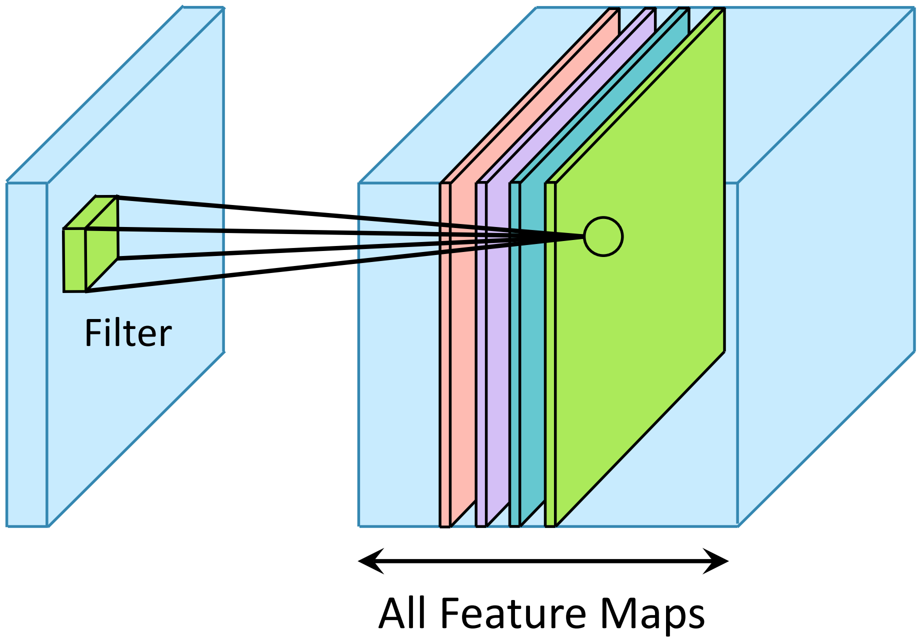
Convolutional Layer

Filter (Feature) Number



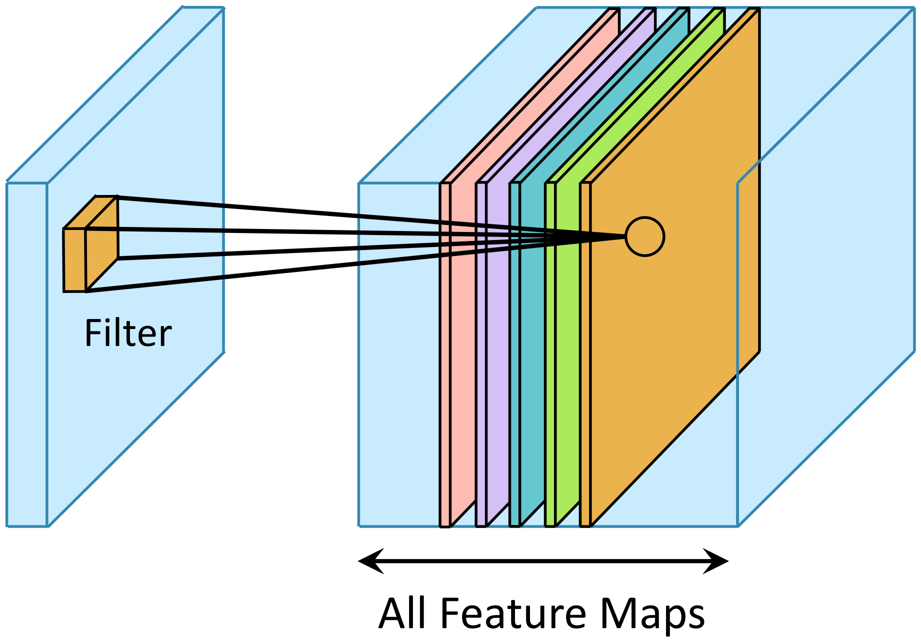
Convolutional Layer

Filter (Feature) Number



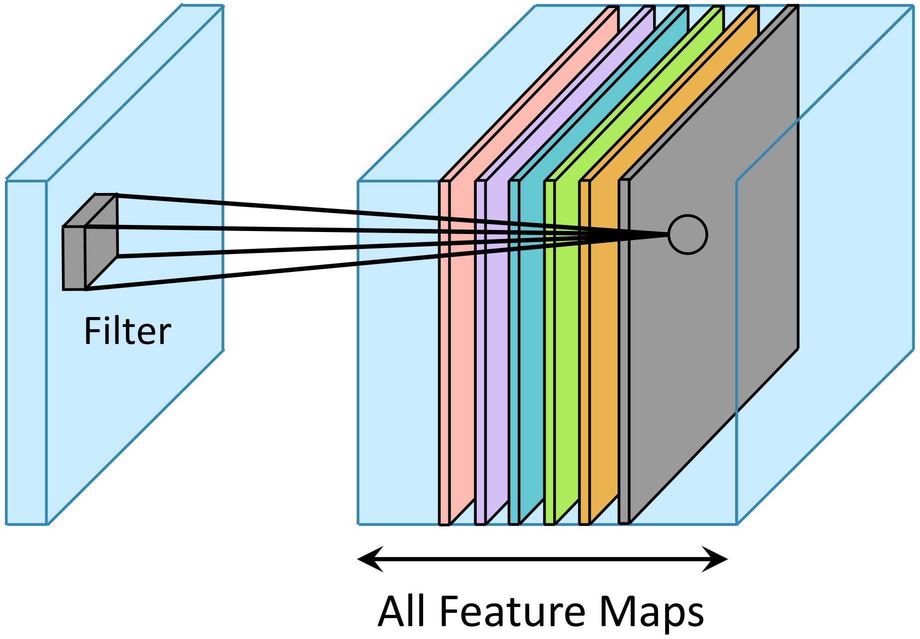
Convolutional Layer

Filter (Feature) Number

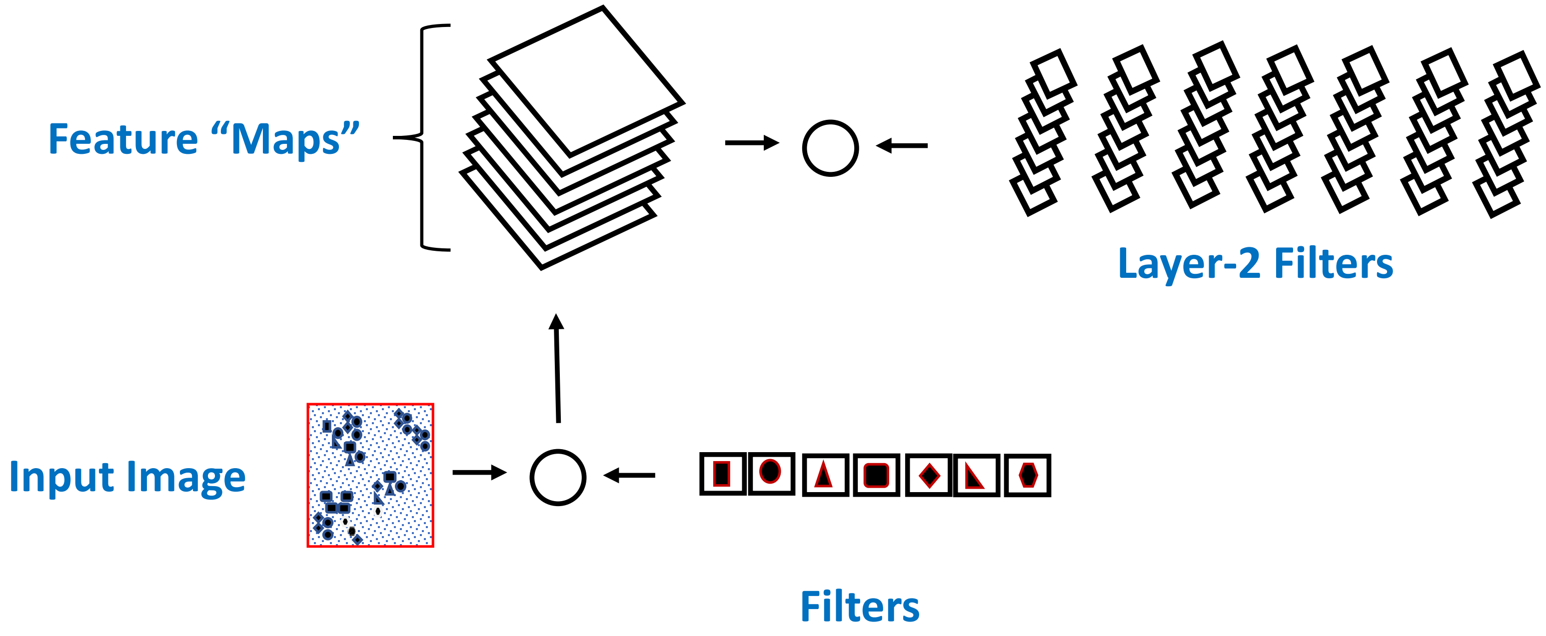


Convolutional Layer

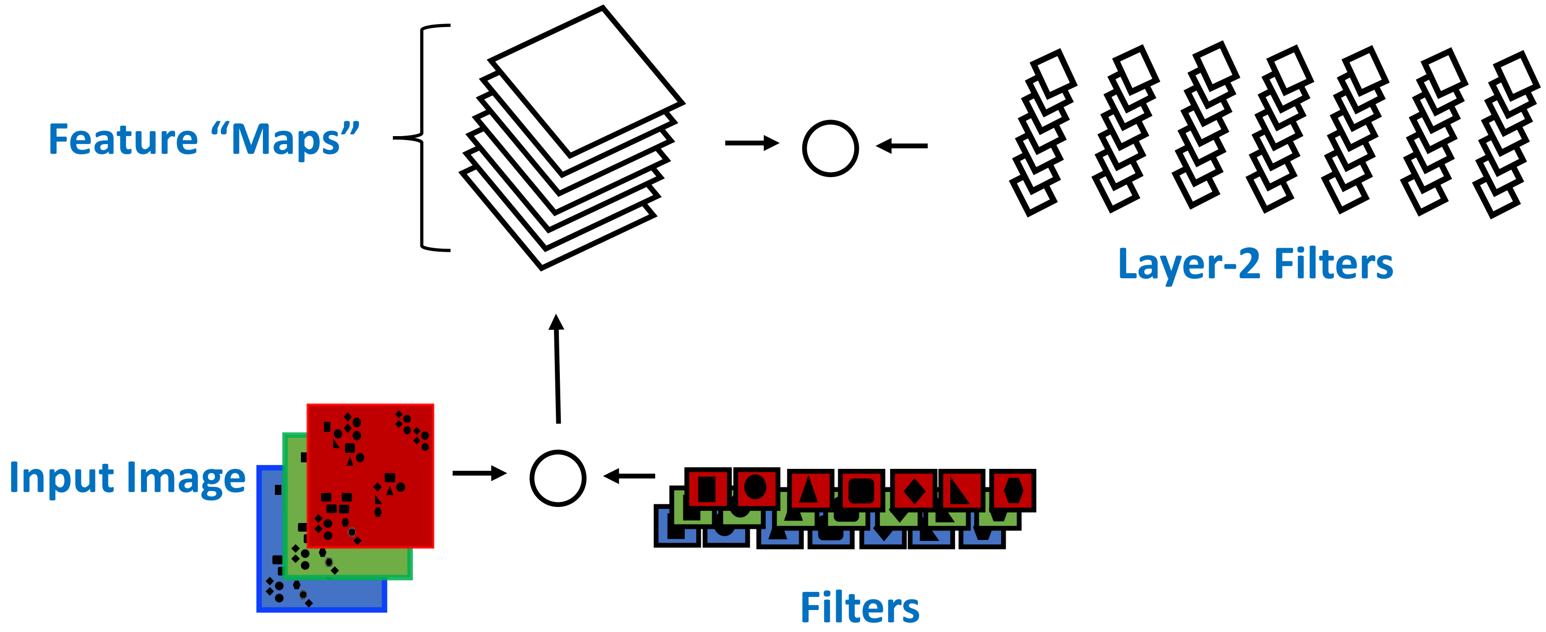
Filter (Feature) Number



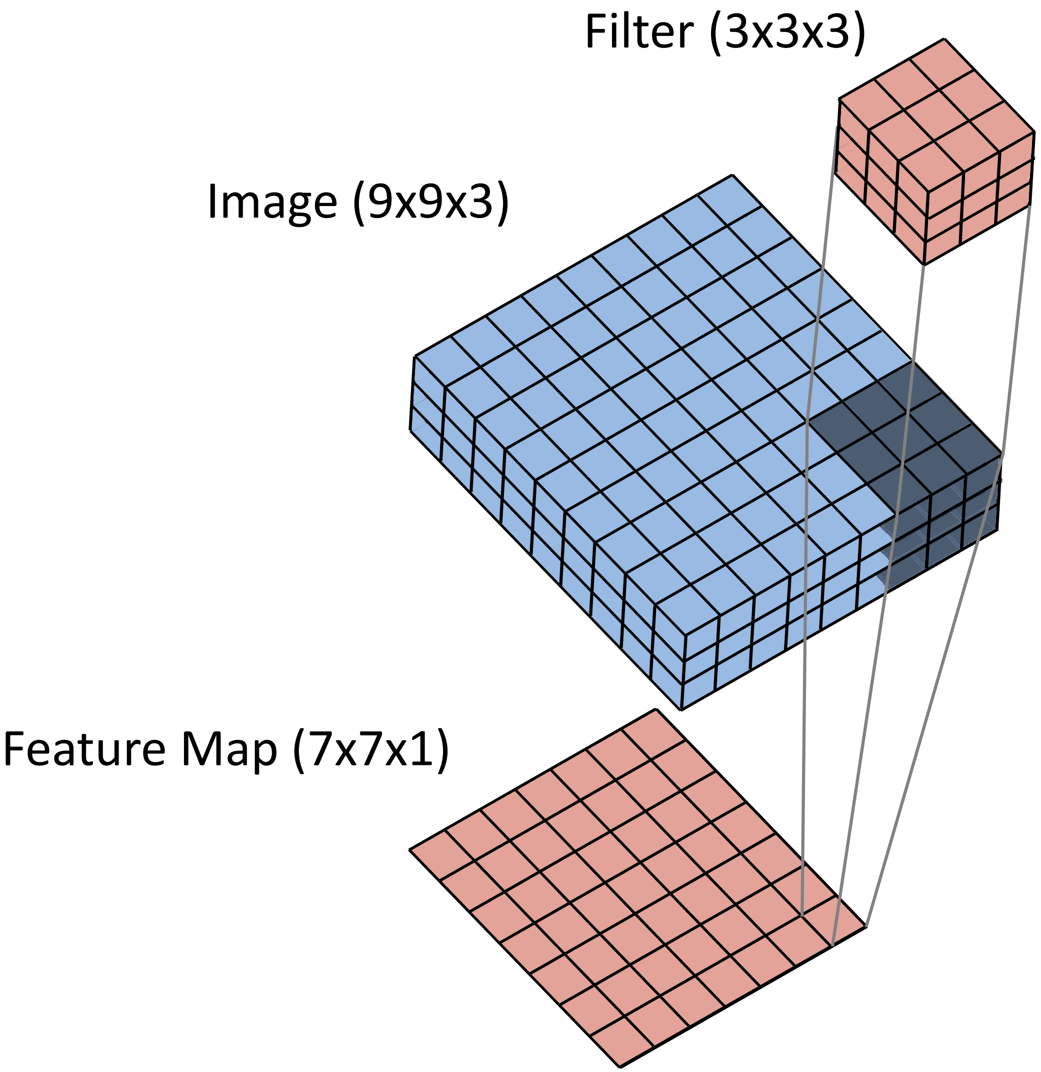
Filters Operate Over Input Volumes



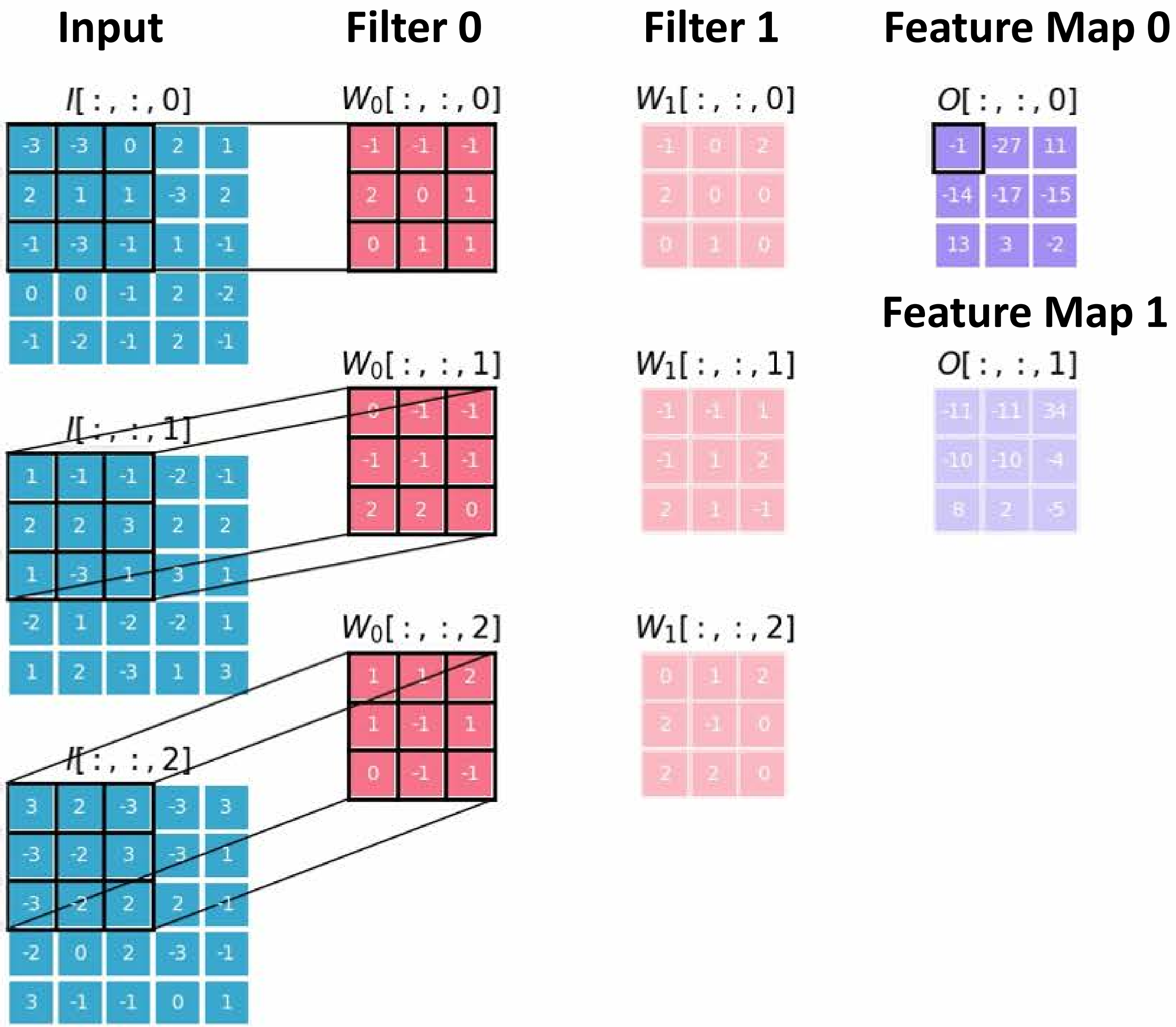
Filters Operate Over Input Volumes



Filters Operate Over Input Volumes

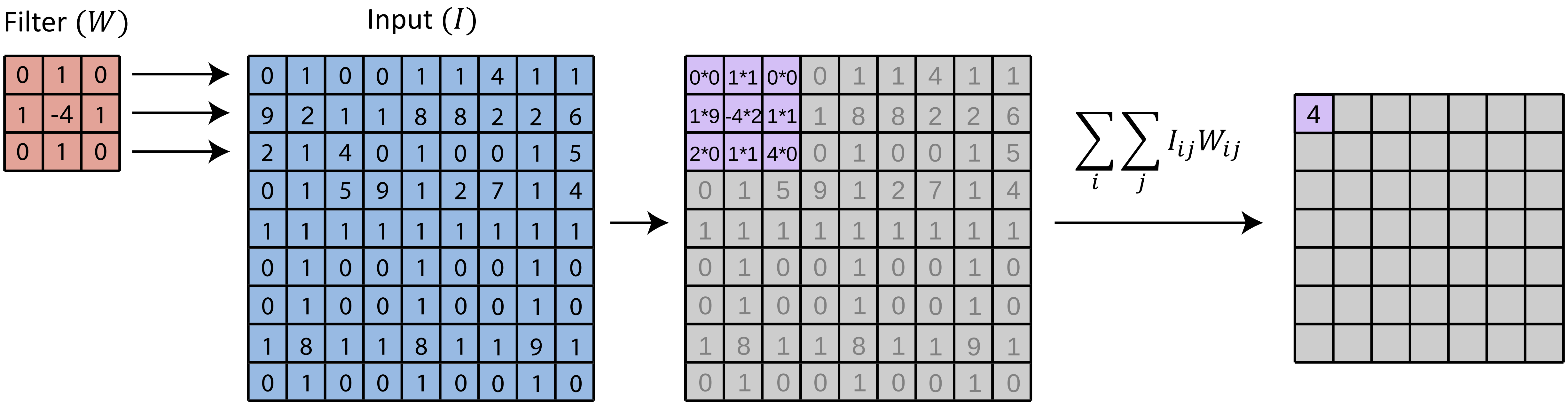


Convolutional Layer

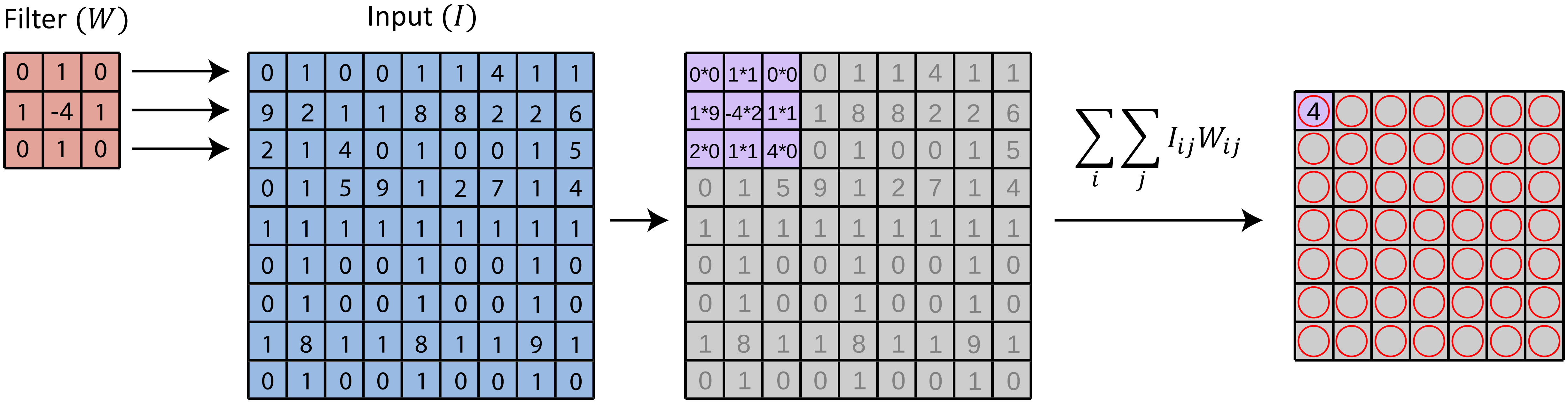


Inspired by Stanford 231n

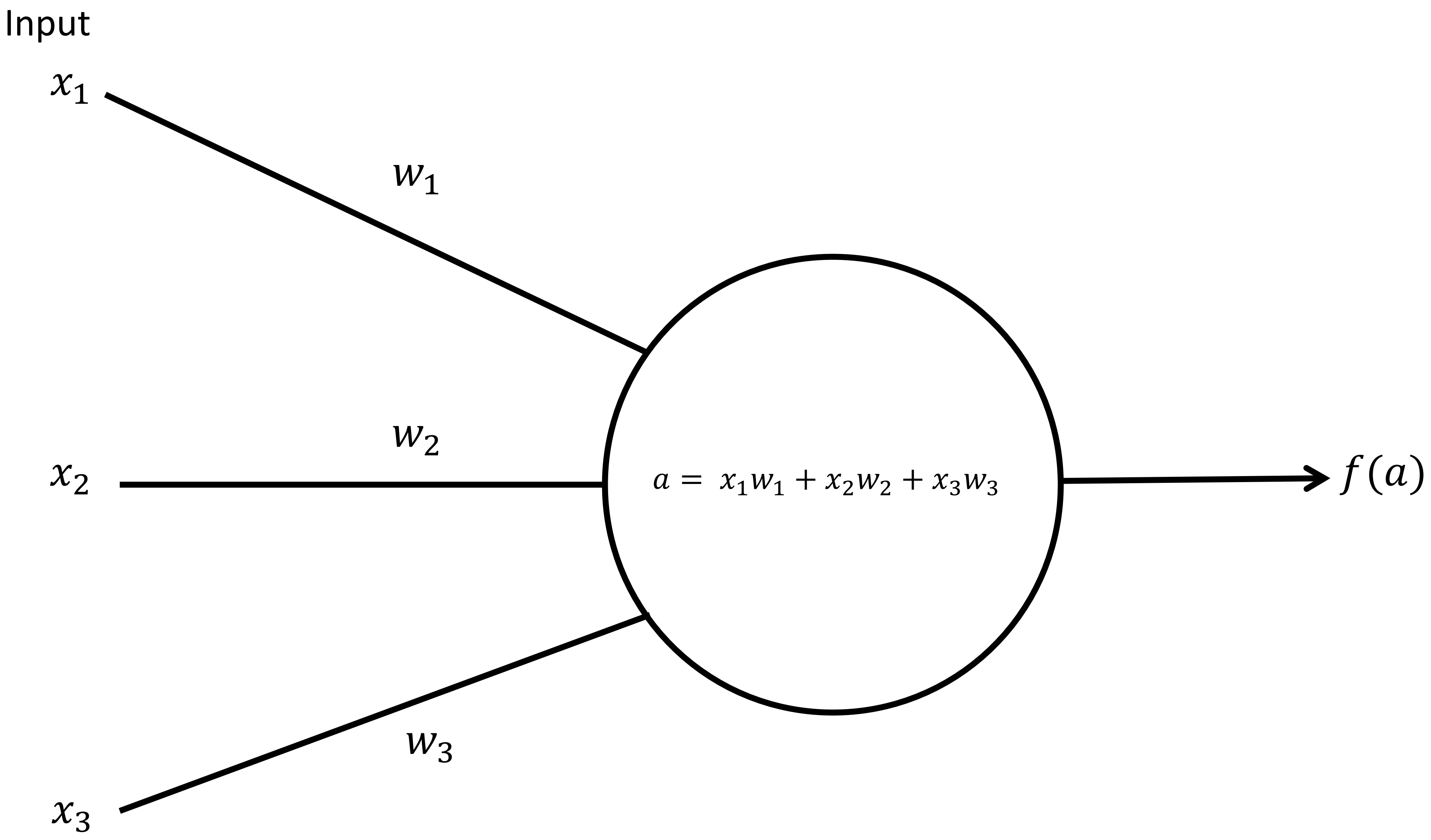
Activation Functions



Activation Functions

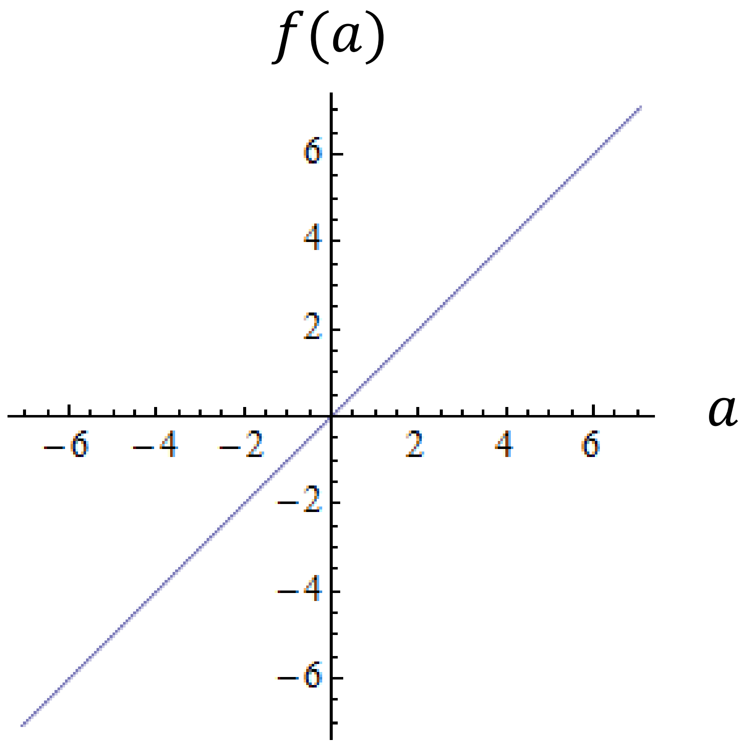


Activation Functions



Activation Functions

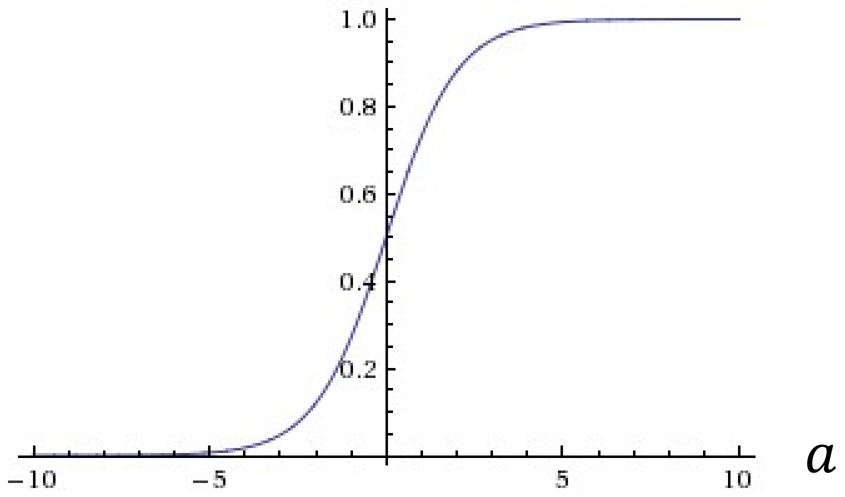
Linear Activation



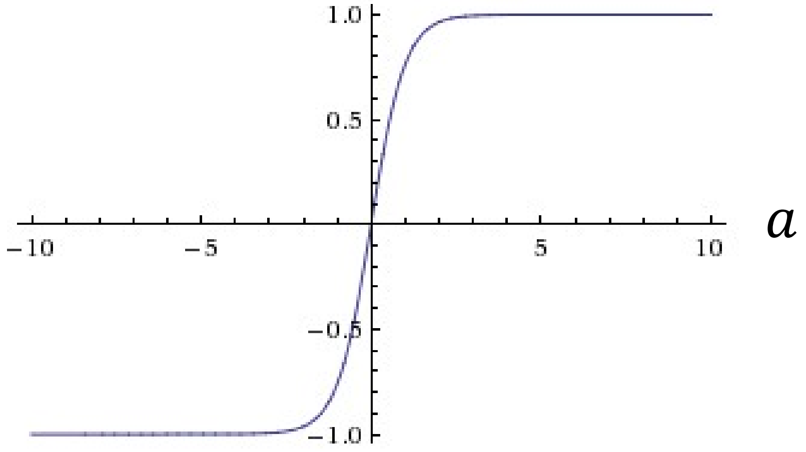
Activation Functions

Non-Linear Activations

$f(a) = \sigma(a)$



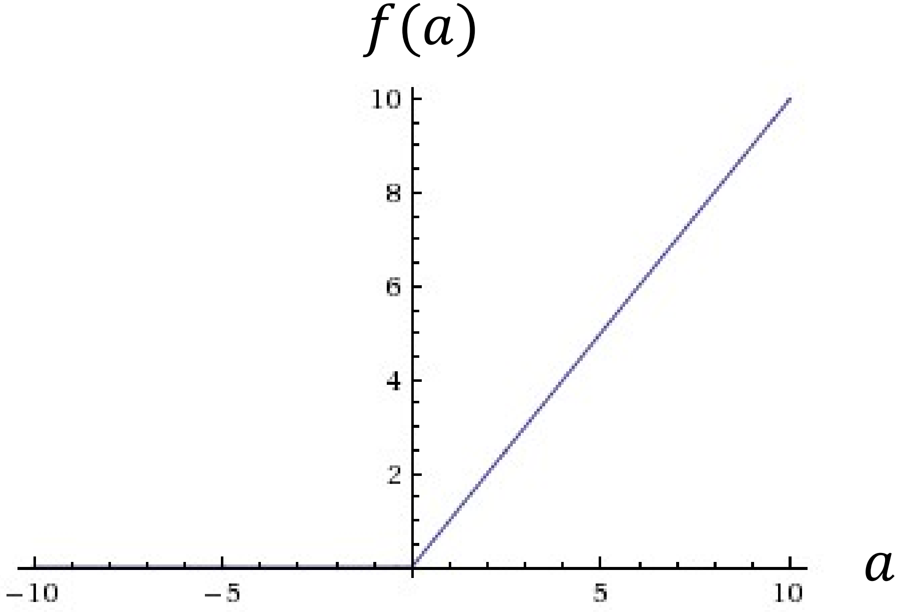
$f(a) = \tanh(a)$



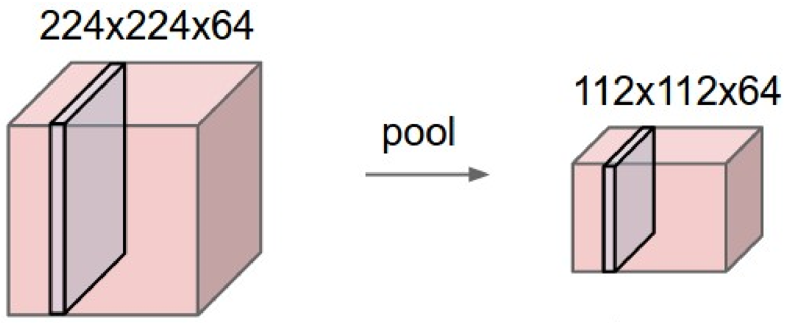
Non-linear activations increase the functional capacity of the neural network

Activation Functions

Non-Linear Activation:
Rectified Linear Unit (ReLU)

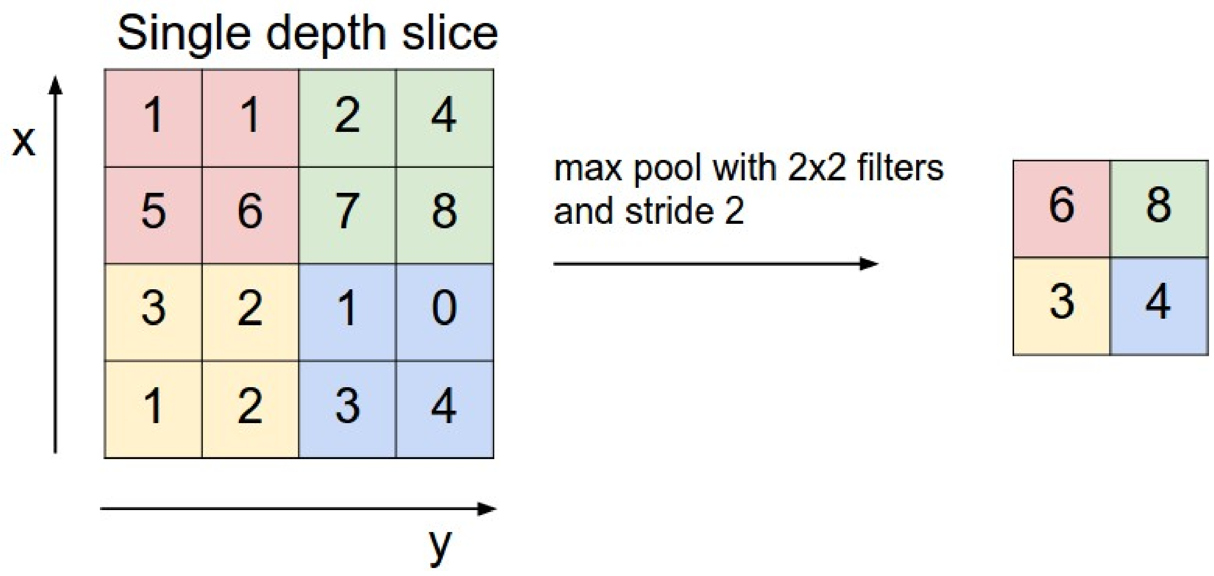


Pooling Layer



- Reduces computational complexity
- Combats overfitting
- Encourages translational invariance

Pooling Layer



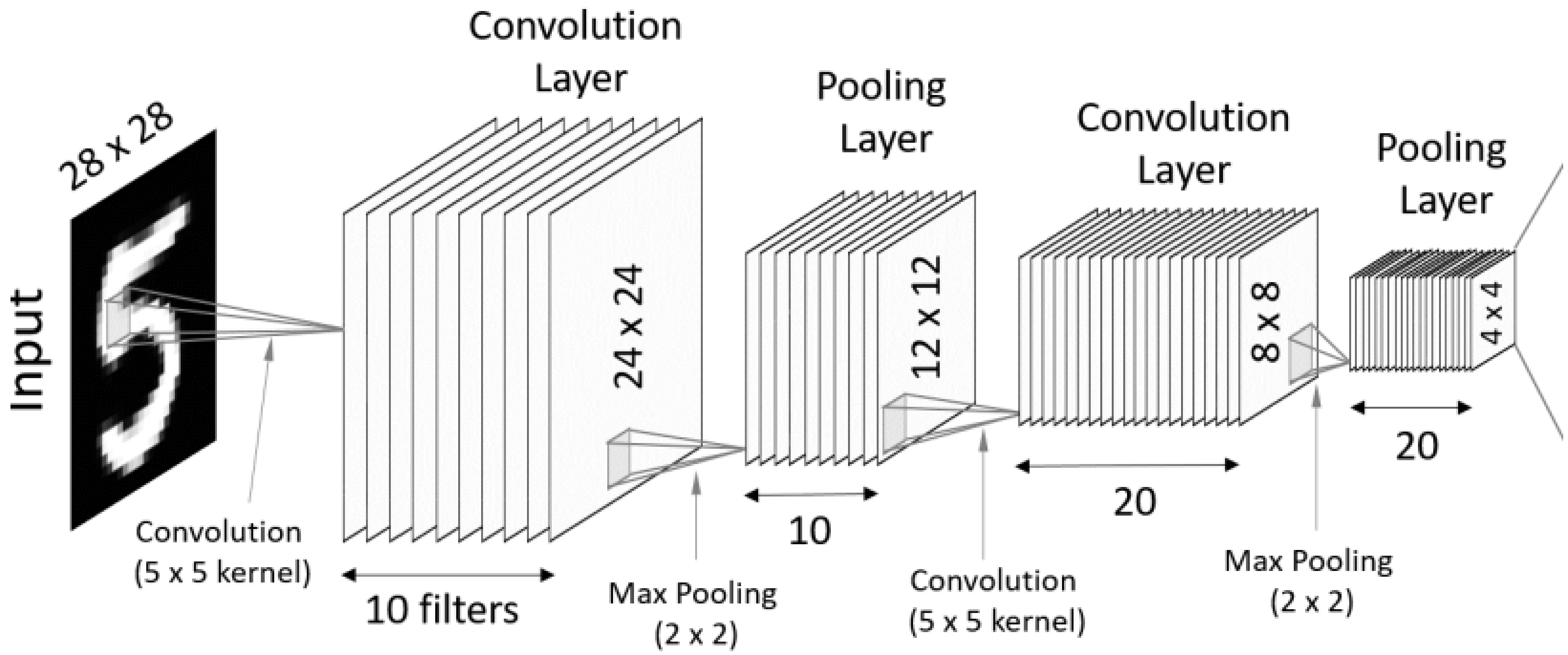
Pooling layers also have **width** and **stride**.

Pooling is typically done by taking the **max** or **mean** across the pooling area

Fully Connected Layer

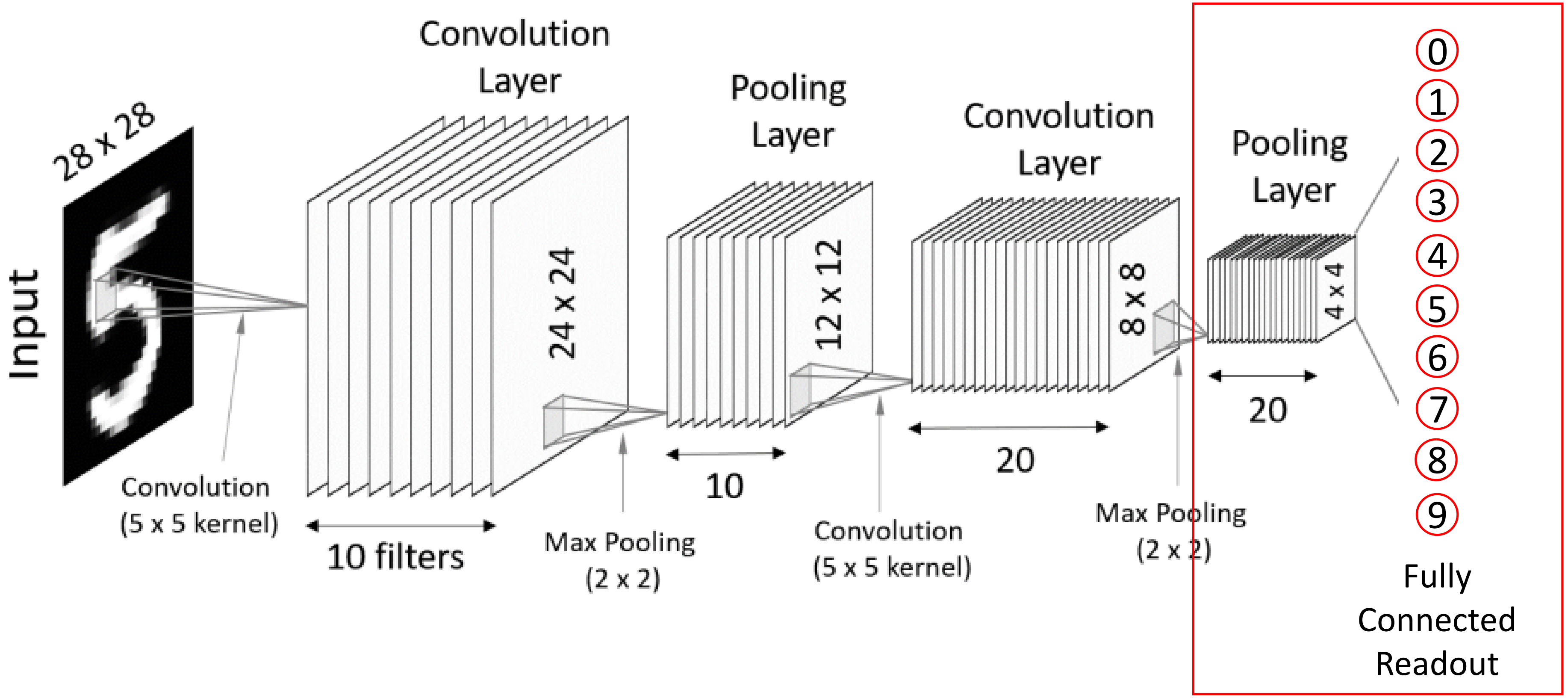
Convolutional and pooling layers are stacked to build up high-level feature representations

How are these high-level features processed to arrive at a final classification?



Fully Connected Layer

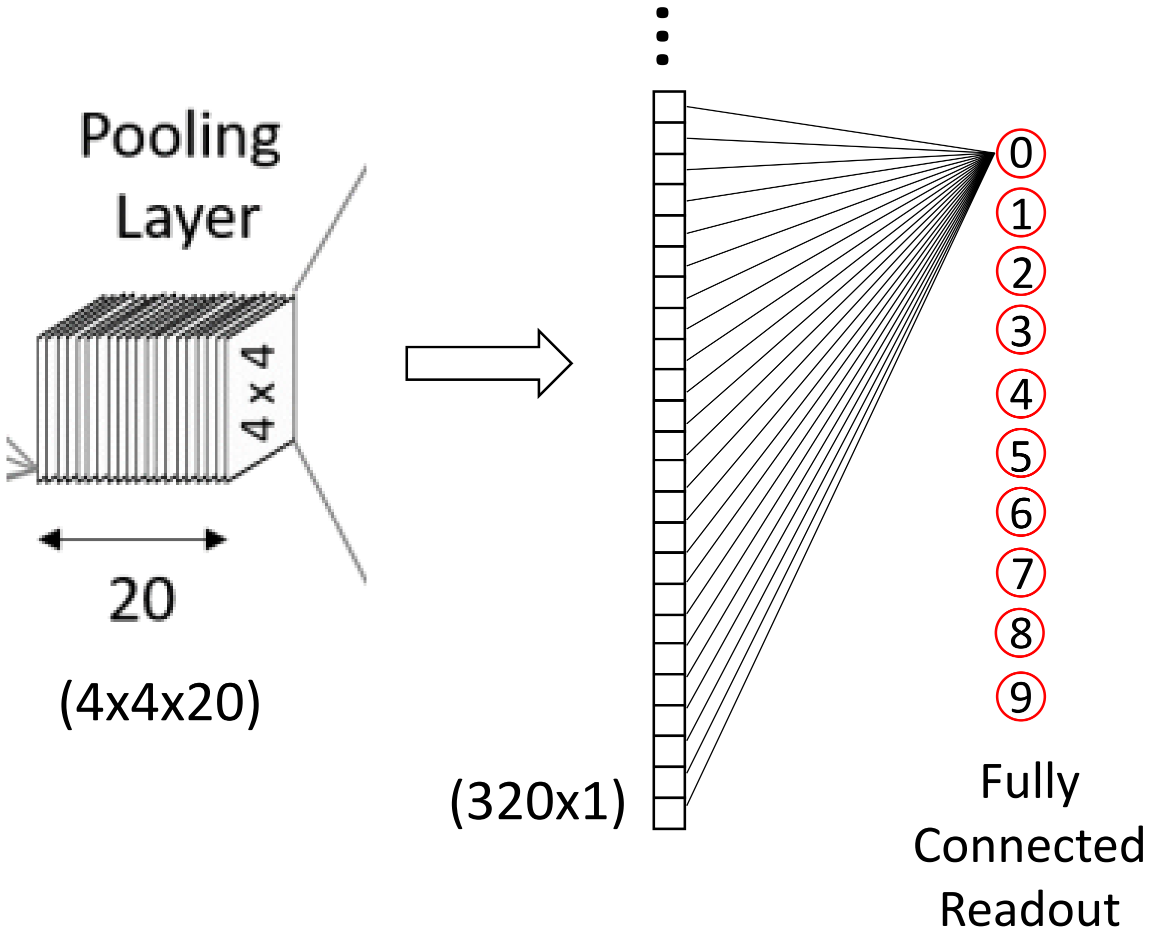
The most basic way is to have a final **fully connected** readout layer with as many neurons as there are classes



Fully Connected Layer

Fully connected means each neuron takes input from all neurons in the final set of feature maps

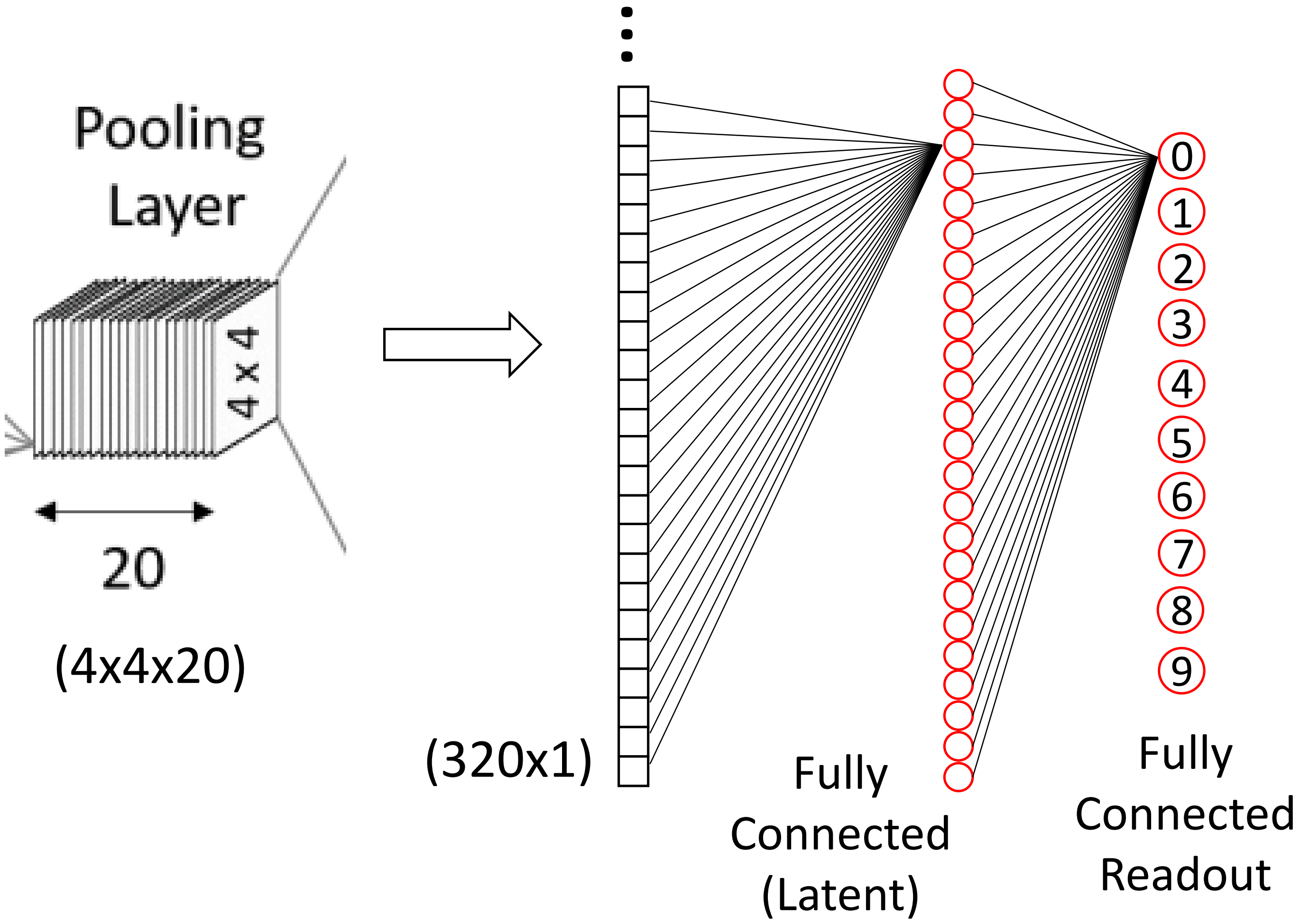
The final set of feature maps are vectorized to create an MLP-like configuration



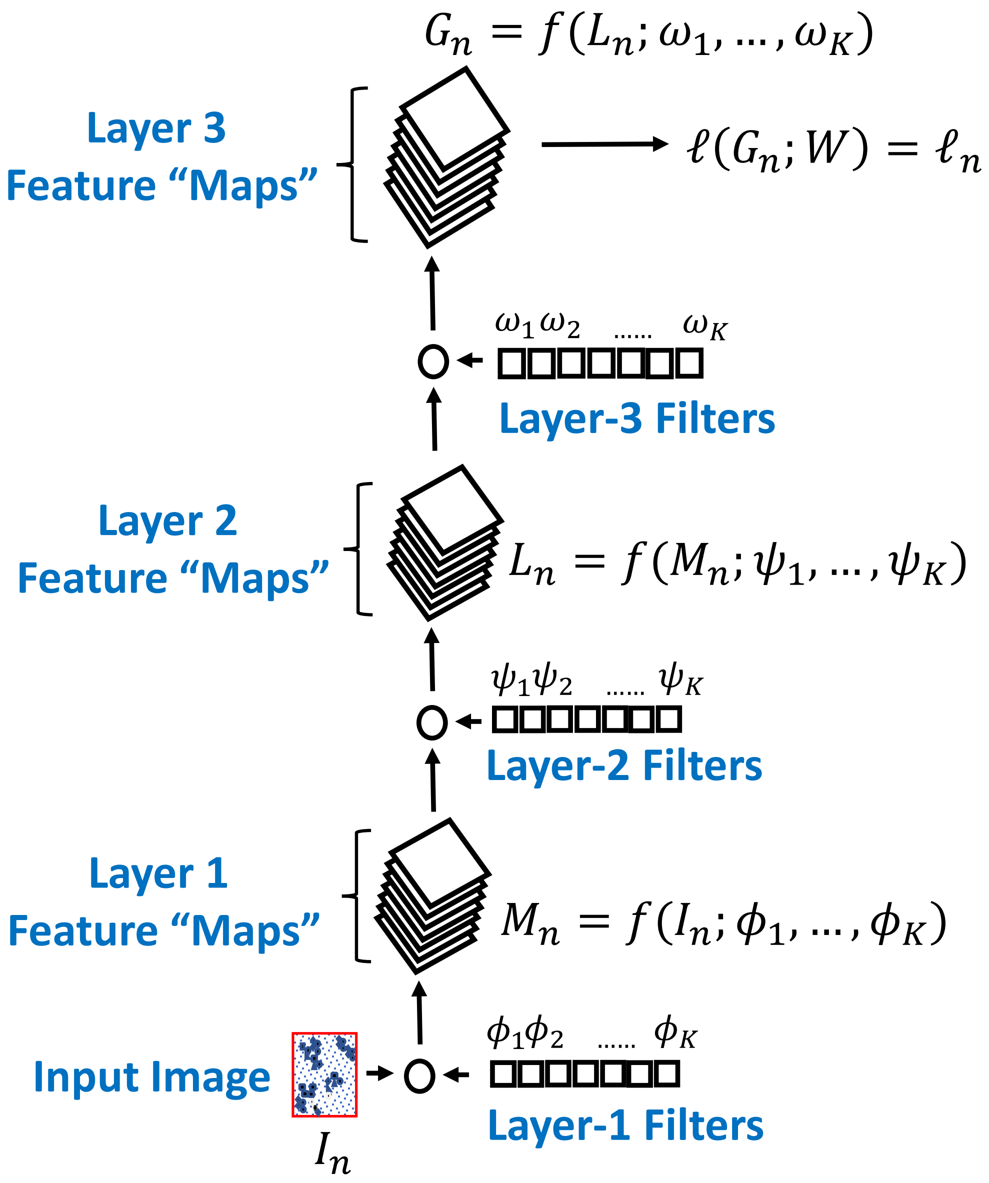
Fully Connected Layer

But it is also common to stack multiple fully connected layers before the final readout layer

Neurons in these intermediate layers can be considered **latent** classes



Review: Training A Deep Convolutional Neural Network

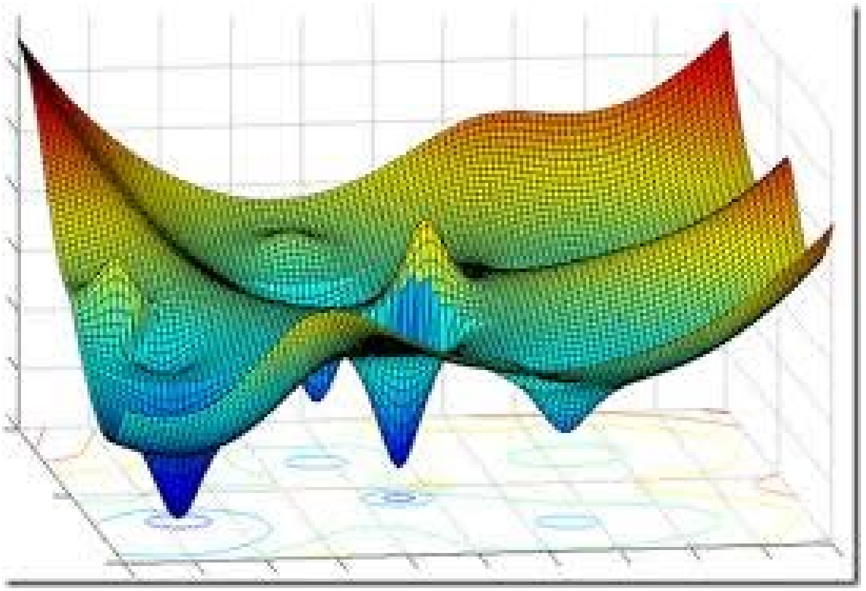


- Assume we have labeled images $\{I_n, y_n\}_{n=1,N}$
- I_n is image n, $y_n \in \{+1, -1\}$ is associated label
- Risk function of model parameters:

$$E(\Phi, \Psi, \Omega, W) = 1/N \sum_{n=1}^{N} loss(y_n, \ell_n)$$

- Find model parameters $\widehat{\Phi}, \widehat{\Psi}, \widehat{\Omega}, \widehat{W}$ that minimize $E(\Phi, \Psi, \Omega, \mathrm{W})$

Cost Function vs. Model Parameters



- High-dimensional function, as a consequence of a large number of model parameters
- Typically many local minima
- May be expensive to compute, for sophisticated models & large quantity of training images

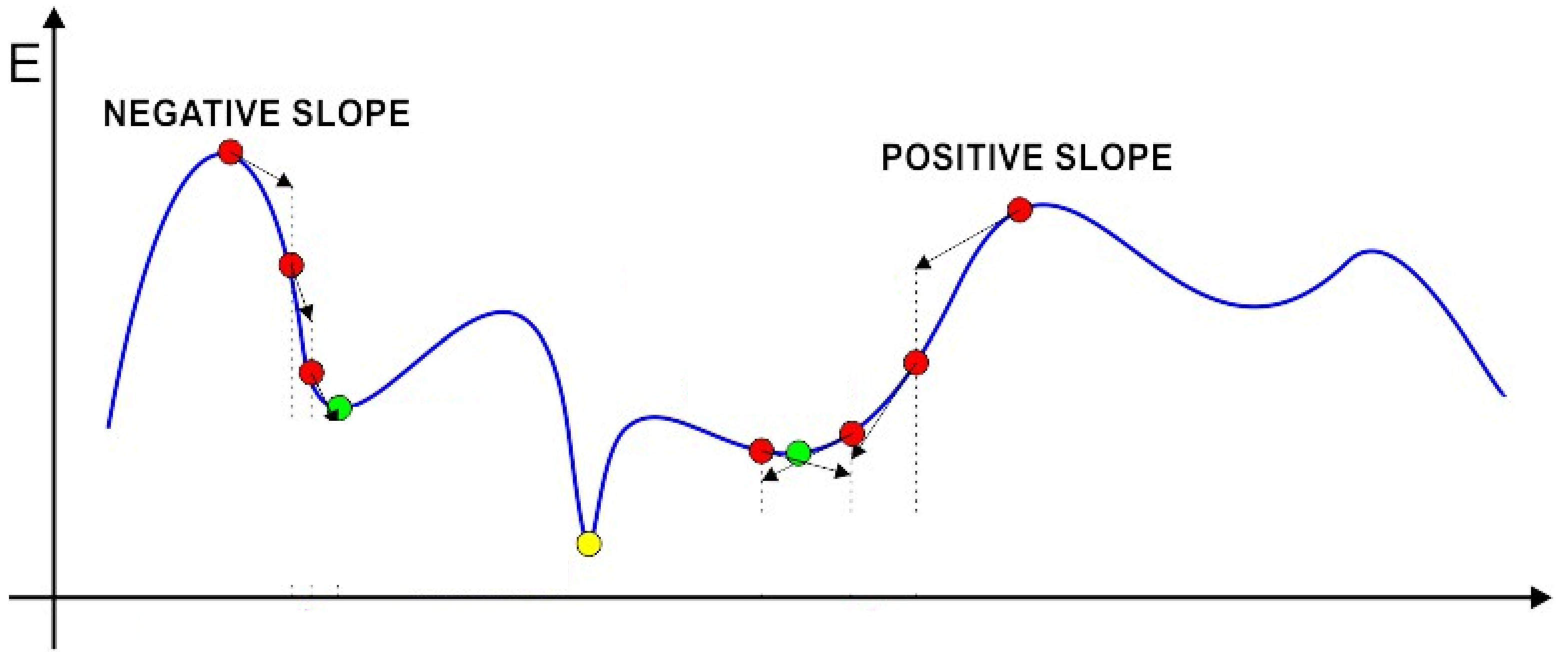
Gradient Descent

$$E(\Phi, \Psi, \Omega, W) = 1/N \sum_{n=1}^{N} loss(y_n, \ell_n)$$

- For large number of training pairs *N*, computation of risk $E(\cdot)$ could be prohibitive

- How do we optimize for Φ, Ψ, Ω, W?

Optimization for $\mathbf{\Theta} = \{\mathbf{\Phi}, \mathbf{\Psi}, \mathbf{\Omega}, \mathbf{W}\}$?



Gradient Descent

$$\Theta_t = \Theta_{t-1} - \alpha \underbrace{\nabla_\Theta E(\Theta_t)}_{\text{Multi-dimensional "slope"}}$$

Massive N ?

Gradient Descent

$$\Theta_t = \Theta_{t-1} - \alpha \nabla_\Theta E(\Theta_t)$$

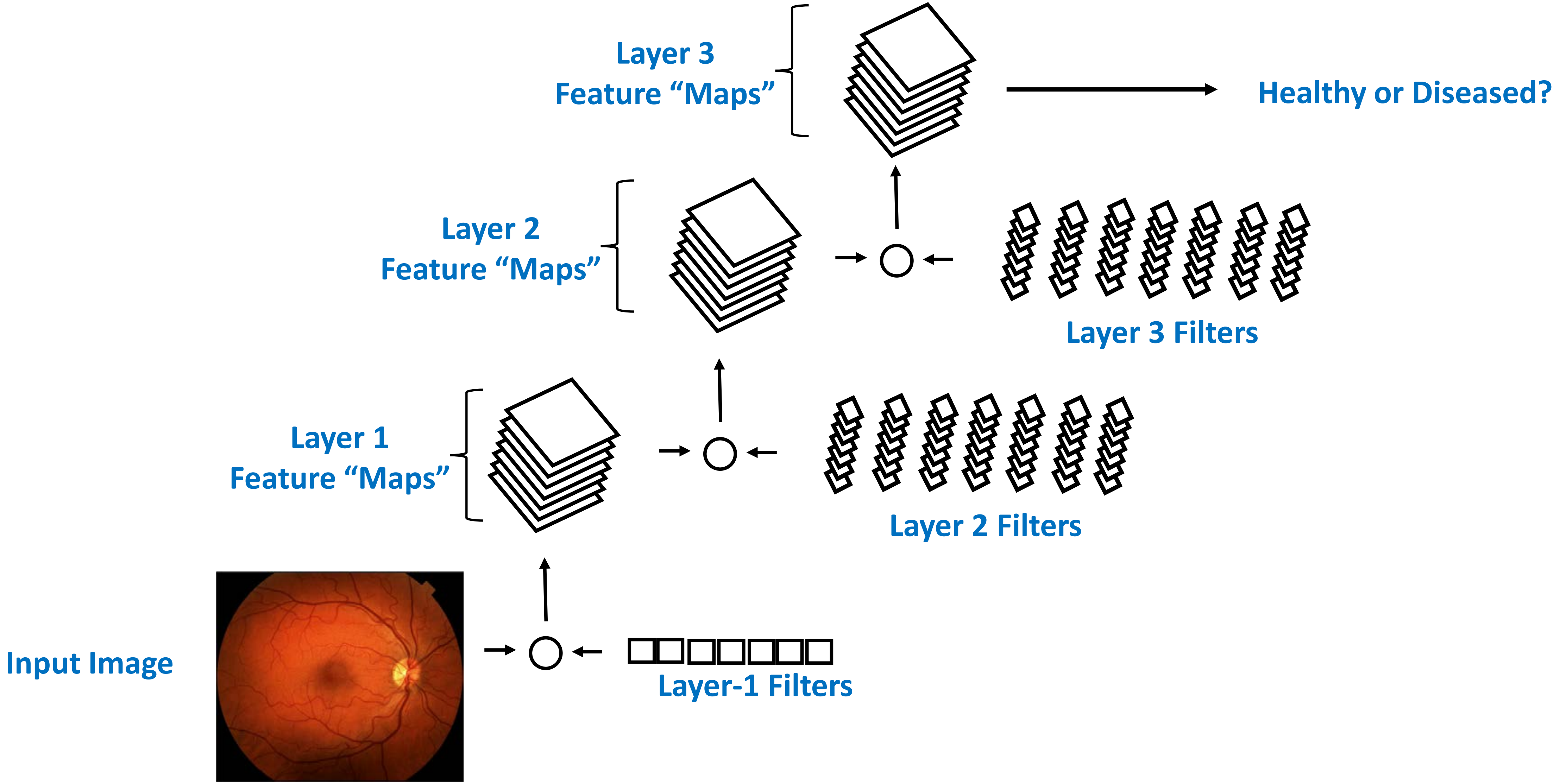
Stochastic Gradient Descent

$$\Theta_t = \Theta_{t-1} - \alpha \nabla_\Theta \hat{E}_t(\Theta_t)$$

$$\hat{E}_t(\Phi, \Psi, \Omega, W) = 1/|S_t| \sum_{n \in S_t} loss(y_n, \ell_n)$$

S_t a *random* subset of data, at iteration *t*

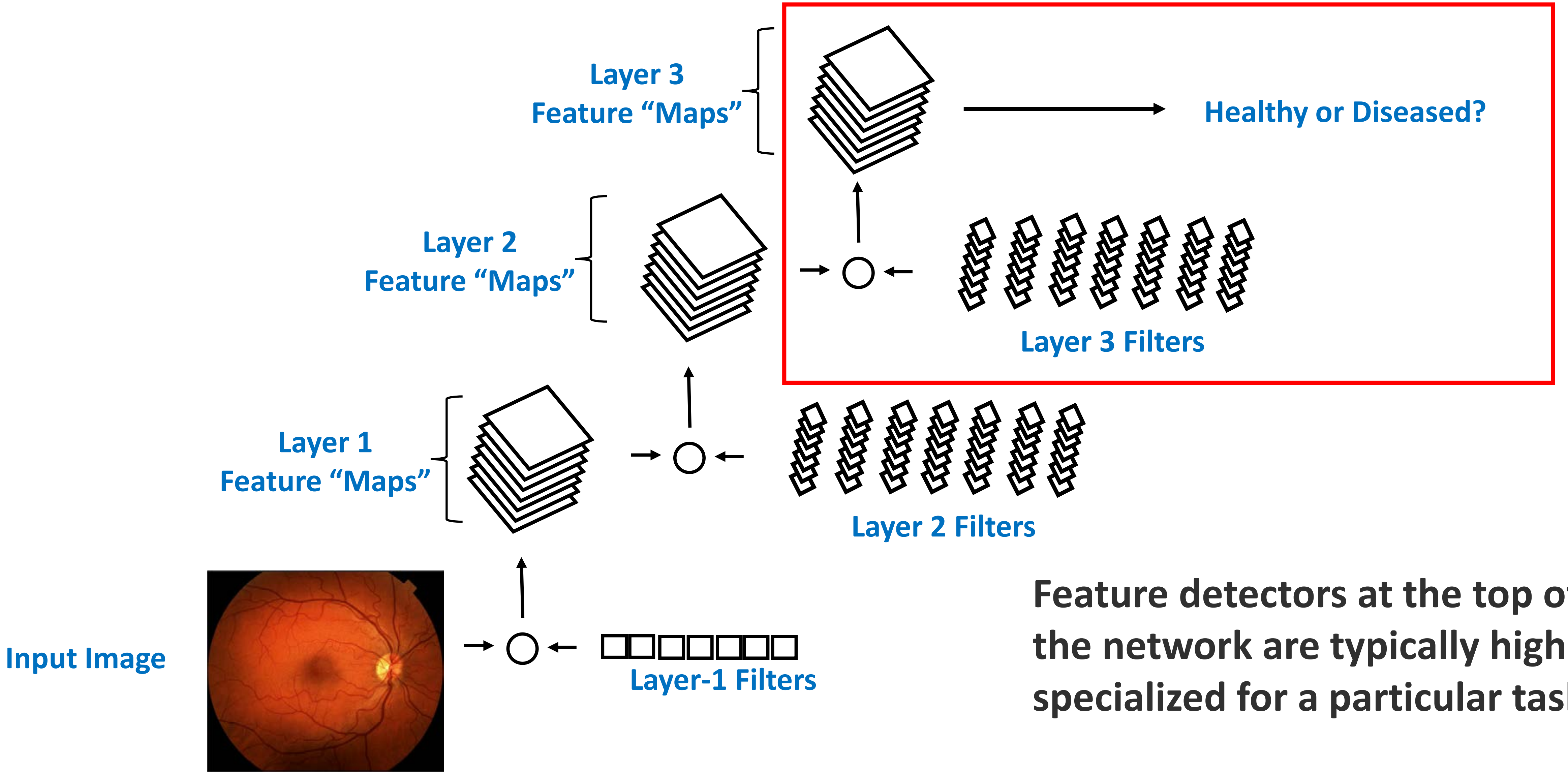
Transfer Learning



Transfer Learning

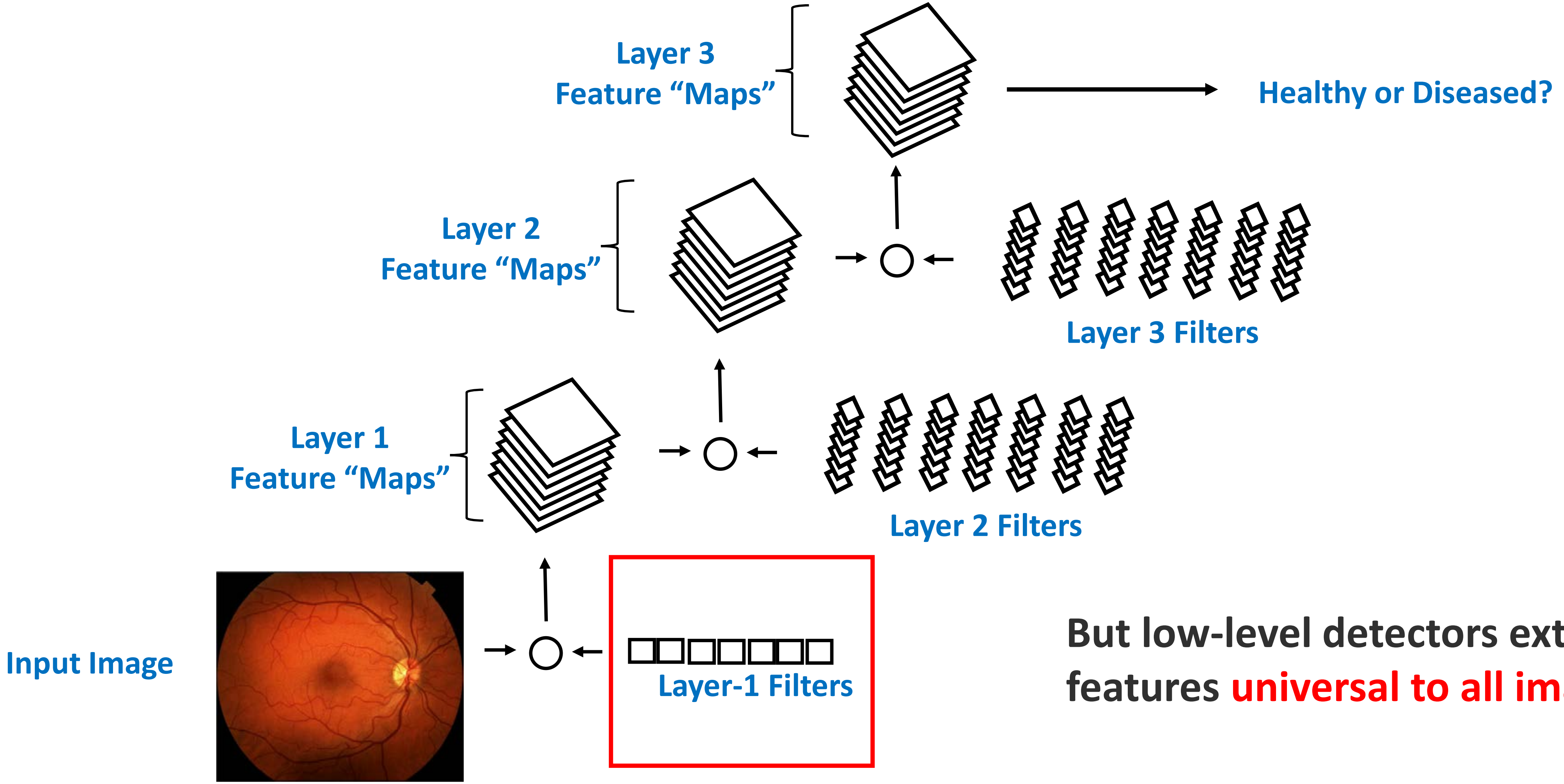
"To speed up the training, batch normalization as well as preinitialization using weights from the same network trained to classify objects in the ImageNet data set were used. Preinitialization also improved performance."

Transfer Learning



Feature detectors at the top of the network are typically highly specialized for a particular task

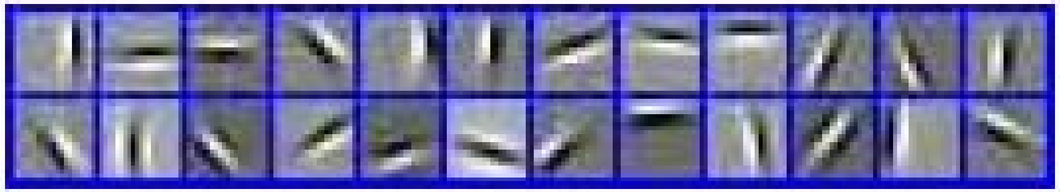
Transfer Learning



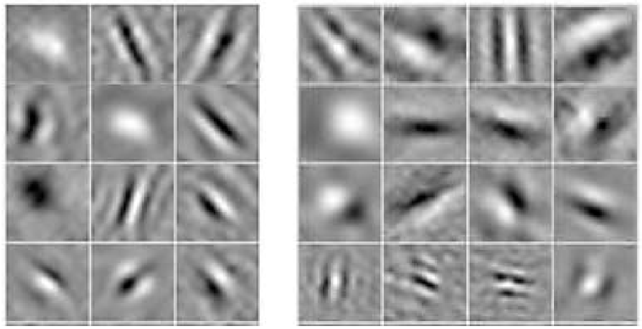
But low-level detectors extract features **universal to all images**

Transfer Learning

Layer 1 Filters,
Convolutional Neural Network



Neuron Receptive Fields,
Macaque Visual Cortex



Summary

- Convolutional neural networks learn to recognize **high-level structure** in images by building **hierarchical representations of features**

- Features are extracted via spatial convolutions with **filters**

- Filters are learned via iterative minimization of a risk function

- Convolutional neural networks have shown capabilities beyond human performance for image analysis