ECE 20875

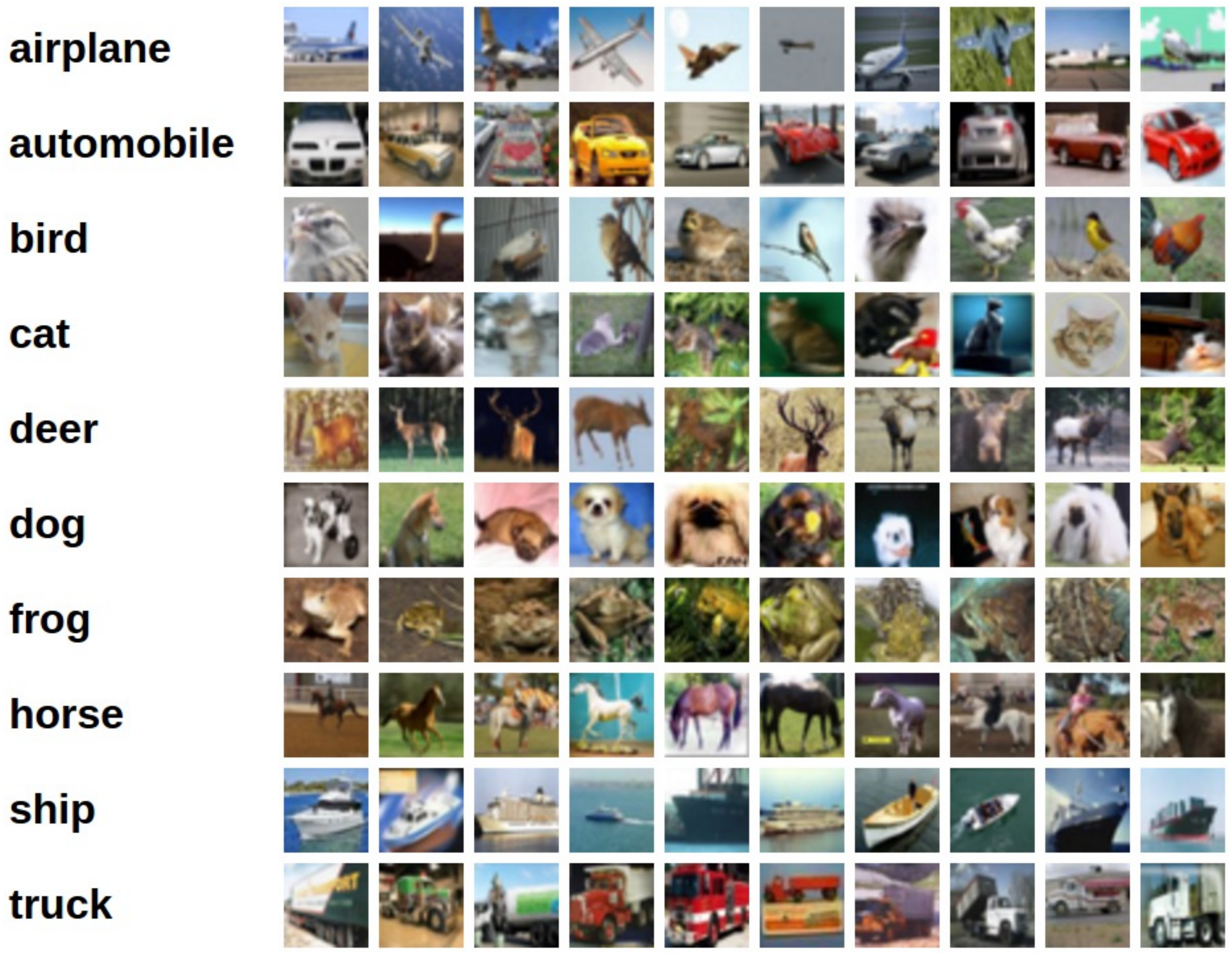
Python for Data Science

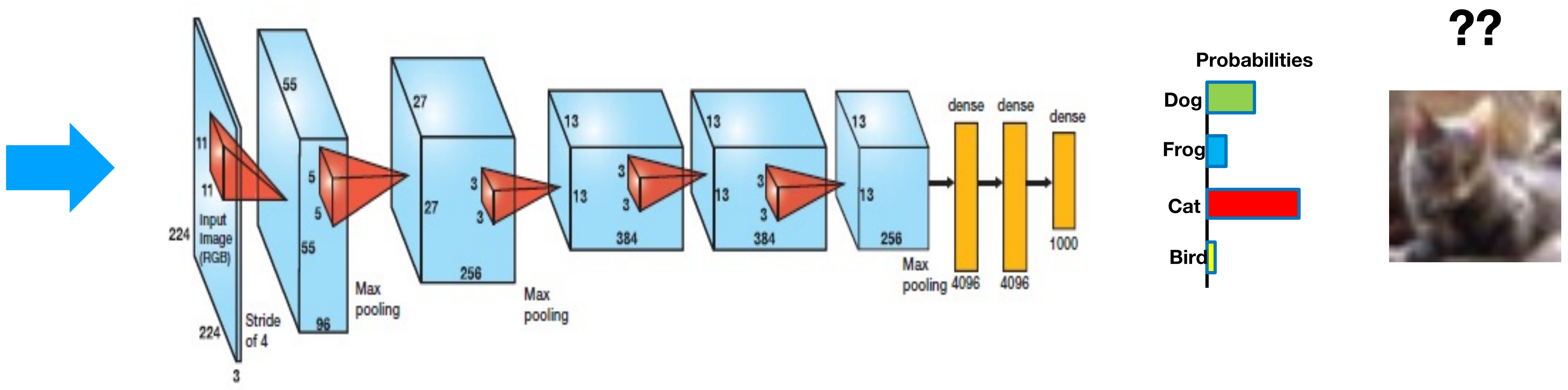
Chris Brinton, Qiang Qiu, and Mahsa Ghasemi

(Adapted from material developed by Profs. Milind Kulkarni, Stanley Chan, Chris Brinton, David Inouye, and Qiang Qiu)

convolutional neural network (CNN)

image classification using NN





Overall procedure:

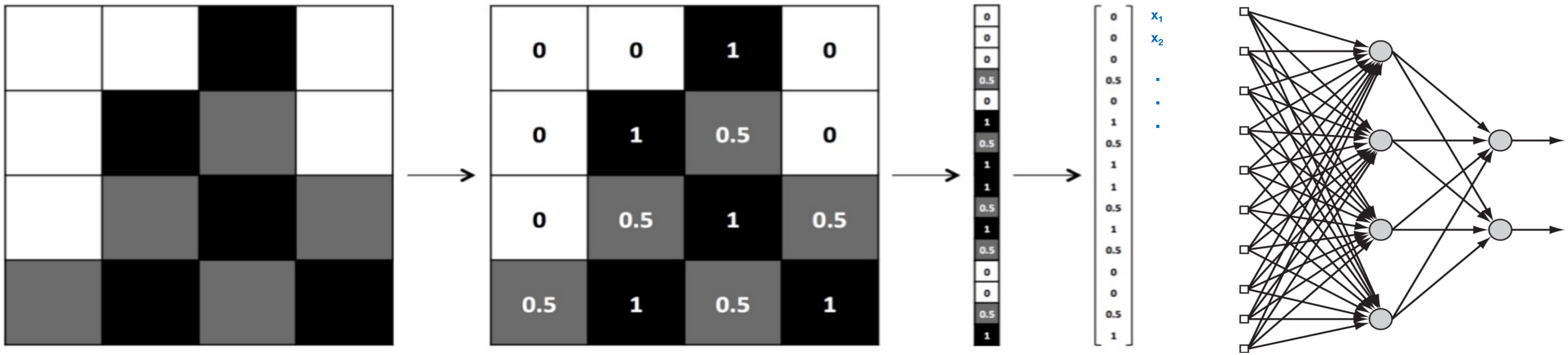
- Collect a large amount of images
- Annotate each image with a class label
- Choose a neural network structure
- Use our training samples to adjust layers and layers of parameters (backpropagation) to minimize a chosen loss

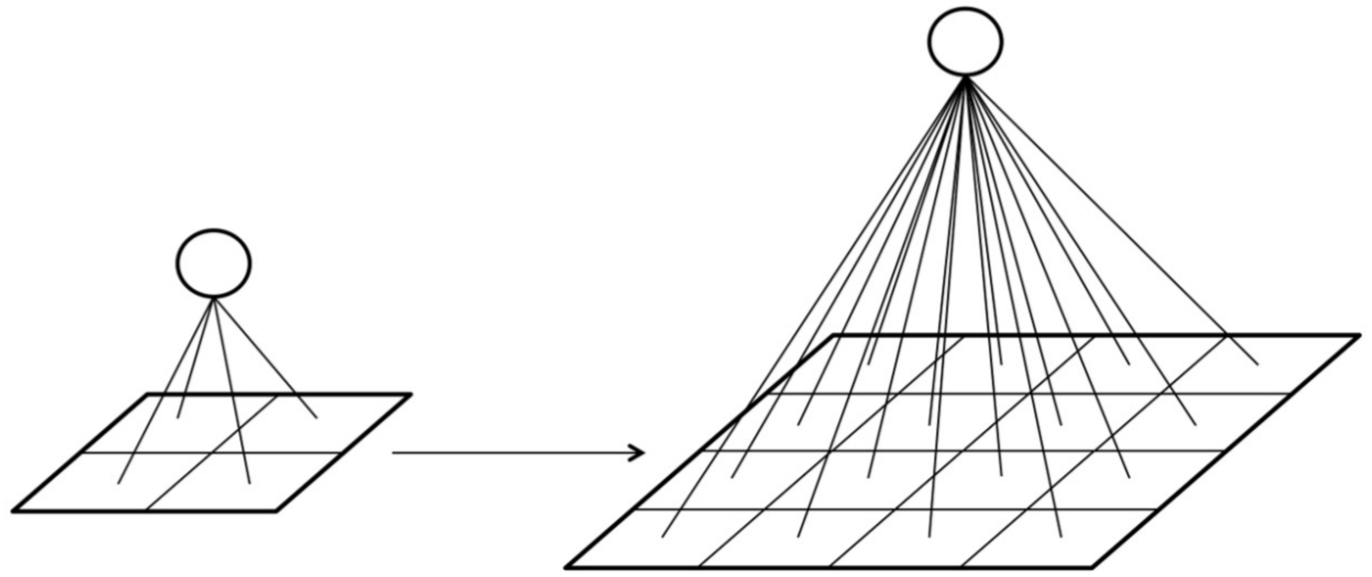
Goal: After training, when the NN sees a new image, its output should assign the highest probability to the respective class.

image vectorization

Image

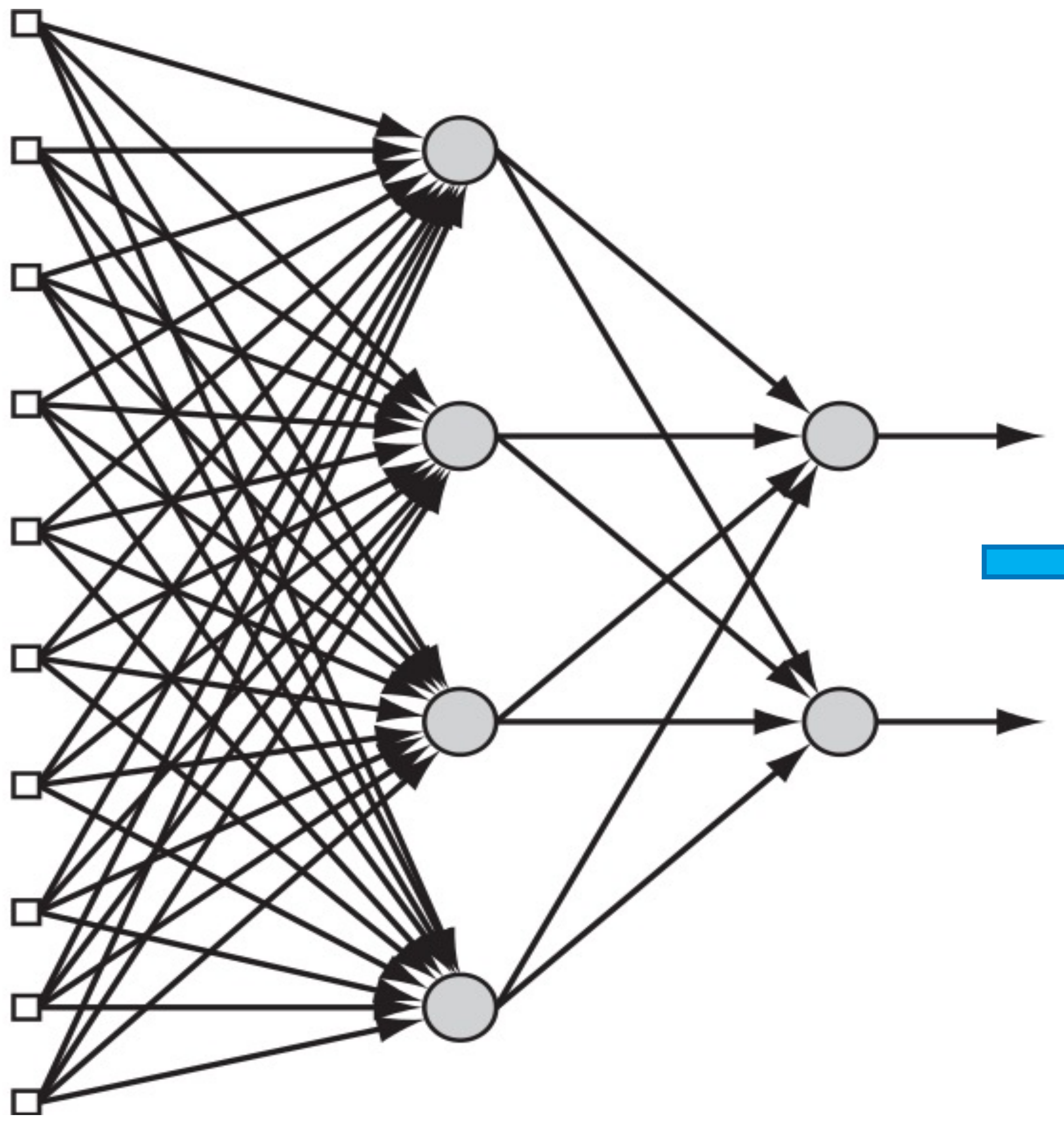
Fully-connected



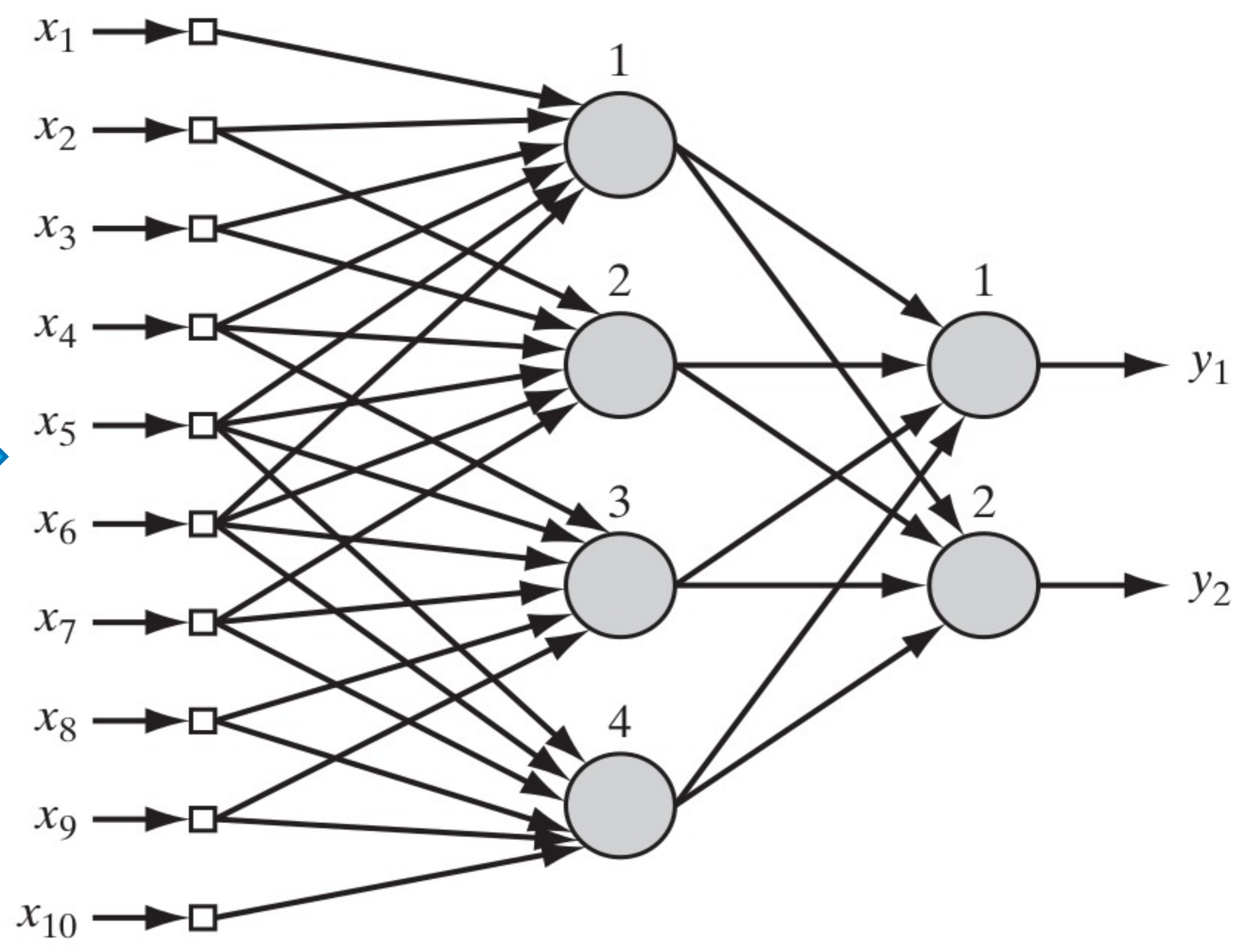


Challenge: The density of connections between layers increases intractably as the size of the image increases!!

local connections and weight sharing



Fully-connected

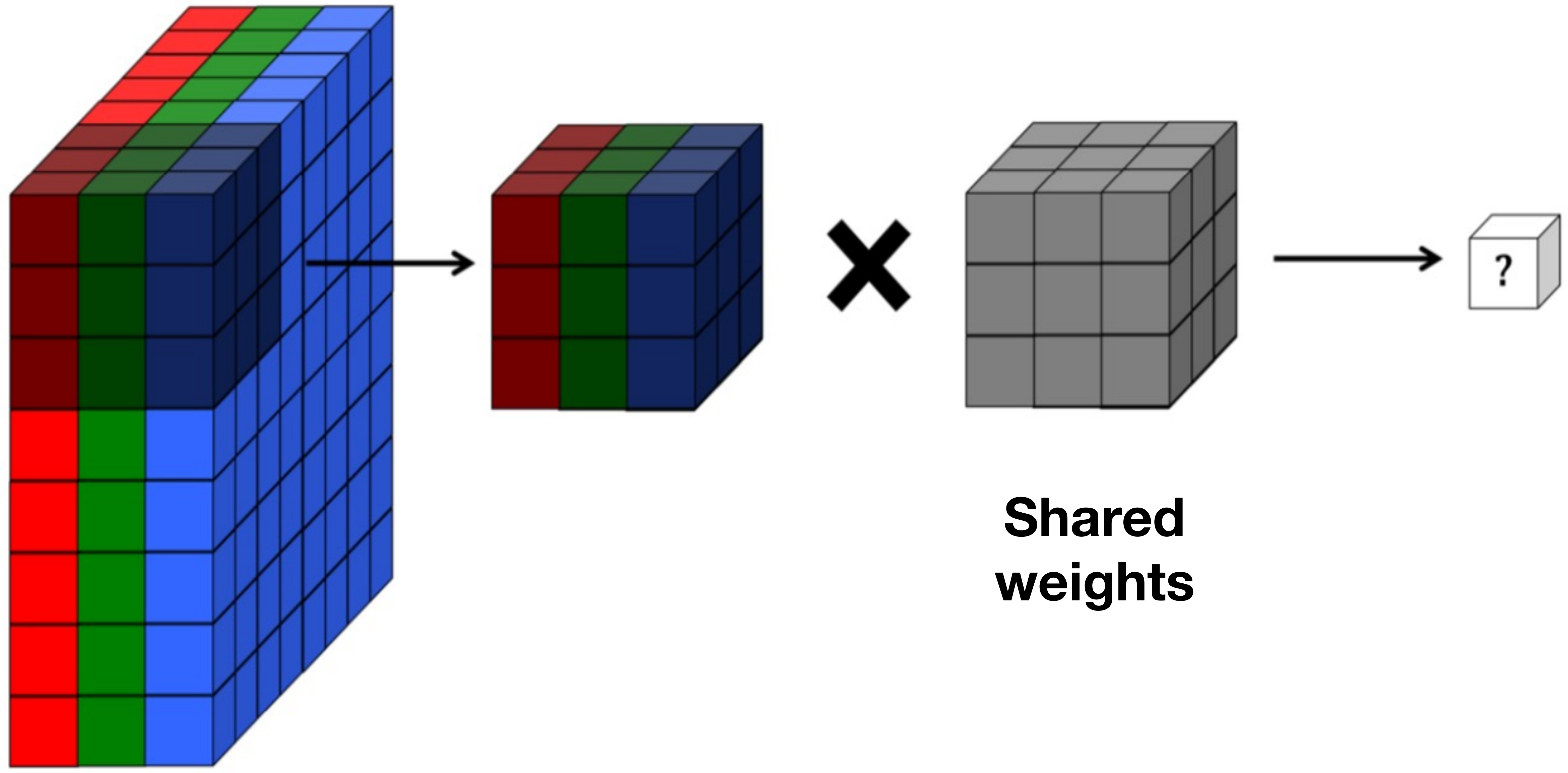


Now, all four hidden neurons
1. Share the same set of 6 weights
2. Use local connections (receptive fields)

An effective way to reduce model parameters:

- (**Local connection**) Each neuron only processes inputs from a local region
- (**Weight sharing**) Neurons within the same layer can share weights

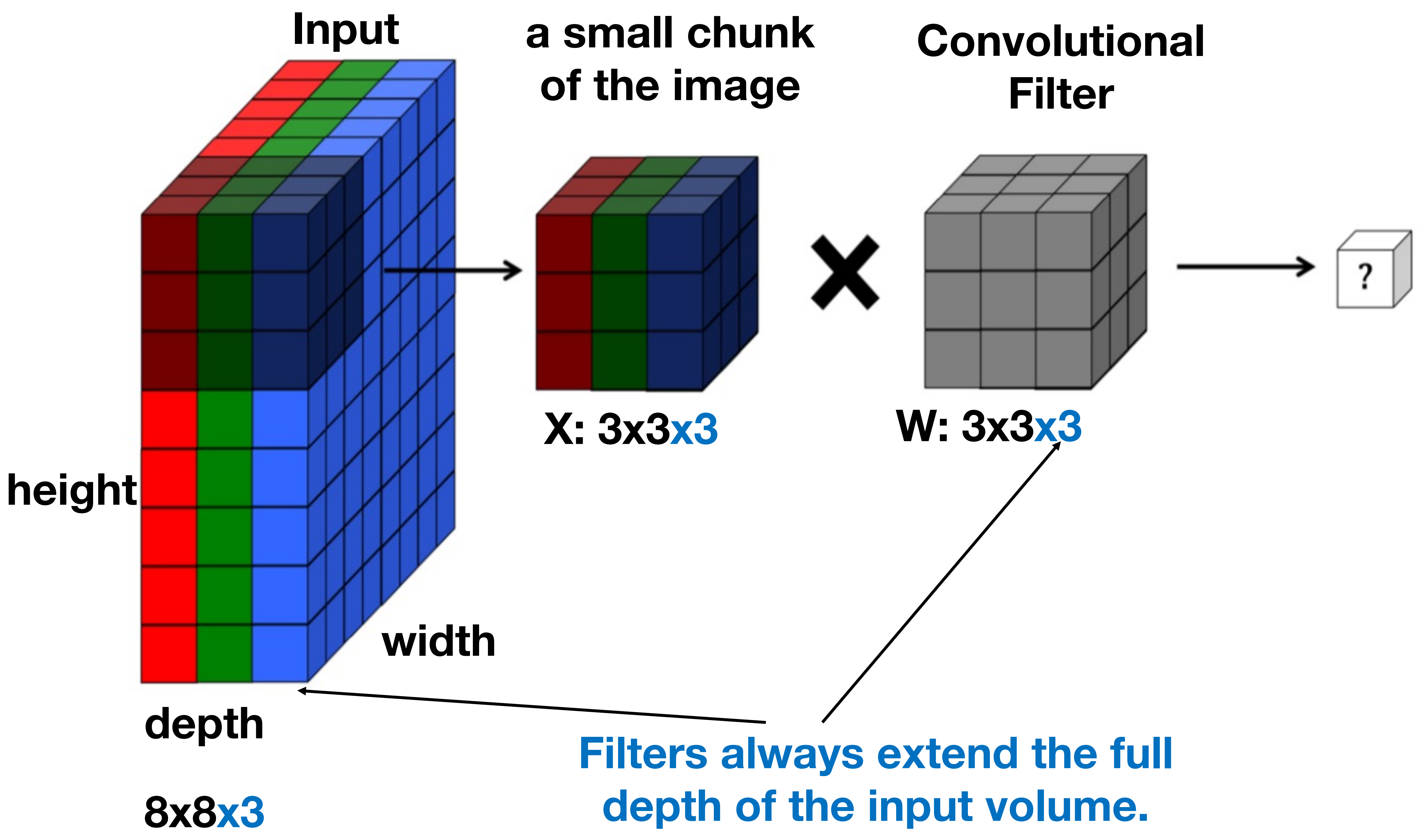
a convolution view



Another view to local connection and weight sharing:

- Convolve (slide) a block of shared weights over all spatial locations
- At each spatial location, output one value (computing dot products)

convolutional filters

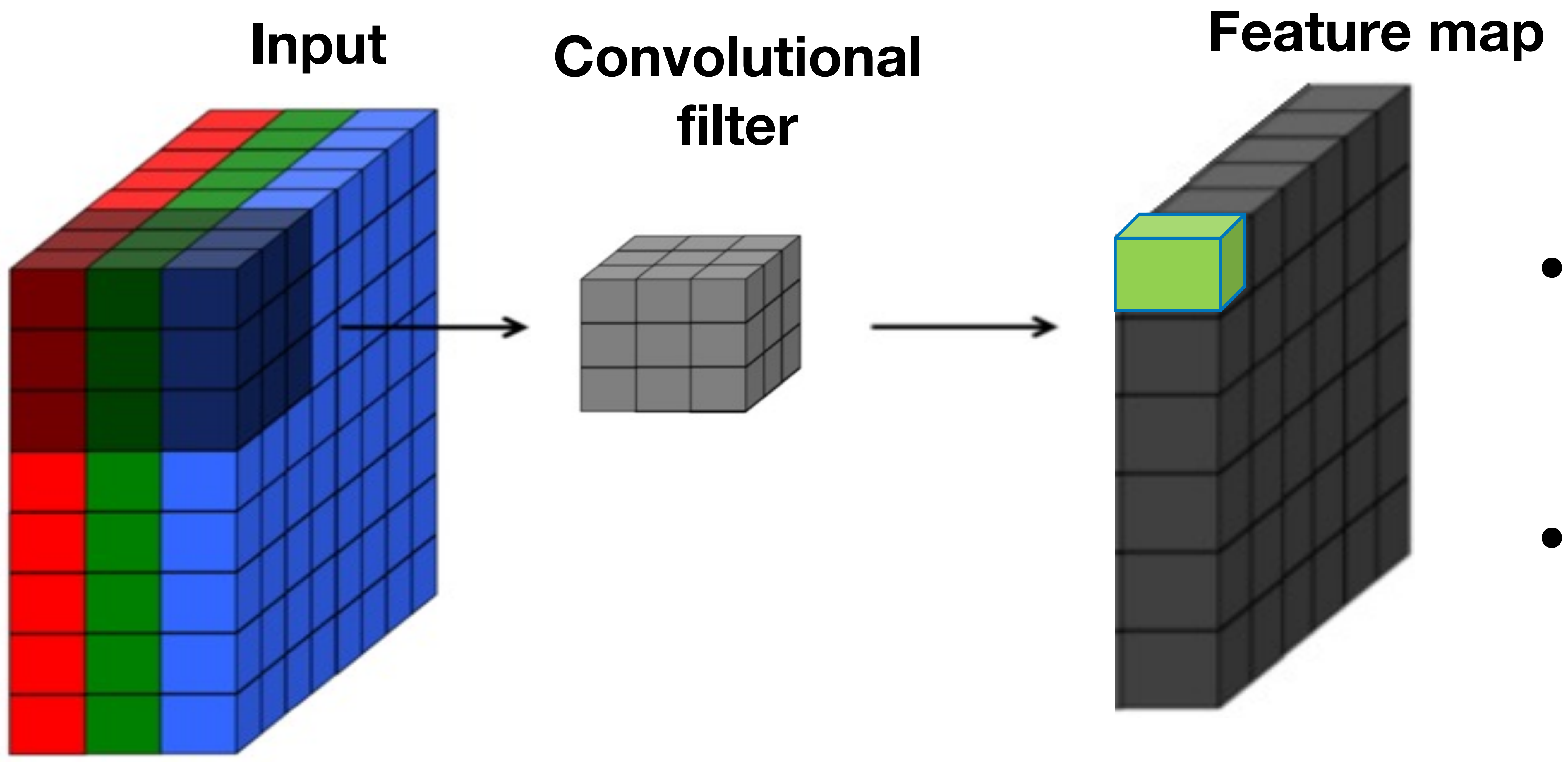


- We call this block of shared weights a **convolutional filter**

- **Convolution**: Convolve a filter with the image, i.e., slide over the image spatially, computing at each position a dot product between the filter and a small chunk of the image (plus bias), $\boldsymbol{W}^T\boldsymbol{X} + b$

- The dot product then goes through an activation function, e.g., *ReLU*, to produce the output

convolution



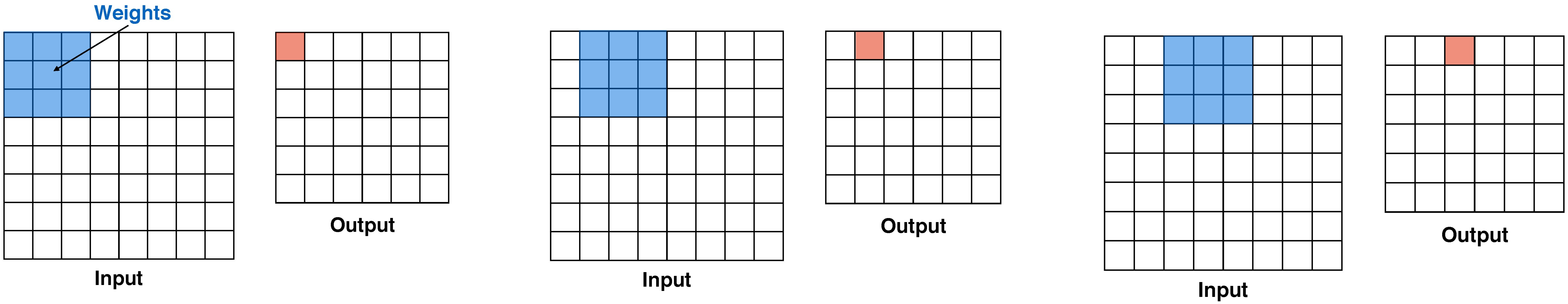
- At each spatial location, output one value

- Convolve (slide) over all spatial locations to generate an image like map, referred to as a **feature map**

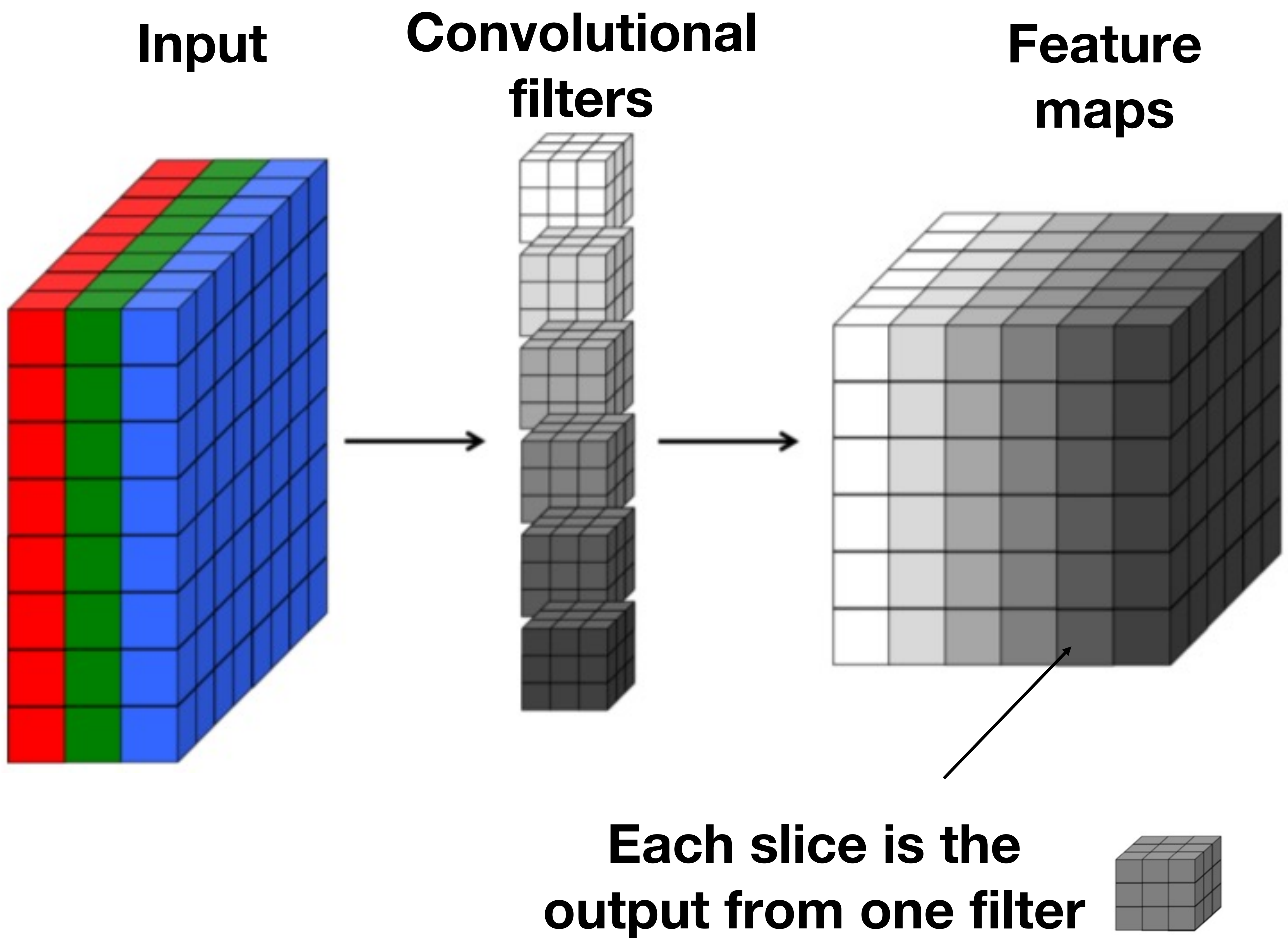
- **A convolutional layer**: Things between an input and a feature map

convolution

During convolution, the weights "slide" along the input to generate each output.



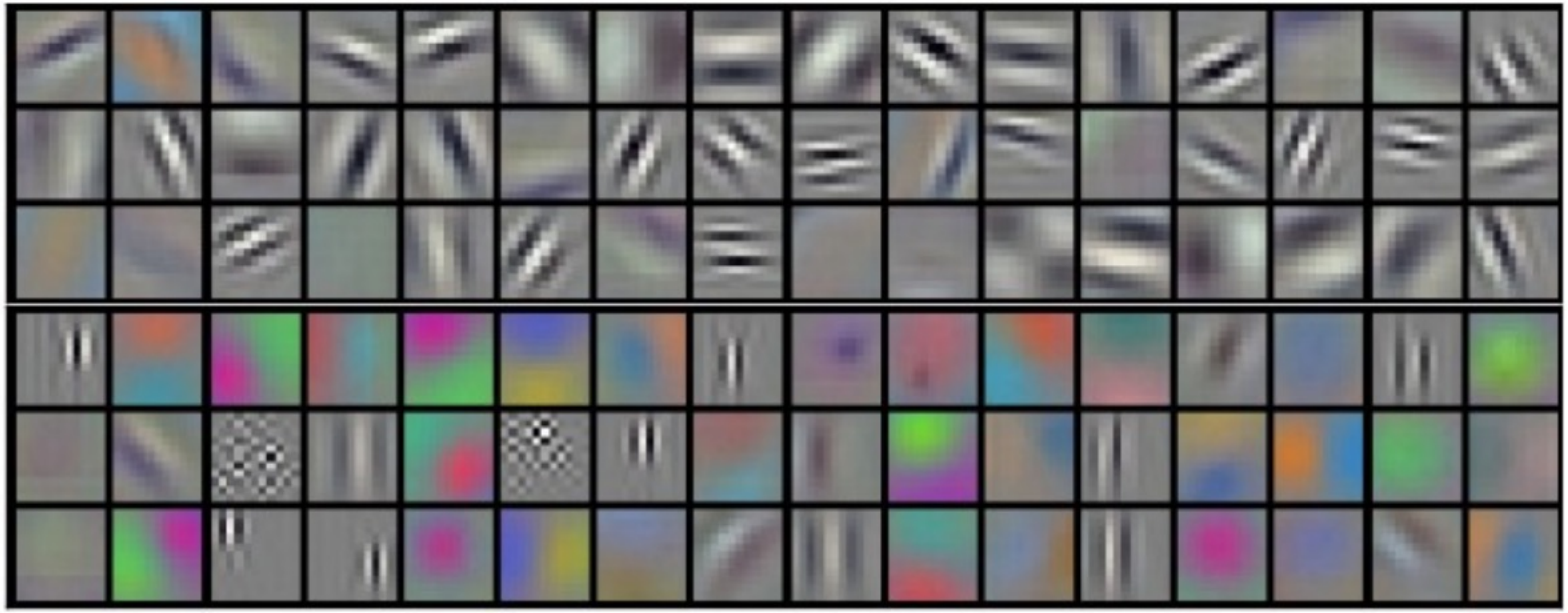
convolution



- Multiple sets of shared weights (filters) are allowed
- Each set of shared weights (filter) give one slice in the output (feature maps)
- In practice, CNN use many filters (~64 to 1024)

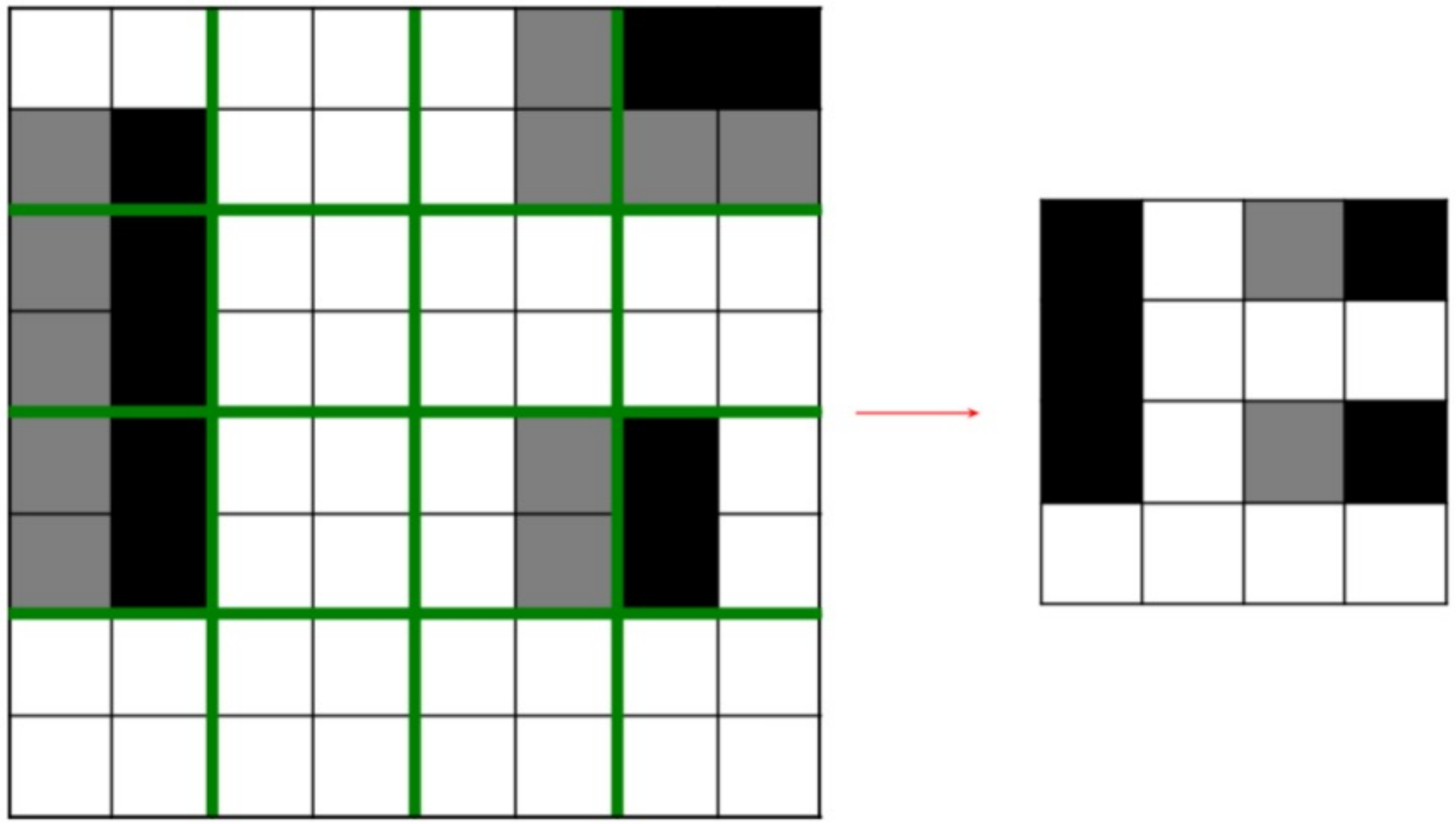
visualizing convolution

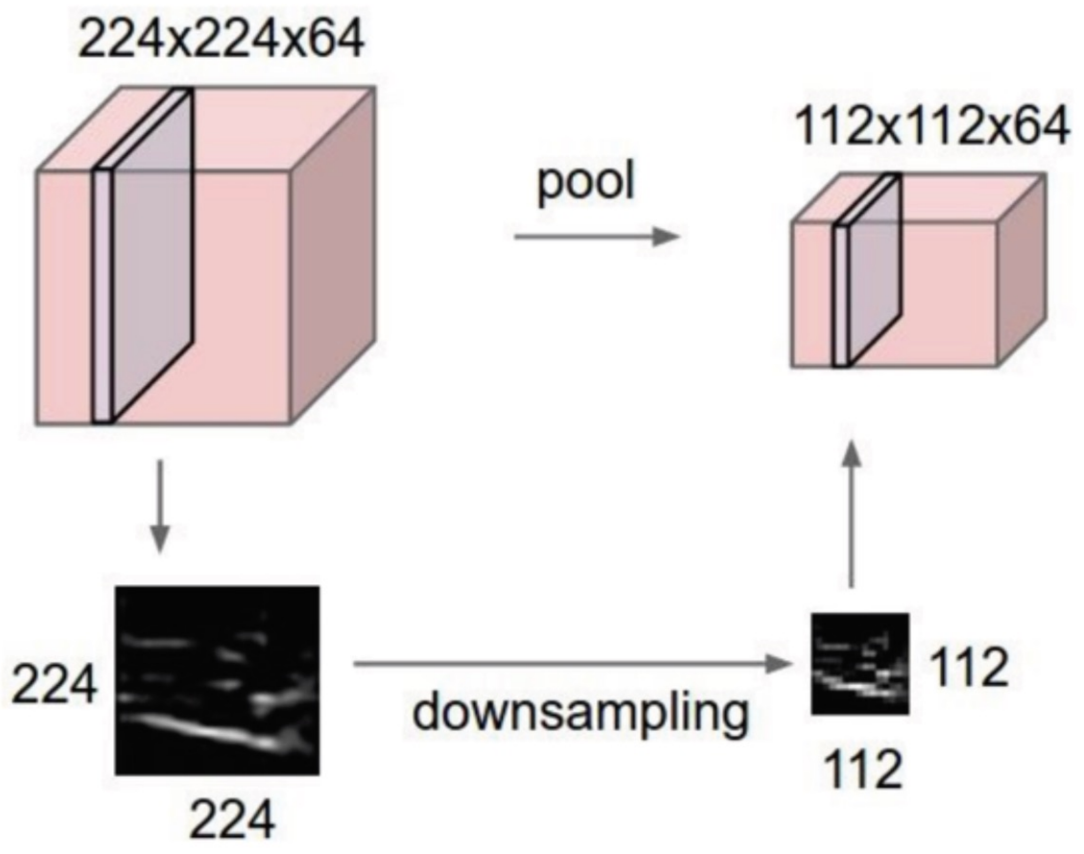
How convolutional filters may look like



pooling

Feature map

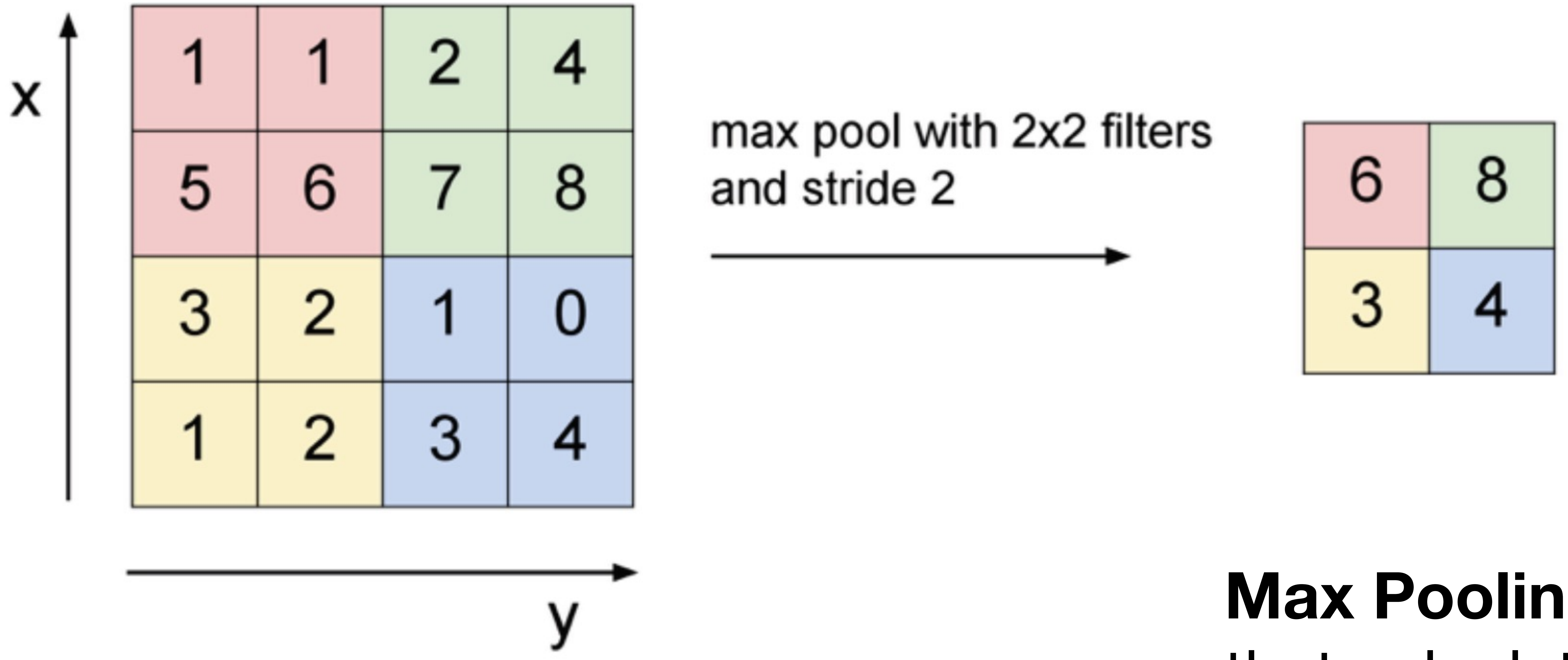




Convolution is often followed by **pooling**:

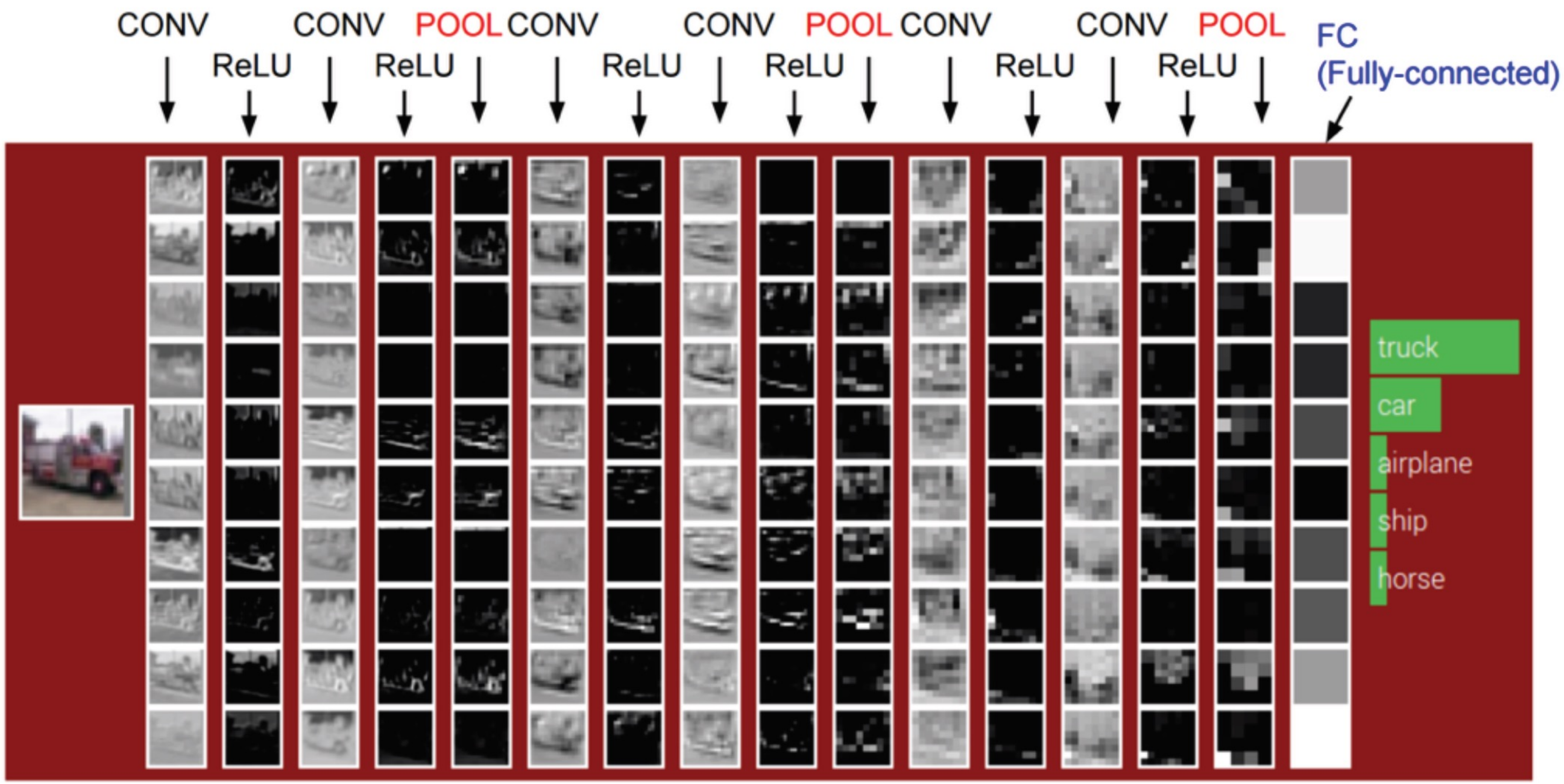
- Create a smaller and more manageable representation while retaining the most important information
- "*max*" is the most common operation
- Operate over each feature map independently

max pooling



Max Pooling is a pooling operation that calculates the maximum value for patches of a feature map.

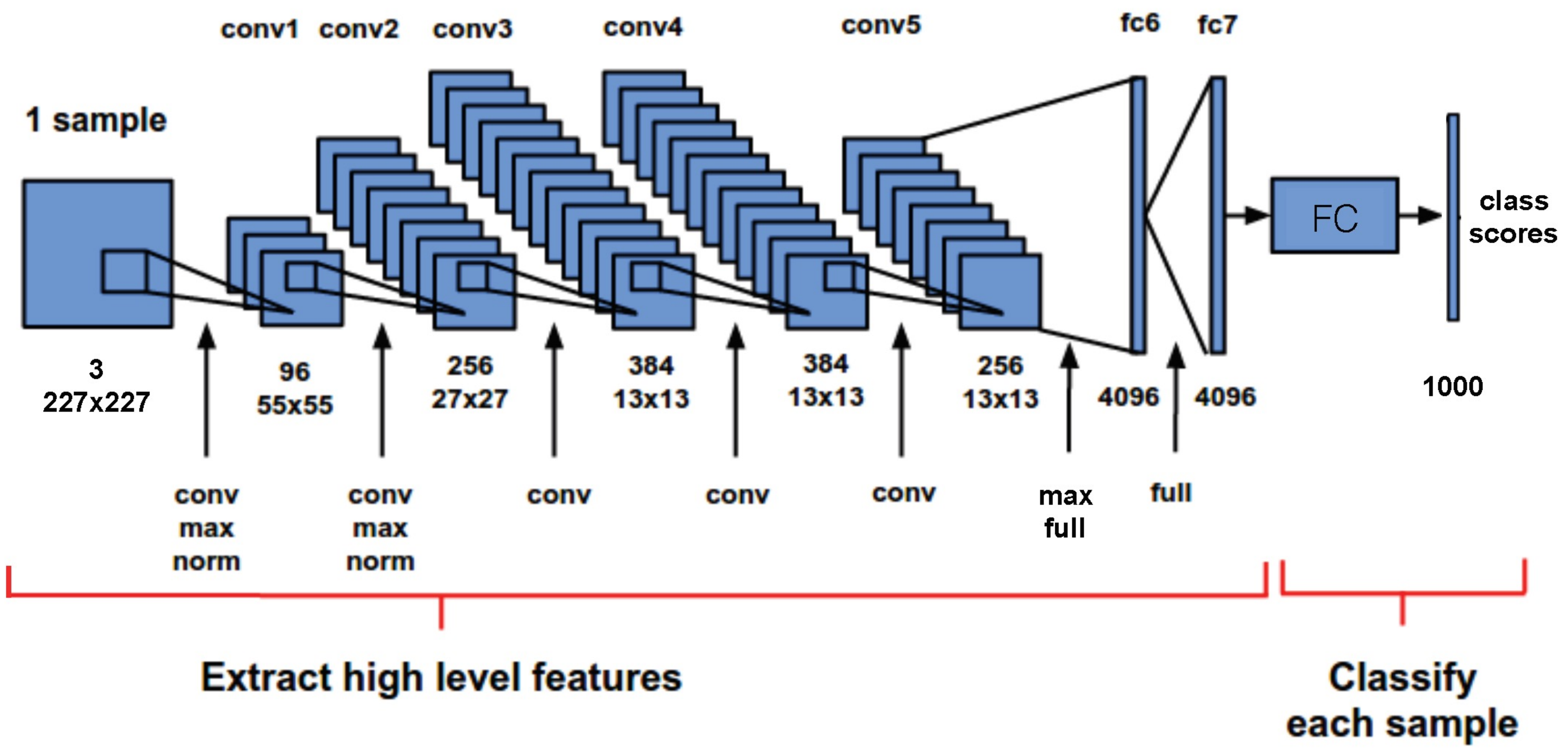
convolutional neural network (CNN)



Stack layers of convolution, activation (ReLU), pooling => CNN

how to train a CNN?

Example: AlexNet [Krizhevsky 2012]



"max": max pooling
"norm": local response normalization
"full": fully connected

Figure: [Karnowski 2015] *(with corrections)*

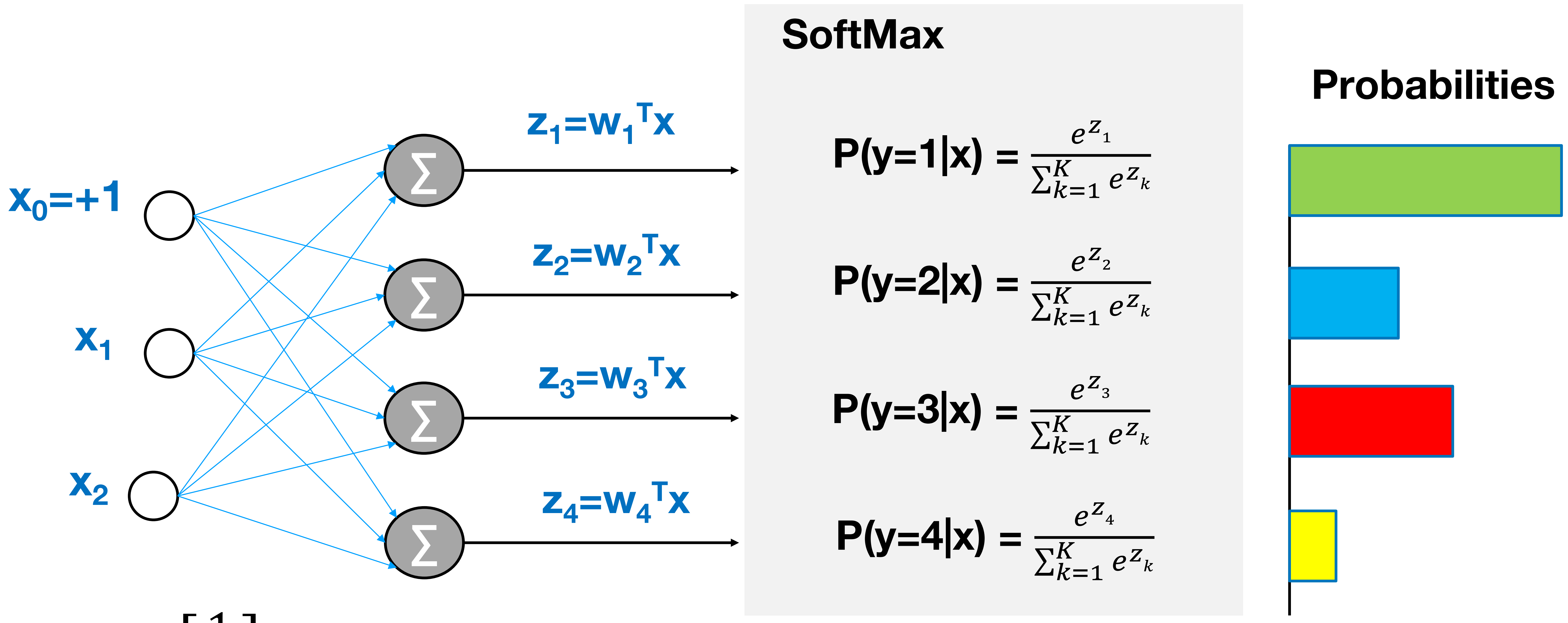
- Split the data
- Choose the network architecture
- Initialize the network weights
- Find a learning rate and regularization strength
- Minimize the loss, e.g., *softmax*

splitting the dataset

- **Train**: gradient descent and fine-tuning of parameters
- **Validation**: determining hyper-parameters (learning rate, regularization strength, etc.) and picking an architecture
- **Test**: estimate real-world performance

softmax Loss (multinomial logistic regression)

A generalization of logistic regression for multi-class classification



$$z_k=w_k^Tx = \begin{bmatrix} w_{k0} \; w_{k1} \; wk_2 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

training

Goal: Minimize loss ⇒ Maximize the probability of true class

- (Per-sample) Negative log-likelihood loss, e.g., for the i-th sample, $(\boldsymbol{x}^i, y^i)$

$$L(\boldsymbol{x}^i, y^i; \boldsymbol{\theta}) = -\log\big(P(y = yi|\boldsymbol{x}^i)\big) = -\log\left(\frac{e^{z_{yi}}}{\sum_{k=1}^{K} e^{z_k}}\right)$$

- **Training**: Minimizing the loss w.r.t parameters over the whole training set using backpropagation

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} L(\boldsymbol{x}^i, y^i; \boldsymbol{\theta})$$

regularization

Regularization reduces overfitting (as we have seen before):

$$L = L_{data} + L_{reg}$$

$\lambda = 0.001$ $\lambda = 0.01$ $\lambda = 0.1$

Higher regularization

examples of regularization terms

- **L2 regularization:** encourages small weights

$$L_{reg} = \lambda \frac{1}{2} \|W\|_2^2$$

- **L1 regularization:** encourages sparse weights

$$L_{reg} = \lambda \|W\|_1 = \lambda \sum_{i,j} |W_{ij}|$$

- **Elastic net:** combines L1 and L2 regularization terms

$$L_{reg} = \lambda_1 \|W\|_1 + \lambda_2 \|W\|_2^2$$

- **Max norm:** clamps (clips) weights to some maximum norm

$$\|W\|_2^2 \leq c$$