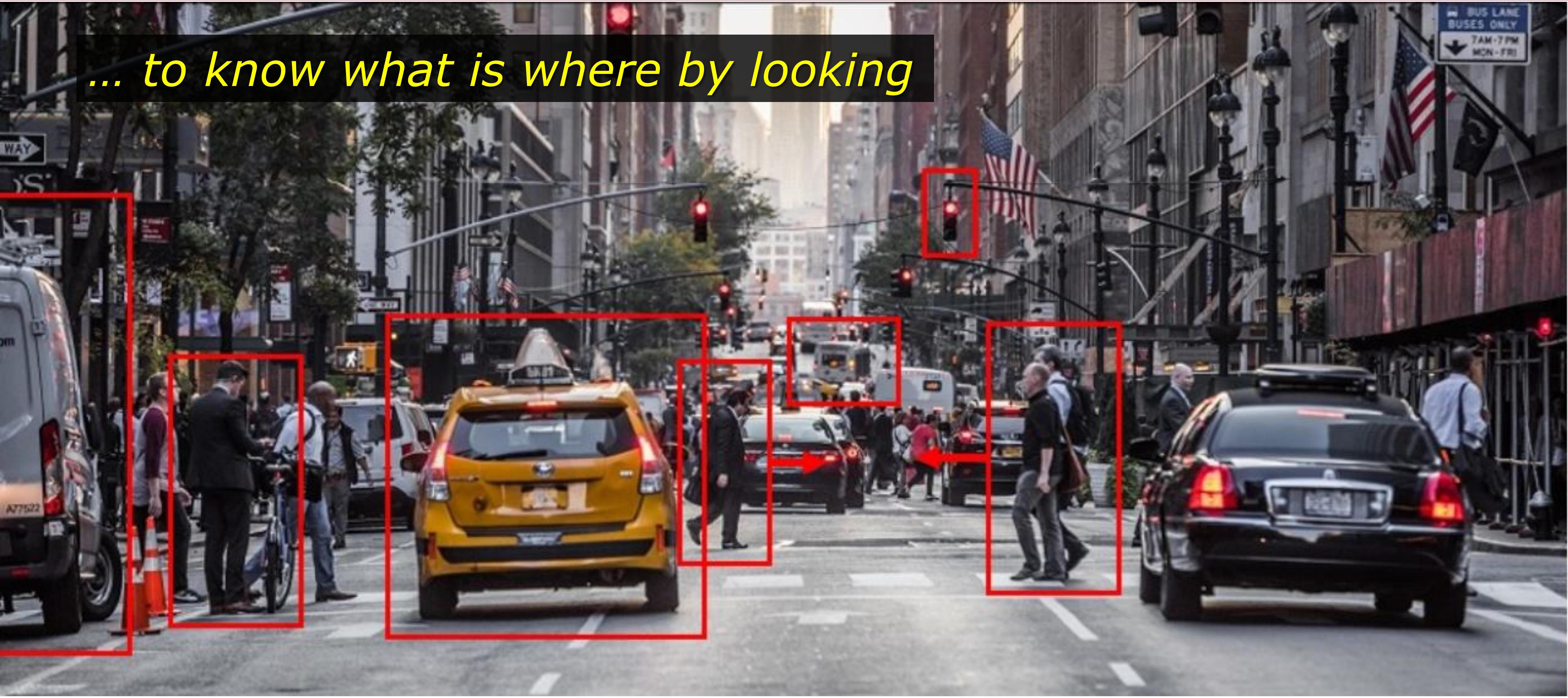
Computer Vision

CV Intro & CNNs

Computer Vision

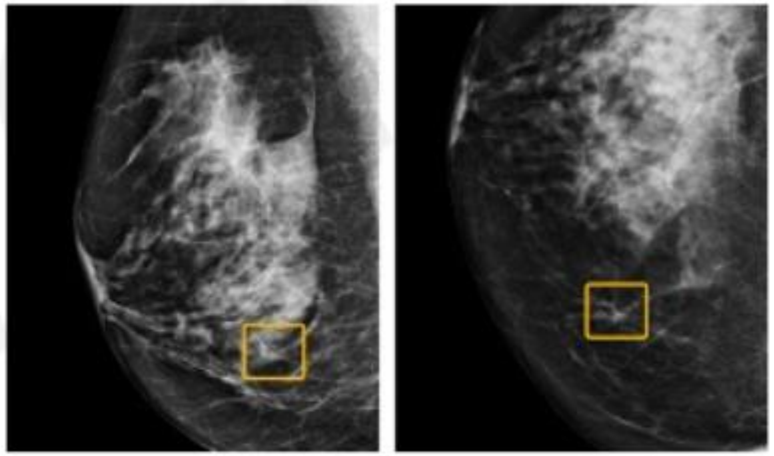


The rise and impact of computer vision

Robotics

Accessibility

Biology & Medicine

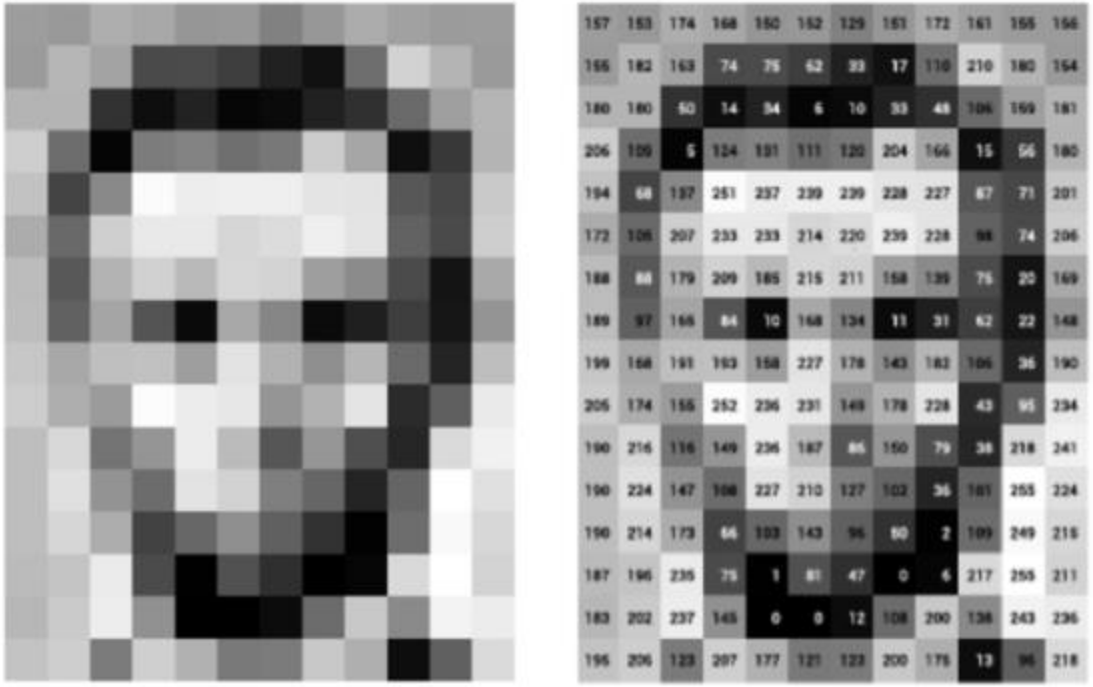


Autonomous driving

Mobile computing

What do computers see?

- Images are numbers

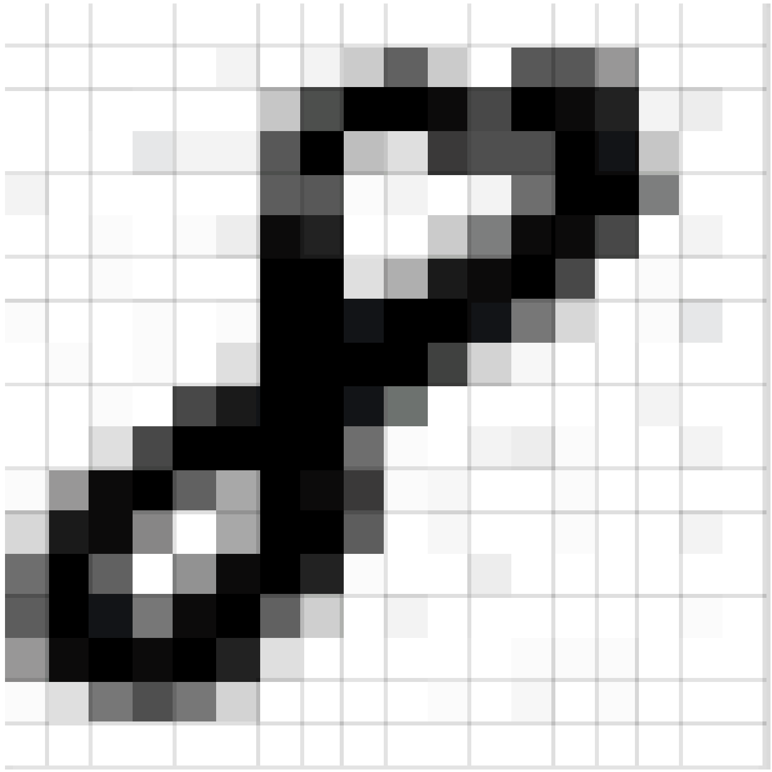


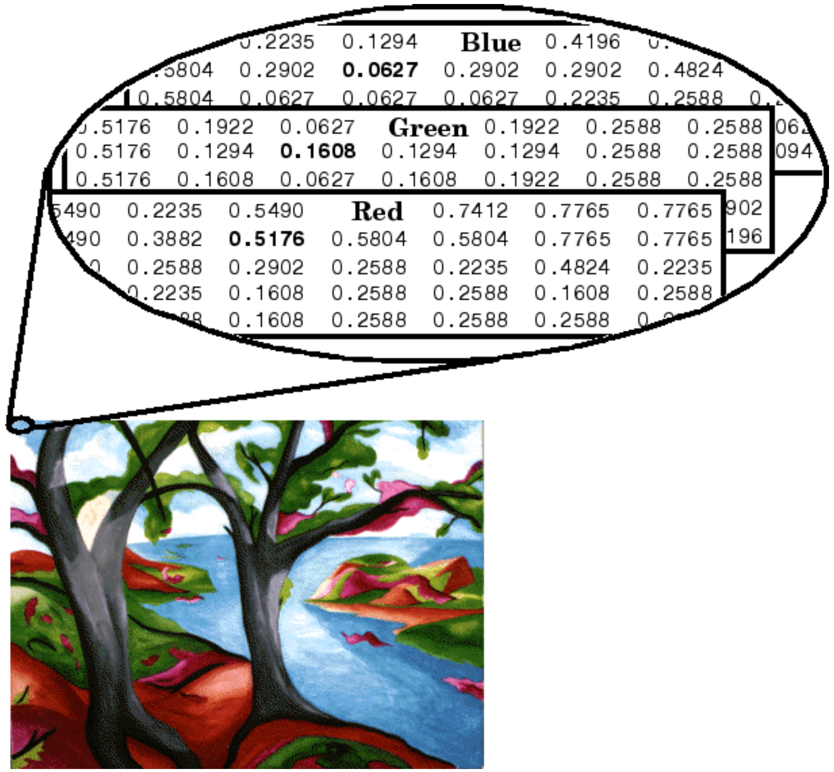
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0, 255]
i.e. 1080 x 1080 x 1 for a grayscale image

What do computers see?

- Images as matrices





An image is just a matrix of numbers [0, 255]
i.e. 1080 x 1080 x 3 for an RGB image

High level feature detection

- Lets identify key features in each image category

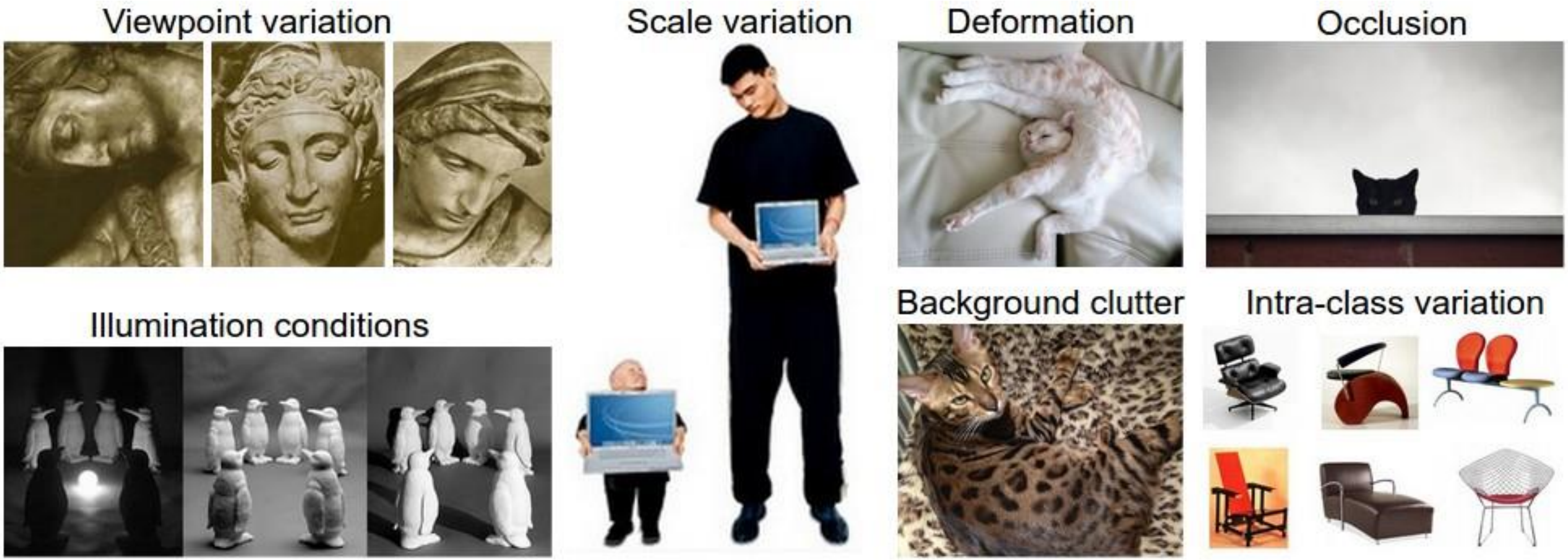
Nose,
Eyes,
Mouth

Wheels,
License Plate,
Headlights

Door,
Windows,
Steps

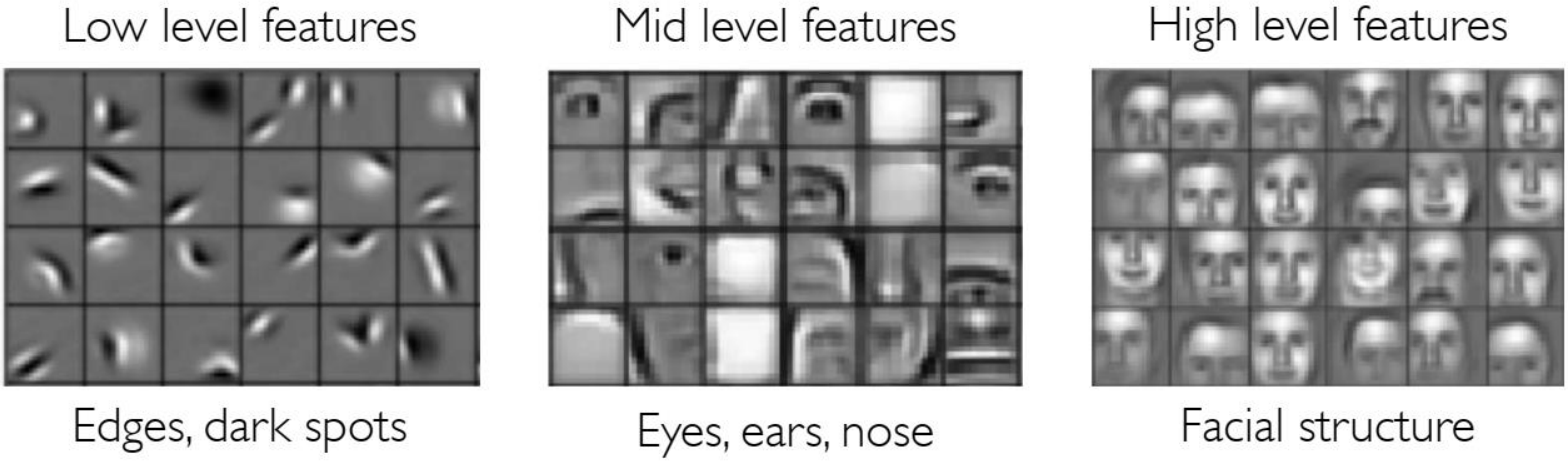
Manual feature extraction

Domain Knowledge → Define Features → Detect features to classify



Learning feature representations

- Can we learn a hierarchy of features directly from the data instead of hand engineering?

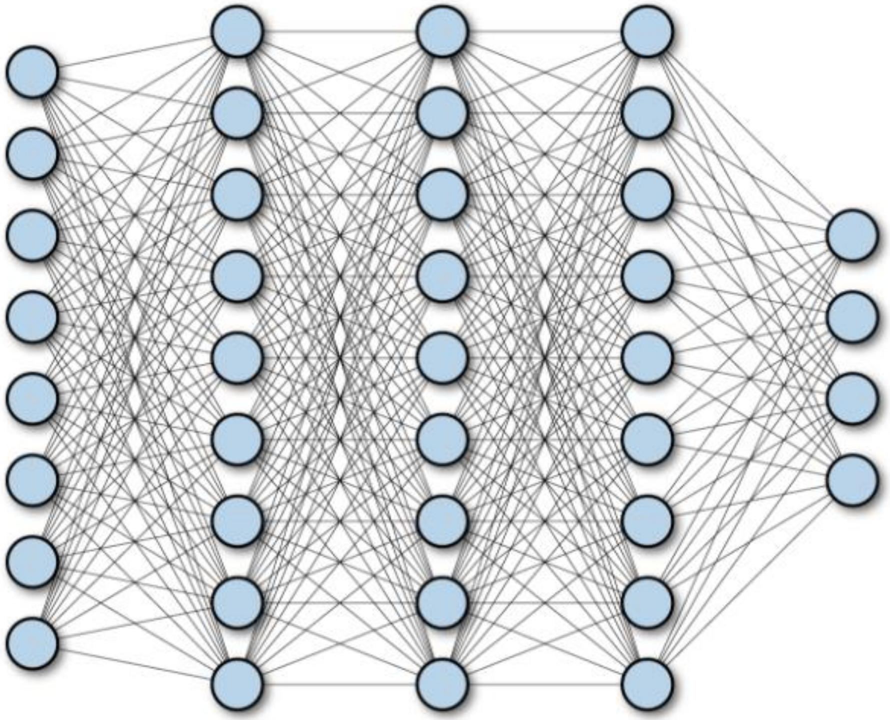


Learning visual features

- Fully connected neural network

Input:

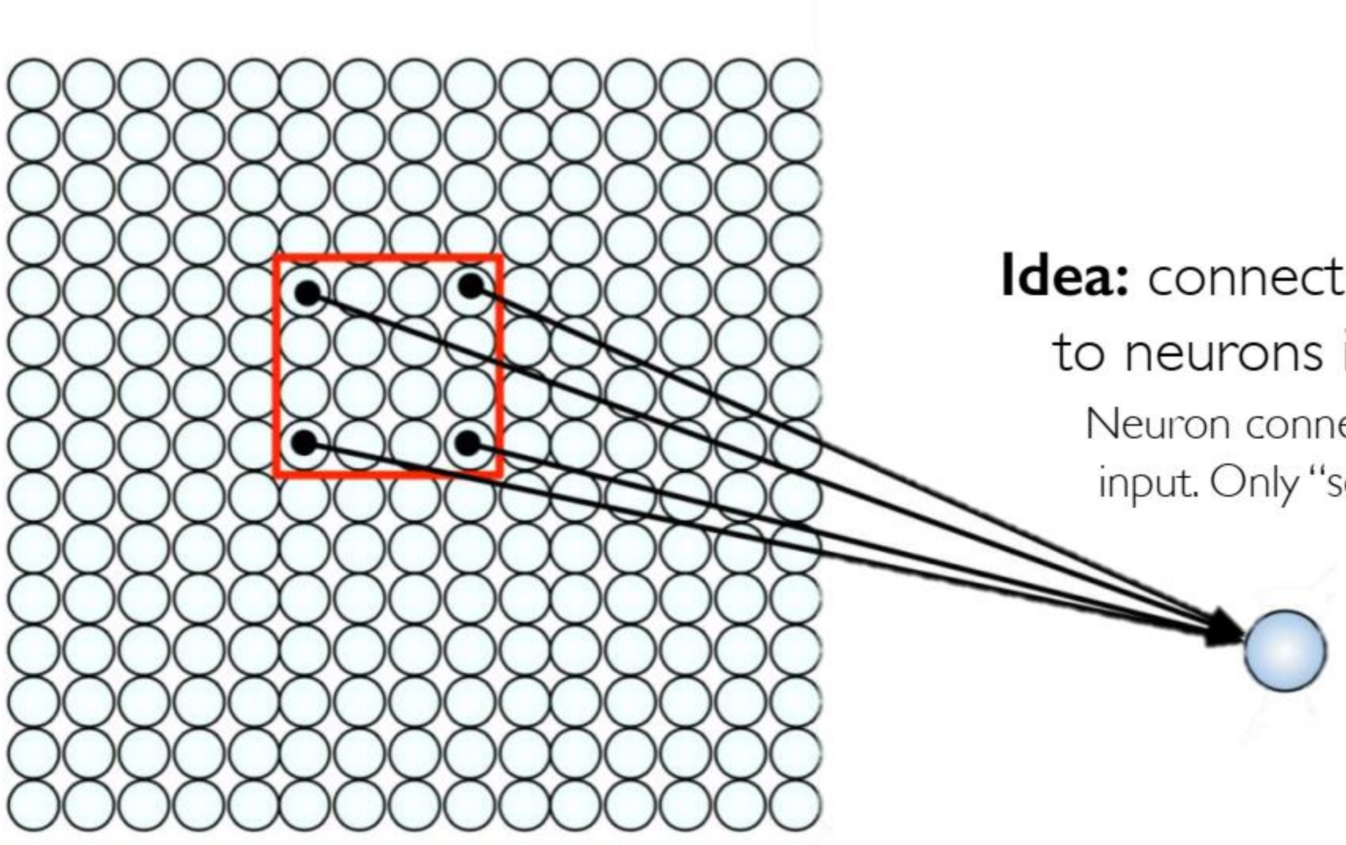
- 2D image
- Vector of pixel values



Learning visual features

- Using spatial structure

Input: 2D image.
Array of pixel values

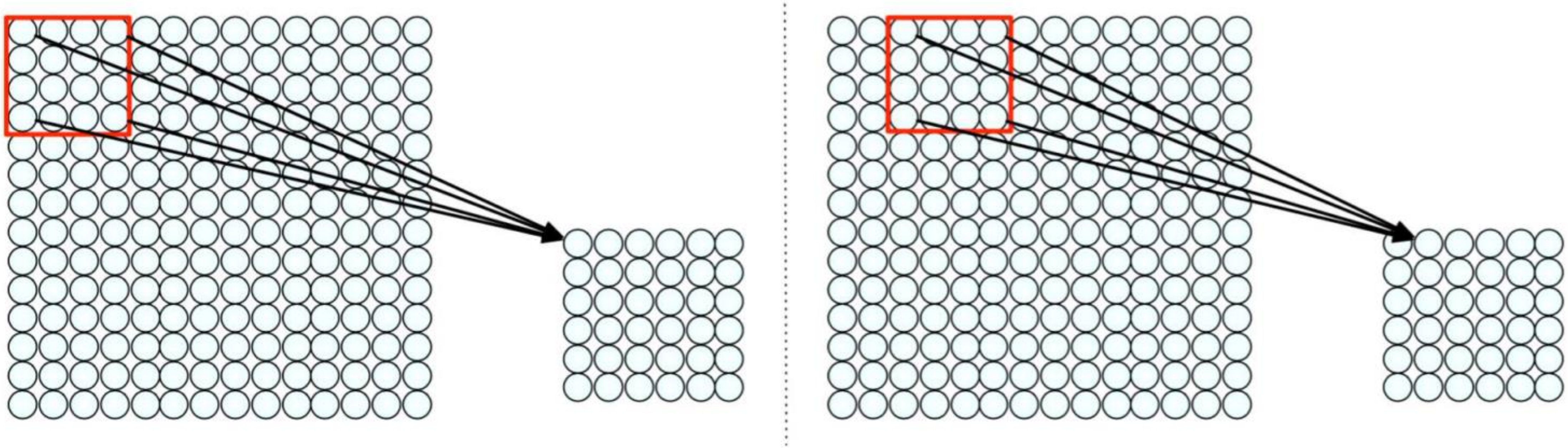


Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.

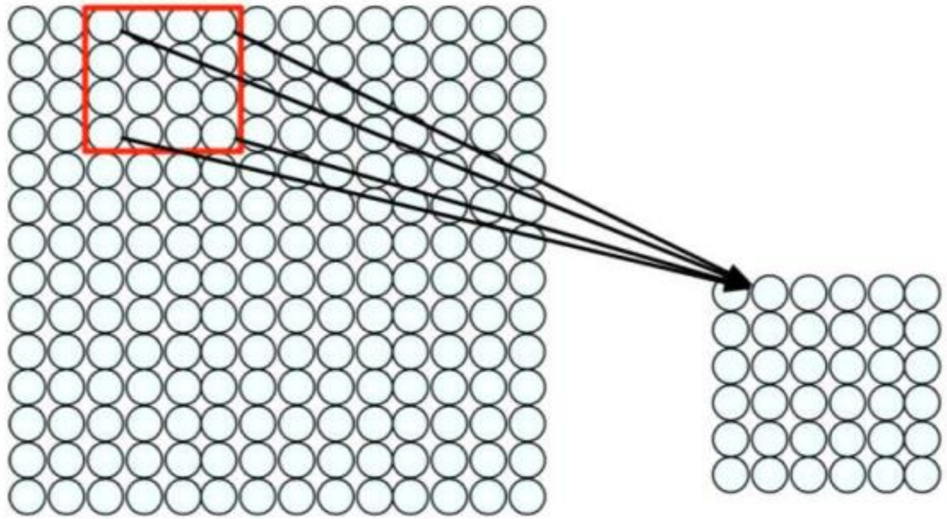
Learning visual features

- Using spatial structure



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.

Feature extraction with convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

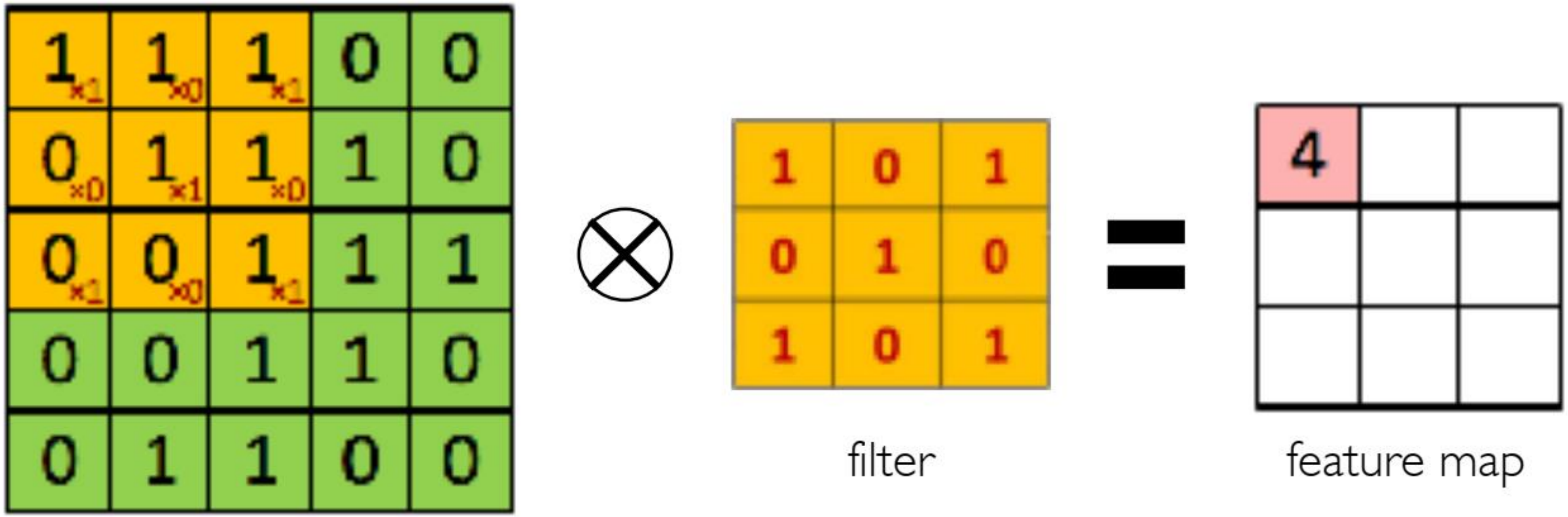
1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

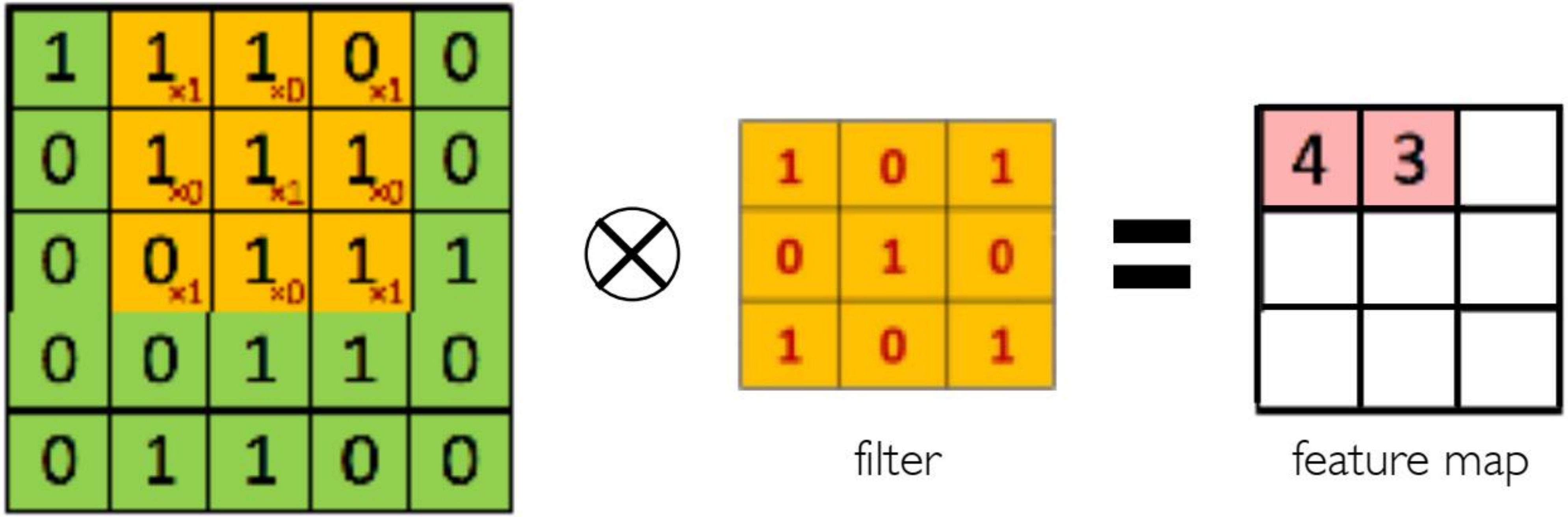
The convolution operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



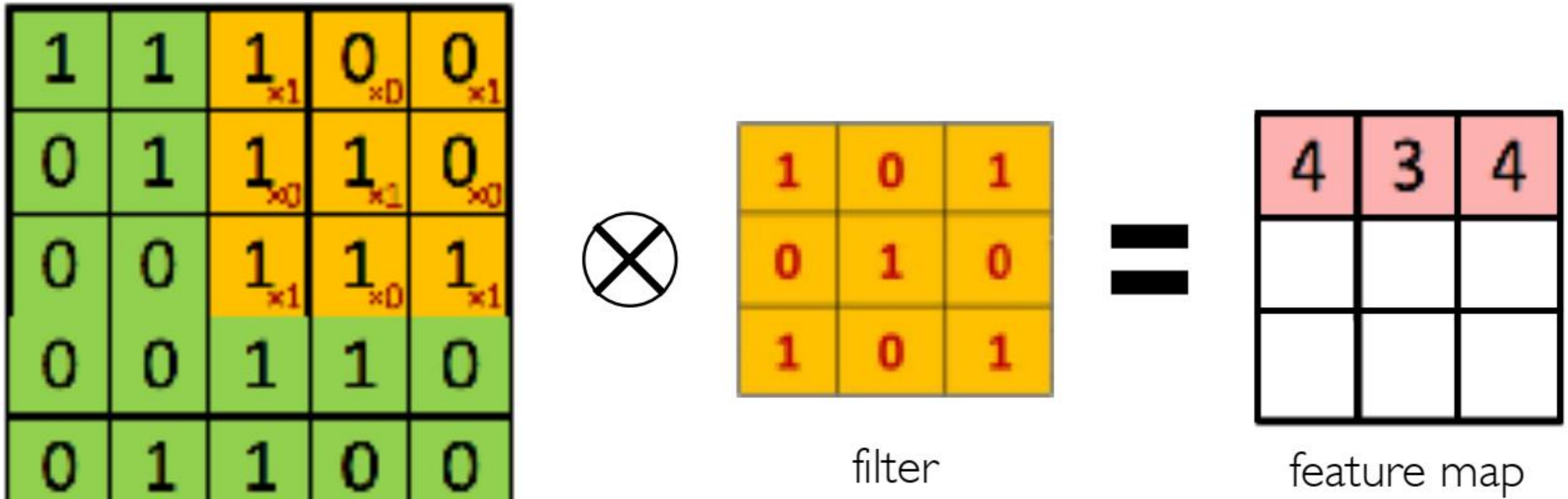
The convolution operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



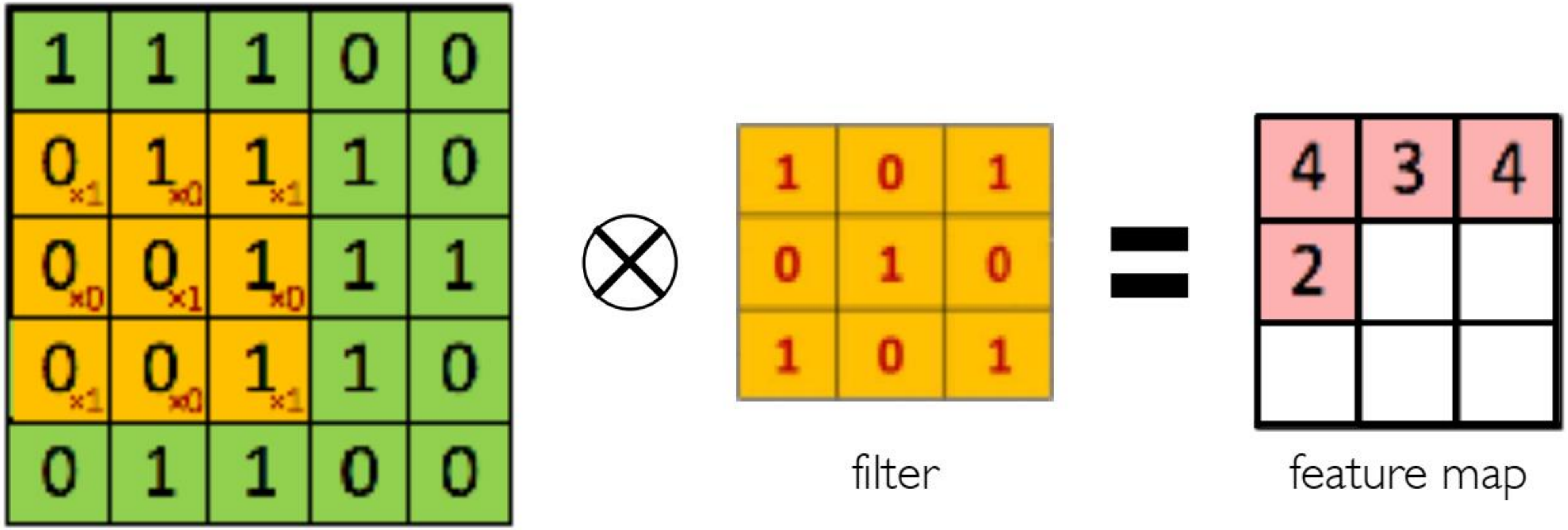
The convolution operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



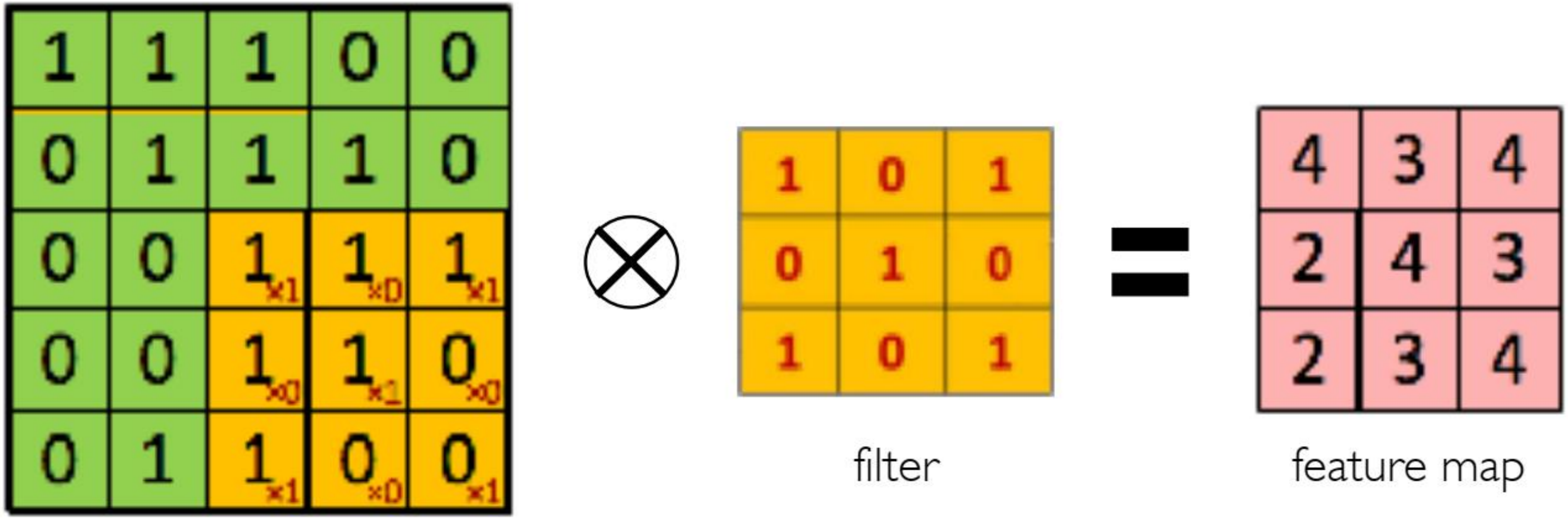
The convolution operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



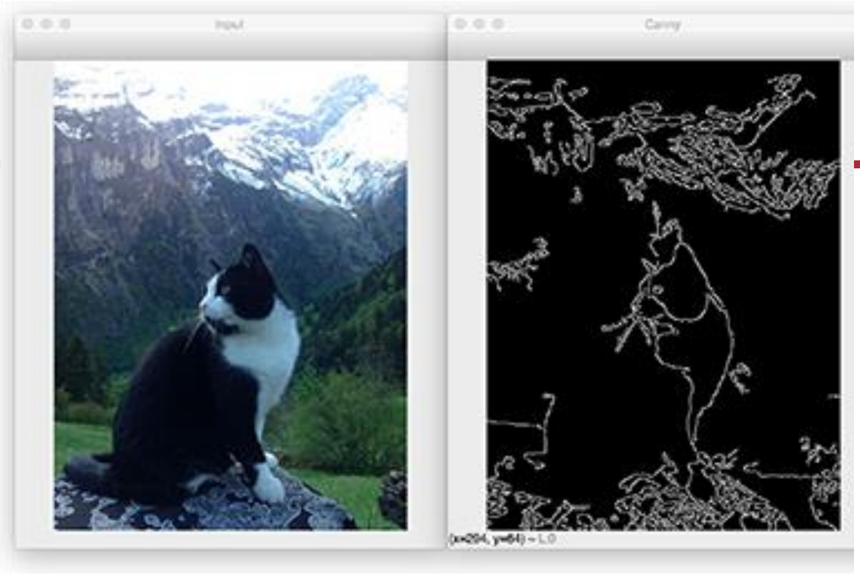
The convolution operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



Purpose of Convolutions

- What does this filter do?



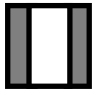
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



Purpose of Convolutions

- Different filters can be used to extract various features

Edge detection

Kernel

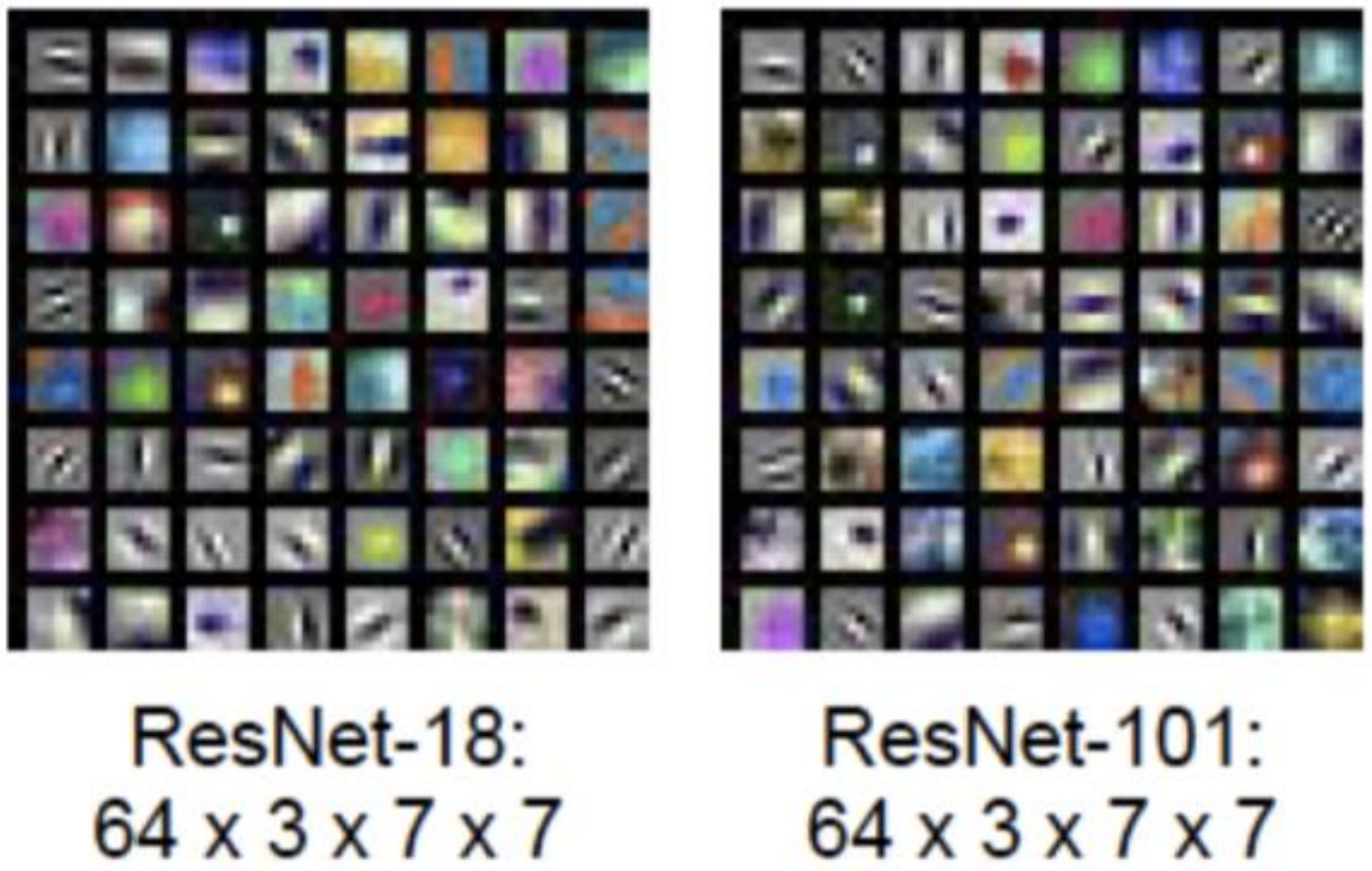
$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

Sharpen

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

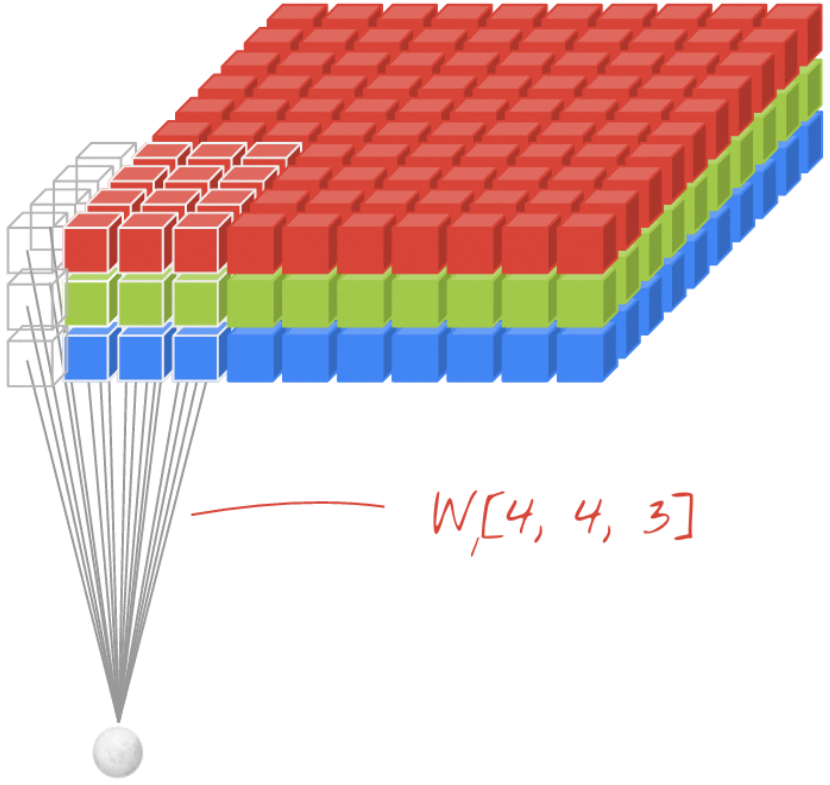
Where does the “Deep Learning” part come in?

- Like dense fully-connected layers, we can just learn these filters!



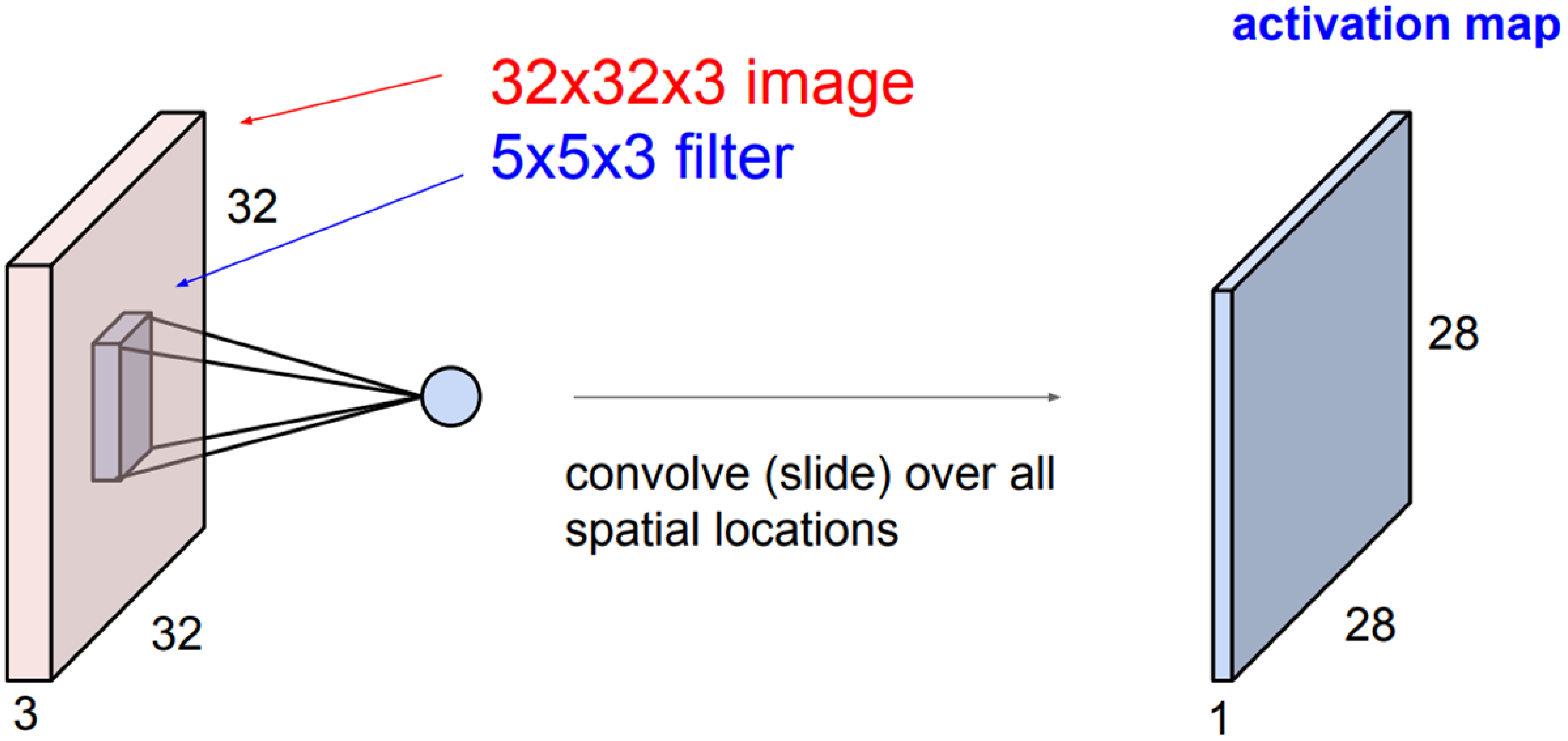
Filters (3D)

- Steps:
  - Compute the dot product for each channel **(same as 2D)**
  - Sum over each channel
- **Note**: The depth of the filter is always the same as the depth of the input image



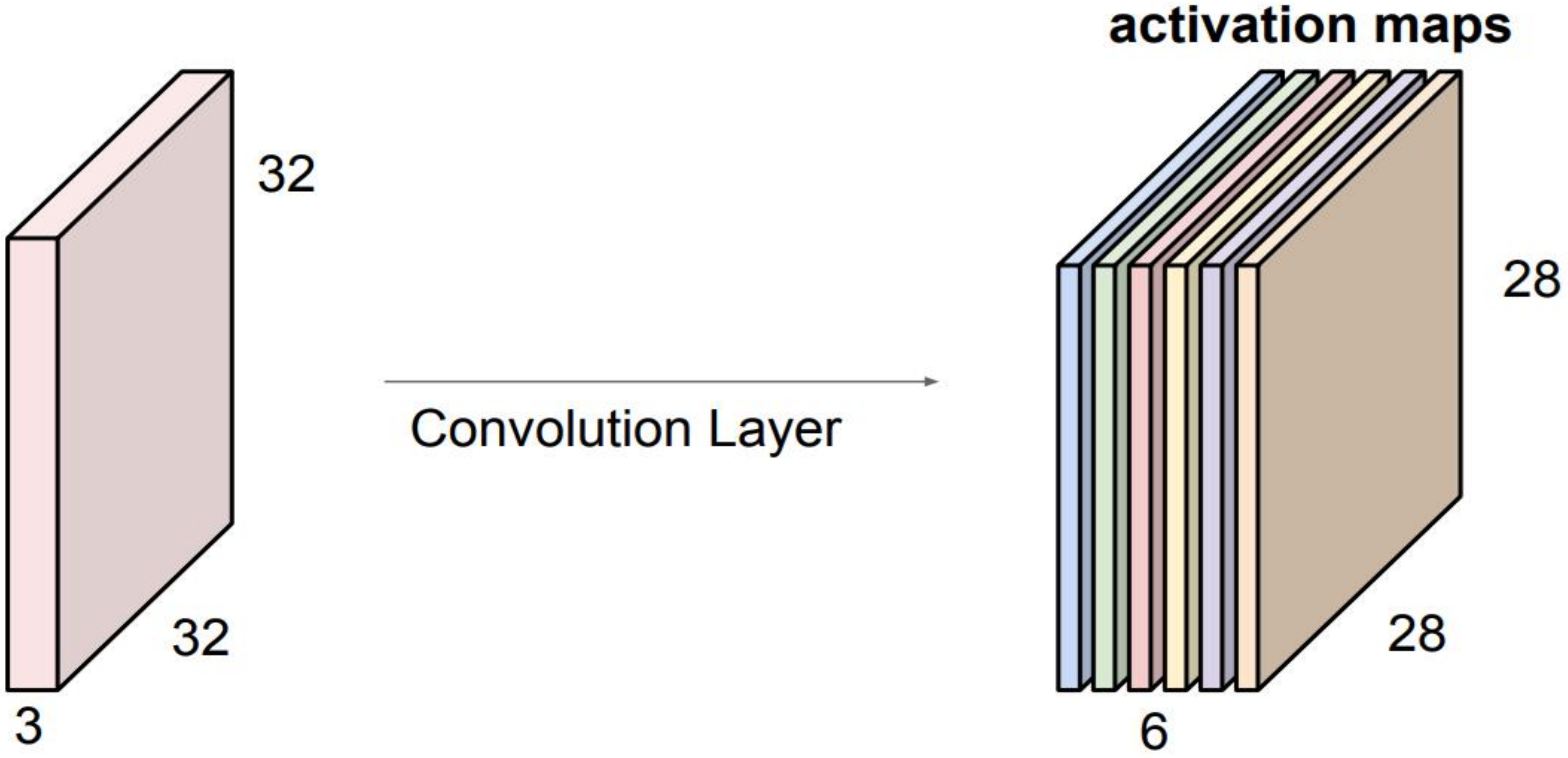
⚠ W1 and W2 are distinct 4 x 4 x 3 filters

Convolutional Layers



Convolutional Layers

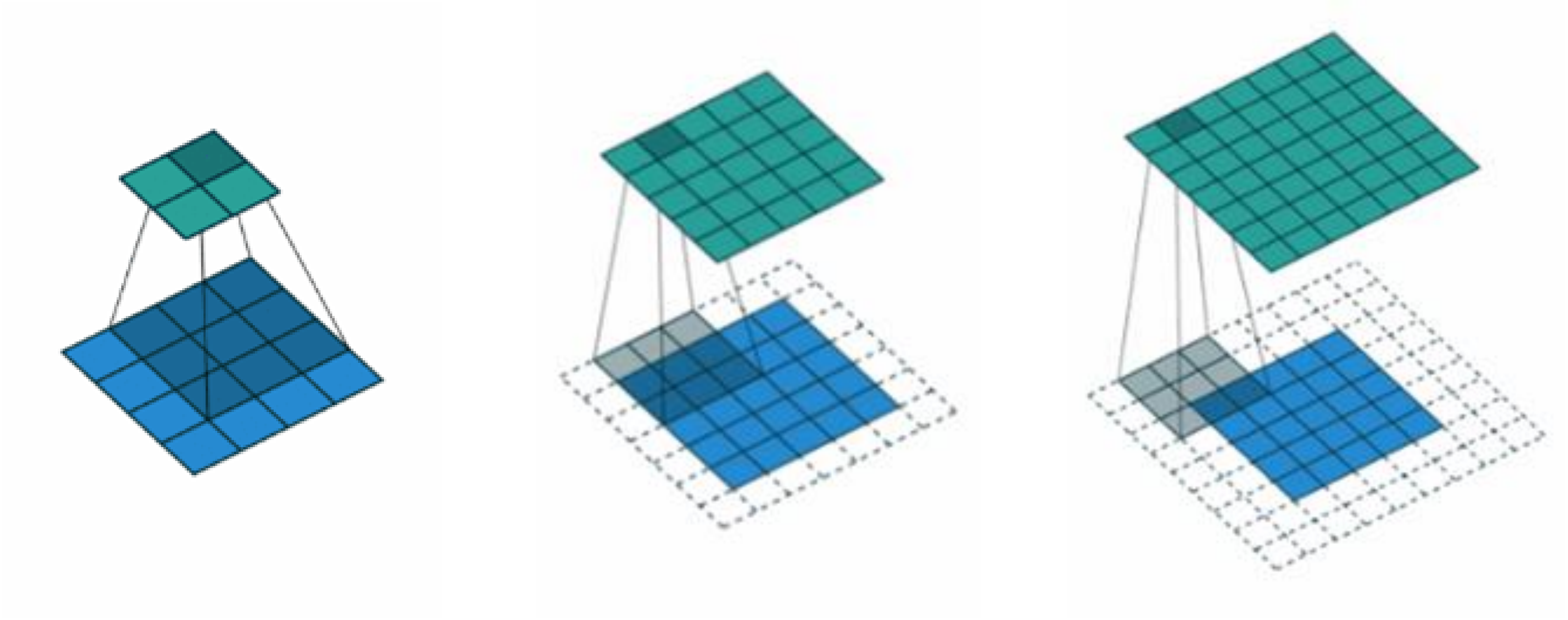
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

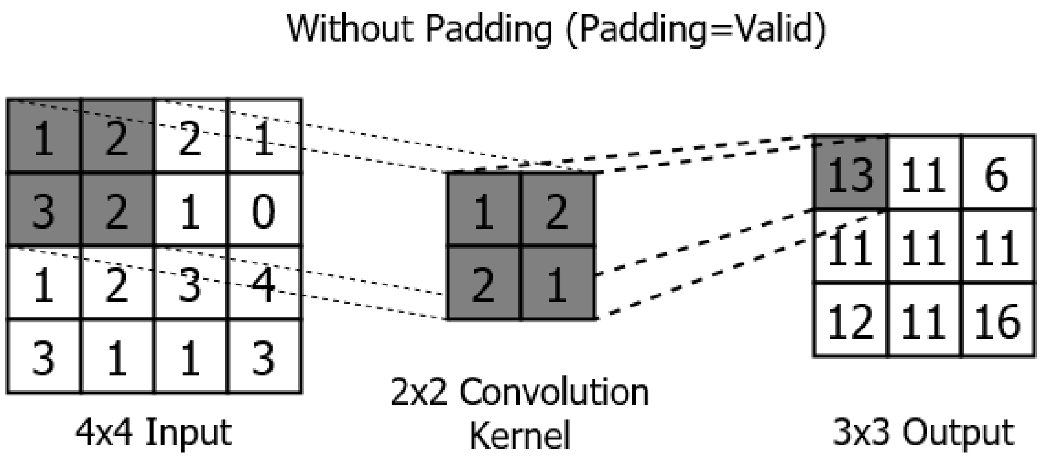
Convolution Layers – Padding

- Convolving an image with a filter results in a block with a smaller height and width — what if we want the height and width as before?

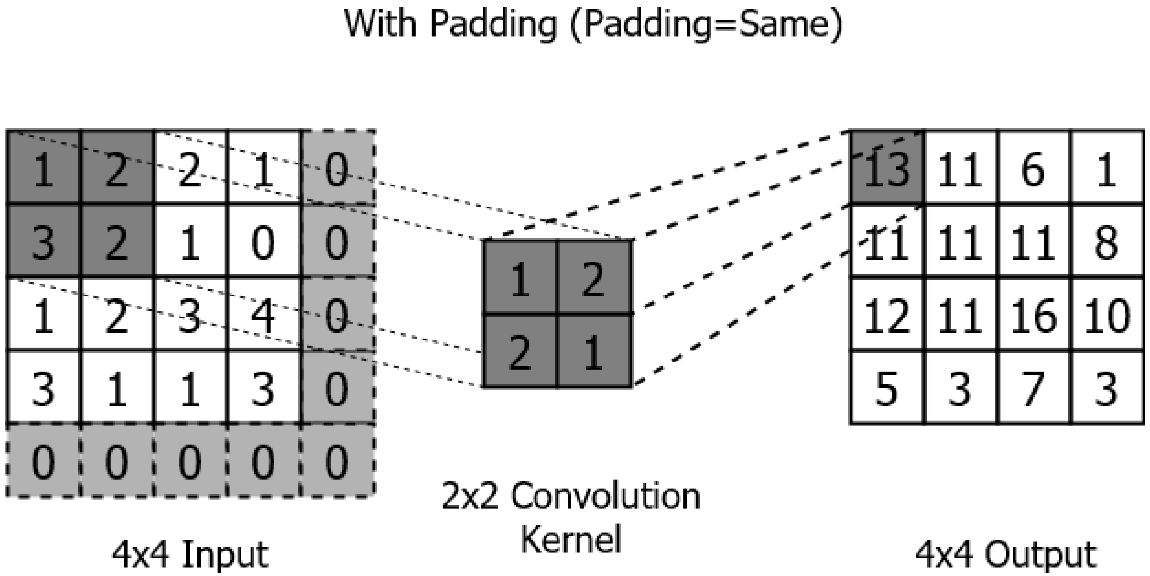


Convolution Layers – Same vs Valid Padding

- Valid padding: no padding

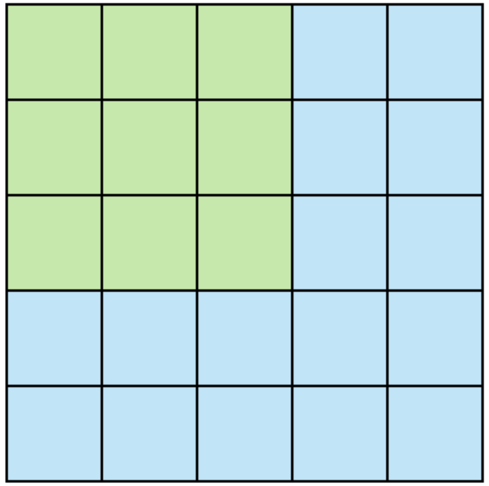


- Same padding: padding with 0s (or possibly some other constant value) to preserve the spatial dimensions of the output

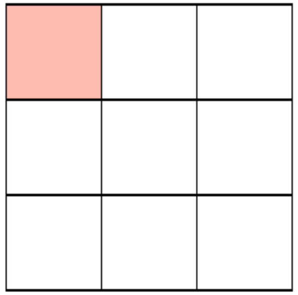


Convolution Layers – Stride

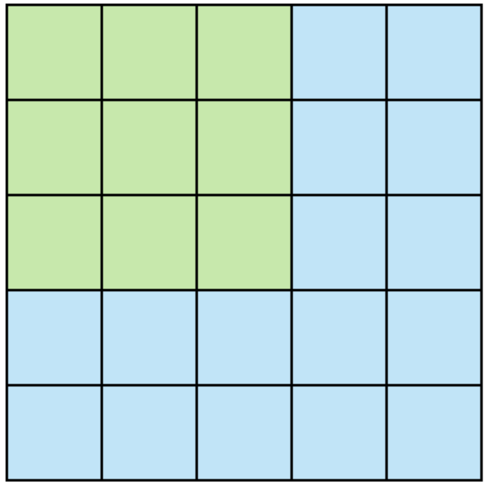
- The number of pixels to slide the filter by (both horizontally and vertically):
- A stride of 1 will shift the filter every pixel
- A stride of 2 will shift the filter every 2 pixels



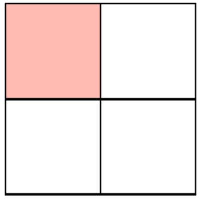
Stride 1



Feature Map



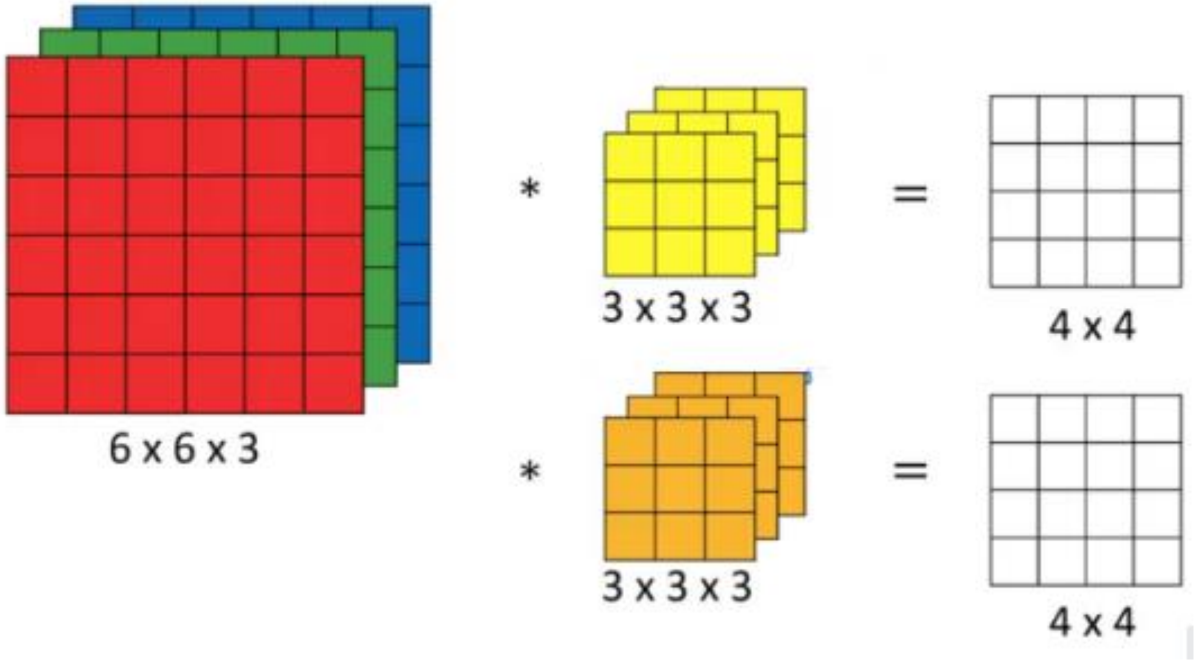
Stride 2



Feature Map

Output Dimensions

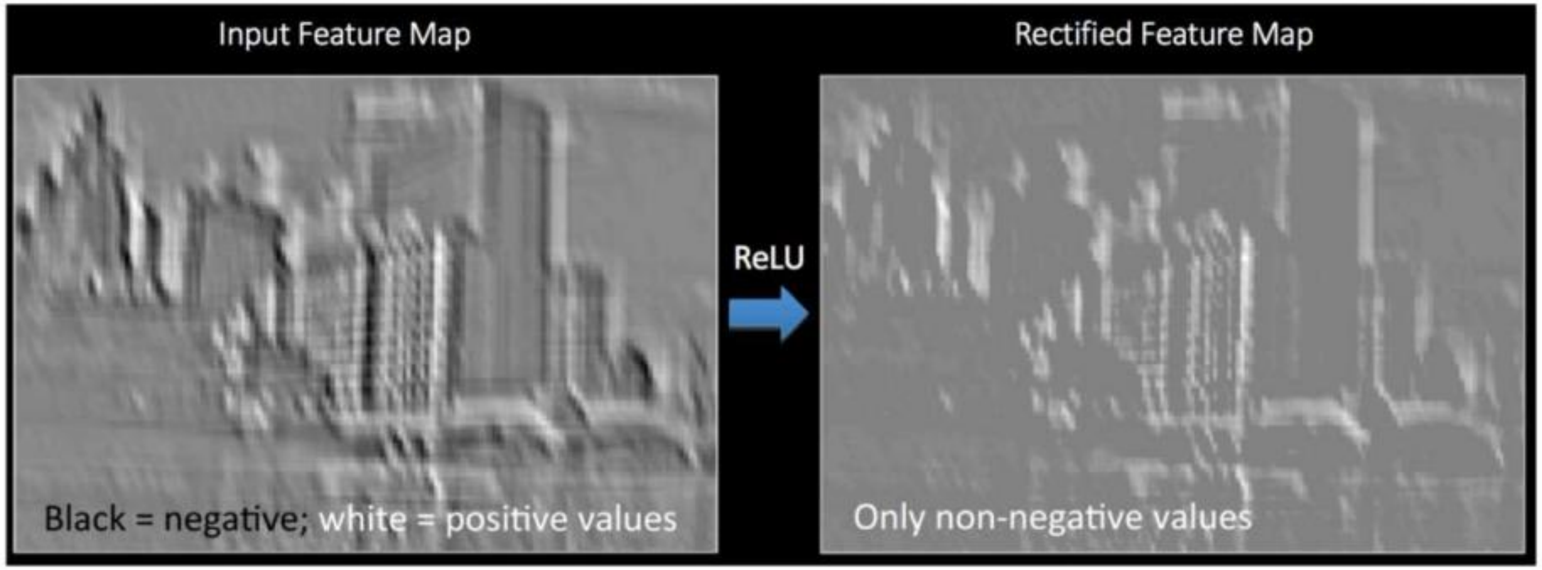
- ceil[(W − F + 2P) / S] + 1
  - W: Input Dimension
  - F: Kernel / Filter Size
  - P: Padding size
  - S: Strides



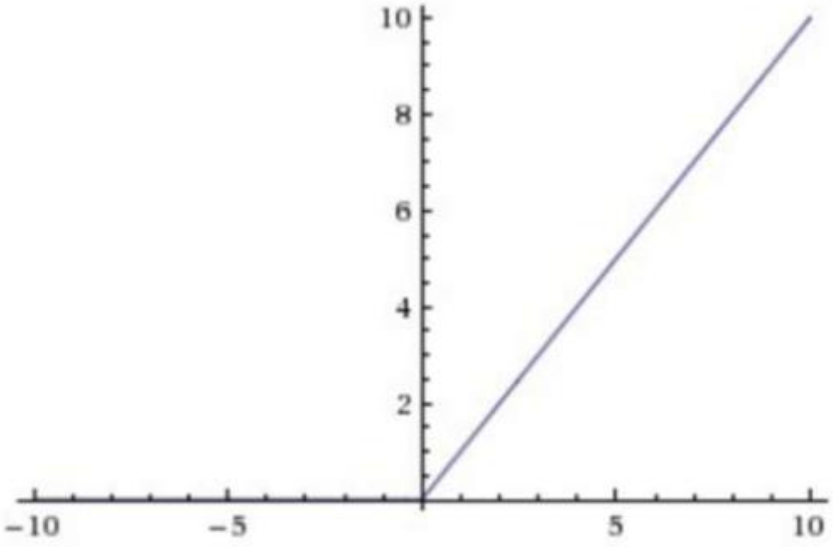
- For the figure on the right:
  - Assume no padding and a stridge of 1
  - W’ = ceil[(6 - 3 + 2 * 0) / 1] + 1 = 4
  - H’ = ceil[(6 - 3 + 2 * 0) / 1] + 1 = 4

Introducing non-linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



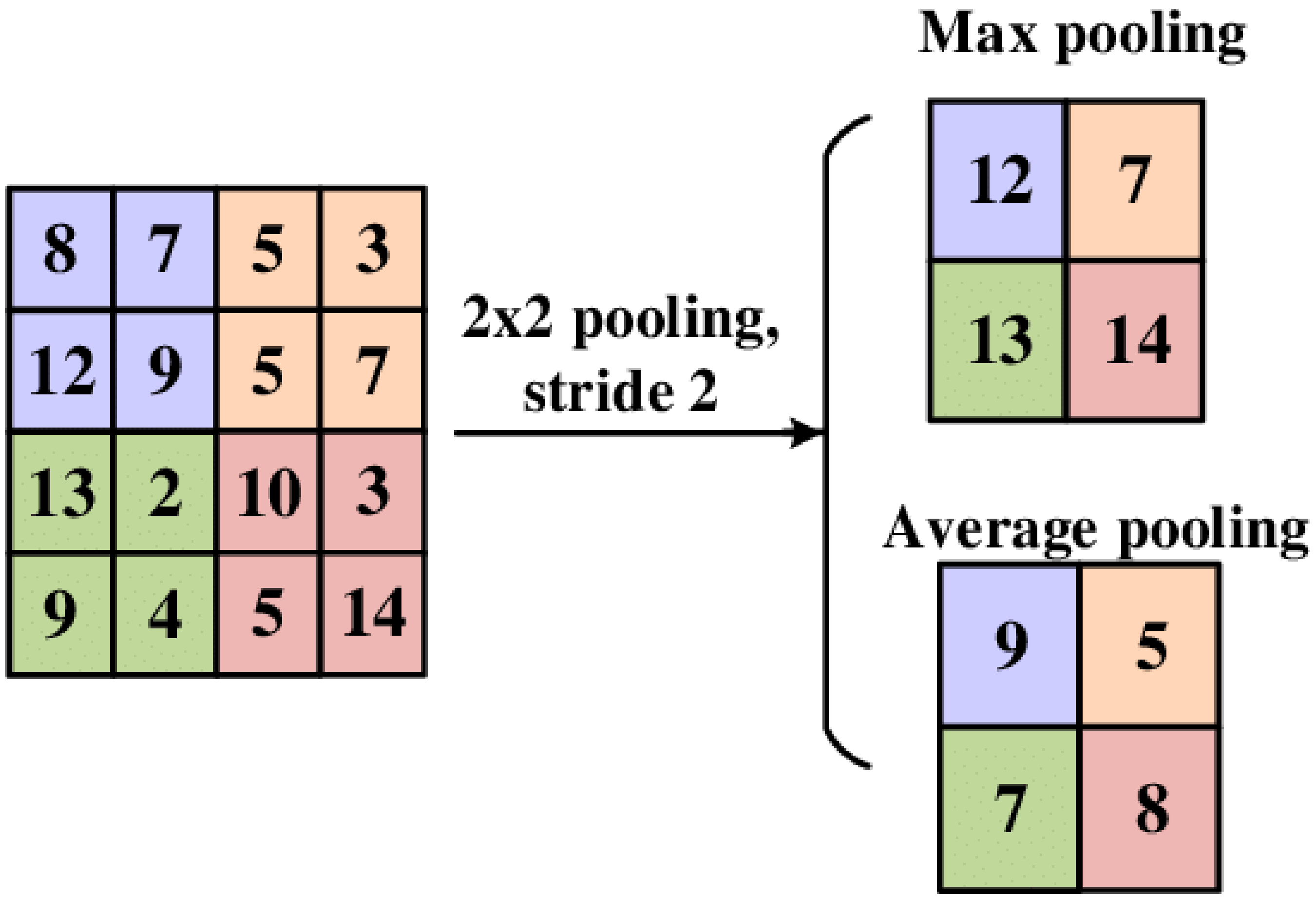
Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

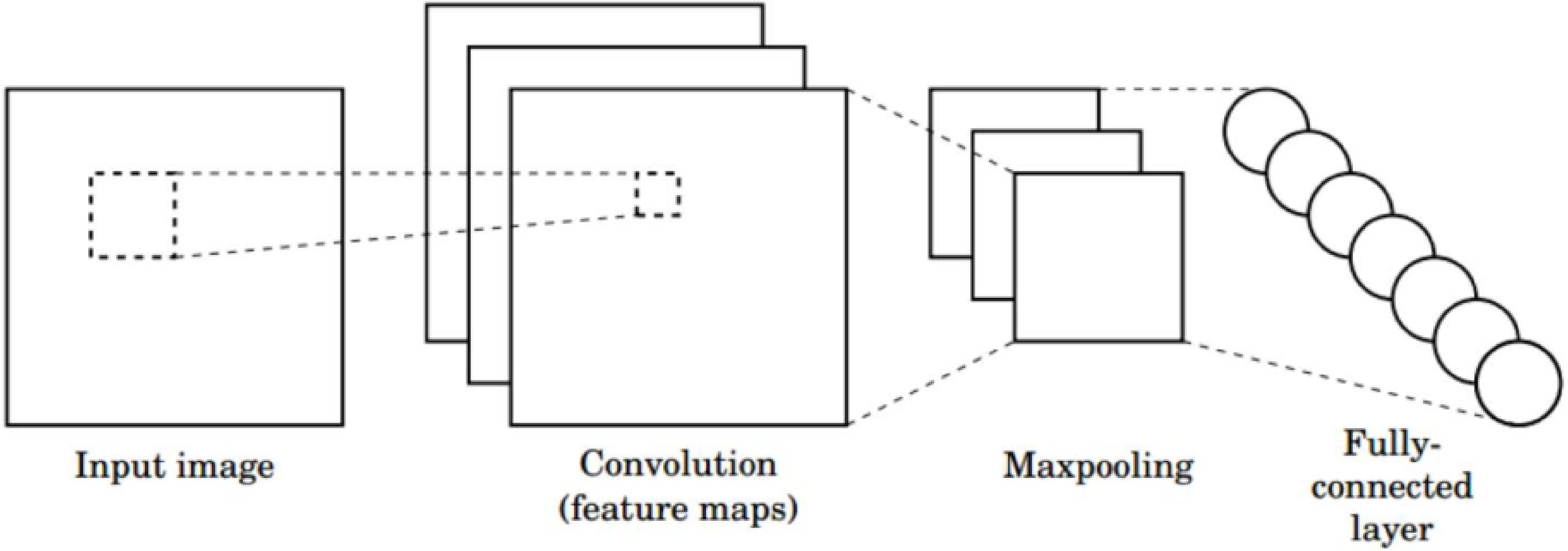
Pooling Layers

- Reduces output size
- Applied to each channel independently
- Neighboring features may be similar
  - Doesn't remove too much information



- Max pooling takes the max
- Average pooling takes the average

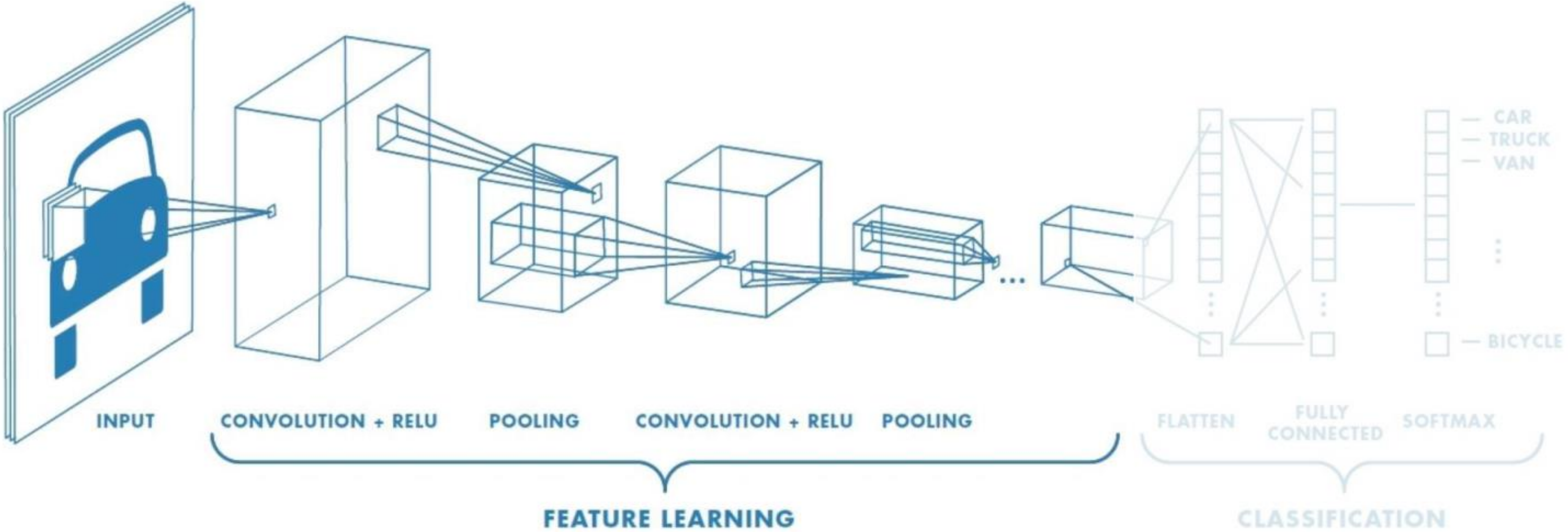
CNNs



1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

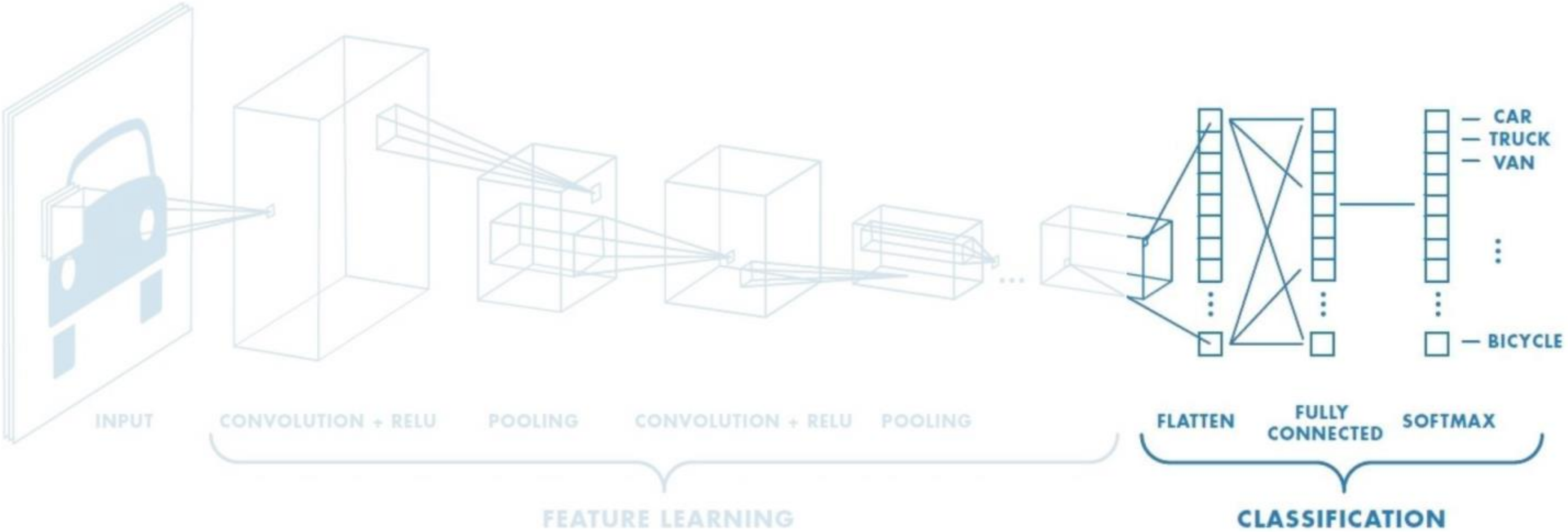
Train model with image data.
Learn weights of filters in convolutional layers.

CNNs for classification: *Feature learning*



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

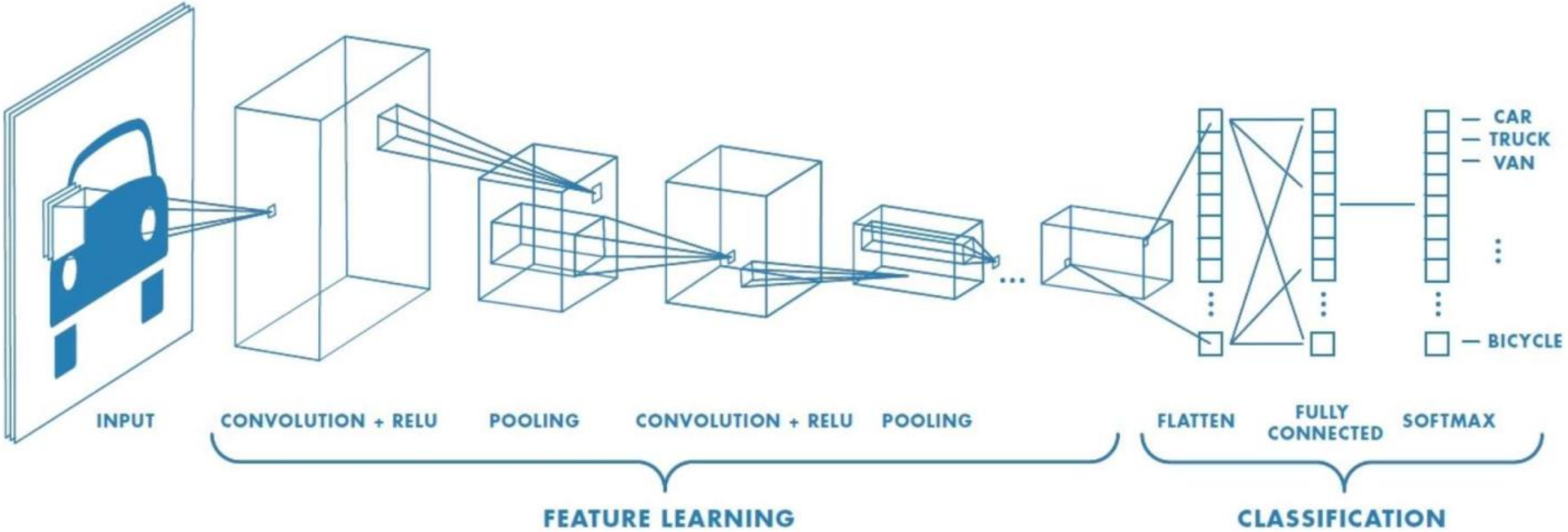
CNNs for classification: *Class probabilities*



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

CNNs: Training with backpropagation

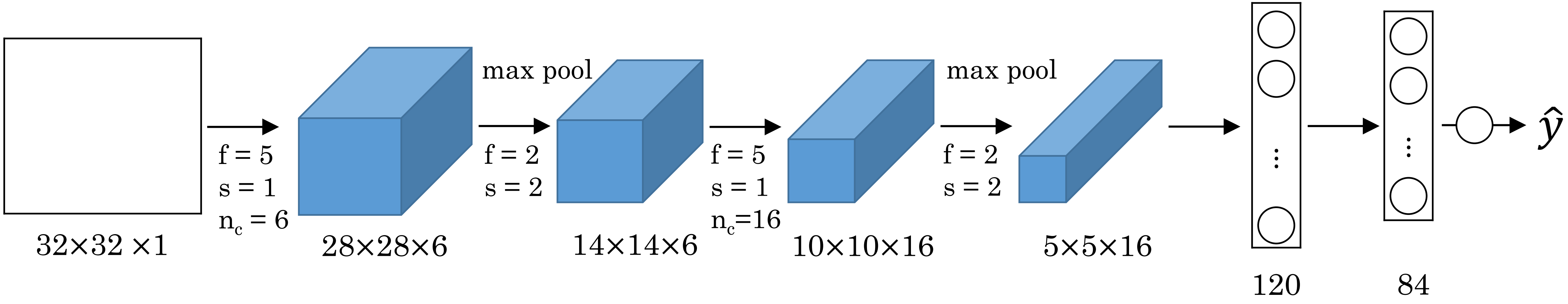


Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

$$\boldsymbol{J}(\boldsymbol{\theta}) = \sum_i y^{(i)} \log\left(\widehat{\boldsymbol{y}}^{(i)}\right)$$

CNN Example



32×32 ×1
f = 5
s = 1
n_c = 6
28×28×6
max pool
f = 2
s = 2
14×14×6
f = 5
s = 1
n_c=16
10×10×16
max pool
f = 2
s = 2
5×5×16
120
84
$\hat{y}$

CNN Example

	Activation Shape	Activation Size	# of parameters
Input	(32,32,3)	3072	
CONV 1 (f=5,s=1,n_c = 8)	(28,28,8)	6272	(5*5*3 + 1) * 8 = 608
POOL 1	(14,14,8)	1568	
CONV 2 (f=5,s=1, n_c= 16)	(10,10,16)	1600	(5*5*8 + 1) * 16 = 3216
POOL 2	(5,5,16)	400	
FC 3	(120,1)	120	400 * 120 + 120 = 48120
FC 4	(84,1)	84	120 * 84 + 84 = 10164
Softmax	(10,1)	10	84*10 + 10 = 850

Thanks