



# Deep Learning for Computer Vision

## MIT 6.S191

Ava Soleimany



# Vision: Evolutionary Origins

Vision: 540 million years of data

vs.

Bipedal movement: 230+ million years of data

Human language: ~100 thousand years of data



Opabinia, 540 mil. years ago

Can we understand neural basis of vision?

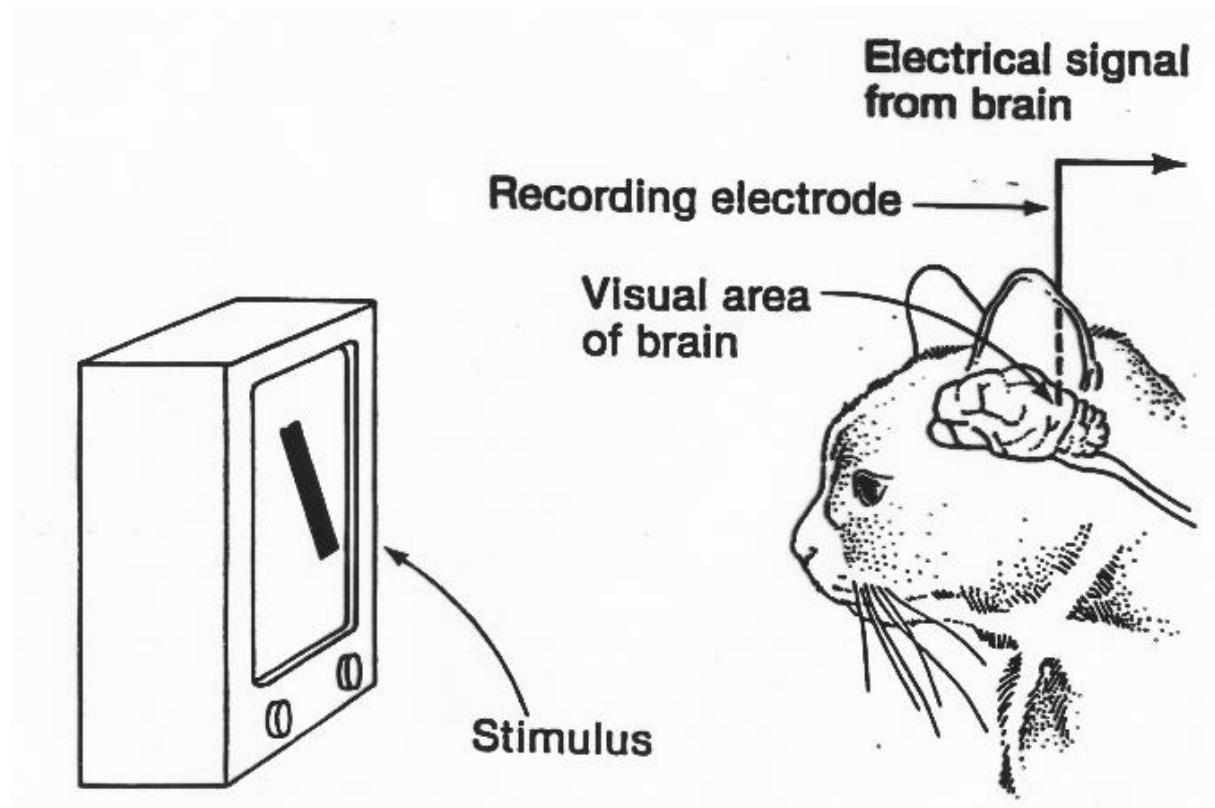
How is visual information processed?

Can we use that structure to inform computer vision?



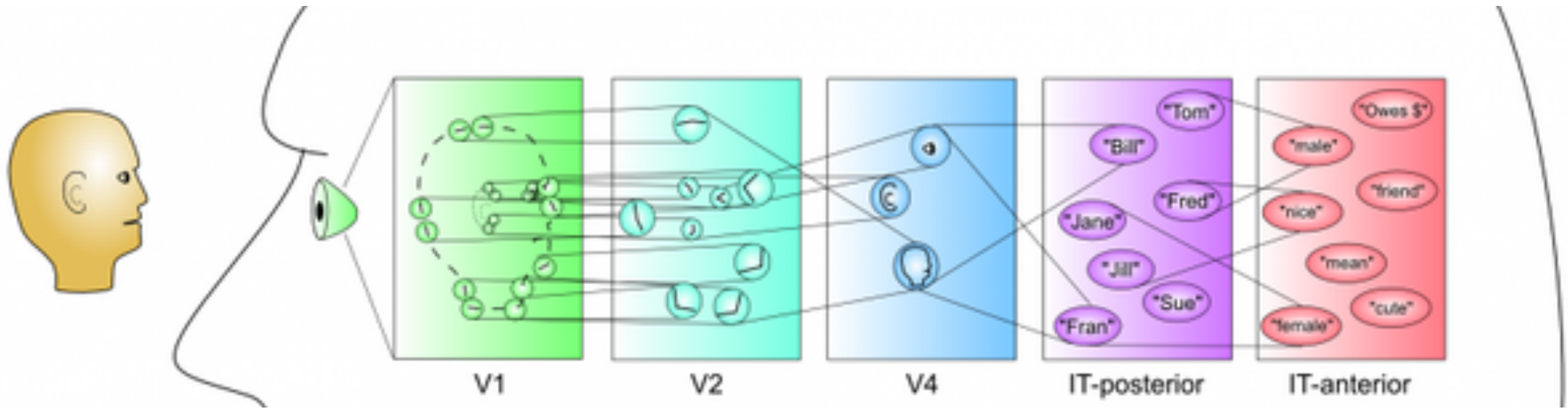
Hubel & Wiesel, Harvard. 1960s

# The Visual Cortex



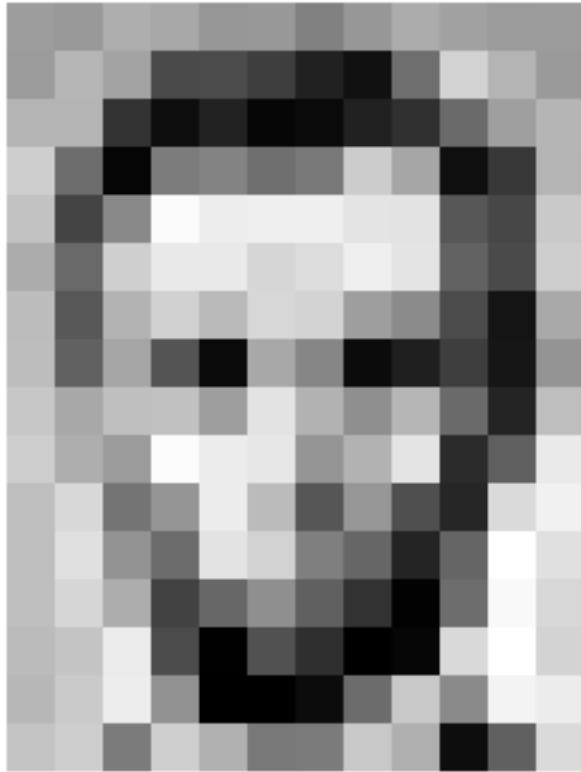
- 1) Spatial Invariance
- 2) Receptive Field
- 3) Hierarchy

# The Visual Cortex

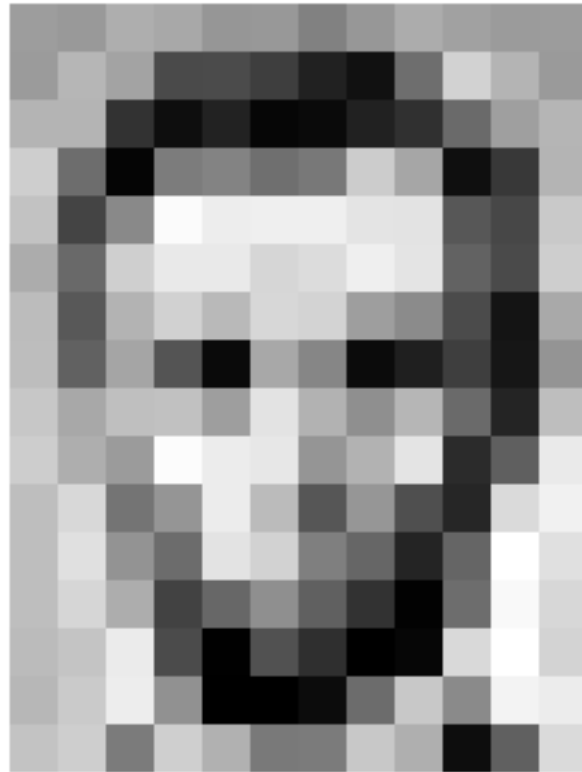


# What Computers “See”

# Images Are Numbers

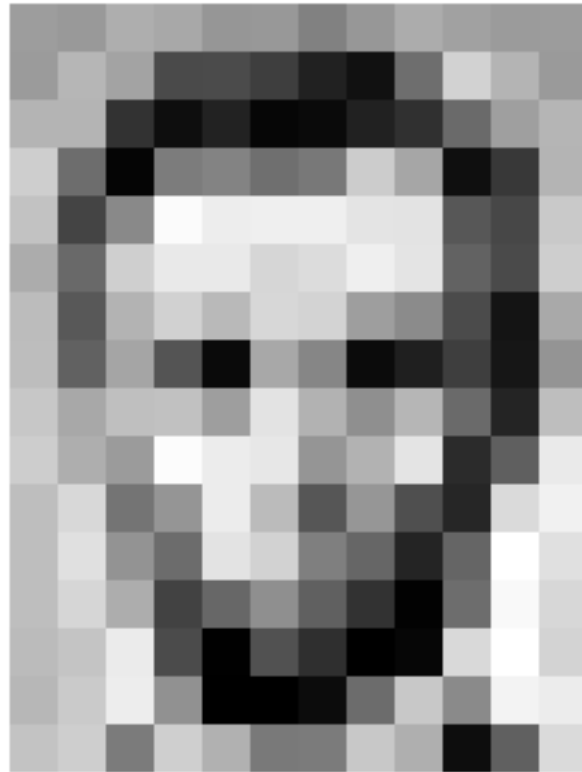


# Images Are Numbers



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	6	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	68	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	85	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	236	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

# 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	106	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

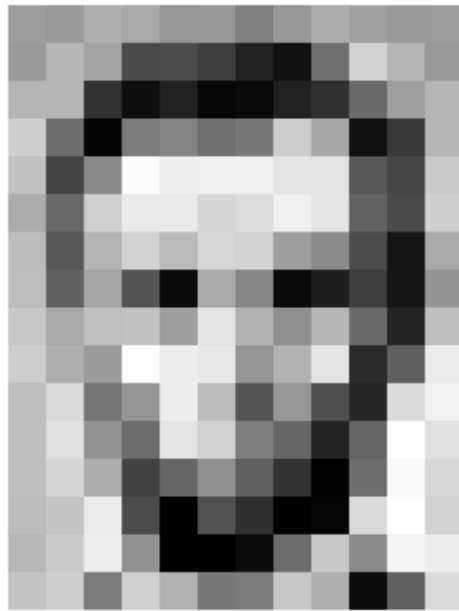
What the computer sees

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	106	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

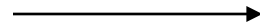
An image is just a matrix of numbers  $[0,255]$ !  
i.e.,  $1080 \times 1080 \times 3$  for an RGB image



# Tasks in Computer Vision



Input Image



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	96	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

Pixel Representation



classification

Lincoln

Washington

Jefferson

Obama

0.8
0.1
0.05
0.05

- **Regression:** output variable takes continuous value
- **Classification:** output variable takes class label. Can produce probability of belonging to a particular class

# High Level Feature Detection

Let's identify key features in each image category



Nose,  
Eyes,  
Mouth

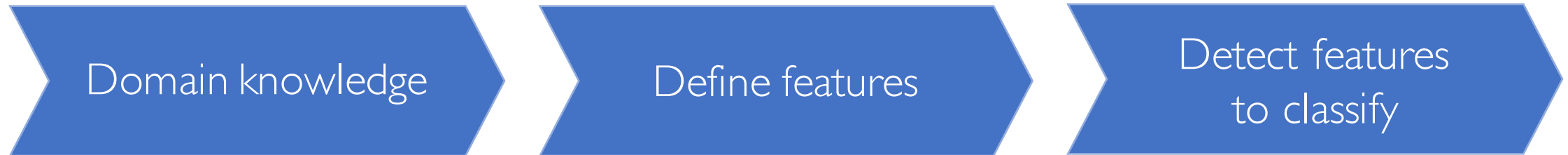


Wheels,  
License Plate,  
Headlights



Door,  
Windows,  
Steps

# Manual Feature Extraction



Problems?

# Manual Feature Extraction

Domain knowledge

Define features

Detect features to classify

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



Background clutter



Intra-class variation



# Manual Feature Extraction

Domain knowledge

Define features

Detect features  
to classify

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



Background clutter



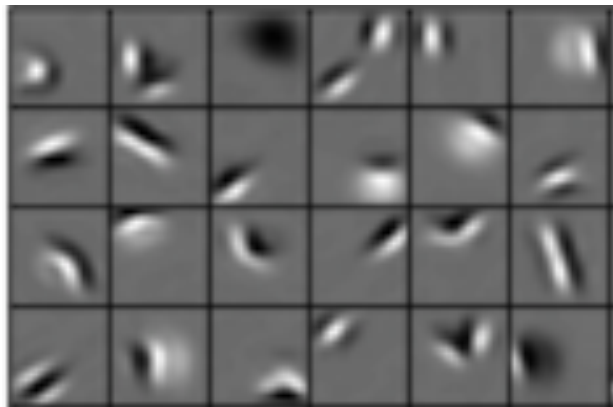
Intra-class variation



# Learning Feature Representations

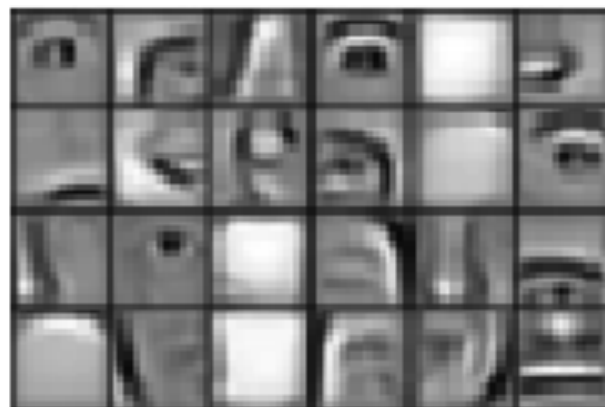
Can we **learn hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

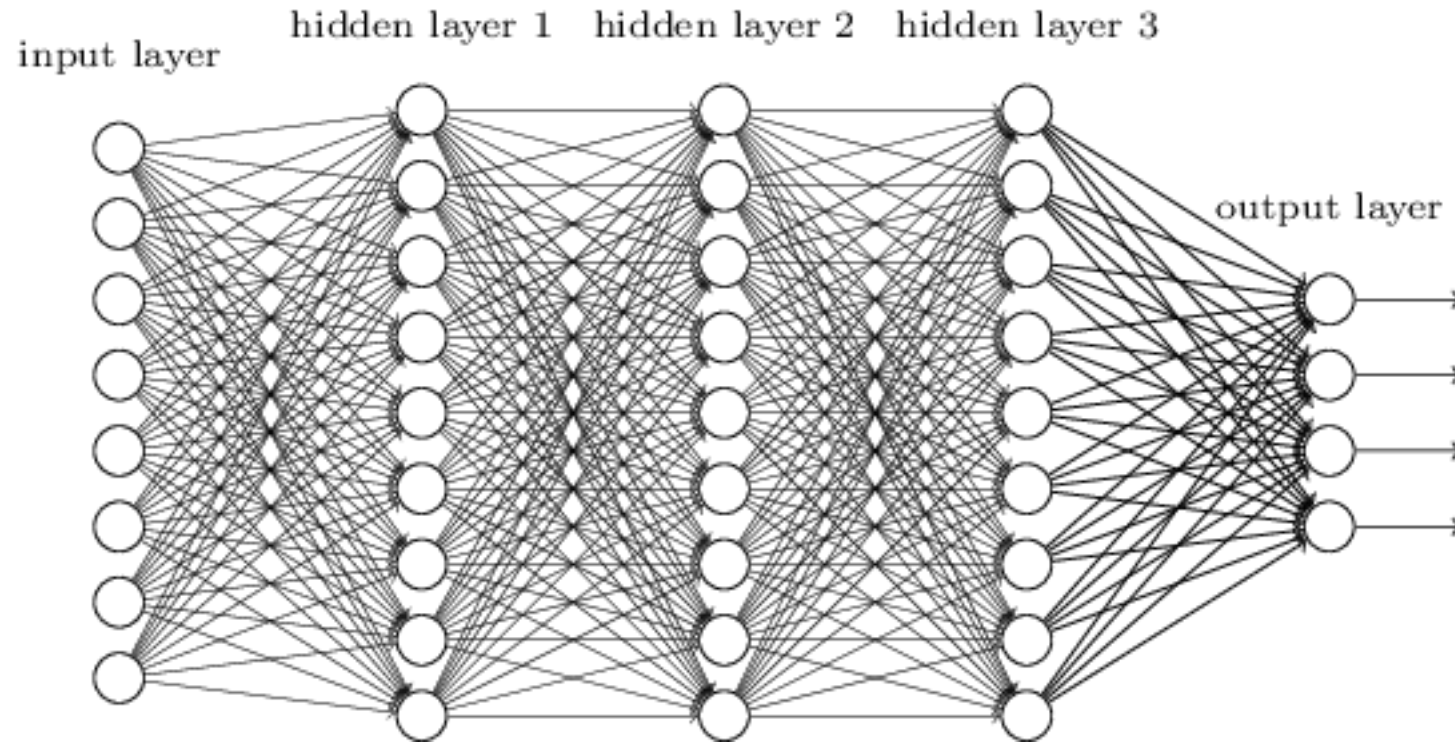
High level features



Facial structure

# Learning Visual Features

# Fully Connected Neural Network

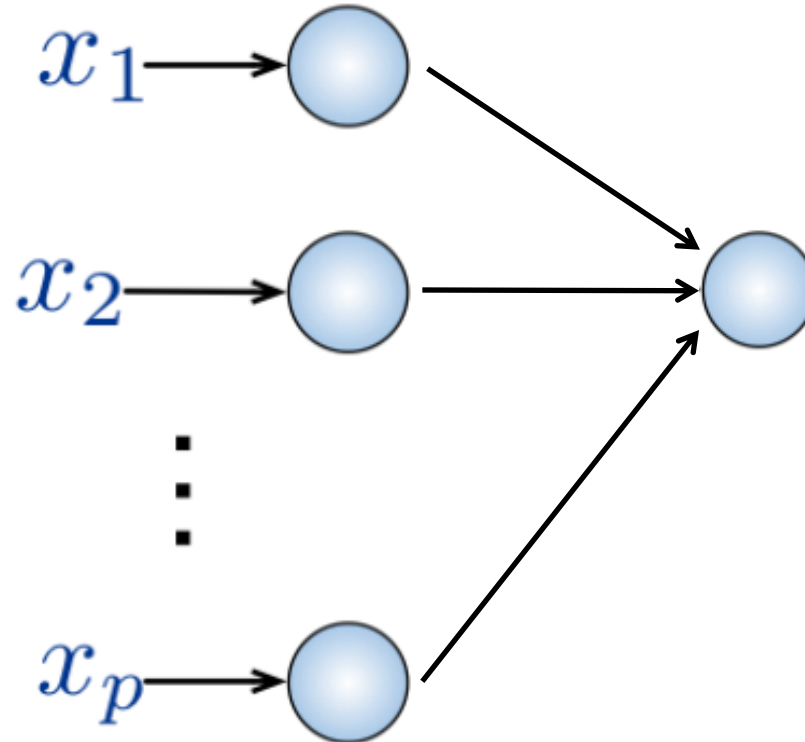




# Fully Connected Neural Network

## Input:

- 2D image
- Vector of pixel values



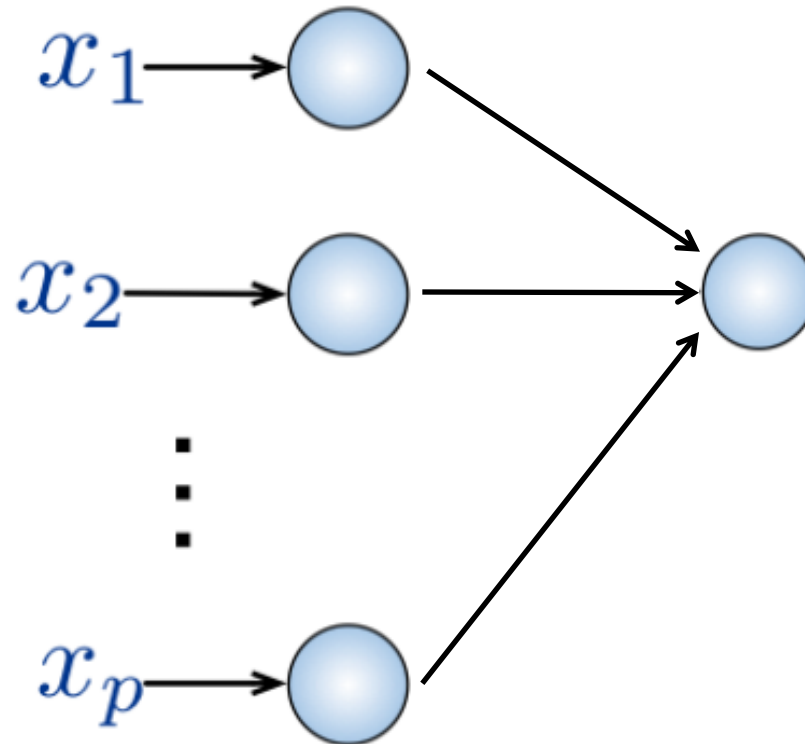
## Fully connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

# Fully Connected Neural Network

## Input:

- 2D image
- Vector of pixel values



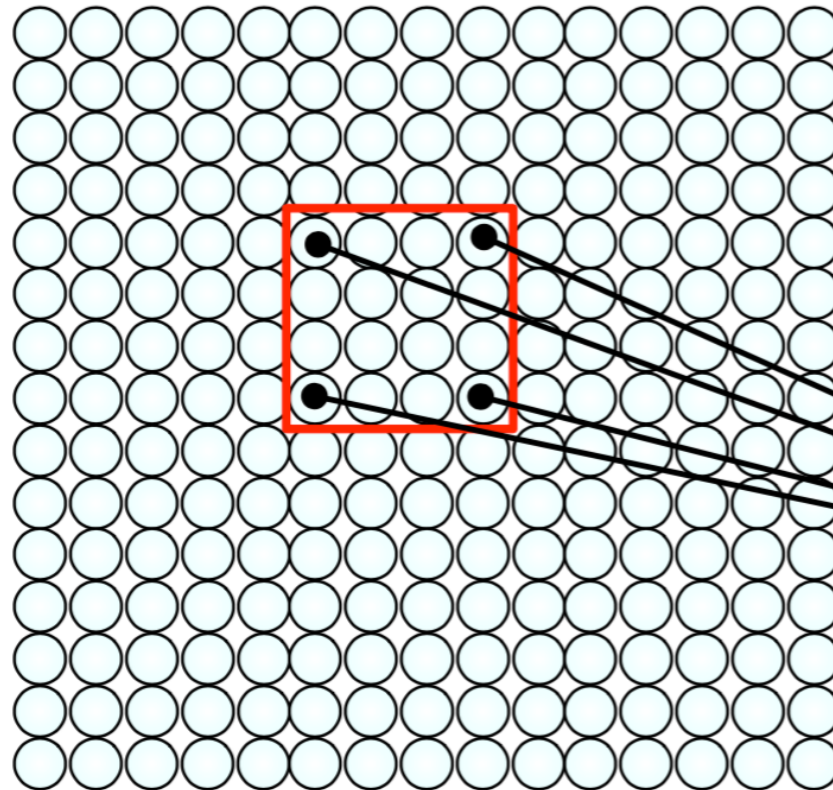
## Fully connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

# Using Spatial Structure

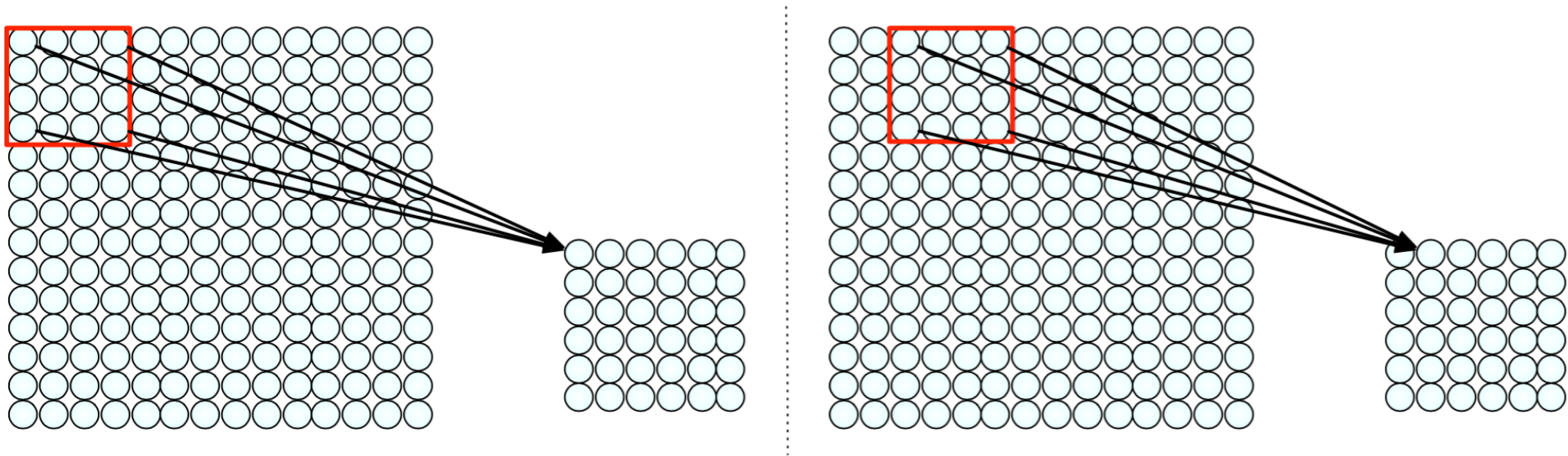
**Input:** 2D image.  
Array of pixel values



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

Neuron connected to region of  
input. Only “sees” these values.

# Using Spatial Structure



Connect patch in input layer to a single neuron in subsequent layer.

Use a sliding window to define connections.

*How can we **weight** the patch to detect particular features?*

# Applying Filters to Extract Features

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

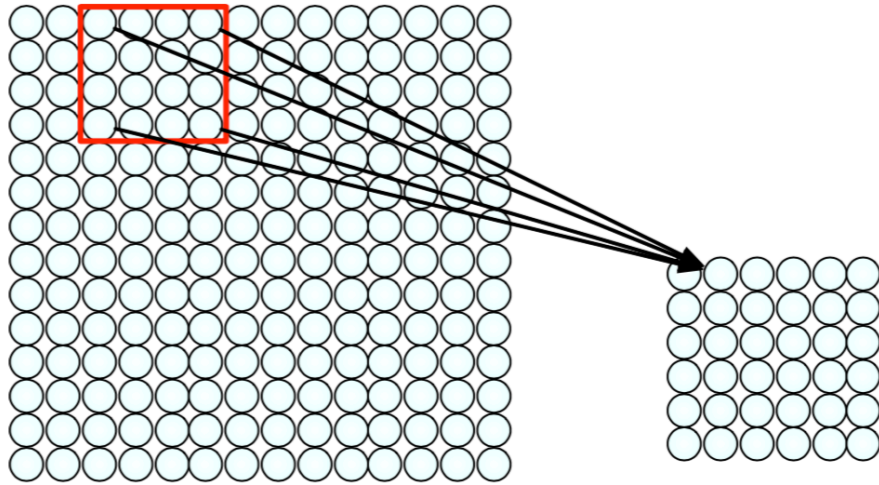
# Applying Filters to Extract Features

- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features

# Applying Filters to Extract Features

- 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  
(features that matter in one part of the input should matter elsewhere)

# 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



# Feature Extraction and Convolution

## A Case Study

# X or X?

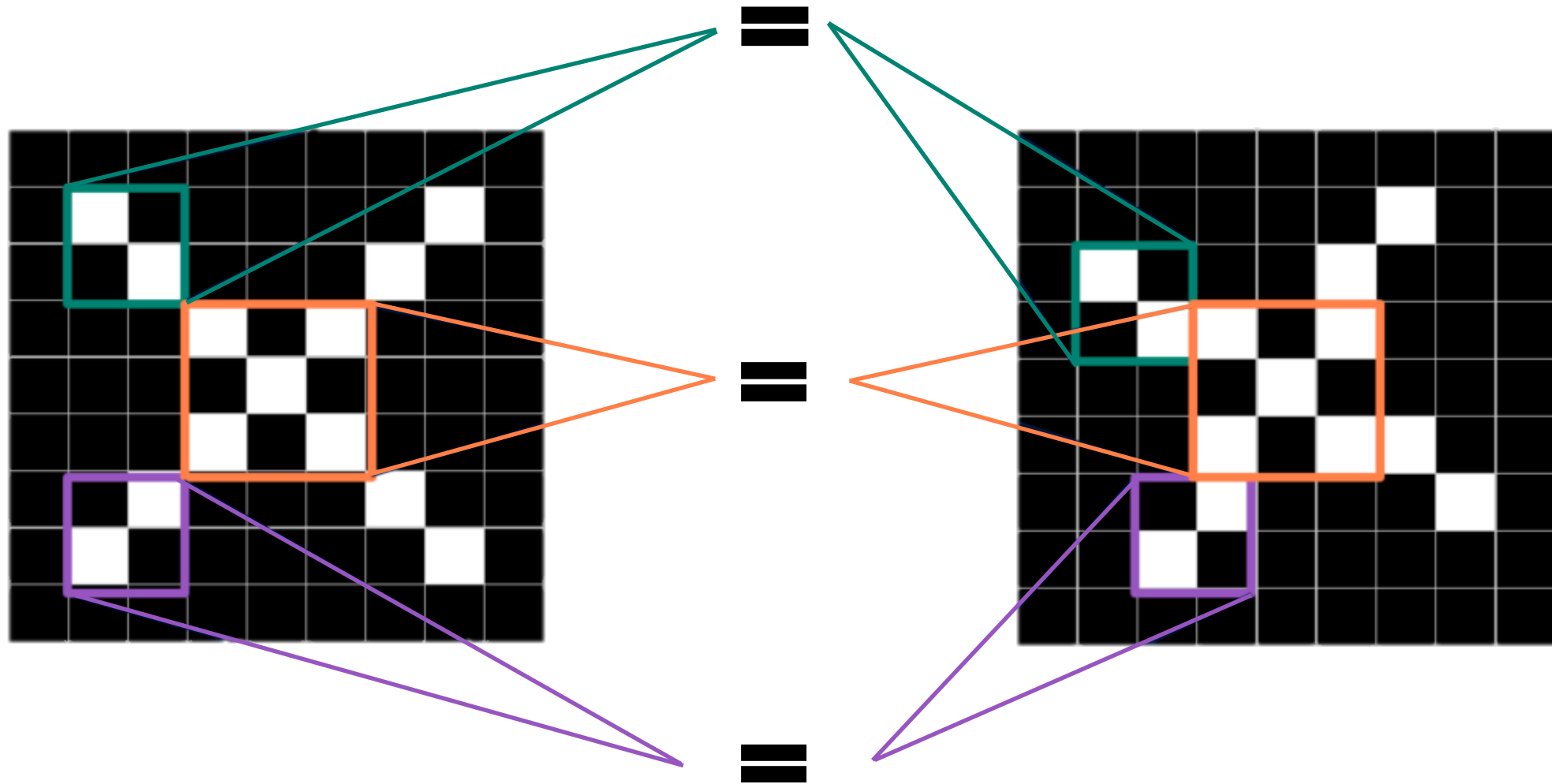
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Image is represented as matrix of pixel values... and computers are literal!  
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

# Features of X



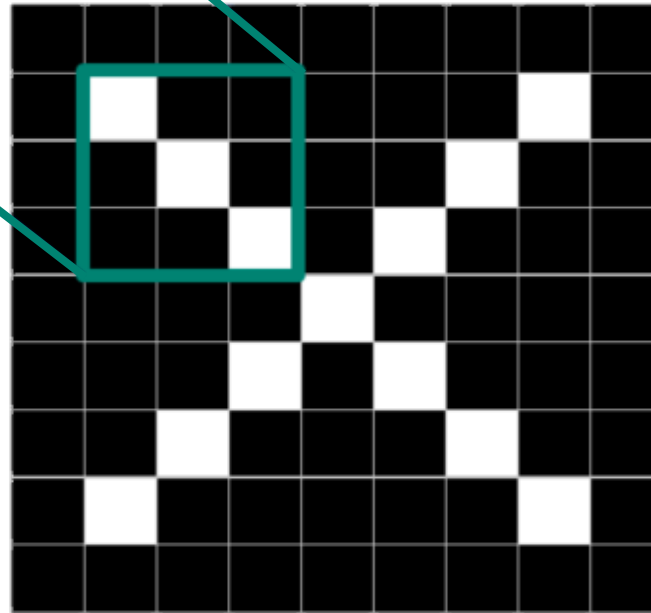
# Filters to Detect X Features

filters

1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1



# The Convolution Operation

1 -1 -1  
-1 1 -1  
-1 -1 1

1 x 1 = 1

The diagram shows a 3x3 kernel (top left) and a 3x3 input patch (middle left). The kernel has values 1, -1, -1 in the first row; -1, 1, -1 in the second row; and -1, -1, 1 in the third row. The input patch has values -1, -1, -1 in the first row; -1, 1, -1 in the second row; and -1, -1, 1 in the third row. A red box highlights the value 1 in the top-right corner of the kernel, and a yellow box highlights the value 1 in the bottom-right corner of the input patch. Lines connect these two boxes to the equation 1 x 1 = 1.

-1	-1	-1	1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



1	1	1
1	1	1
1	1	1

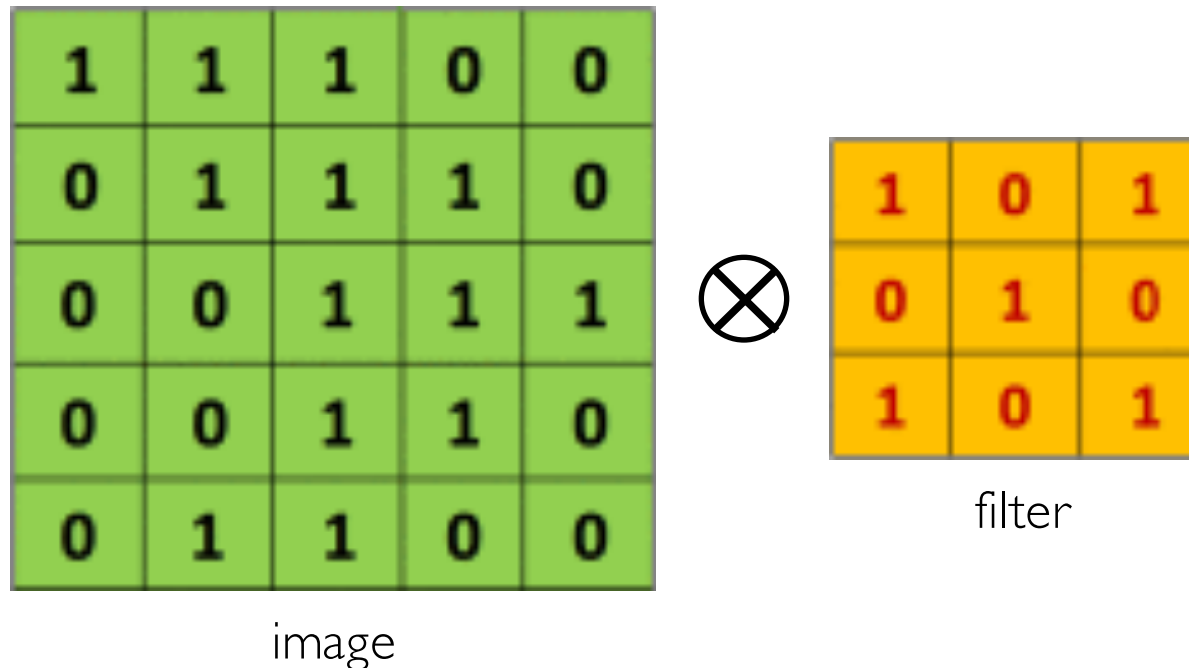
= 9

Element wise multiply

Add outputs

# The Convolution Operation

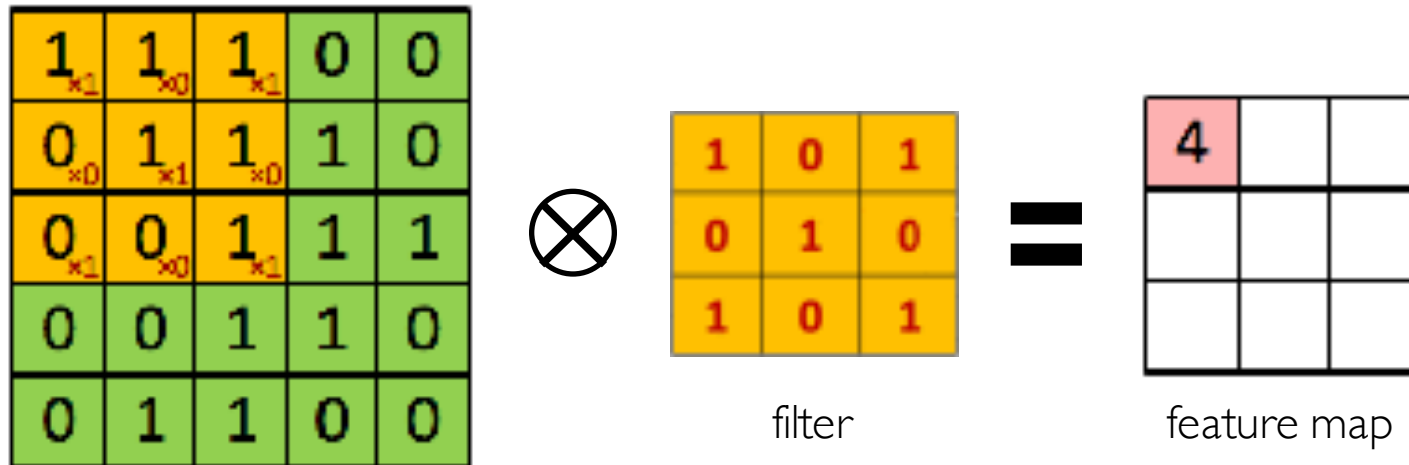
Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:



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

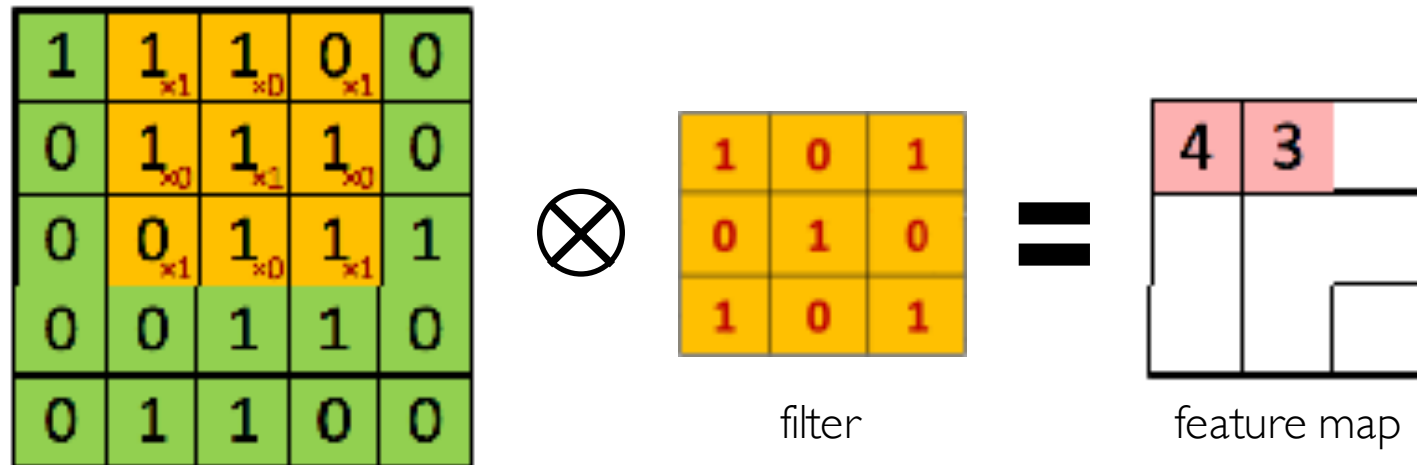
# The Convolution Operation

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



# The Convolution Operation

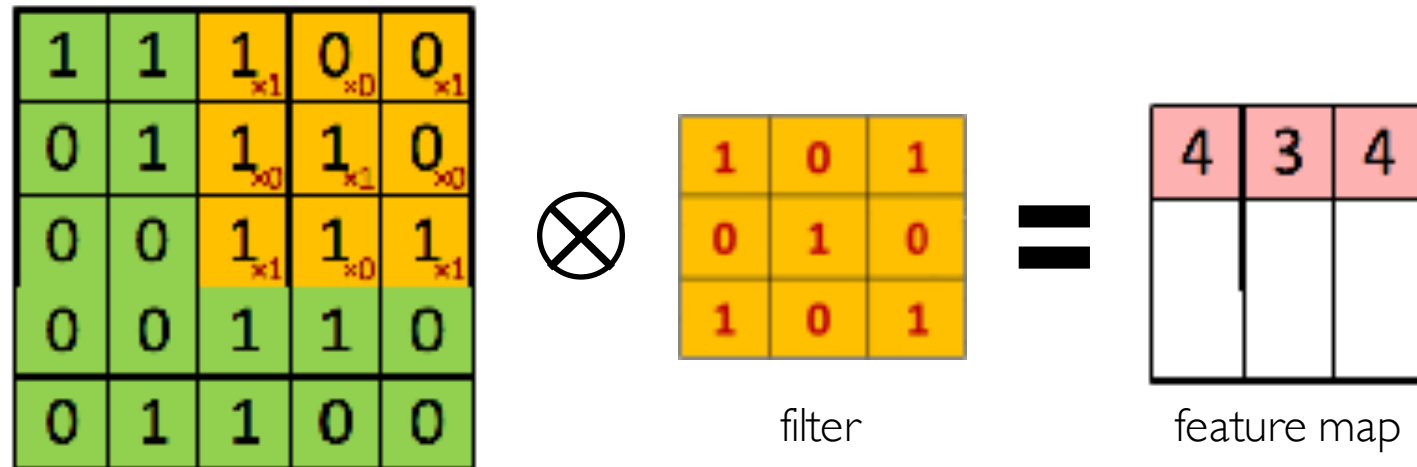
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:





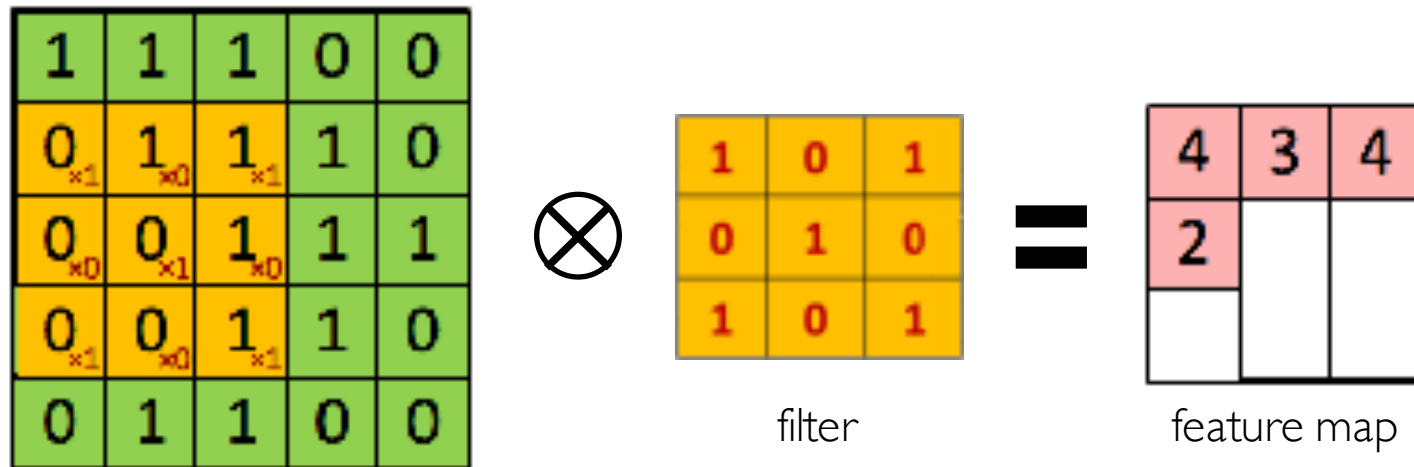
# The Convolution Operation

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



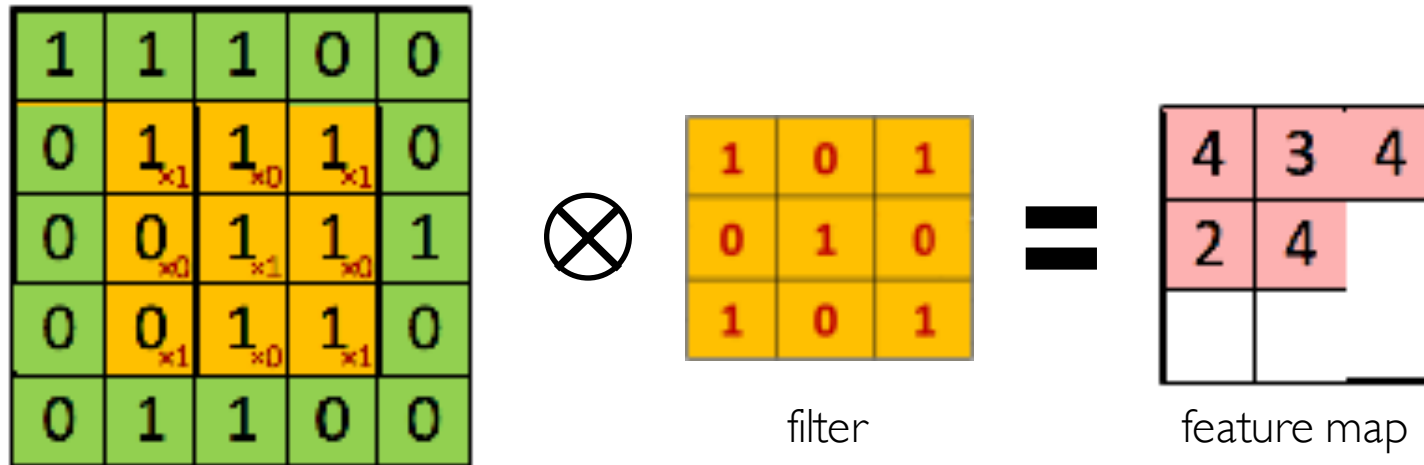
# The Convolution Operation

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



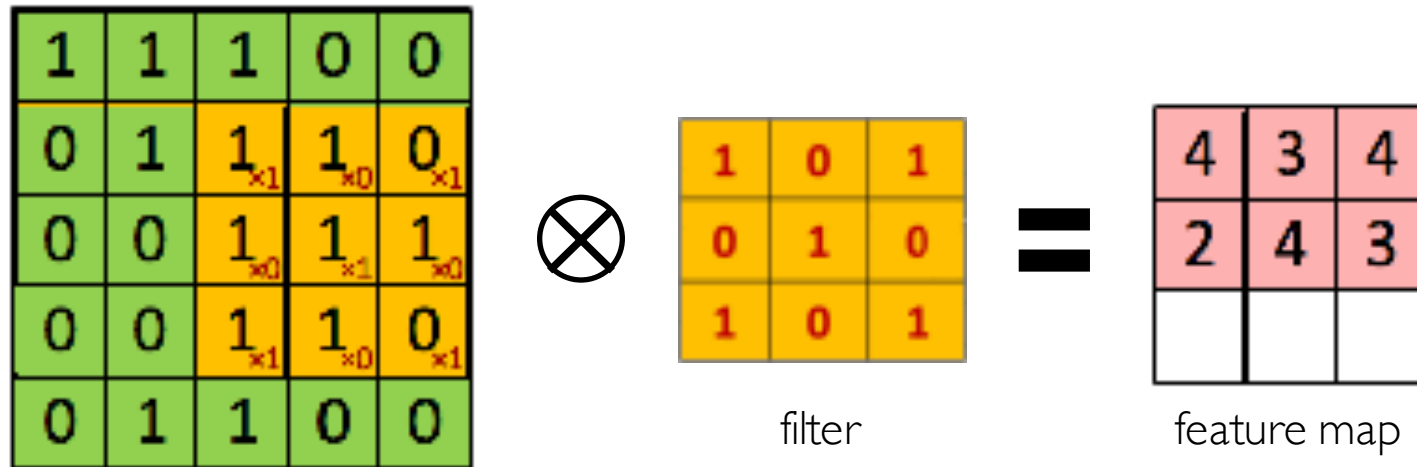
# The Convolution Operation

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



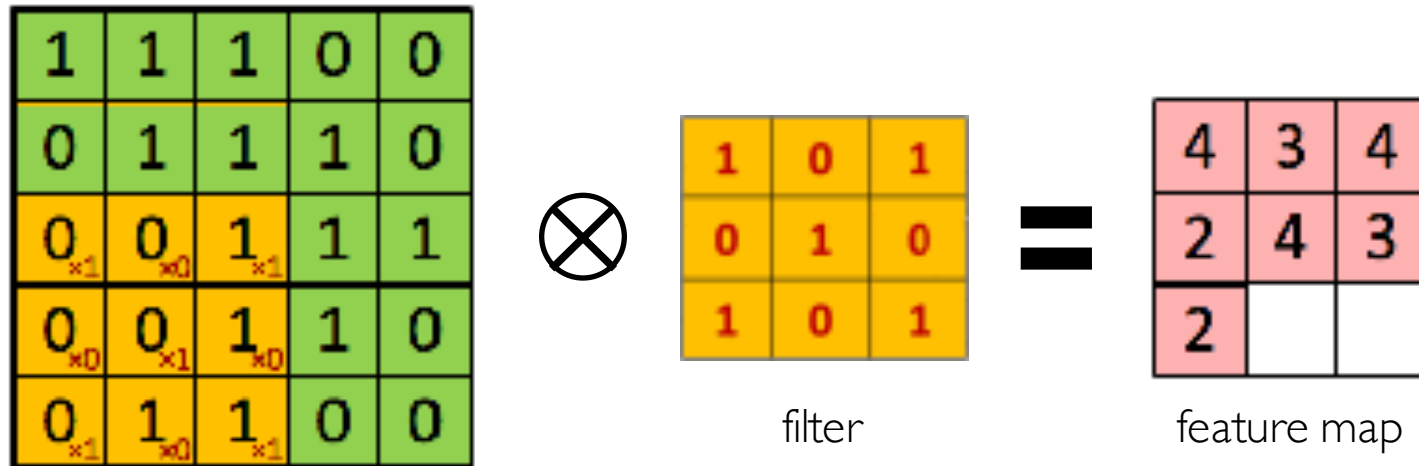
# The Convolution Operation

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



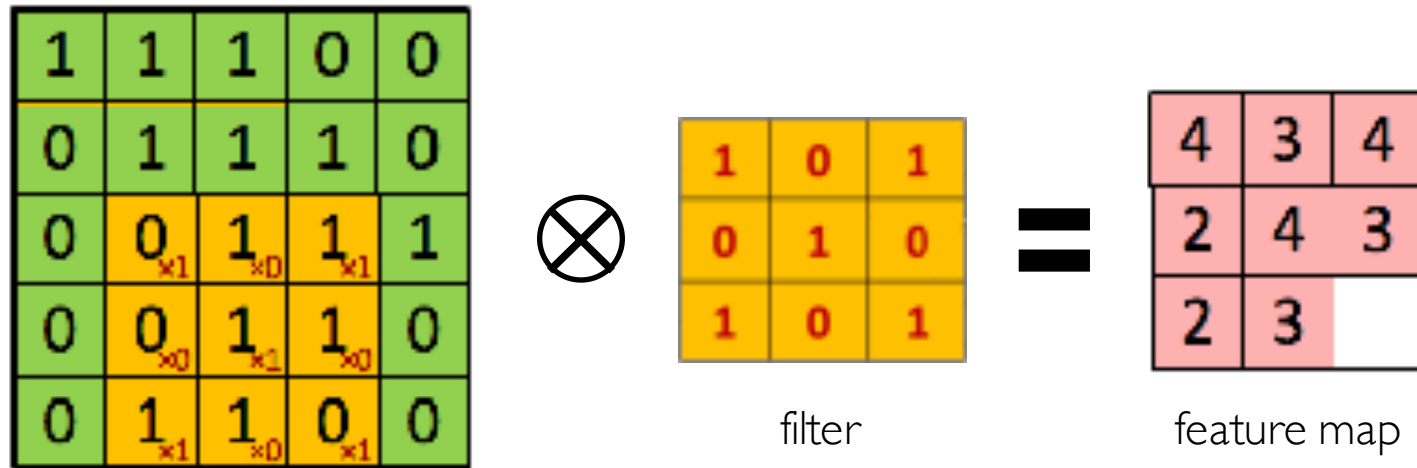
# The Convolution Operation

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



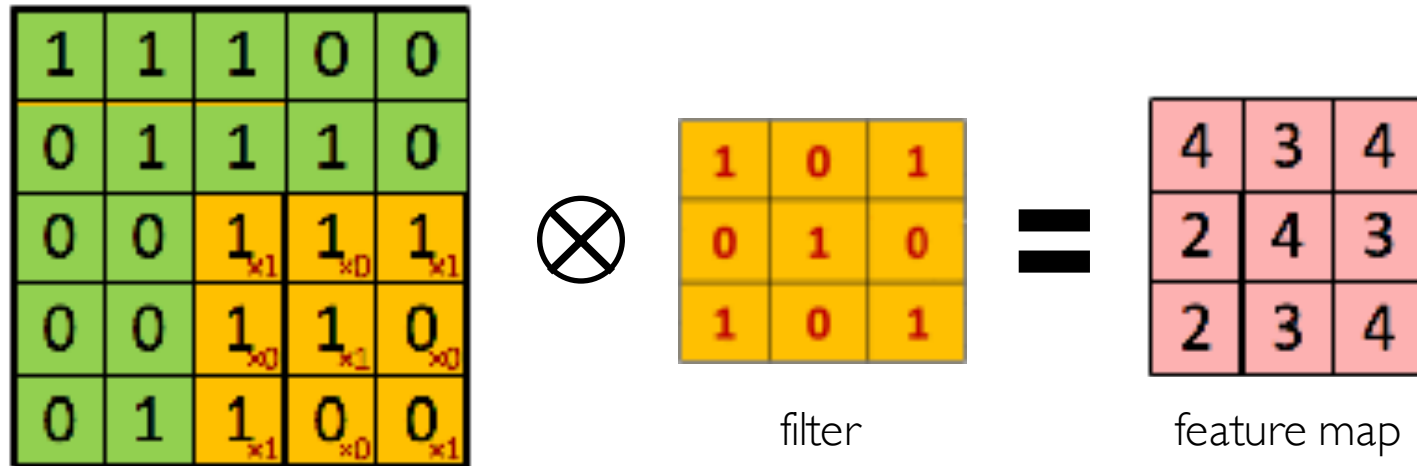
# The Convolution Operation

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:



# Producing Feature Maps



Original



Sharpen



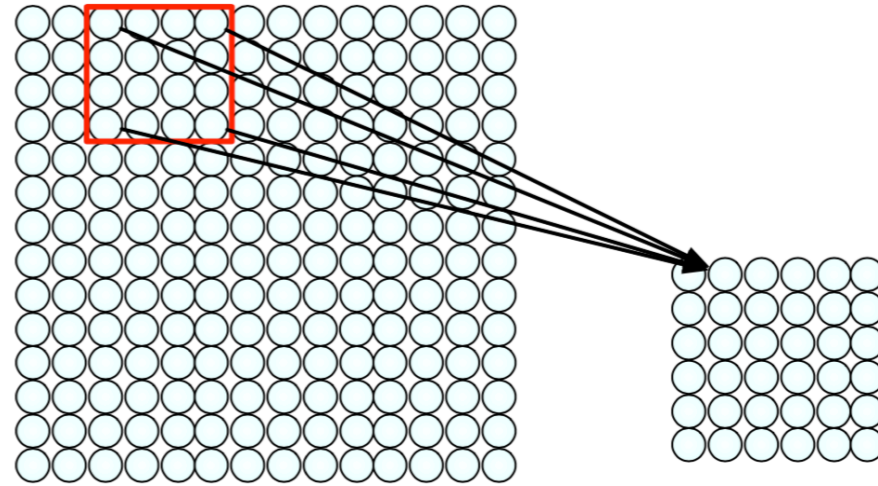
Edge Detect



“Strong” Edge Detect



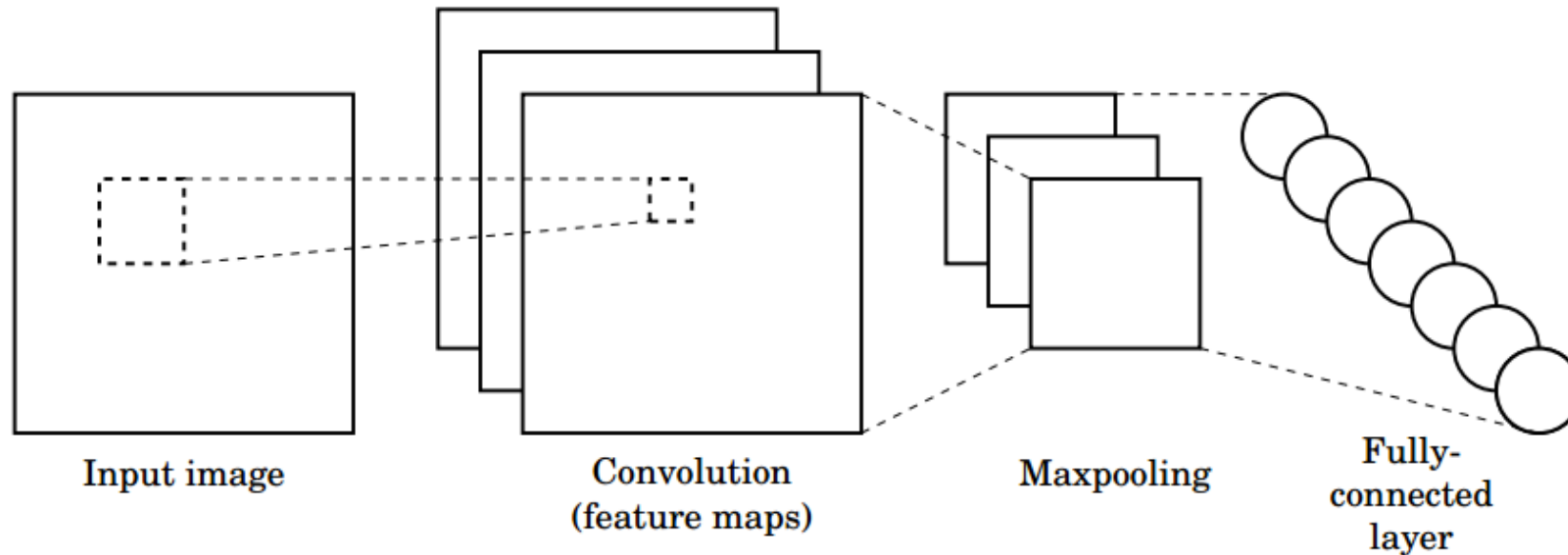
# Feature Extraction with Convolution



- 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

# Convolutional Neural Networks (CNNs)

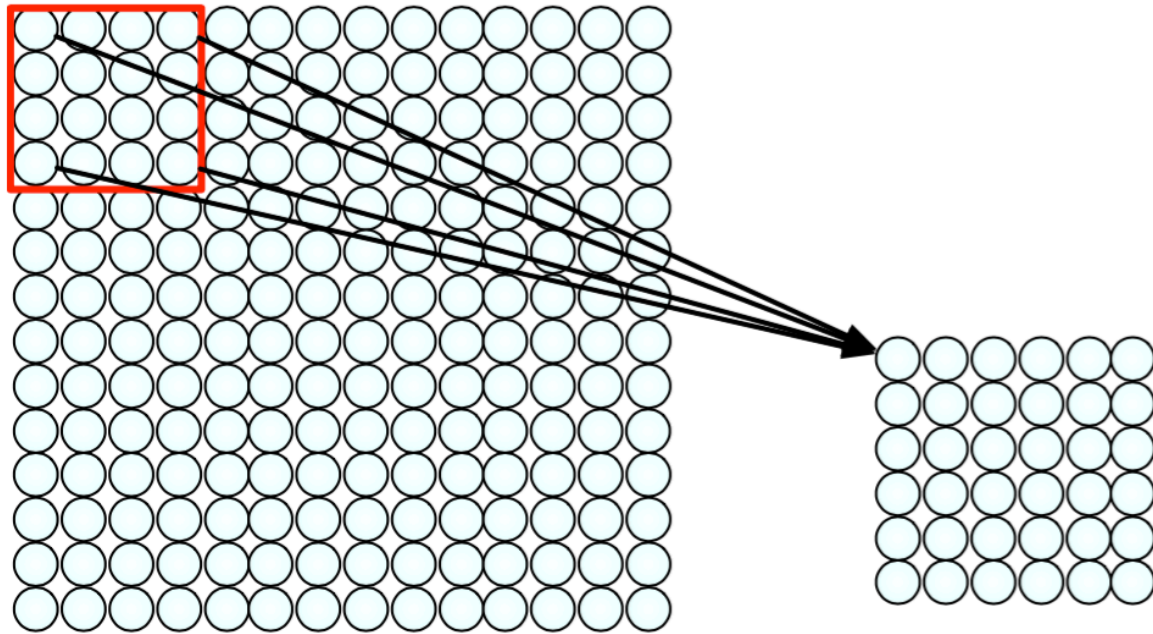
# CNNs for Classification



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.**

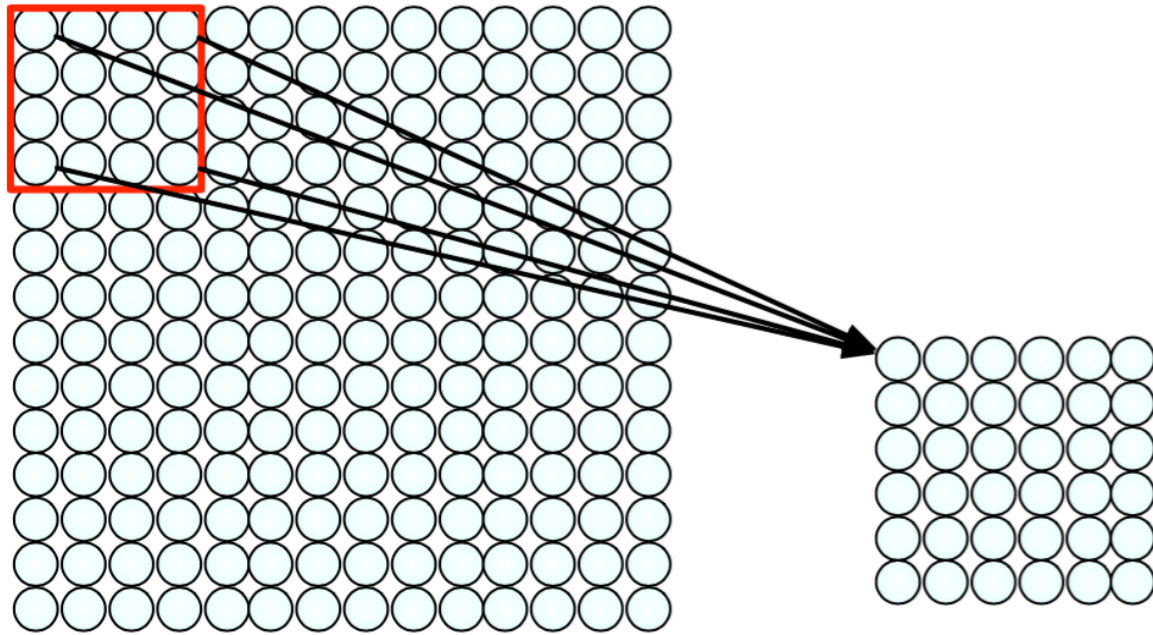
# Convolutional Layers: Local Connectivity



For a neuron in hidden layer:

- Take inputs from patch the neuron “sees”
- Compute weighted sum
- Apply bias

# Convolutional Layers: Local Connectivity



For a neuron in hidden layer:

- Take inputs from patch the neuron “sees”
- Compute weighted sum
- Apply bias

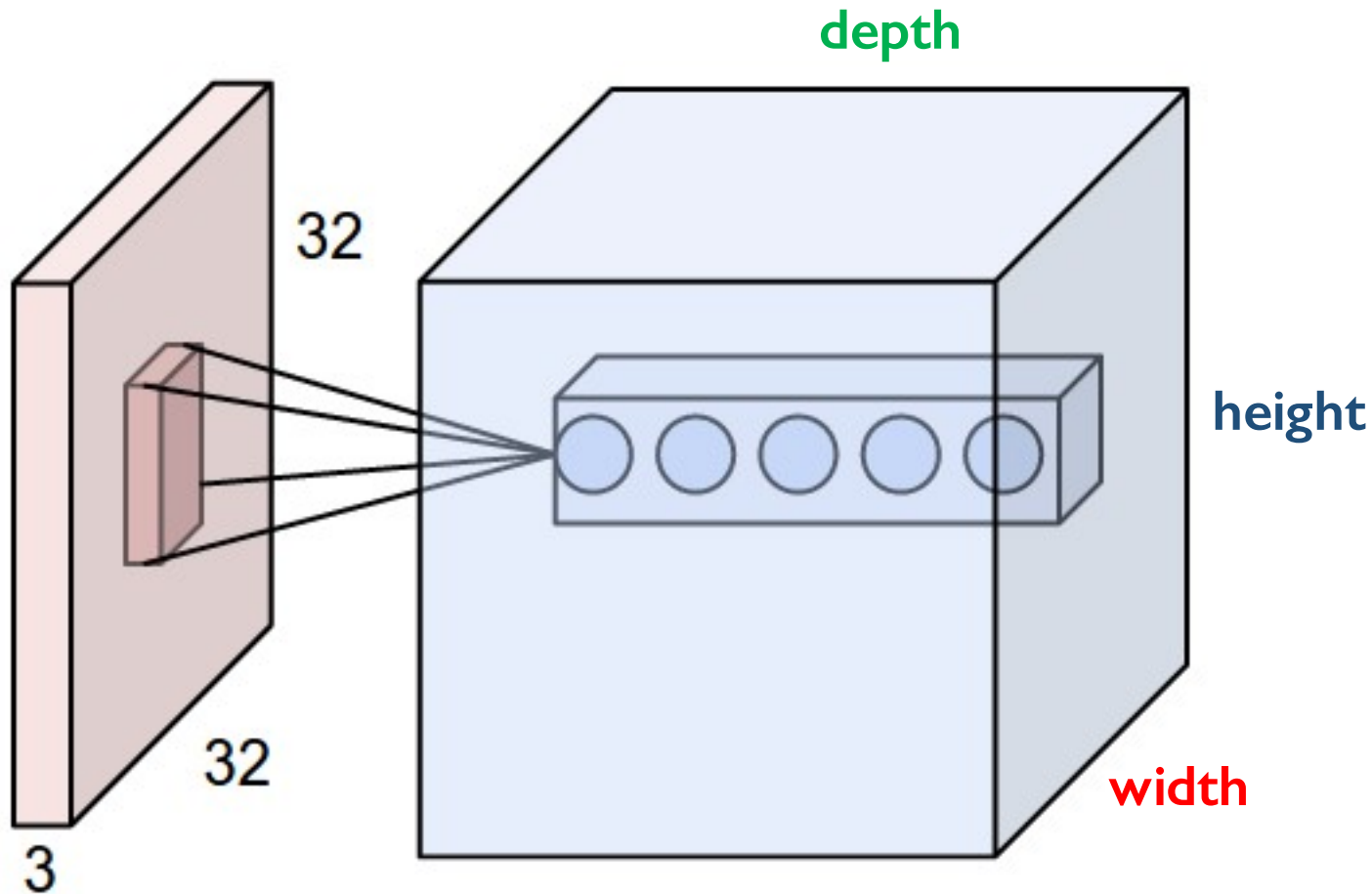
4x4 filter: matrix  
of weights  $\theta_{ij}$

$$\sum_{i=1}^4 \sum_{j=1}^4 \theta_{ij} x_{i+p, j+q} + b$$

for neuron (p,q) in hidden layer

applying a window of weights  
computing linear combinations  
activating with non-linear function

# CNNs: Spatial Arrangement of Output Volume



**Layer Dimensions:**

$$h \times w \times d$$

where  $h$  and  $w$  are spatial dimensions  
 $d$  (depth) = number of filters

**Stride:**

Filter step size

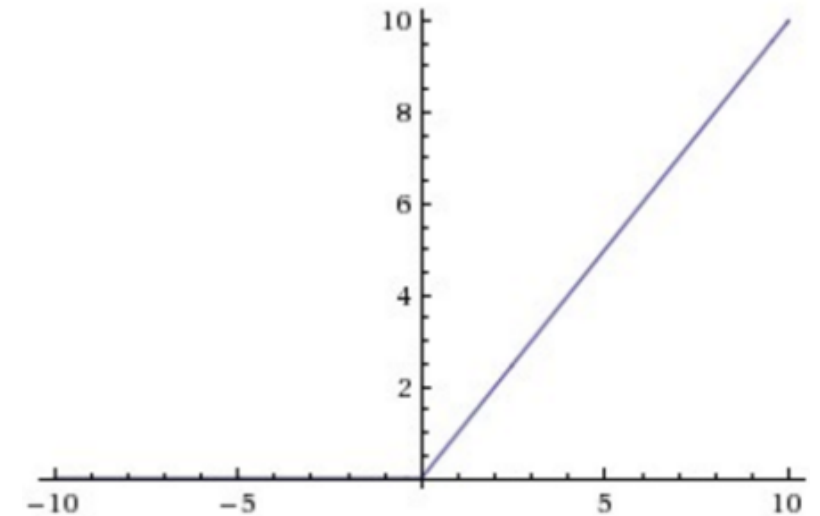
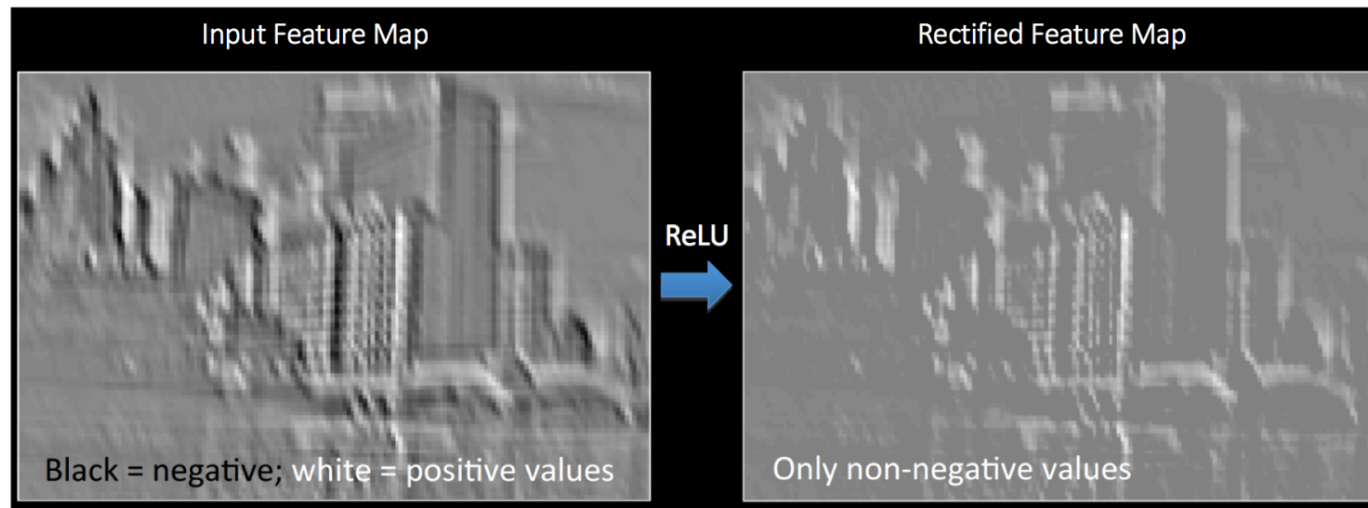
**Receptive Field:**

Locations in input image that  
a node is path connected to

# Introducing Non-Linearity

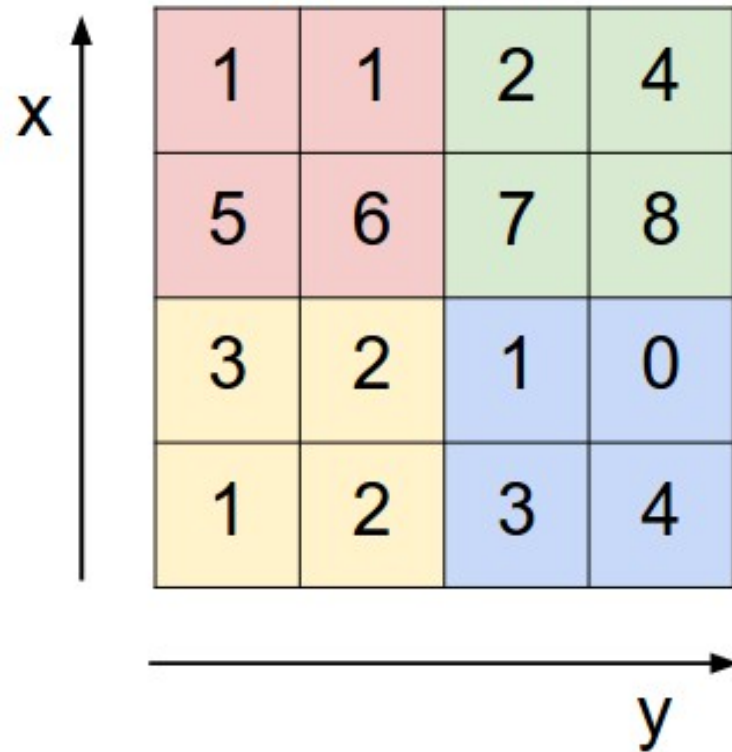
- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation**

Rectified Linear Unit (ReLU)



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

# Pooling



max pool with 2x2 filters  
and stride 2



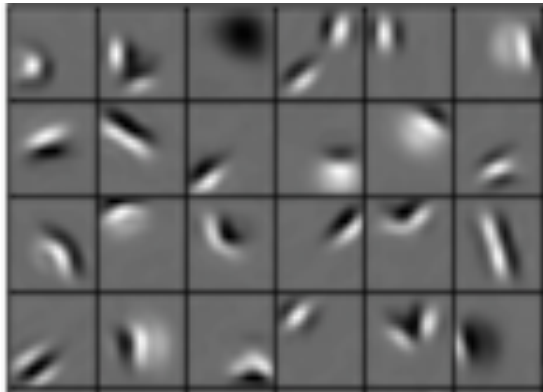
- 1) Reduced dimensionality
- 2) Spatial invariance

How else can we downsample and preserve spatial invariance?



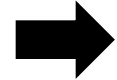
# Representation Learning in Deep CNNs

Low level features

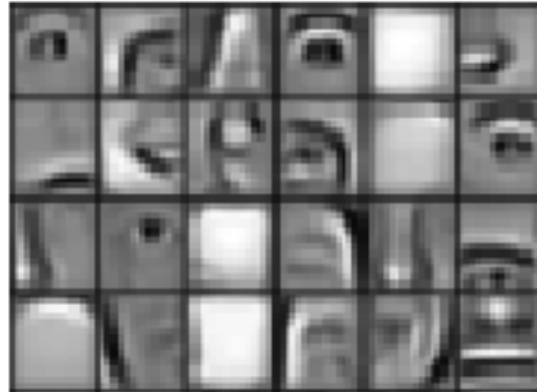


Edges, dark spots

Conv Layer 1

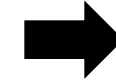


Mid level features



Eyes, ears, nose

Conv Layer 2



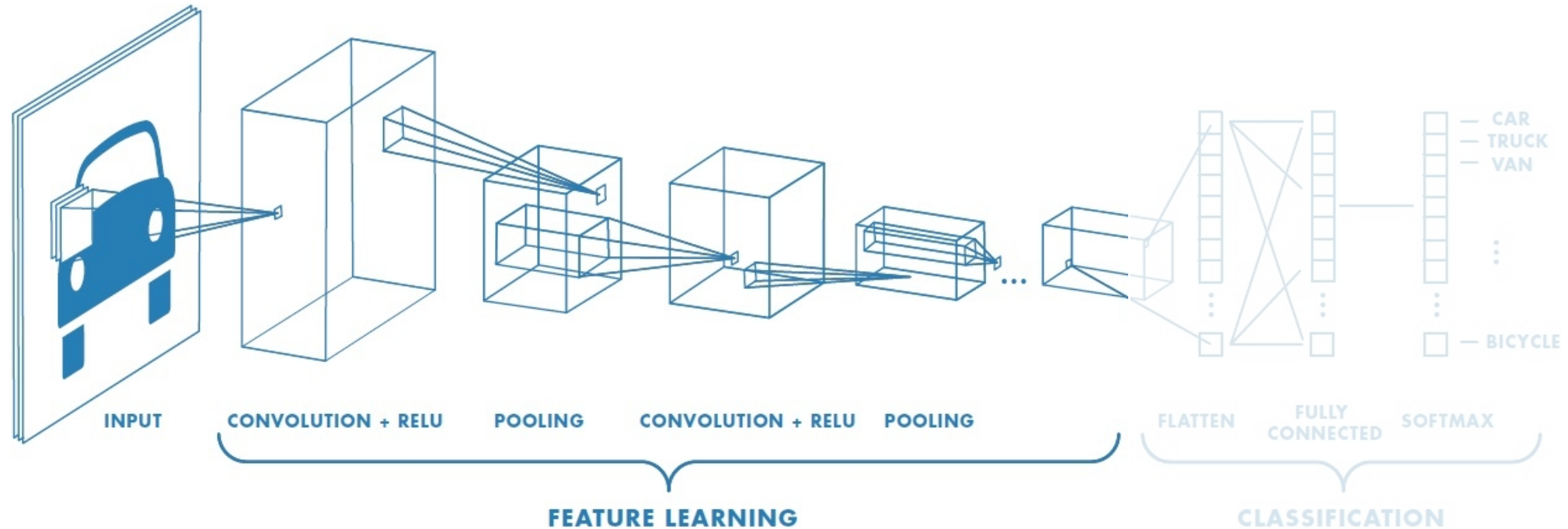
High level features



Facial structure

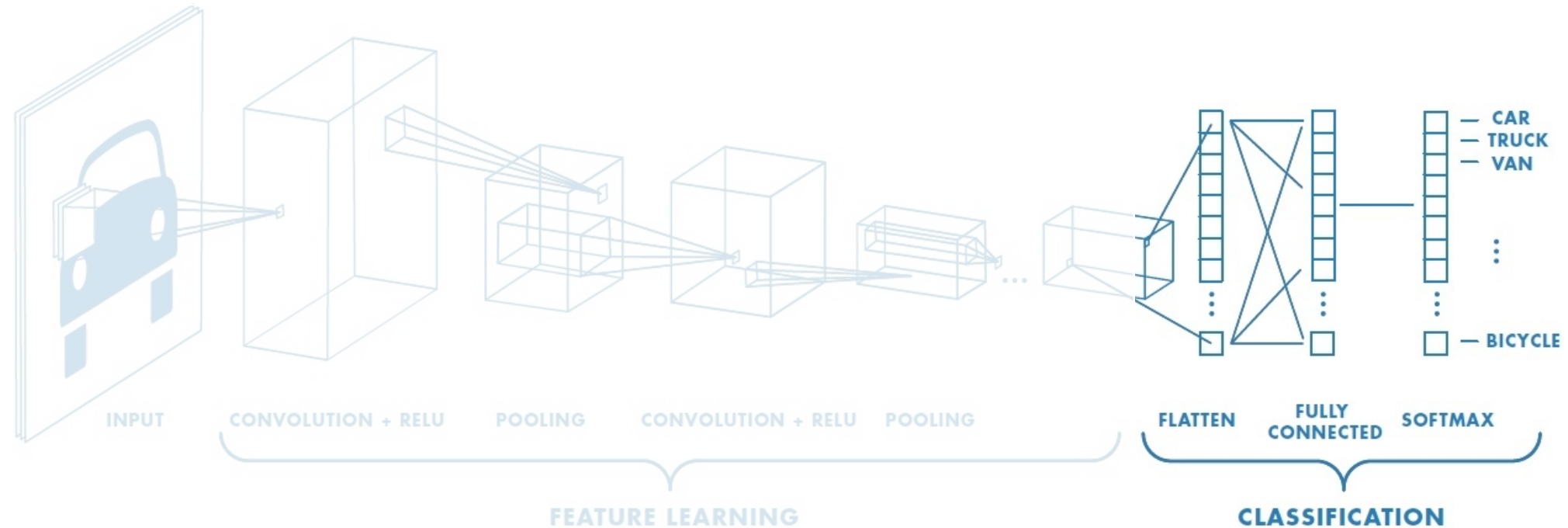
Conv Layer 3

# 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**

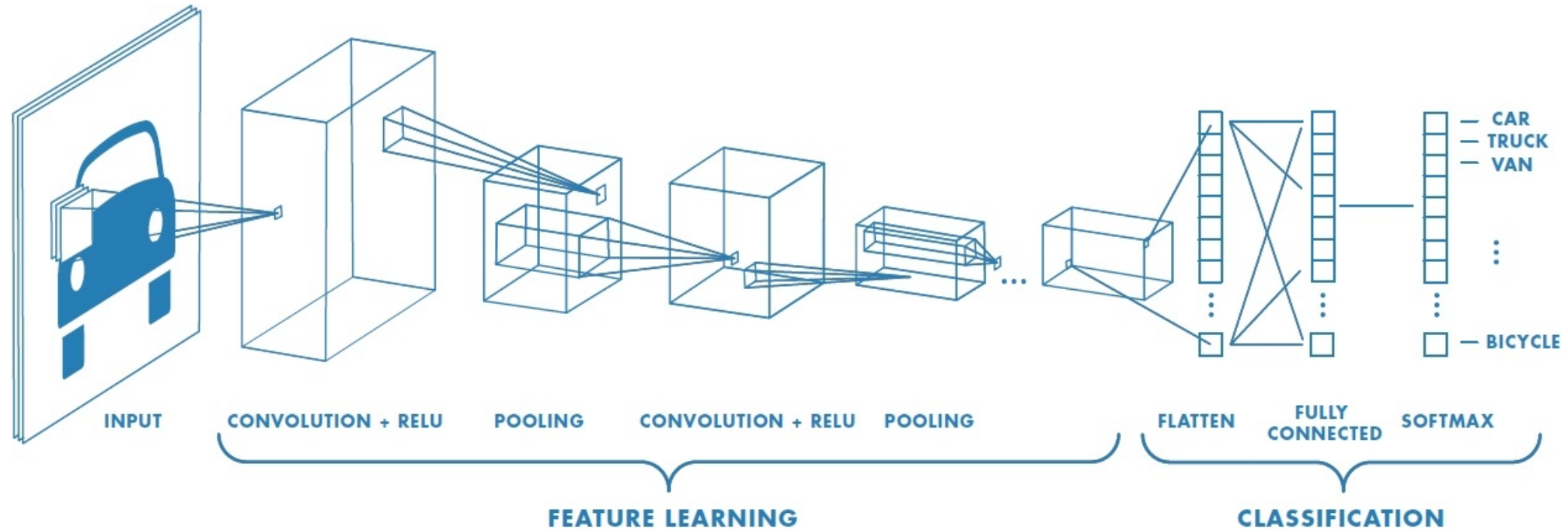
# CNNs for Classification: Class Probabilities



- 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

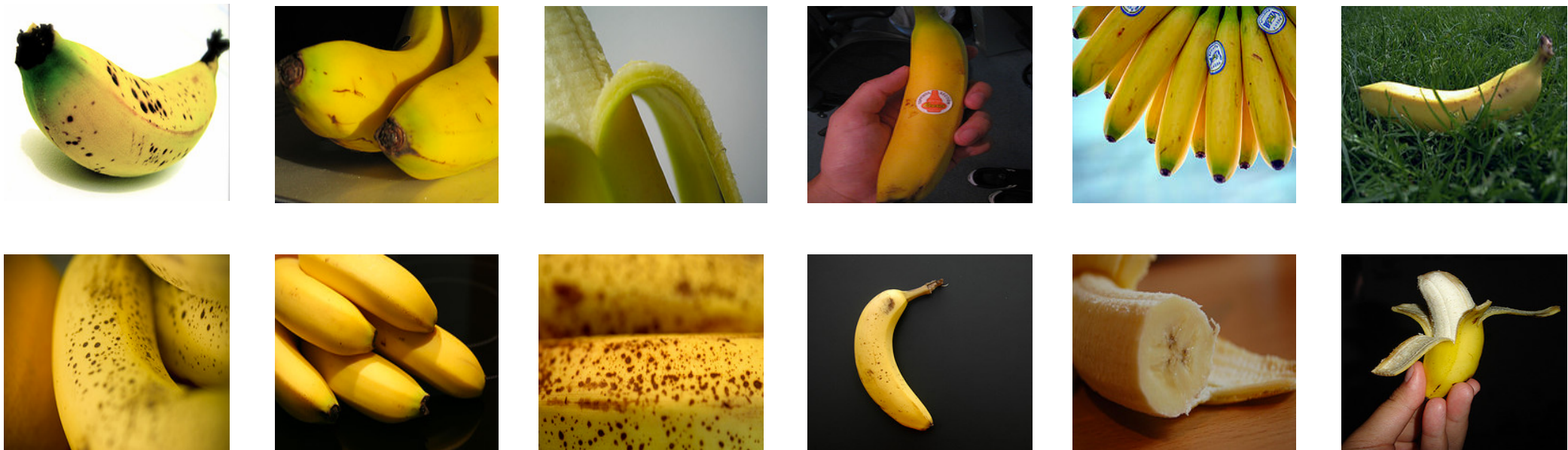
$$J(\theta) = \sum_i y^{(i)} \log(\hat{y}^{(i)})$$

# CNNs for Classification: ImageNet

# ImageNet Dataset

Dataset of over 14 million images across 21,841 categories

*“Elongated crescent-shaped yellow fruit with soft sweet flesh”*



1409 pictures of bananas.

# ImageNet Challenge

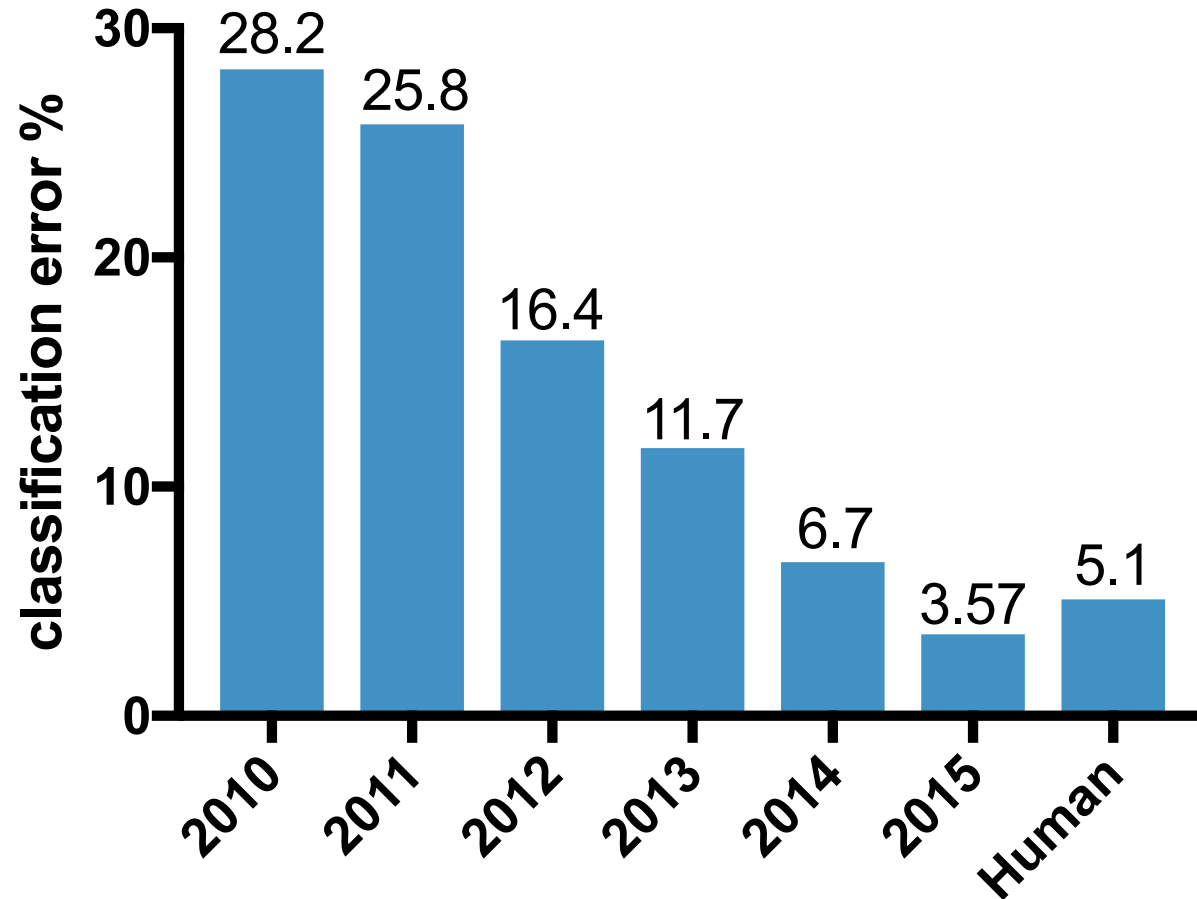


**Classification task:** produce a list of object categories present in image. 1000 categories.  
“Top 5 error”: rate at which the model does not output correct label in top 5 predictions

Other tasks include:

single-object localization, object detection from video/image, scene classification, scene parsing

# ImageNet Challenge: Classification Task



2012: AlexNet. First CNN to win.  
- 8 layers, 61 million parameters

2013: ZFNet  
- 8 layers, more filters

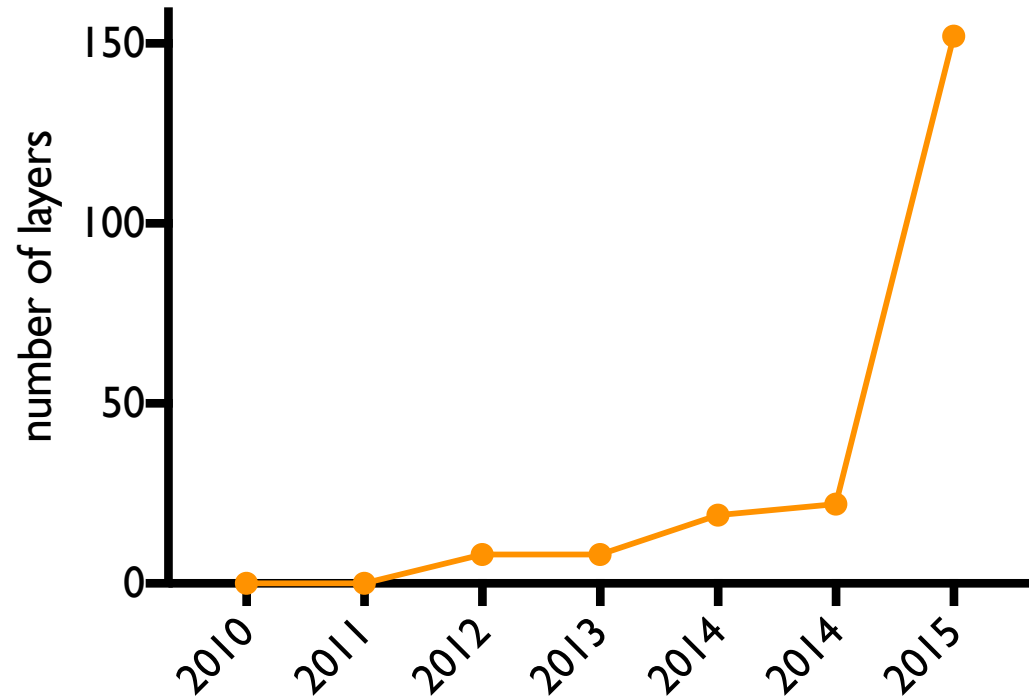
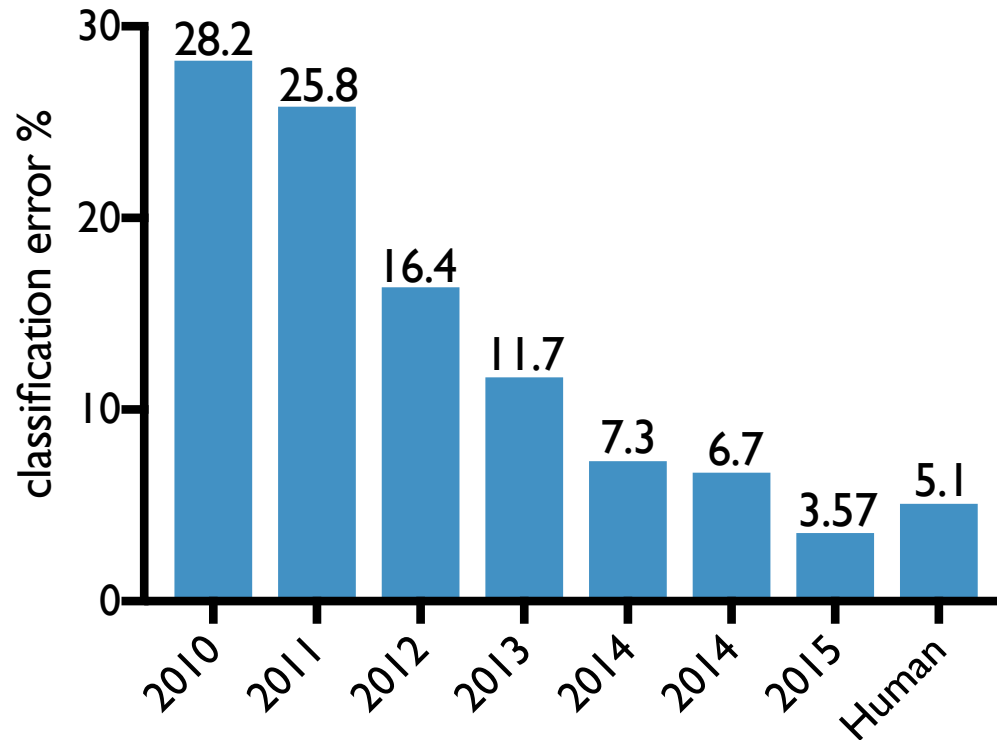
2014: VGG  
- 19 layers

2014: GoogLeNet  
- “Inception” modules  
- 22 layers, 5 million parameters

2015: ResNet  
- 152 layers

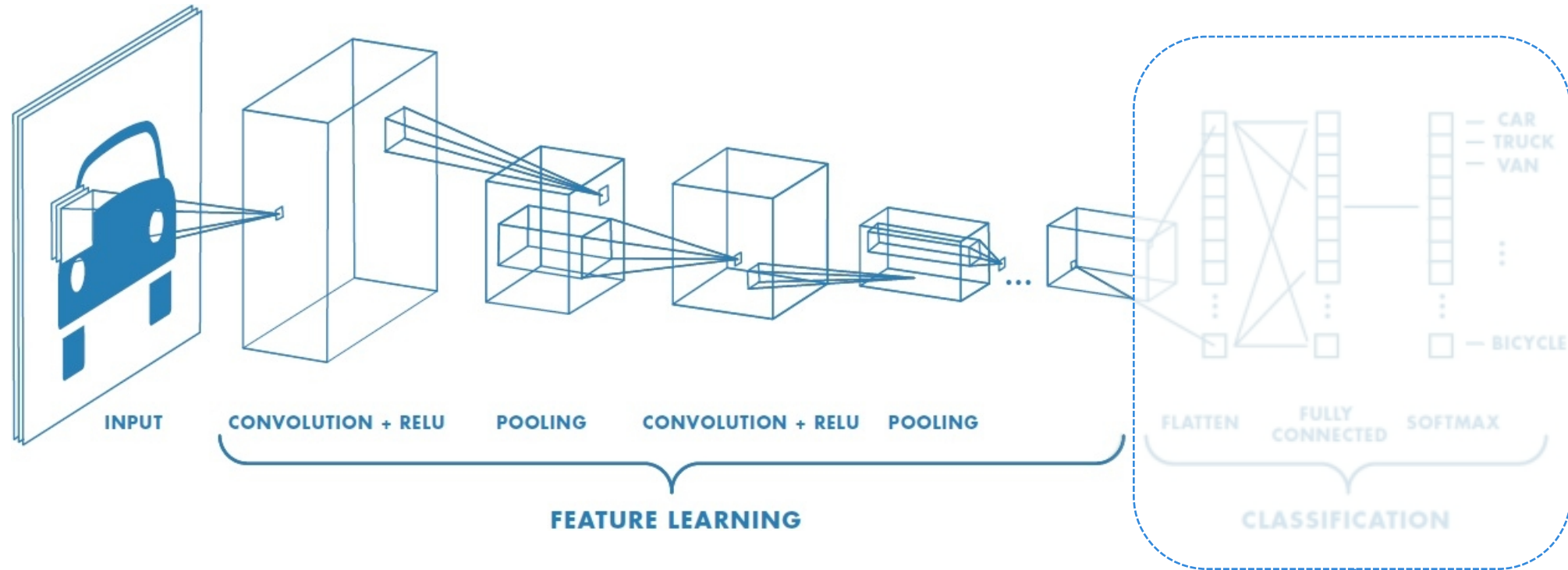


# ImageNet Challenge: Classification Task



# An Architecture for Many Applications

# An Architecture for Many Applications



Different image classification domains  
Object detection with R-CNNs  
Segmentation with fully convolutional networks  
Image captioning with RNNs

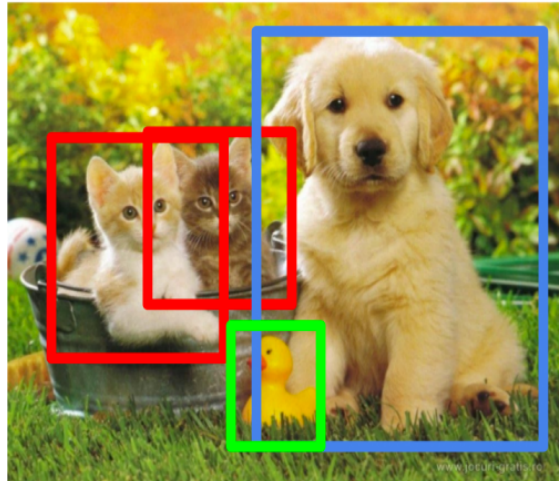
# Beyond Classification

Semantic Segmentation



CAT

Object Detection



CAT, DOG, DUCK

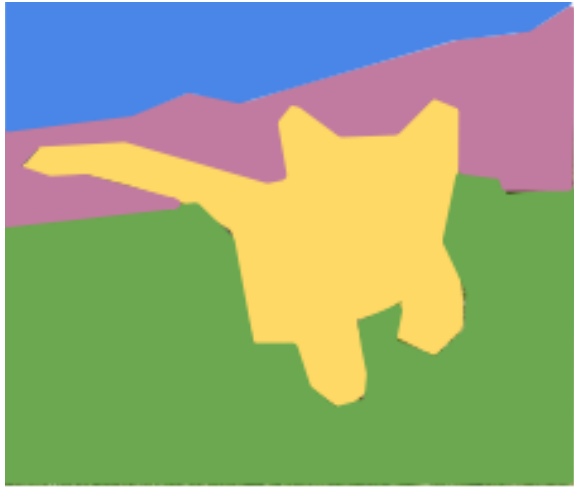
Image Captioning



The cat is in the grass.

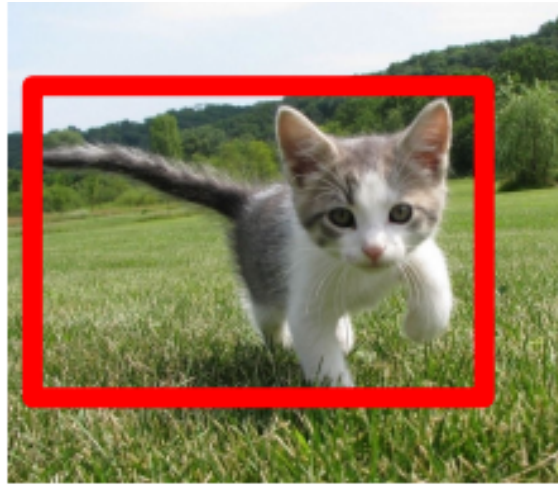
# Beyond Classification

Semantic Segmentation



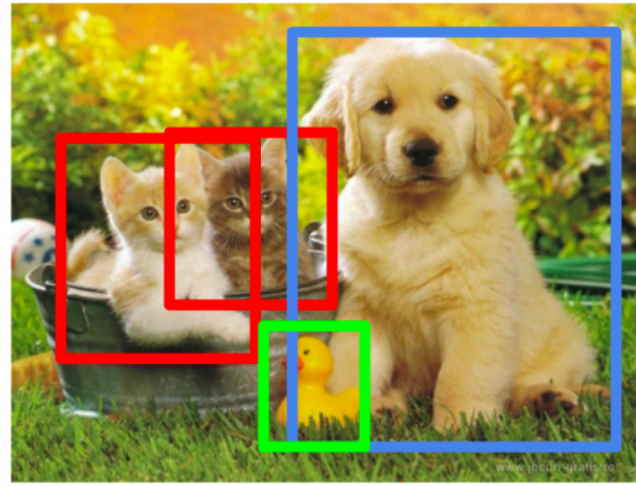
CAT

Classification + Localization



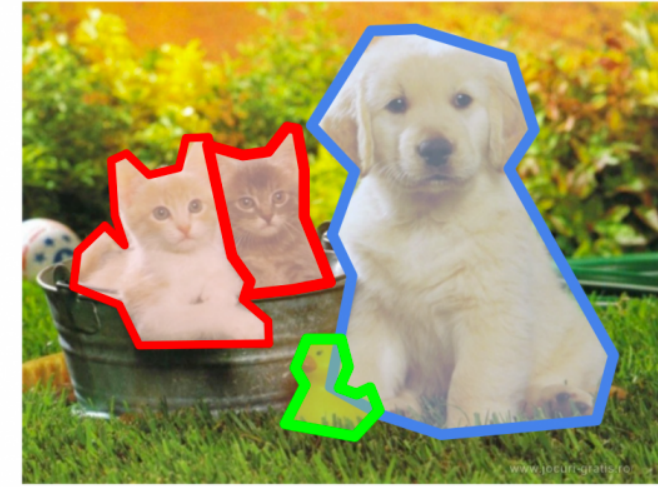
CAT

Object Detection



CAT, DOG, DUCK

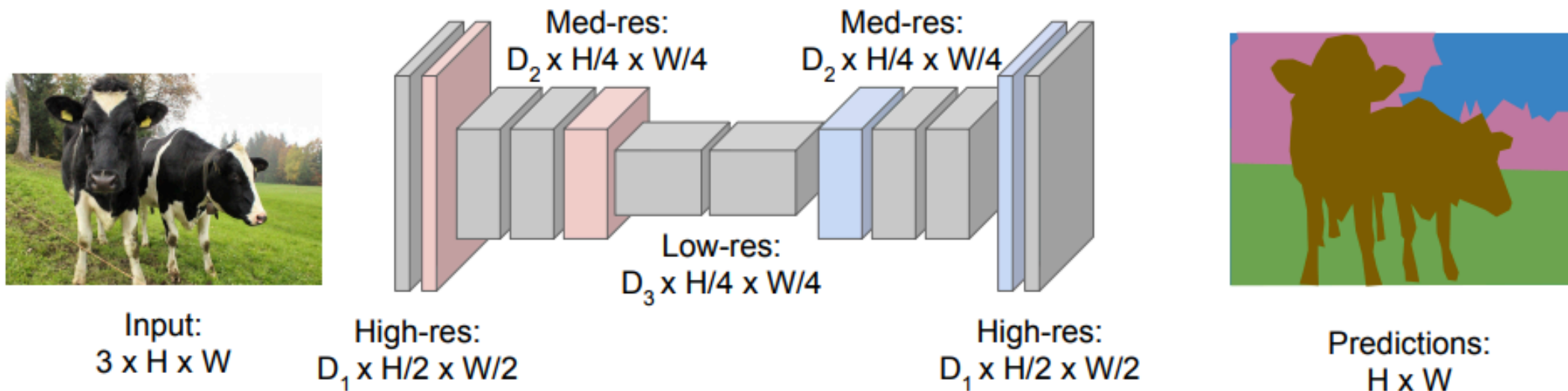
Instance Segmentation



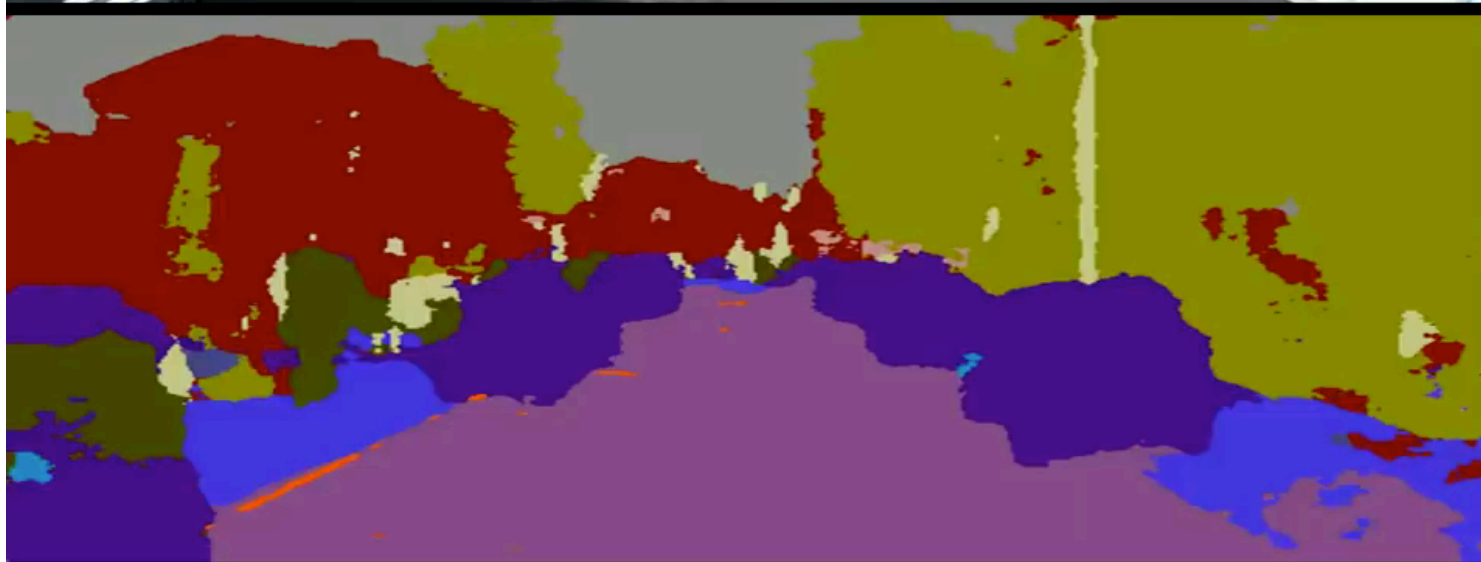
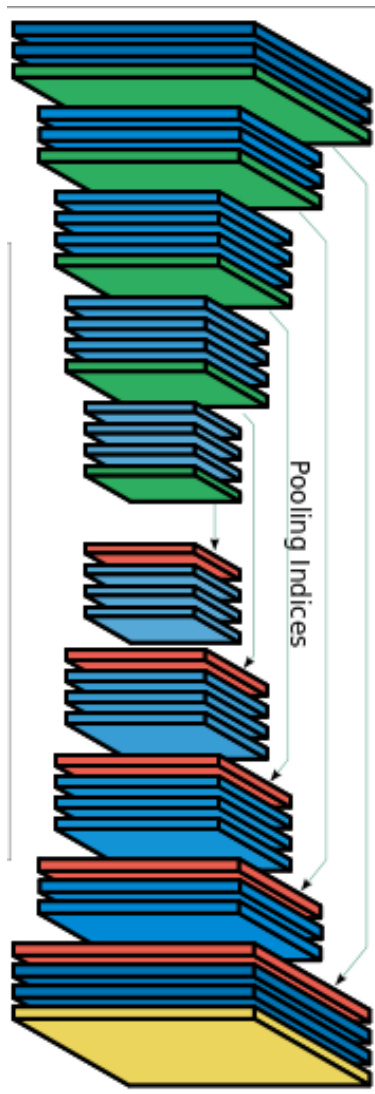
CAT, DOG, DUCK

# Semantic Segmentation: FCNs

FCN: Fully Convolutional Network  
Network designed with all convolutional layers,  
with **downsampling** and **upsampling** operations



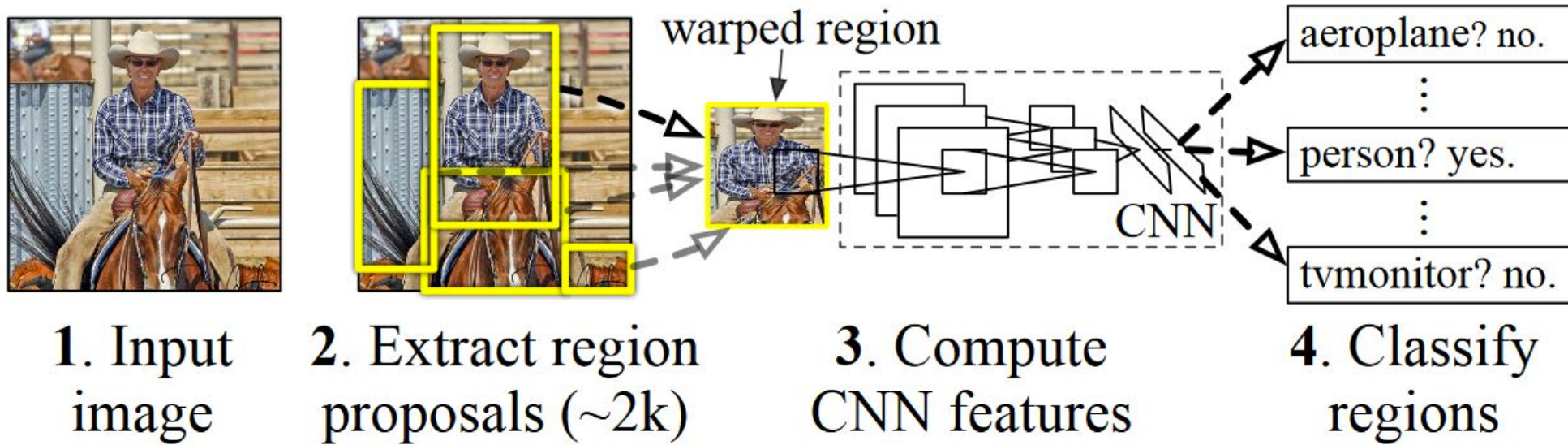
# Driving Scene Segmentation



- Sky
- Building
- Pole
- Road Marking
- Road
- Pavement
- Tree
- Sign Symbol
- Fence
- Vehicle
- Pedestrian
- Bike

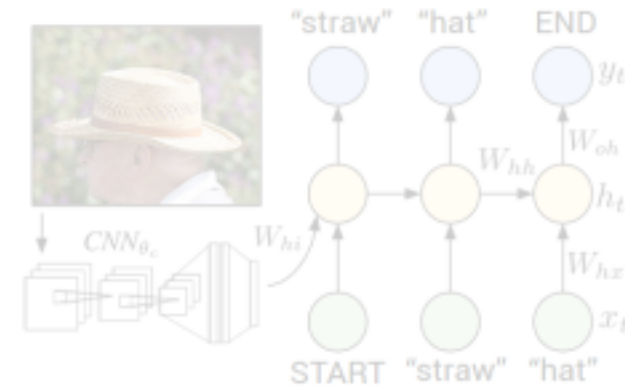
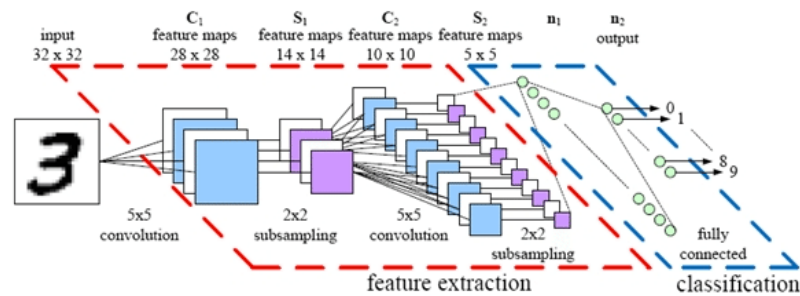
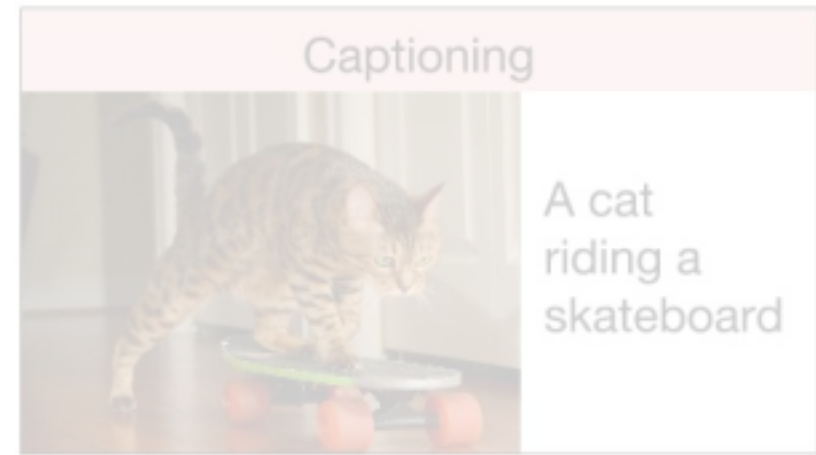
# Object Detection with R-CNNs

R-CNN: Find regions that we think have objects. Use CNN to classify.

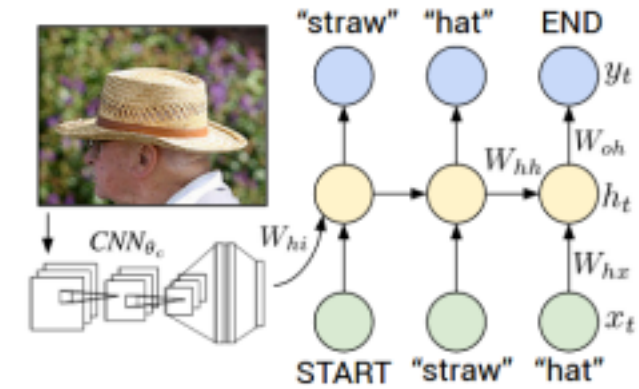
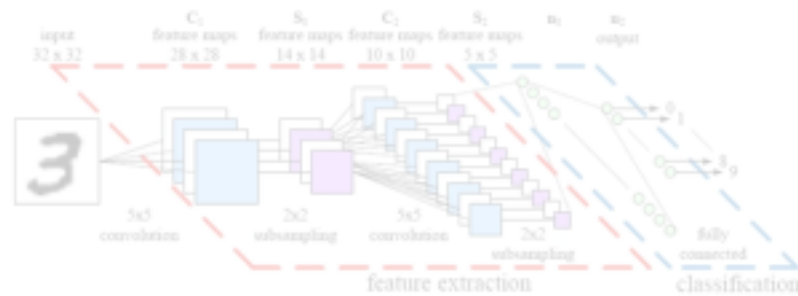
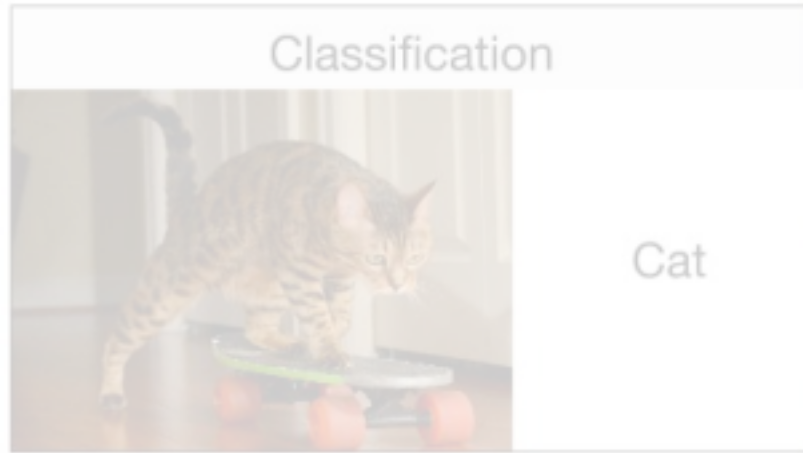




# Image Captioning using RNNs



# Image Captioning using RNNs



# Visualizing Convolutional Neural Networks

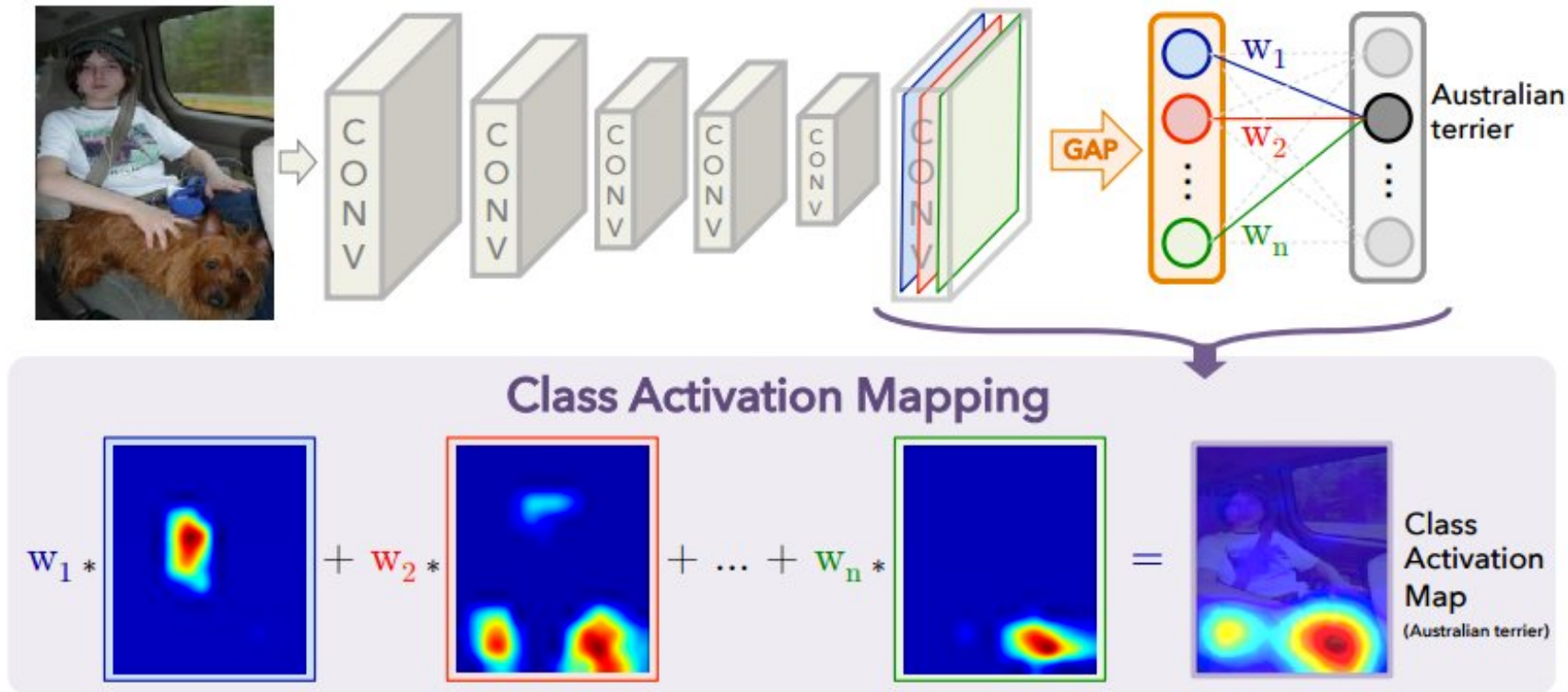
# Peering Inside CNNs



# Class Activation Maps (CAMs)



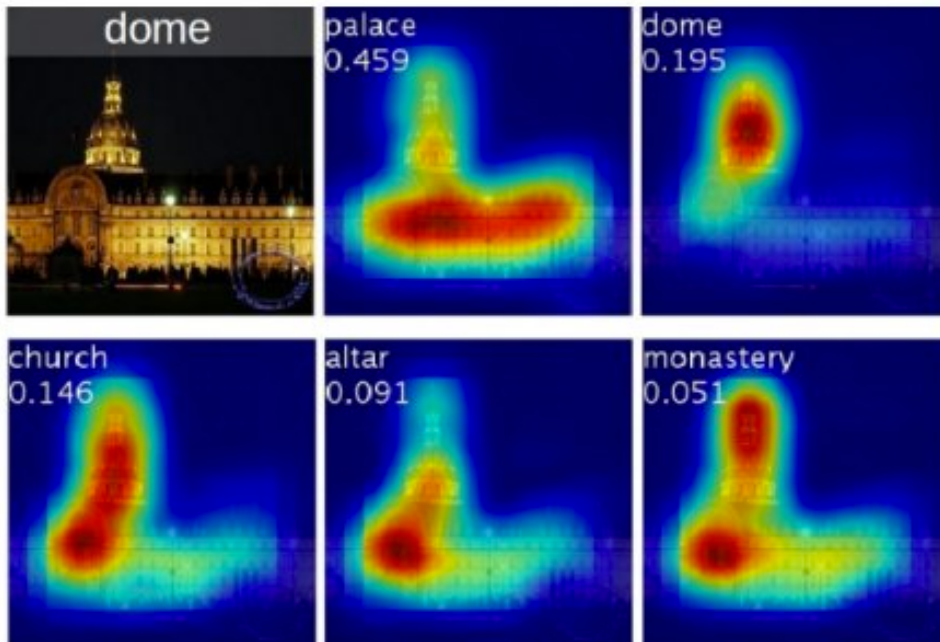
A **class activation map (CAM)** for a given class highlights the image regions used by the CNN to identify that class



# Class Activation Maps (CAMs)



A **class activation map (CAM)** for a given class highlights the image regions used by the CNN to identify that class



Class activation maps of top 5 predictions



Class activation maps for one object class

# Deep Learning for Computer Vision: Impact and Summary

# Data, Data, Data



ImageNet:  
22K categories. 14M images.



MNIST: handwritten digits



places: natural scenes

**Airplane**

**Automobile**

**Bird**

**Cat**

**Deer**

**Dog**

**Frog**

**Horse**

**Ship**

**Truck**

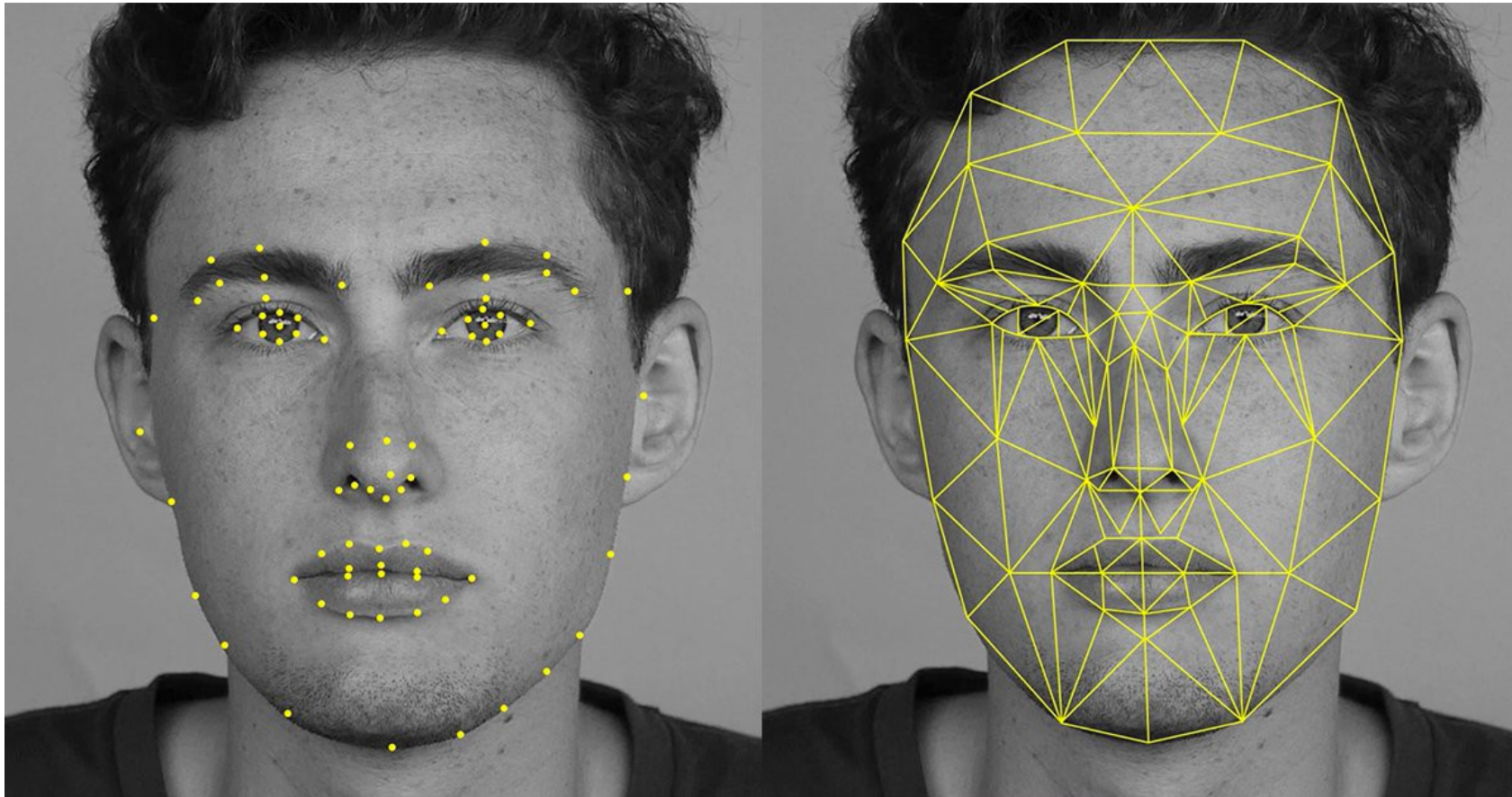
CIFAR-10



# Deep Learning for Computer Vision: Impact



# Impact: Face Recognition



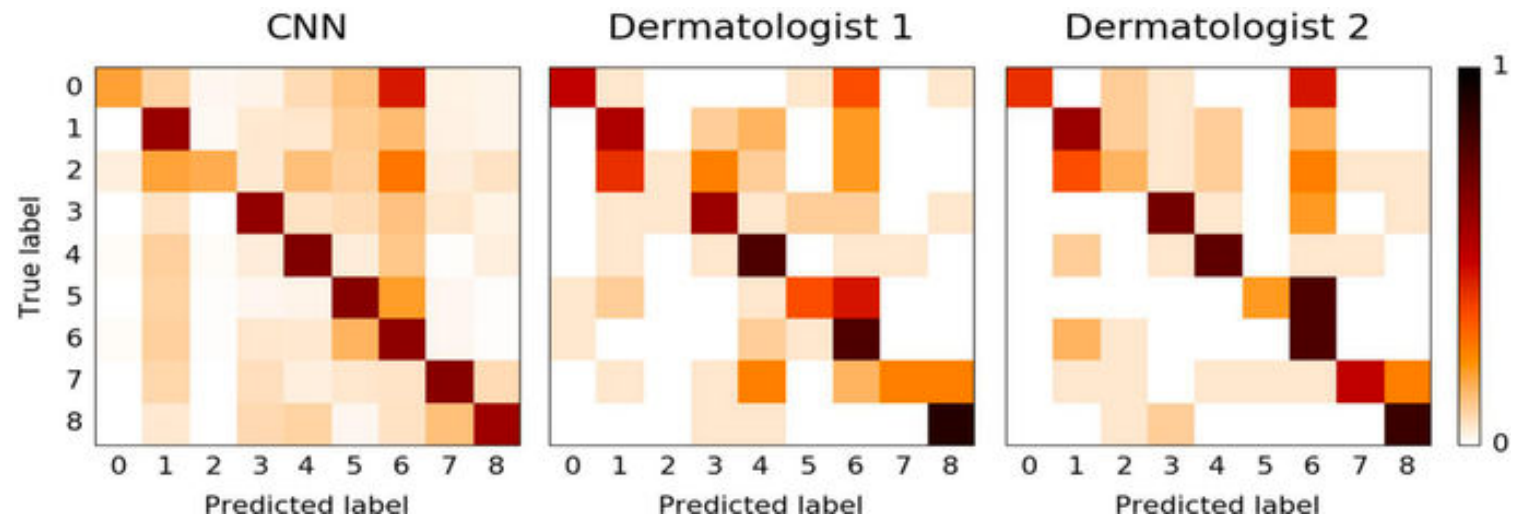
# Impact: Self-Driving Cars



# Impact: Medicine



Dermatologist-level classification of skin cancer with deep neural networks



# Deep Learning for Computer Vision: Review

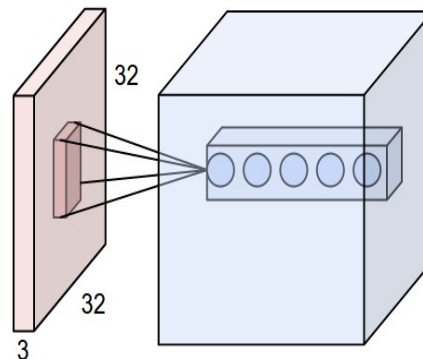
## Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction



## CNNs

- CNN architecture
- Application to classification: ImageNet



## Applications

- Segmentation, object detection, image captioning
- Visualization



References:  
<https://goo.gl/ZuBkGx>