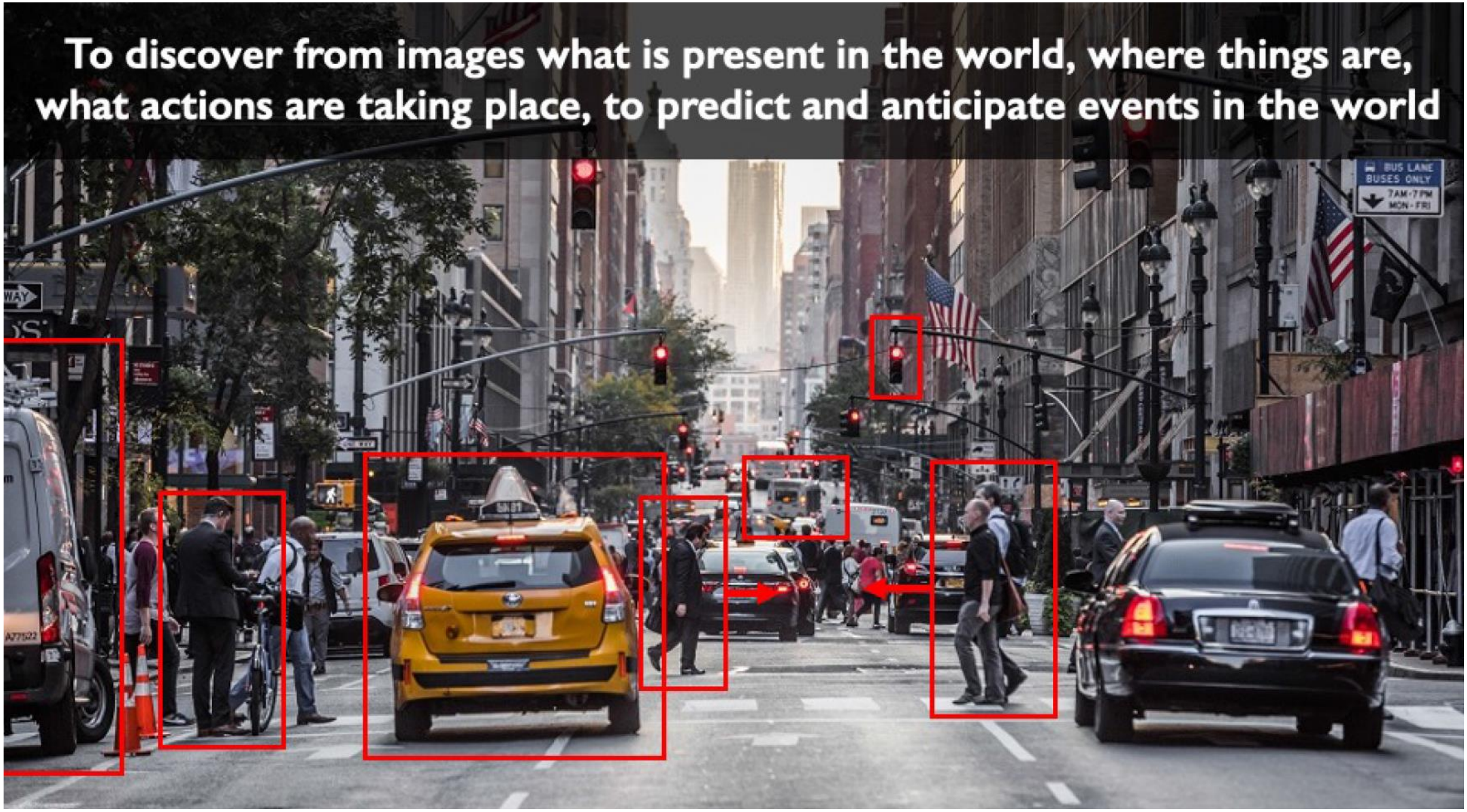
Machine learning

Deep Computer Vision: To know what is where by looking

Mohammad-Reza A. Dehaqani

dehaqani@ut.ac.ir

Based on MIT 6.S191

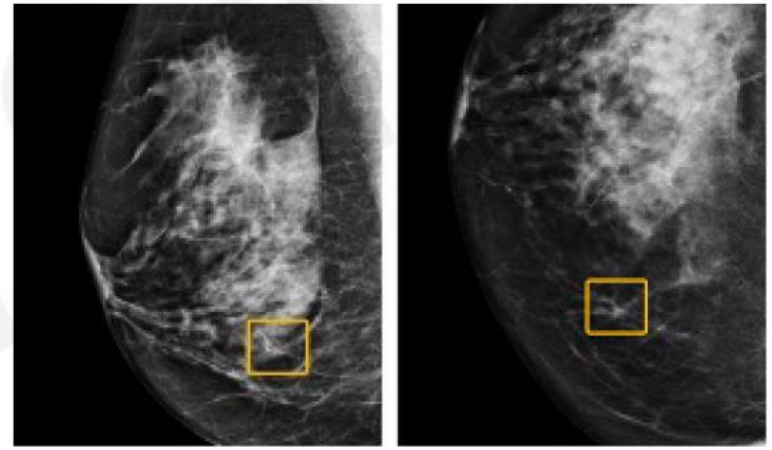

To discover from images what is present in the world, where things are, what actions are taking place, to predict and anticipate events in the world
BUS LANE
BUSES ONLY
7AM-7PM
MON-FRI

The rise and impact of computer vision

Robotics

Accessibility

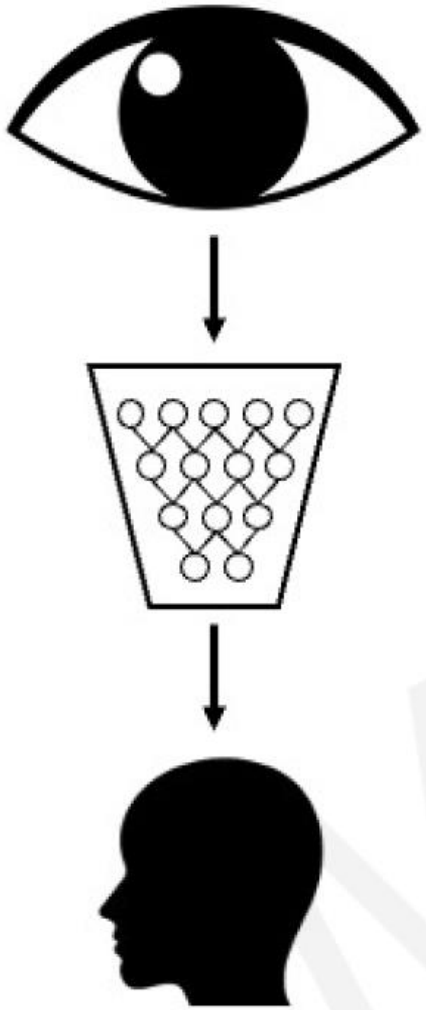
Biology & Medicine

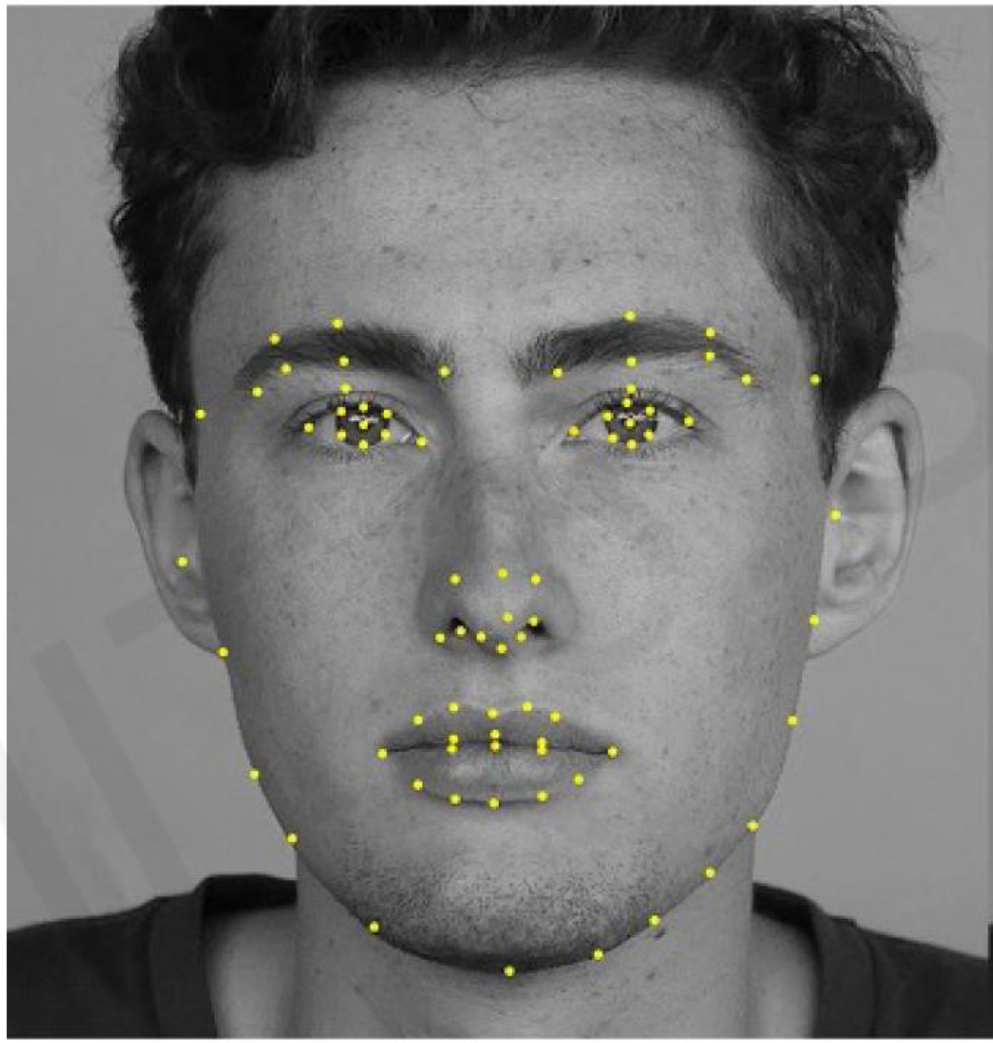


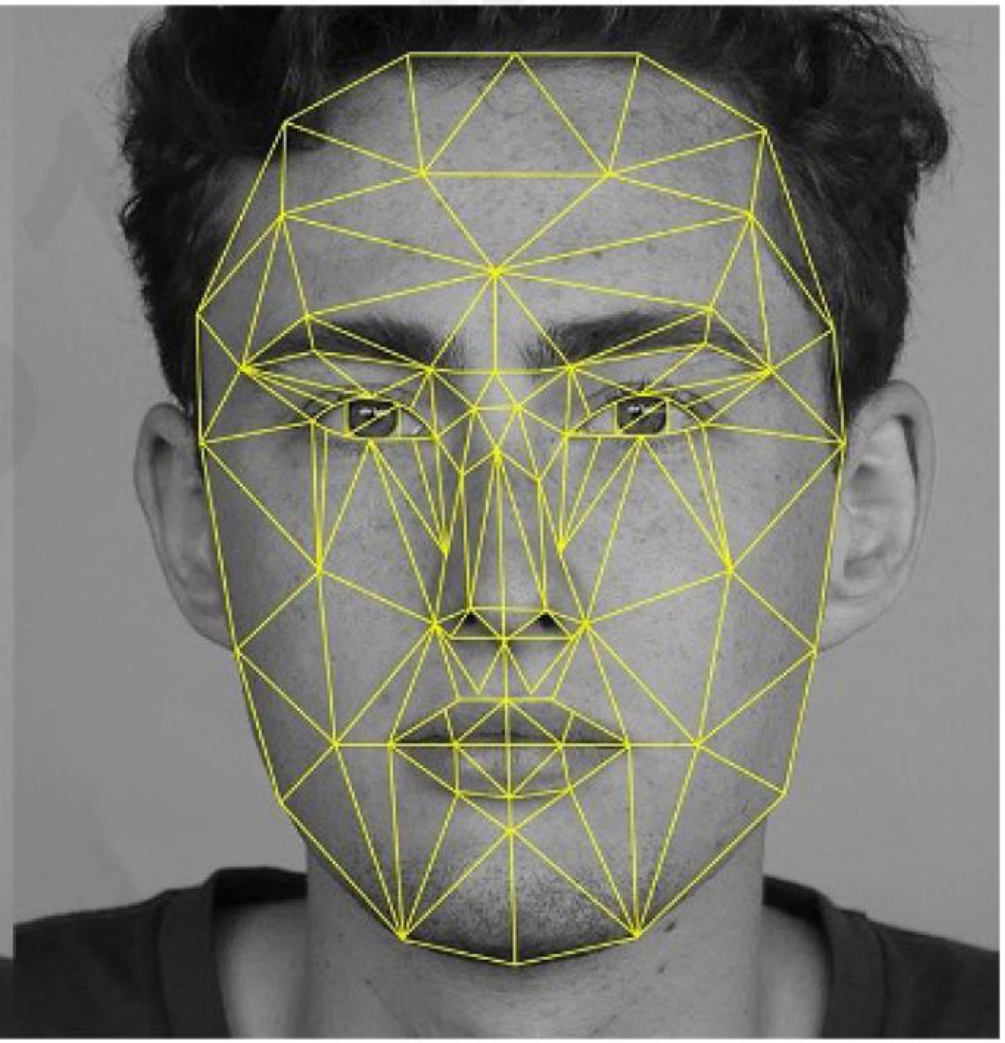
Autonomous driving

Mobile computing

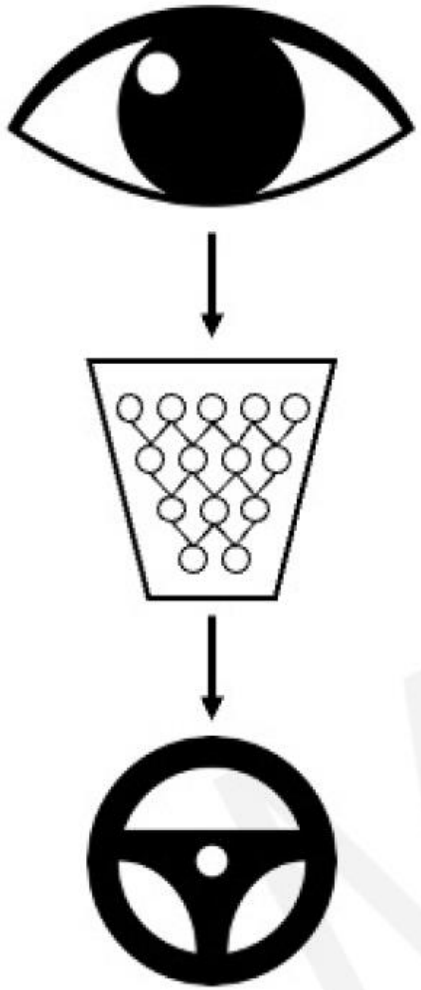
Impact: Facial Detection & Recognition



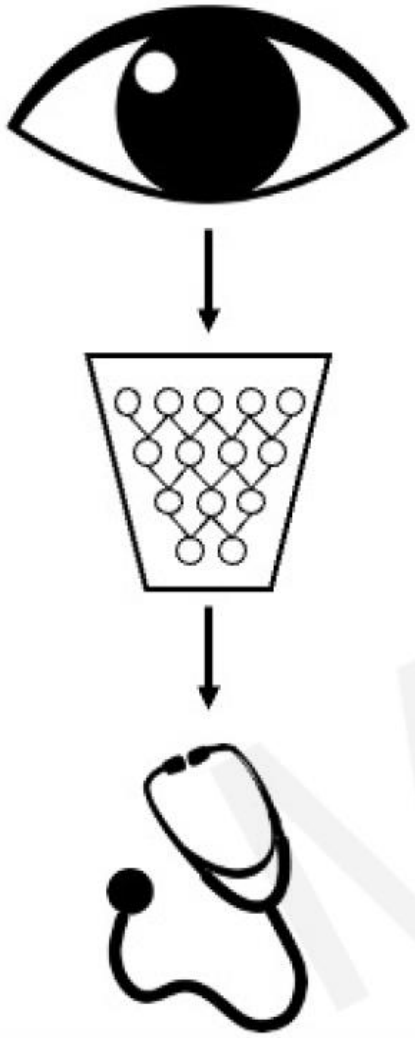




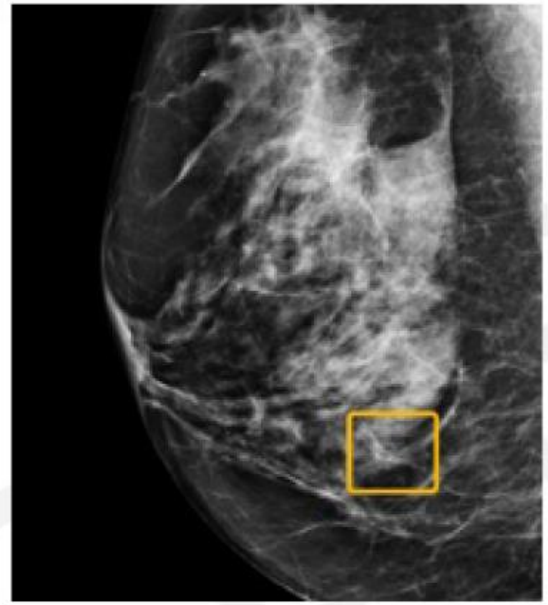
Impact: Self-Driving Cars



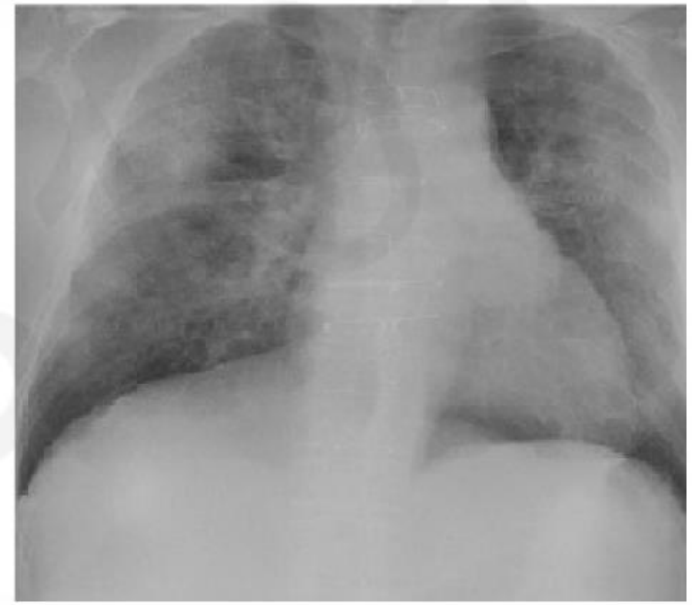
Impact: Medicine, Biology, Healthcare



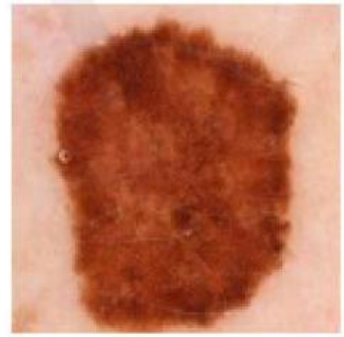
Breast cancer

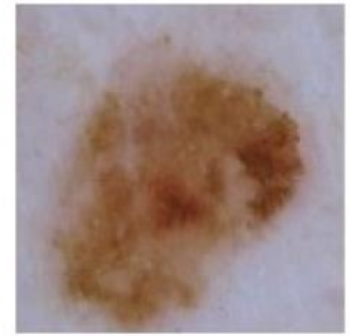


COVID-19



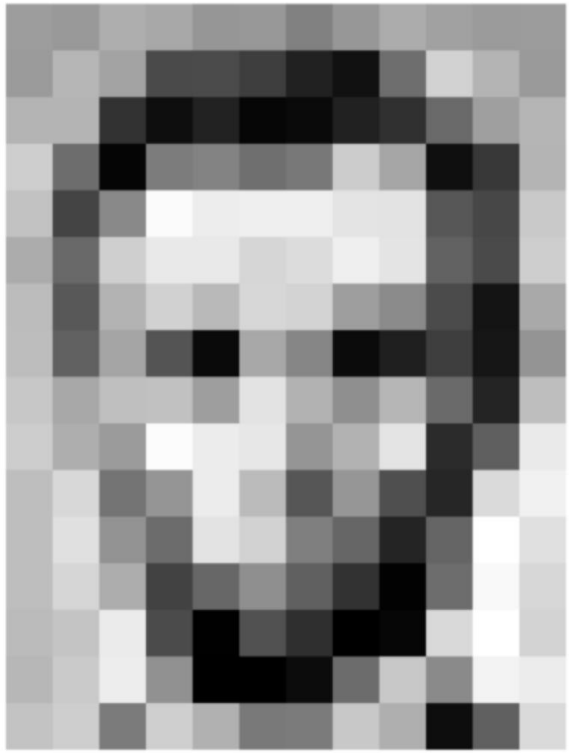
Skin cancer

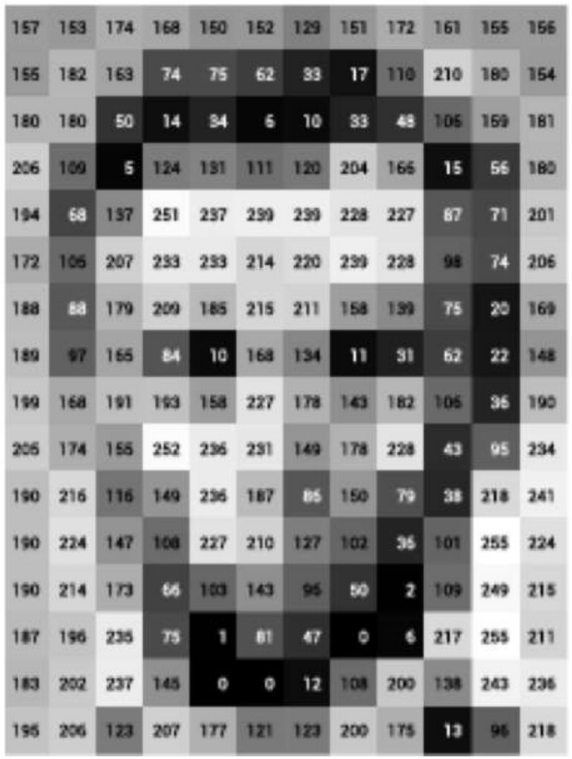




What computer “See”

Images are Numbers





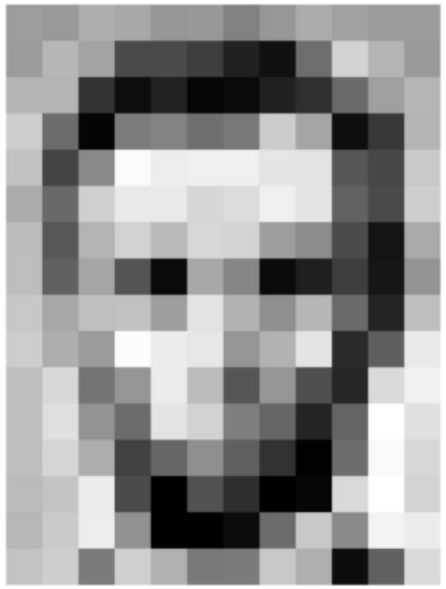
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	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

What the computer sees

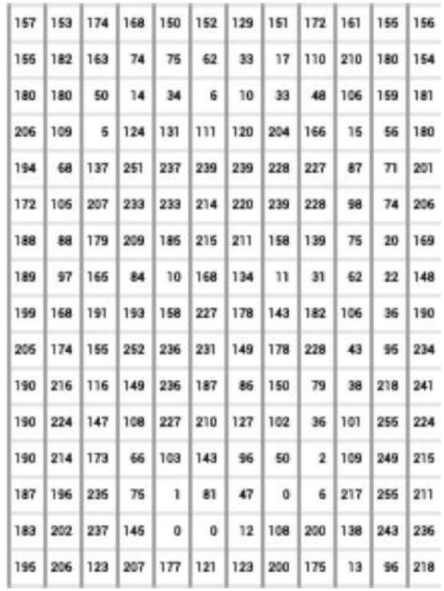
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	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., 1080x1080x3 for an RGB image

Tasks in Computer Vision



Input Image



Pixel Representation

classification

Lincoln 0.8
Washington 0.1
Jefferson 0.05
Obama 0.05

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

High Level Feature Detection

Let's identify key features in each image category

Wheels,
License Plate,
Headlights

Door,
Windows,
Steps

Manual Feature Extraction; Problems

Domain knowledge → Define features → Detect features to classify

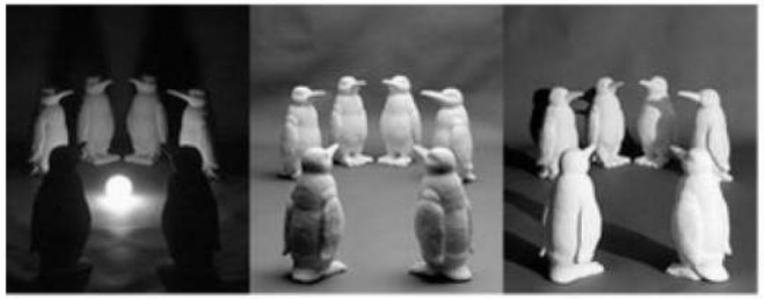
Viewpoint variation

Scale variation

Deformation

Occlusion

Illumination conditions



Background clutter

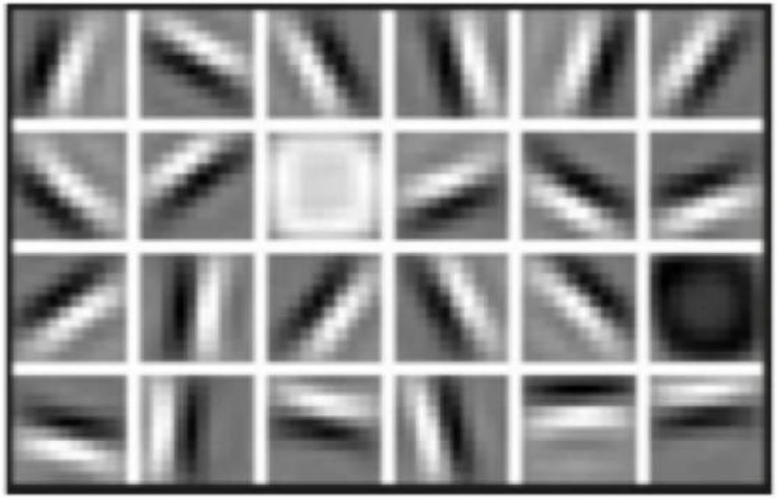
Intra-class variation

Learning Feature Representations

Hand engineered features are time consuming, brittle and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

High Level Features

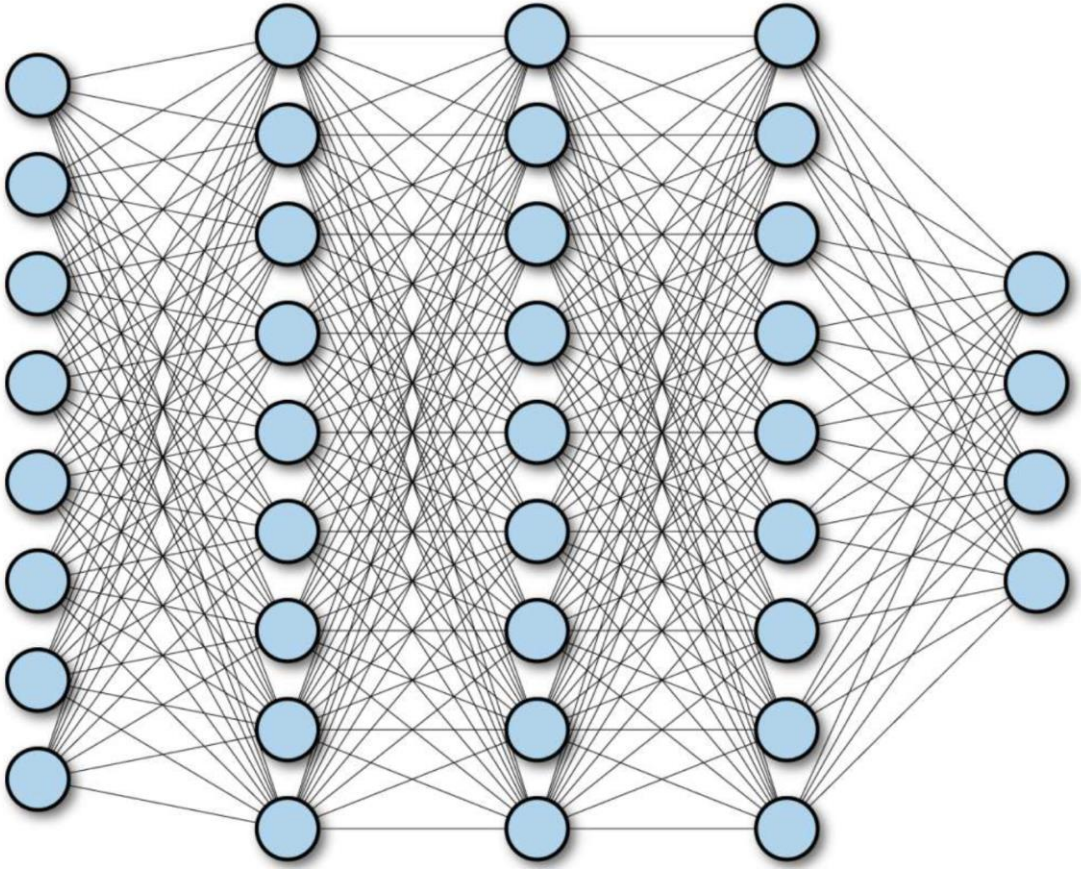
Facial Structure

Learning Visual Features

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values



- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, **many** parameters!
- Do we really need all the edges? Can some of these be shared?

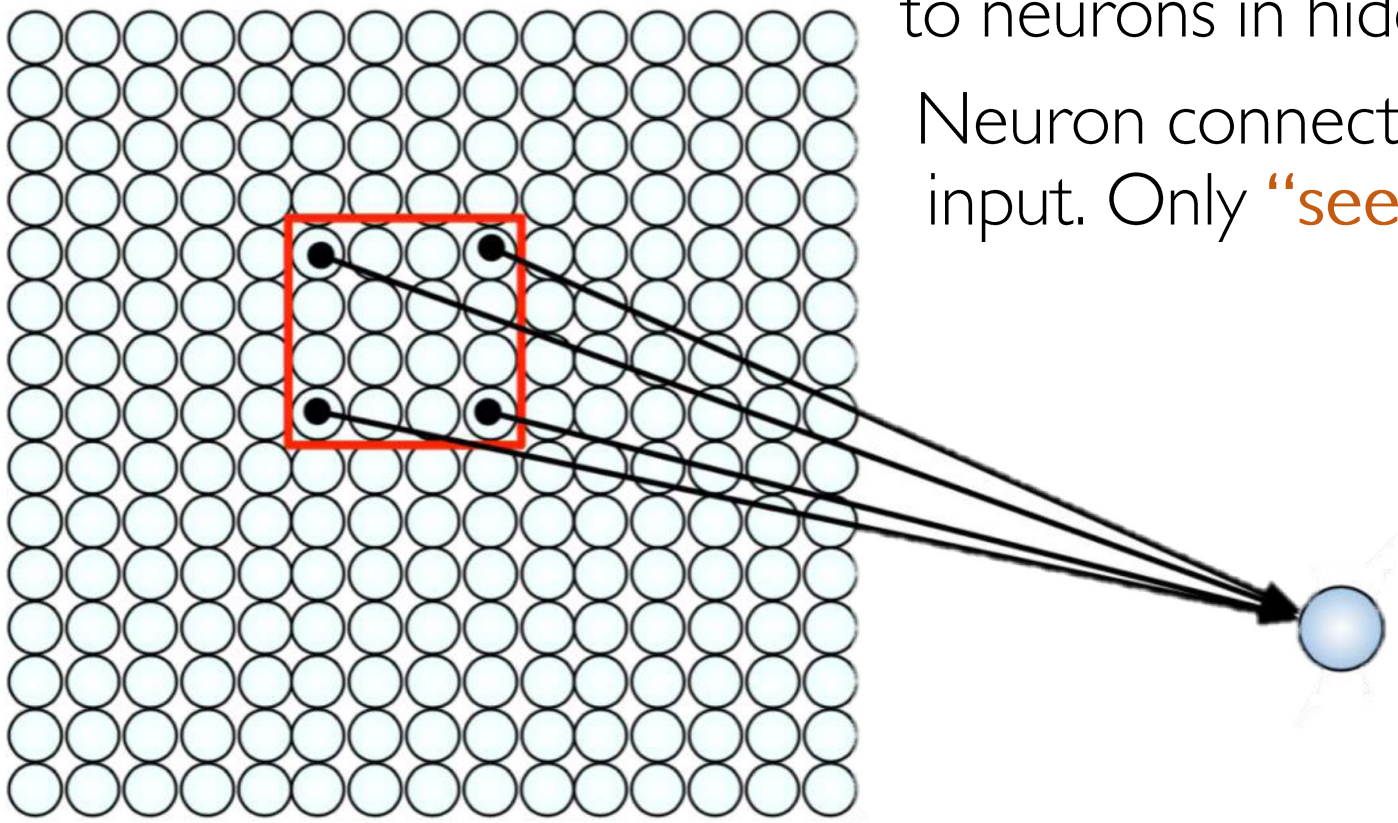
How can we use **spatial structure** in the input to inform the architecture of the network?

Using Spatial Structure

Input: 2D image.
Array of pixel values

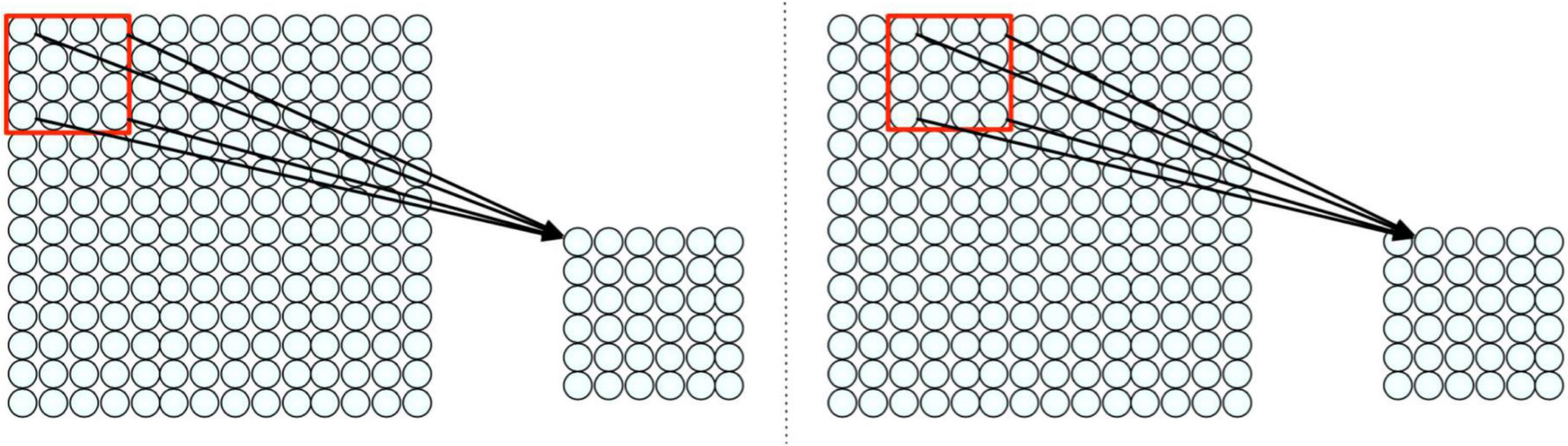
Idea: connect patches of input to neurons in hidden layer

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



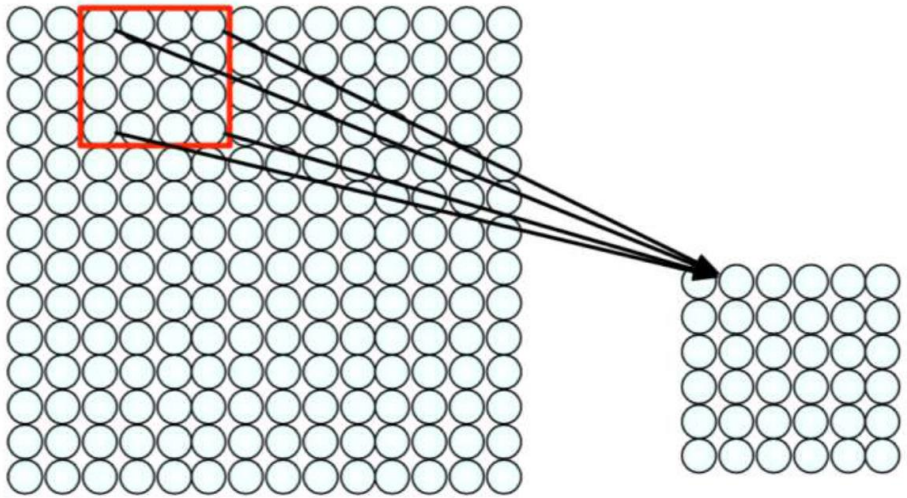
Connect patch in input layer to a single neuron in subsequent layer.

Appling a sliding window to use spatial structure and define connections



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

Feature Extraction with Convolution



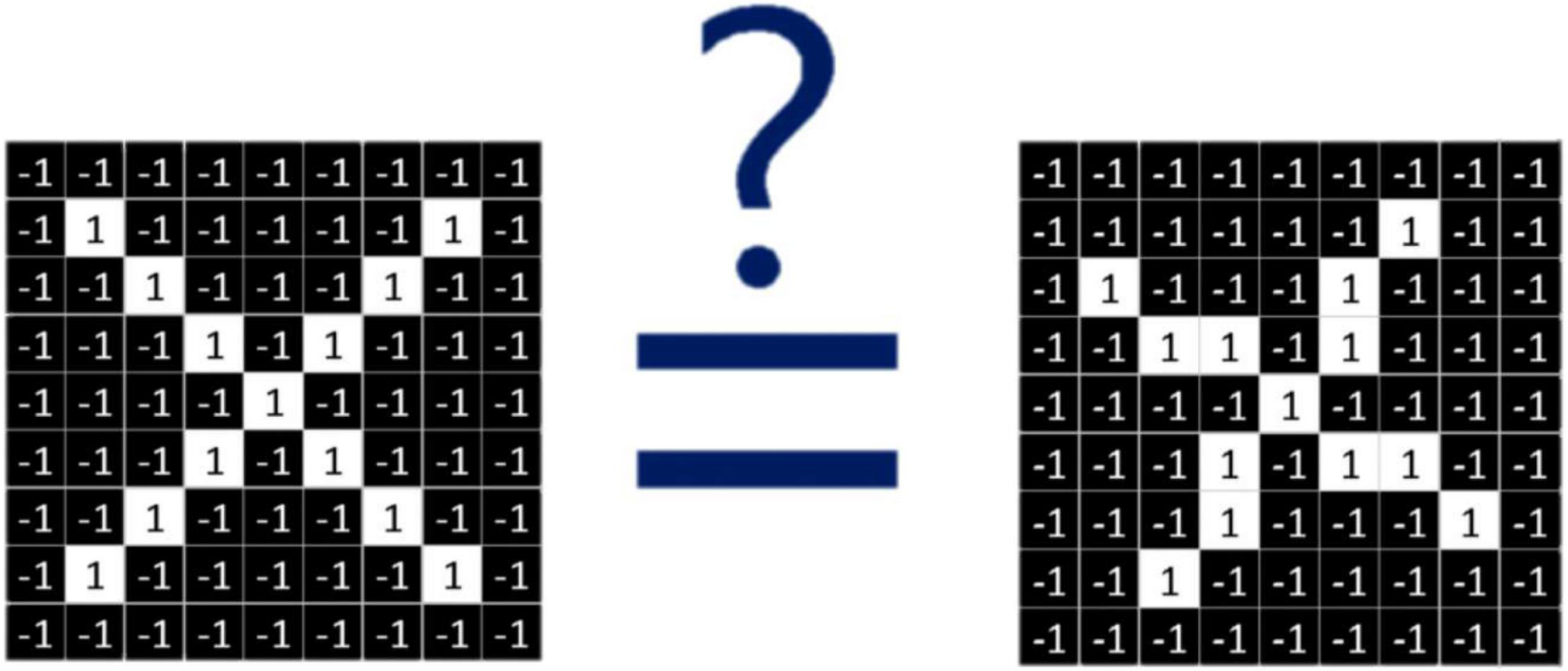
- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

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

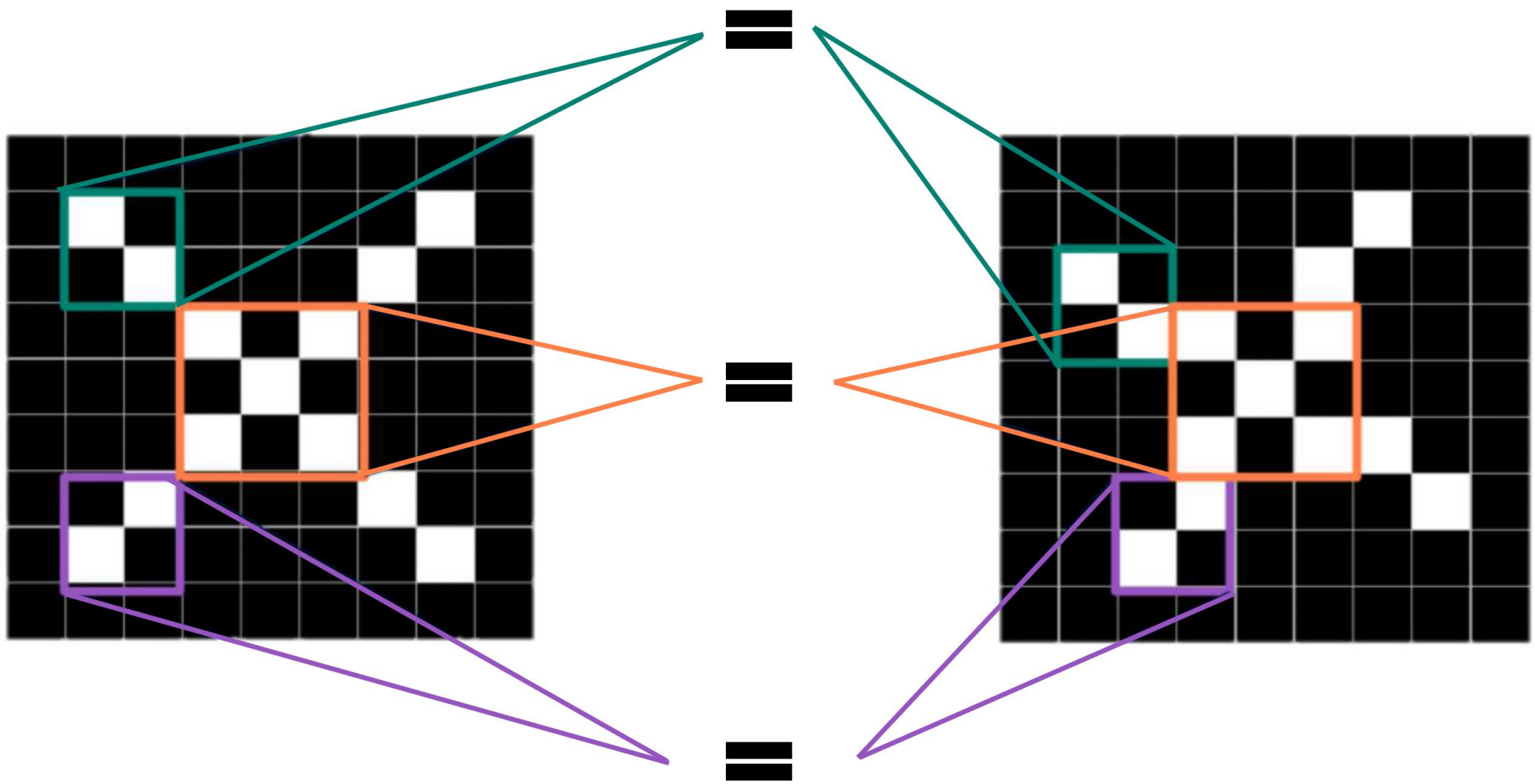
Feature Extraction and Convolution
A Case Study

X or X?



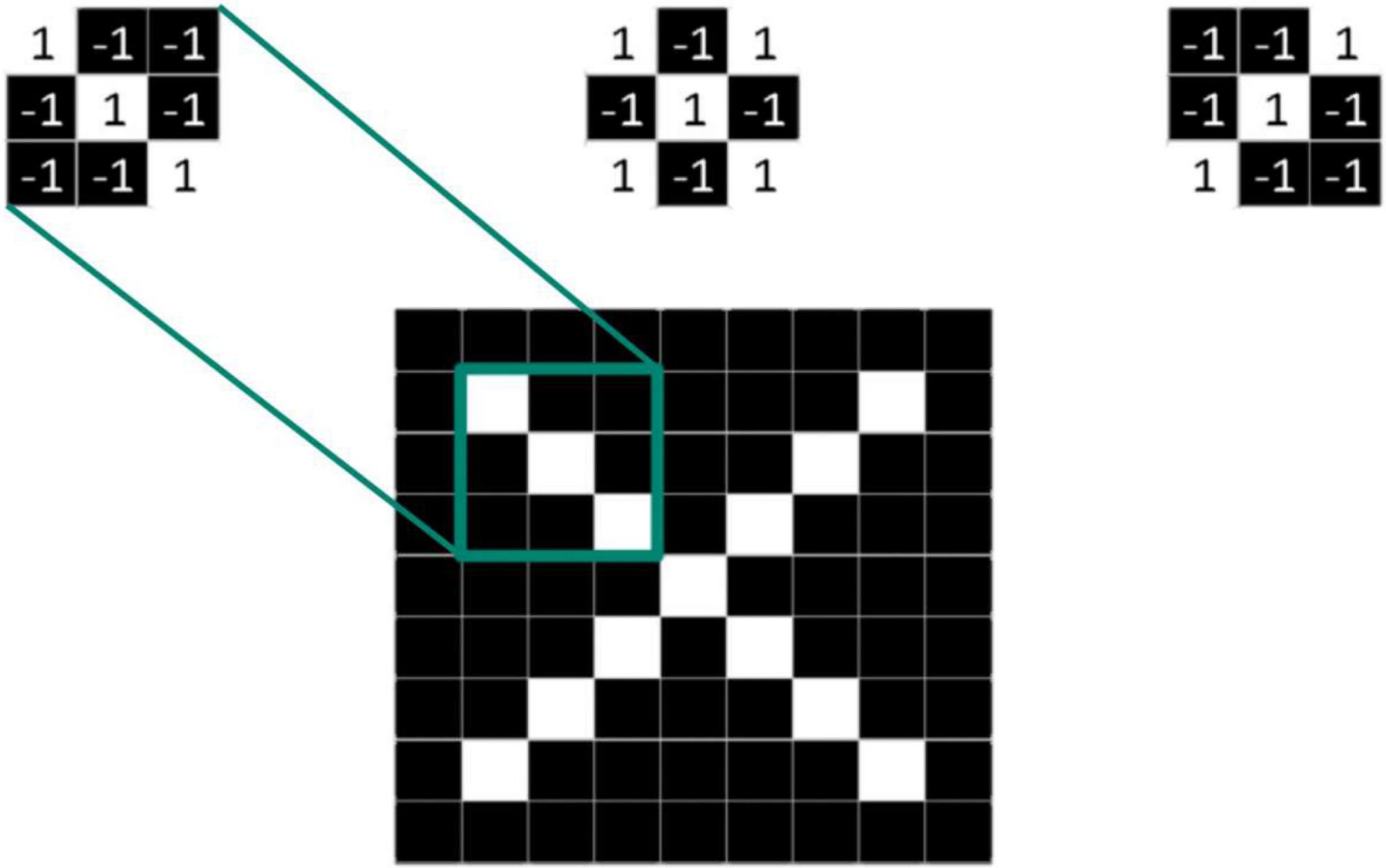
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

Features of X

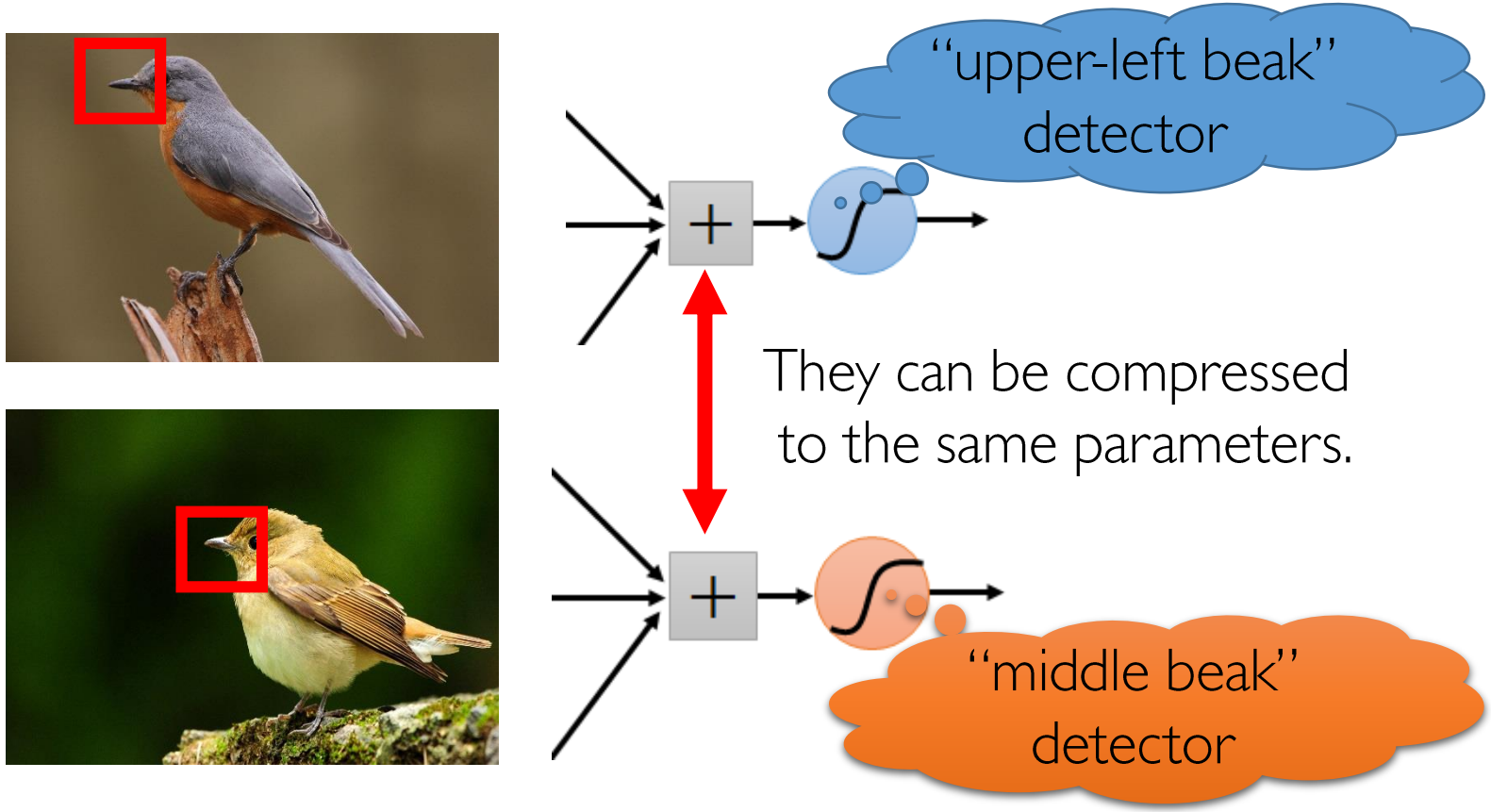


Filters to Detect X Features

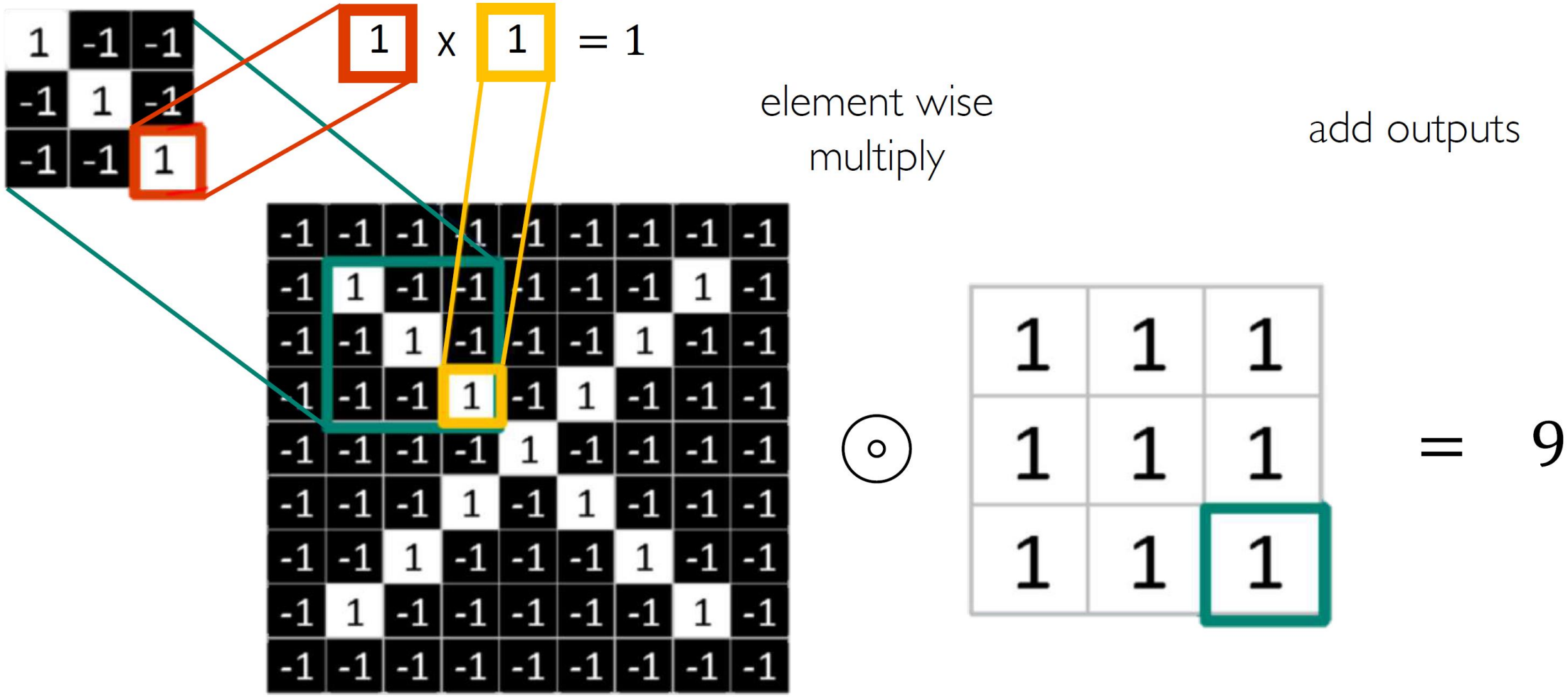
filters



Same pattern appears in different **places**: They can be compressed!
What about training a lot of such "small" detectors and each detector must "move around".

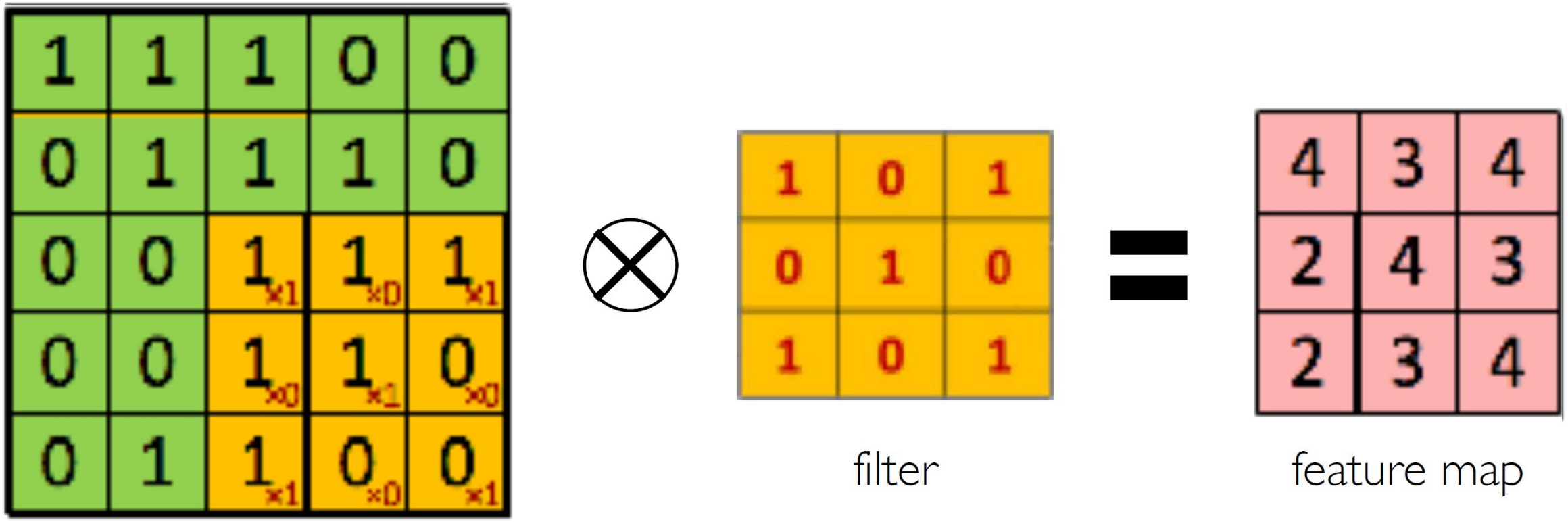


The Convolution Operation



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

Convolutional Neural Networks (CNNs)

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

Filter 1

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

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

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

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product →

3 -1

6 x 6 image

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

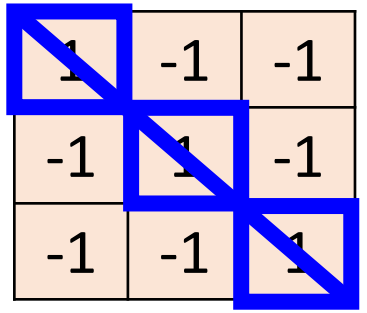
Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

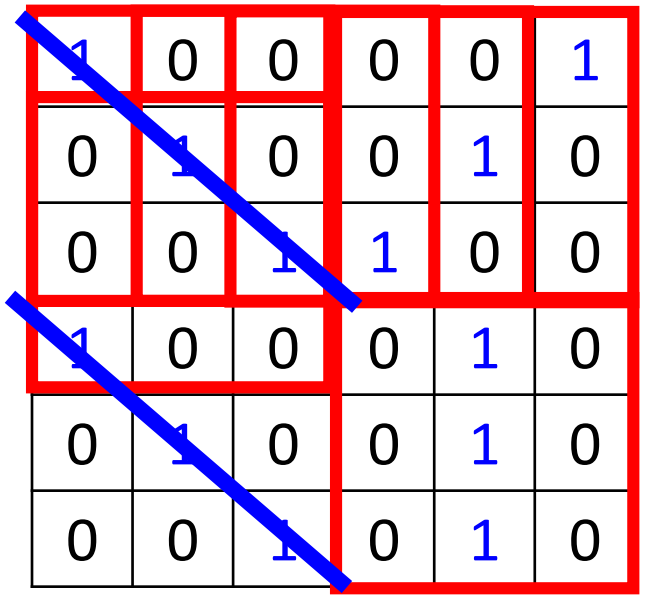
6 x 6 image

3 -3

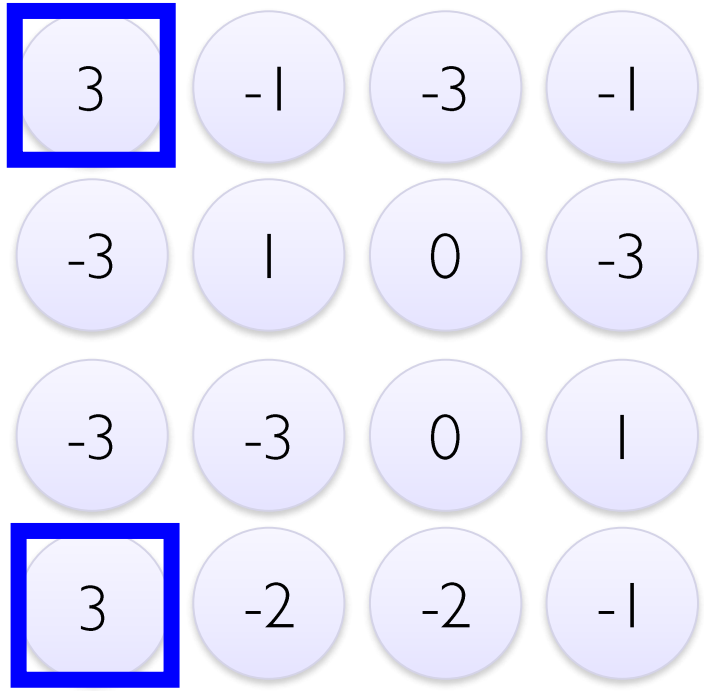


Filter 1

stride=1



6 x 6 image



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

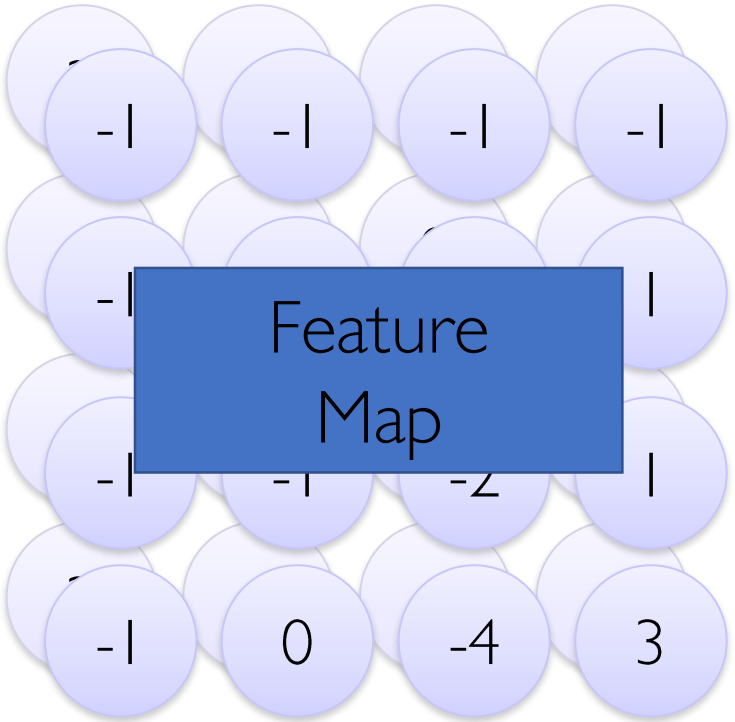
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

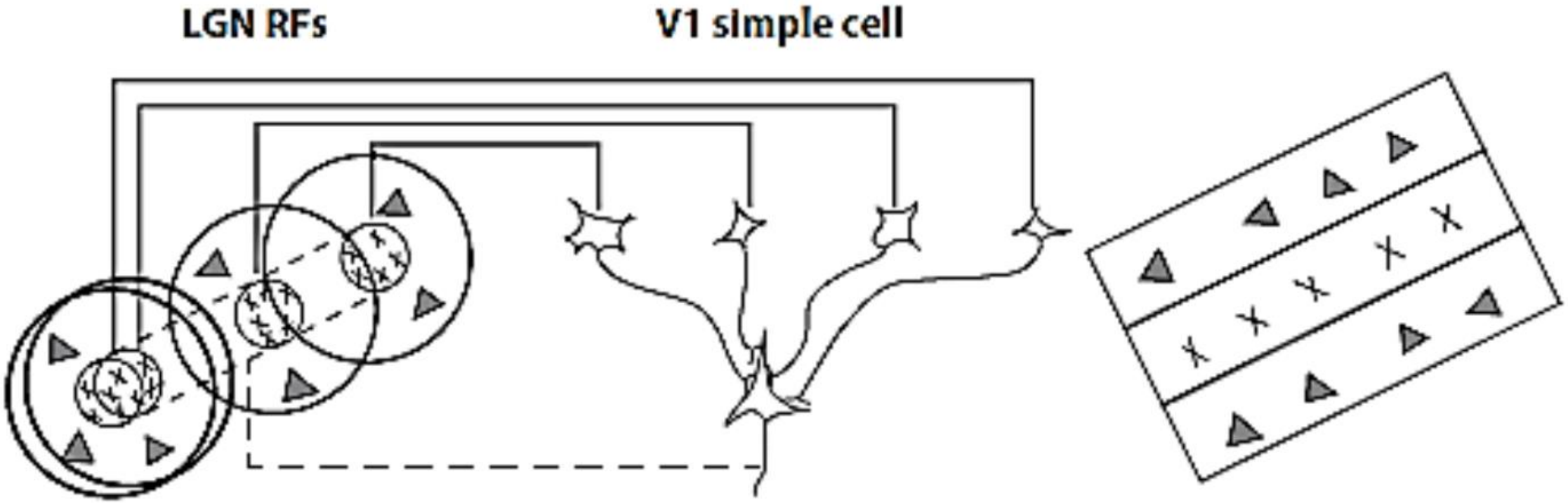
6 x 6 image

Repeat this for each filter



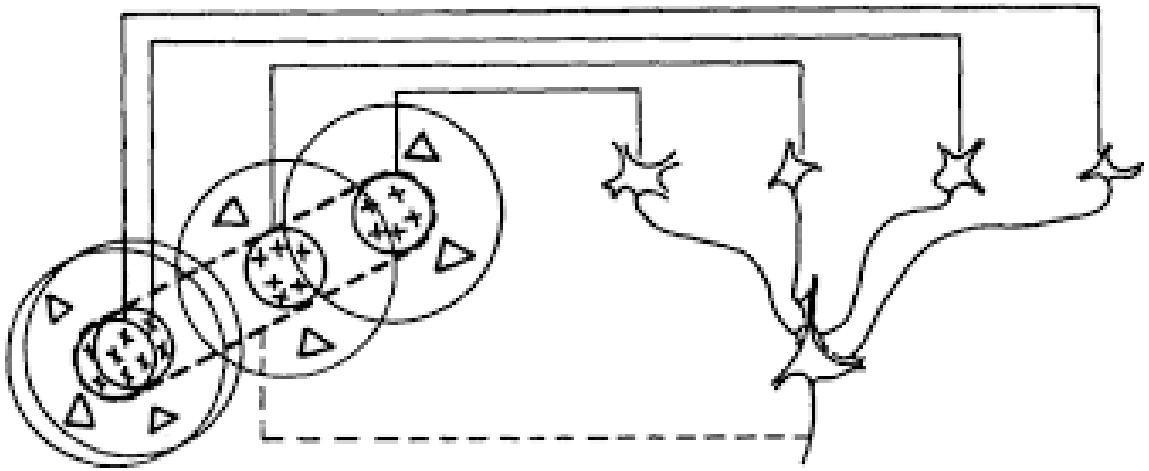
Two 4 x 4 images
Forming 2 x 4 x 4 matrix

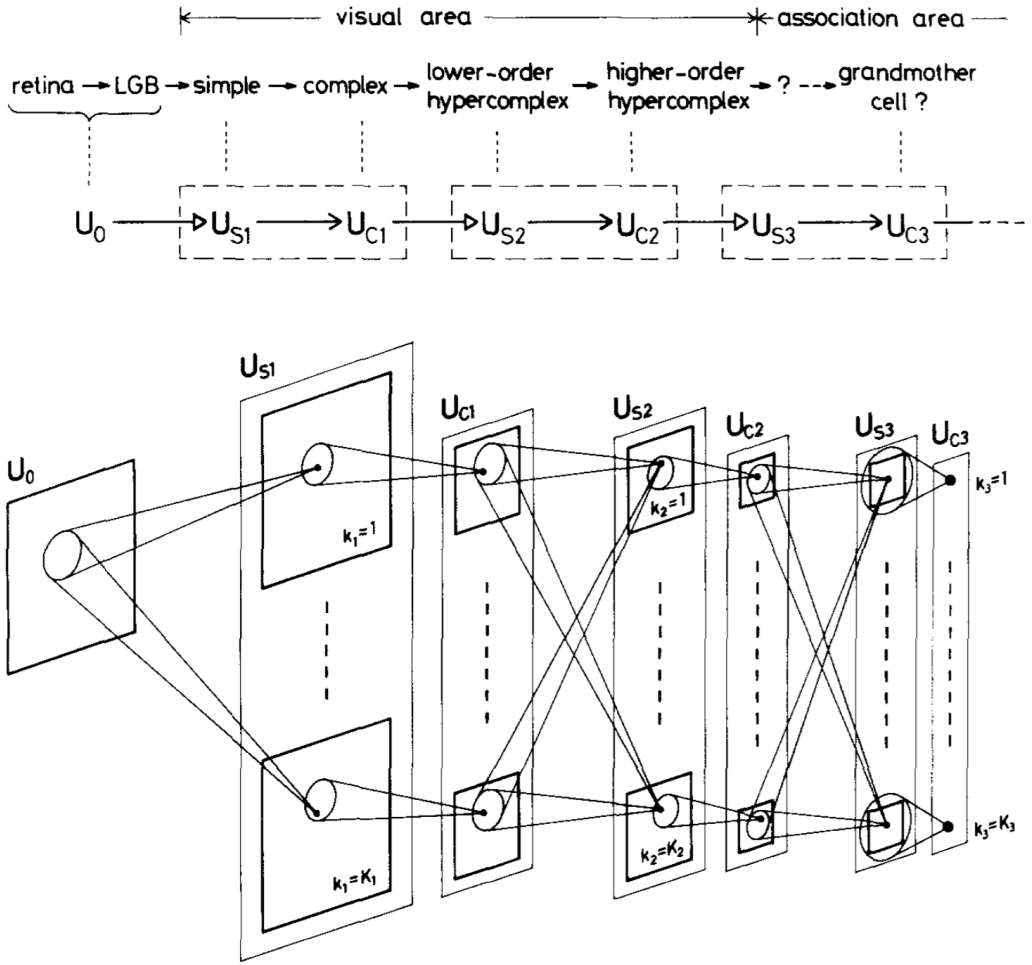
Complex feature extraction in brain



Neocognitron; Emergence of convolutional neural network (1982)

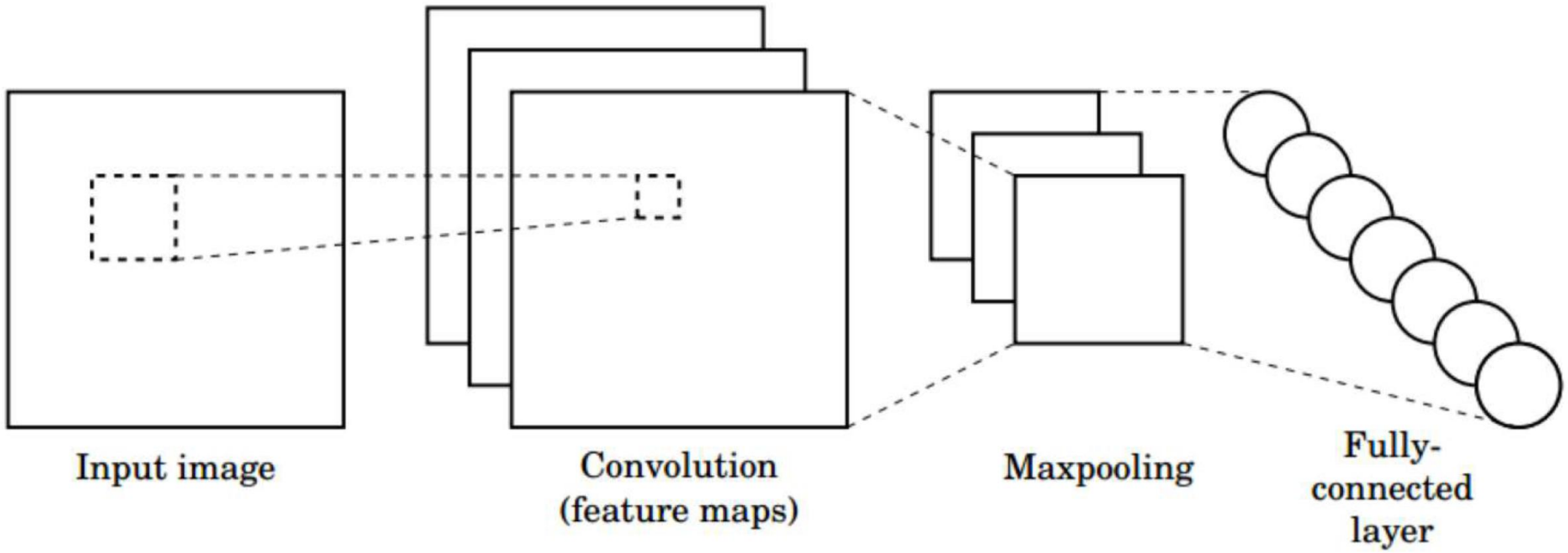
simple-to-complex pooling in Fukushima's neocognitron





Fukushima, Kunihiko, and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition

CNNs for Classification

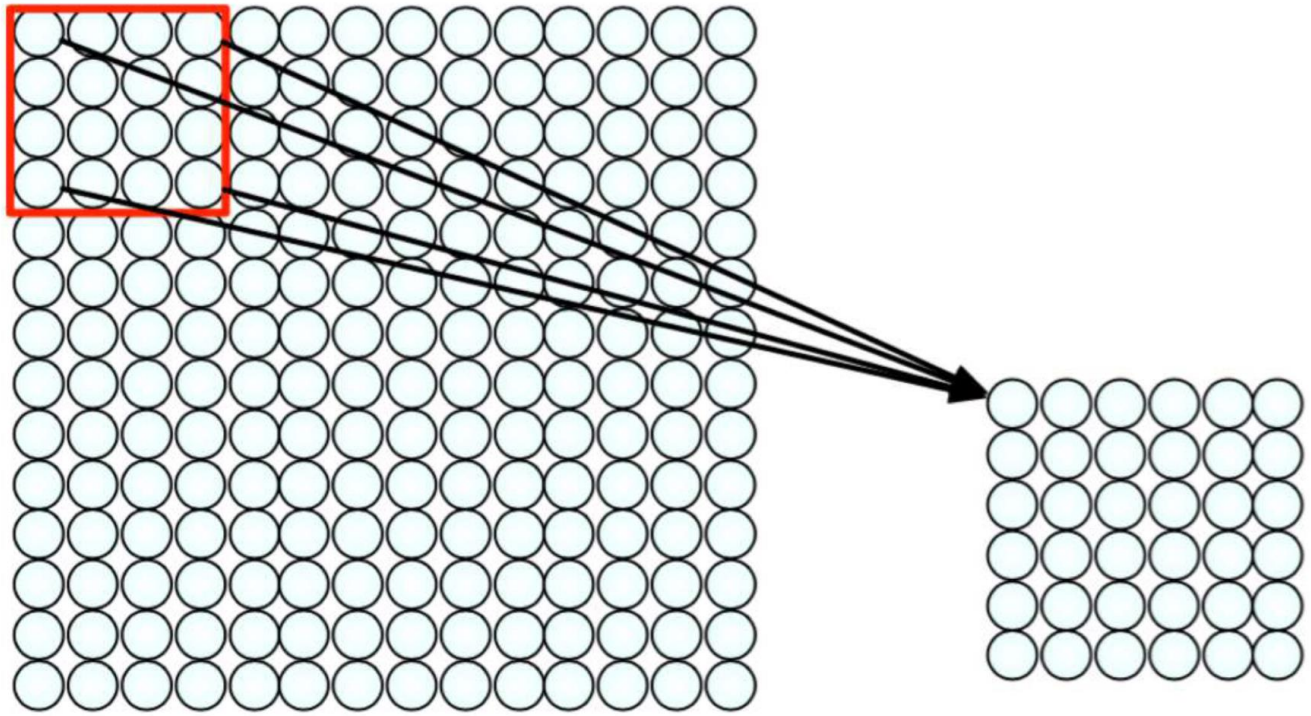


1. Convolution: Apply **filters** with learned weights to generate feature maps.
2. Non-linearity: Often ReLU.
3. Pooling: Downsampling operation on each feature map.

Train model with image data.

Learn weights of filters in convolutional layers

Convolutional Layers: Local Connectivity



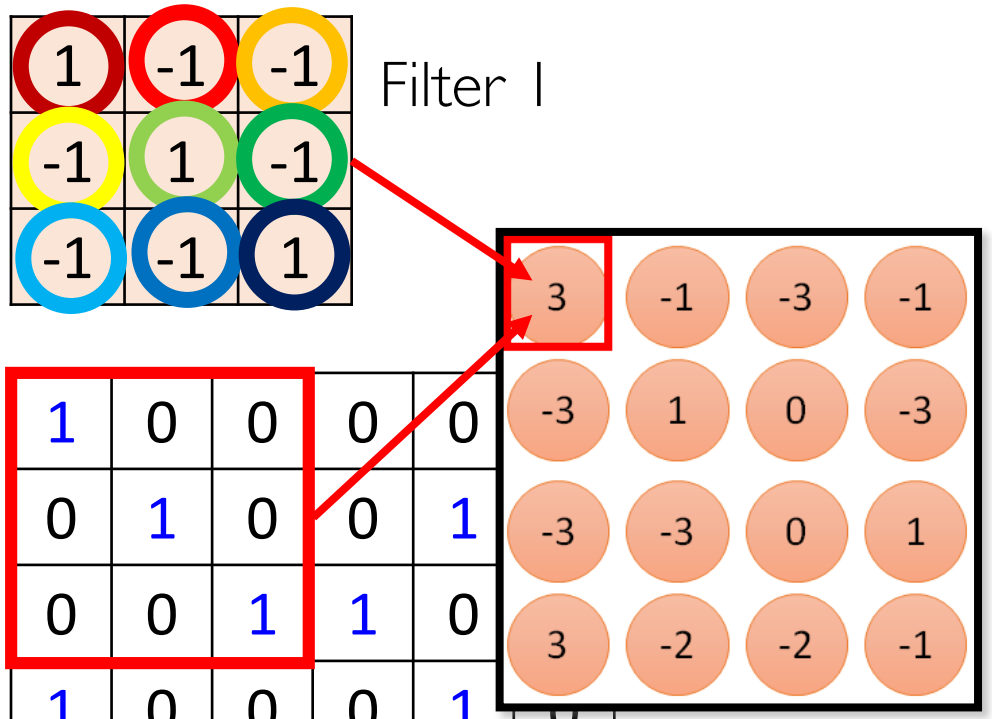
4x4 filter: matrix of weights w_{ij}

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

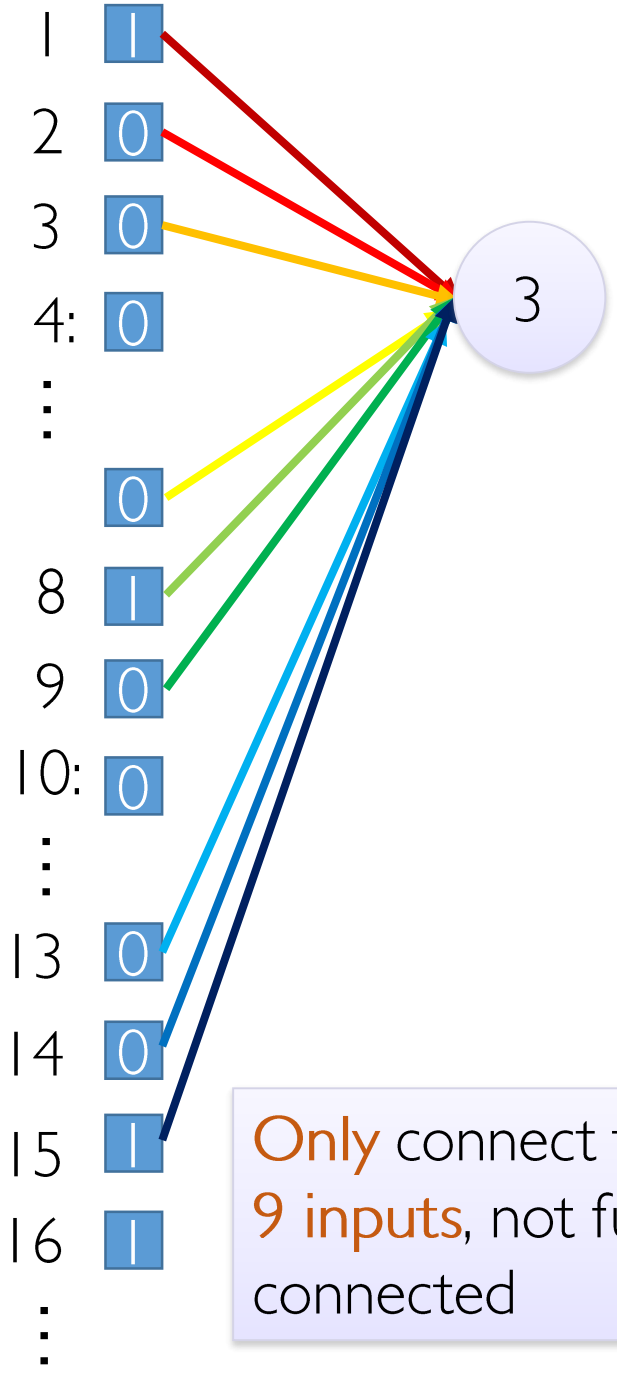
for neuron (p,q) in hidden layer

- For a neuron in hidden layer:
  - Take inputs from patch
  - Compute weighted sum
  - Apply bias

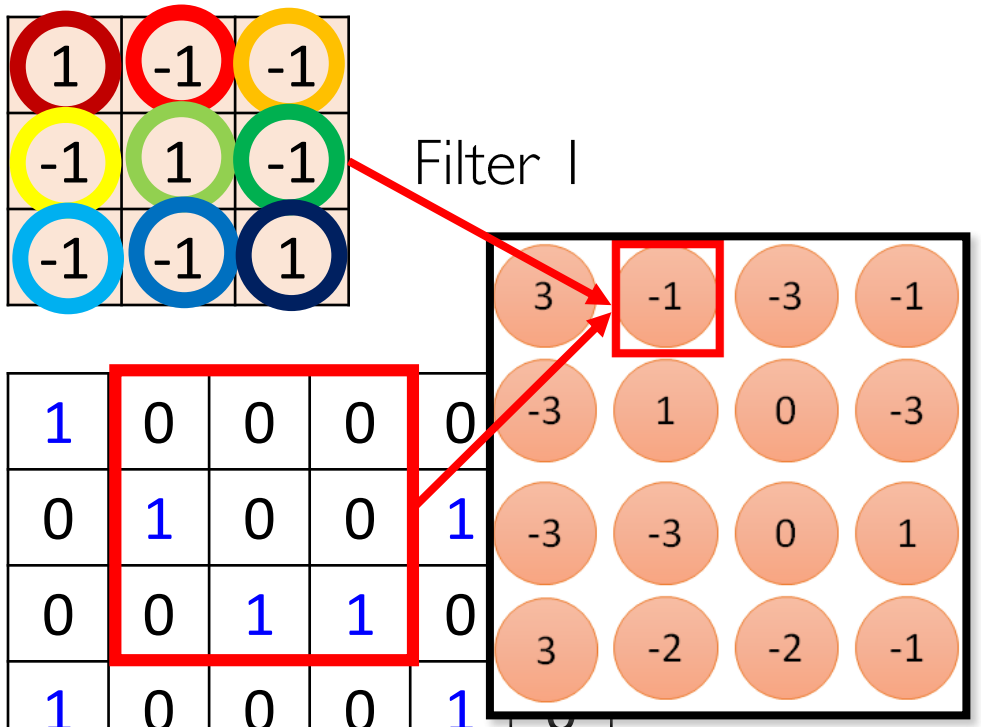
1) applying a window of **weights**
2) computing **linear** combinations
3) **activating** with non-linear function



fewer parameters!

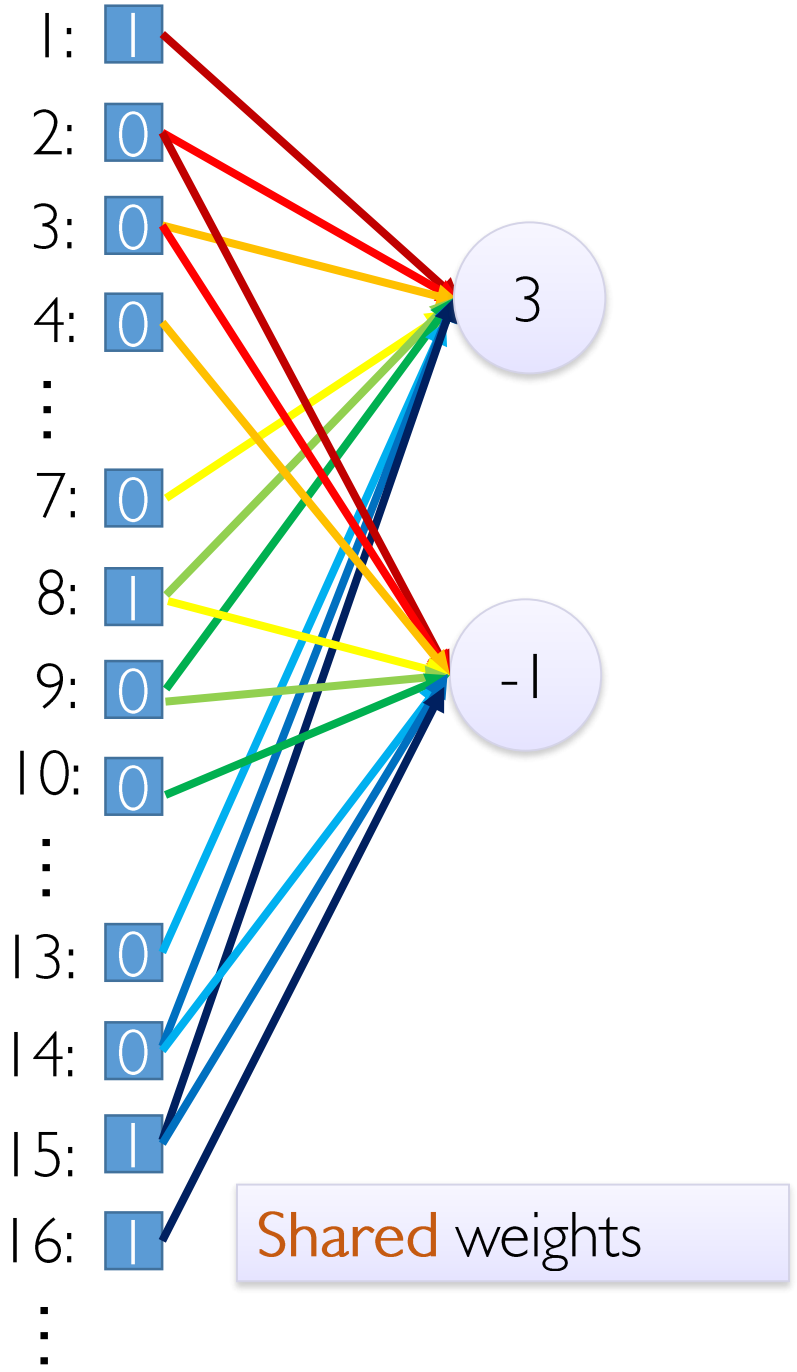


Only connect to 9 inputs, not fully connected

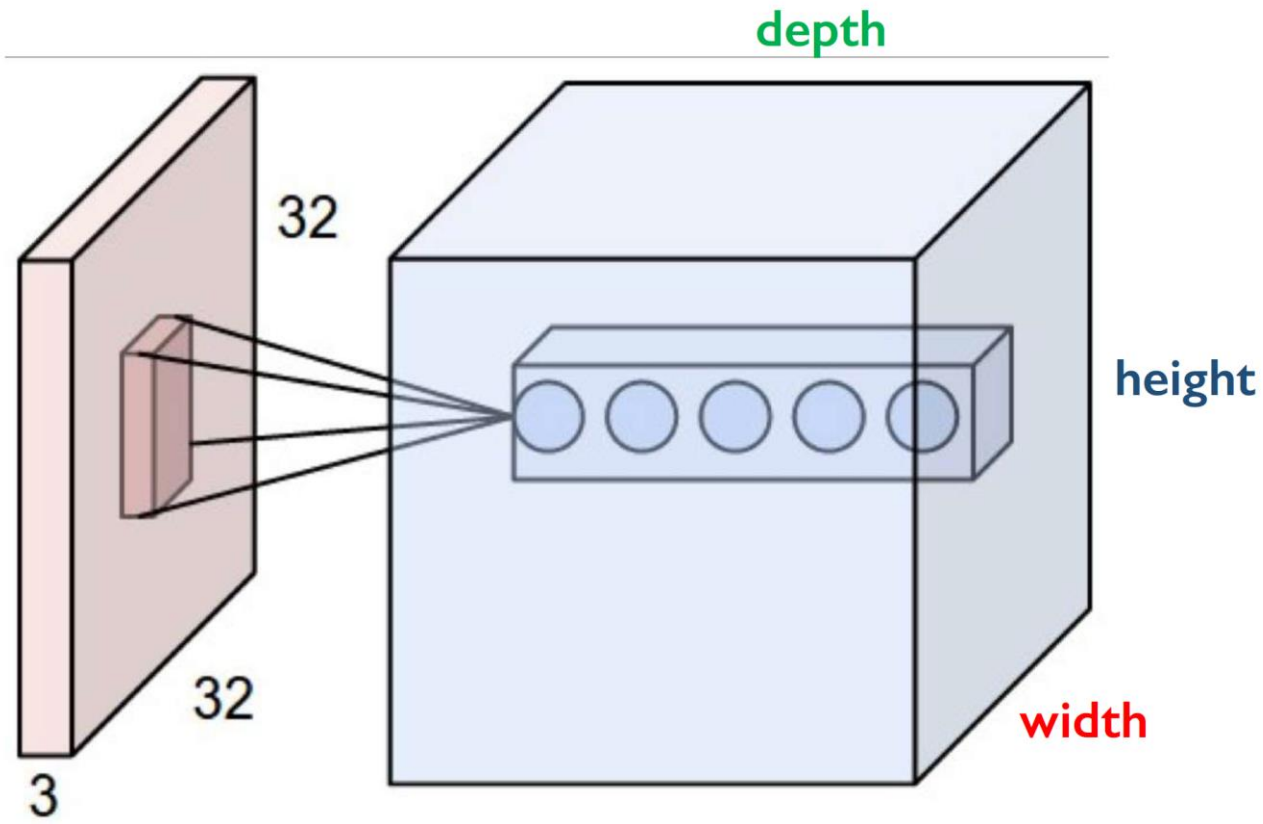


Fewer parameters

Even fewer parameters



CNNs: Spatial Arrangement of Output Volume



Layer Dimensions:

h * W * d

where h and w are **spatial dimensions**

d (**depth**) = number of filters

Receptive Field:

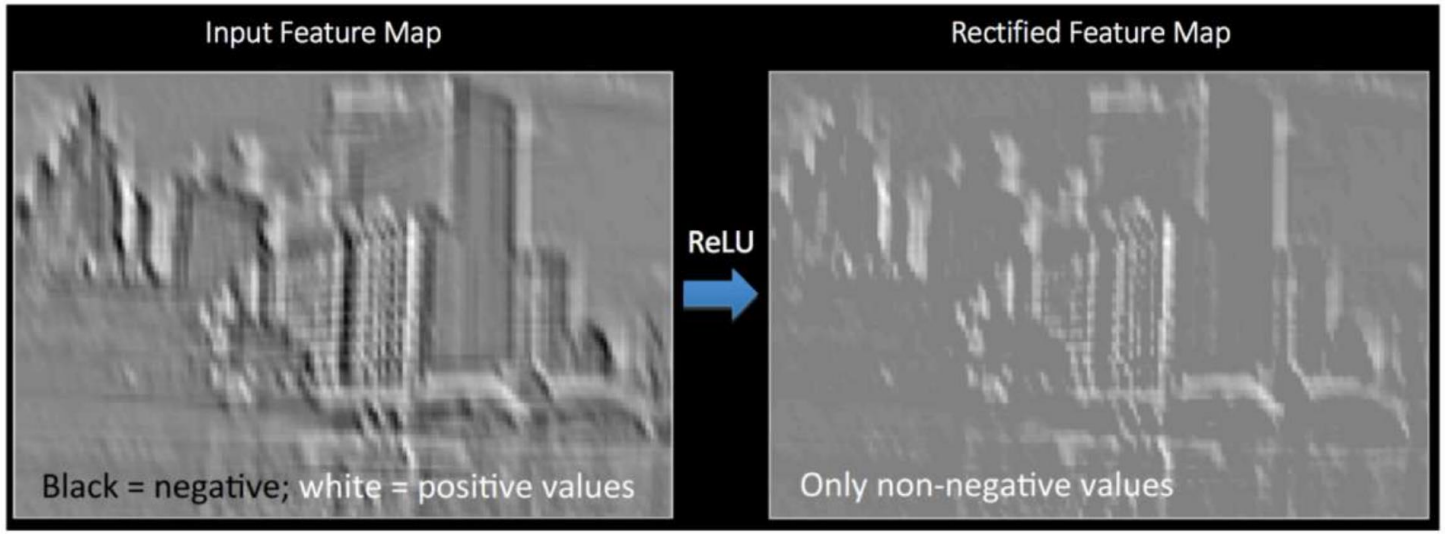
Locations in input image that a node is path connected to

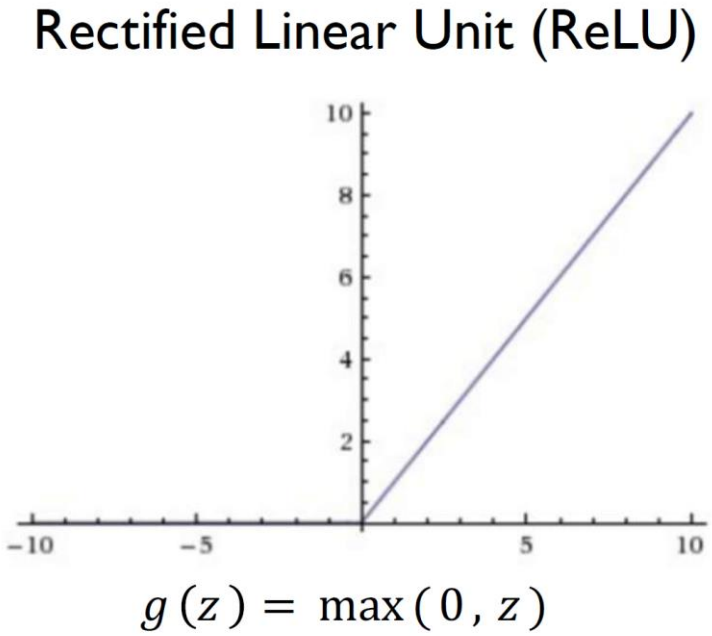
Stride:

Filter step size

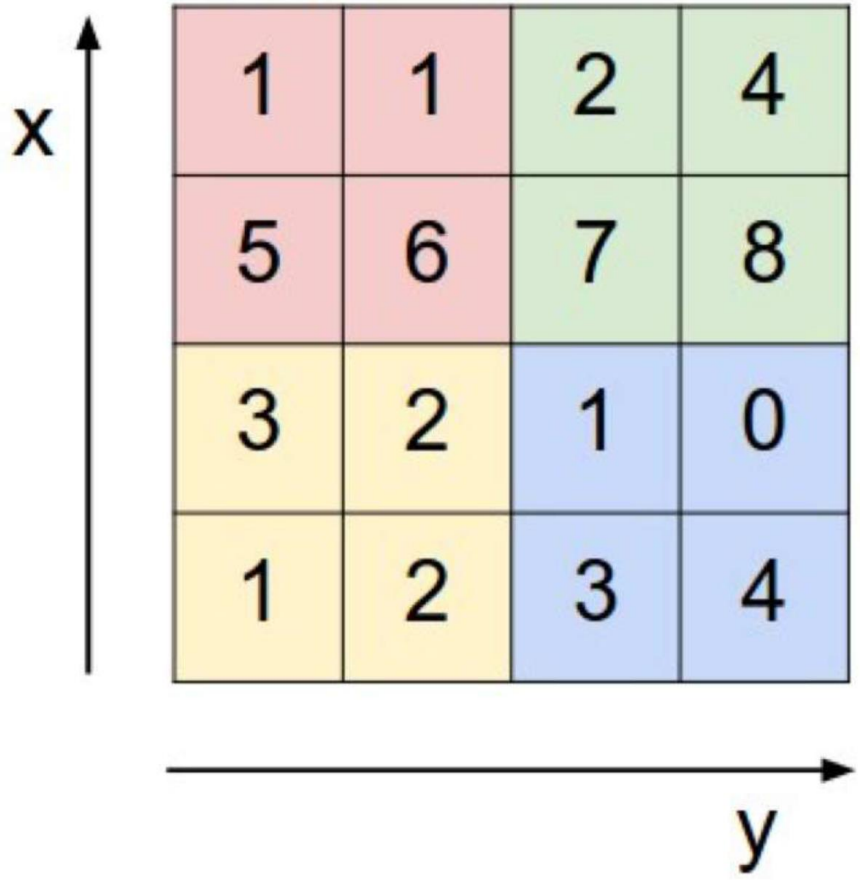
Introducing Non-Linearity

- 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!

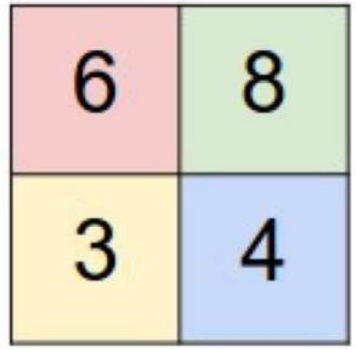




Pooling



max pool with 2x2 filters
and stride 2



1) Reduced dimensionality
2) Spatial invariance

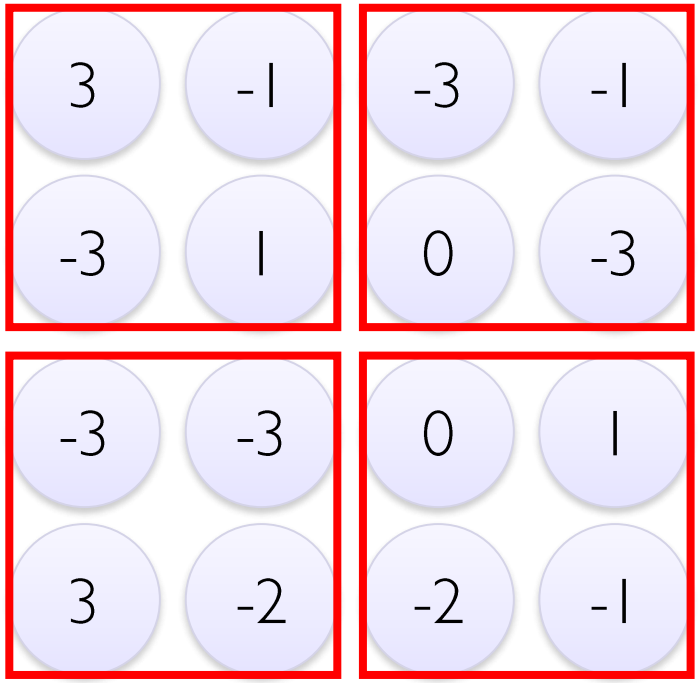
Max Pooling

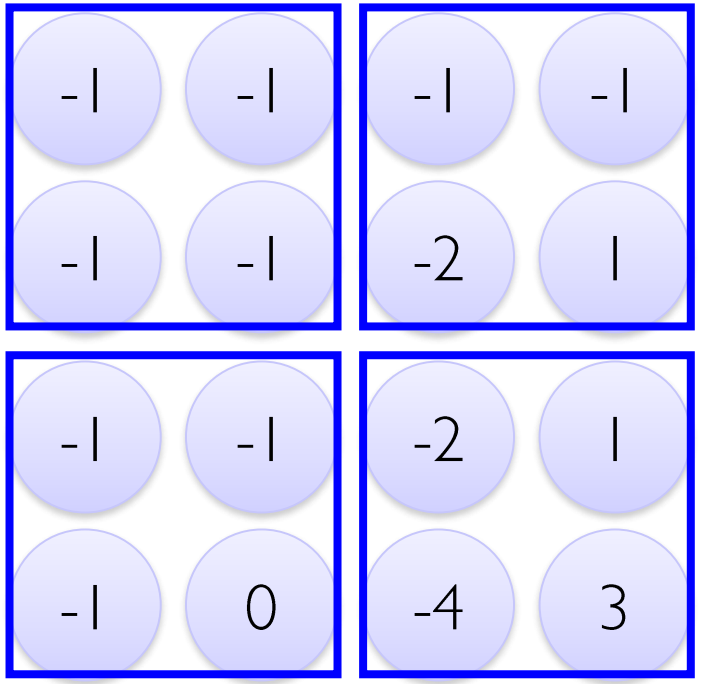
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

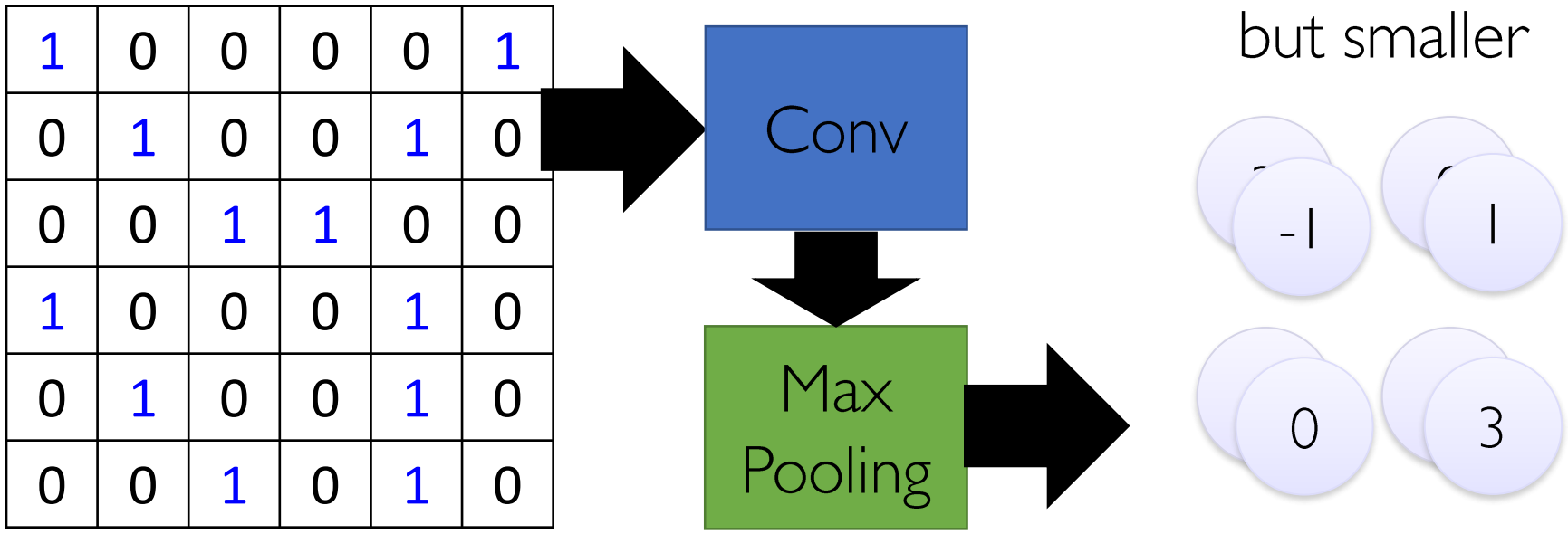
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2





Max Pooling



6 x 6 image

Each filter is a channel

Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

We can subsample the pixels to make image smaller

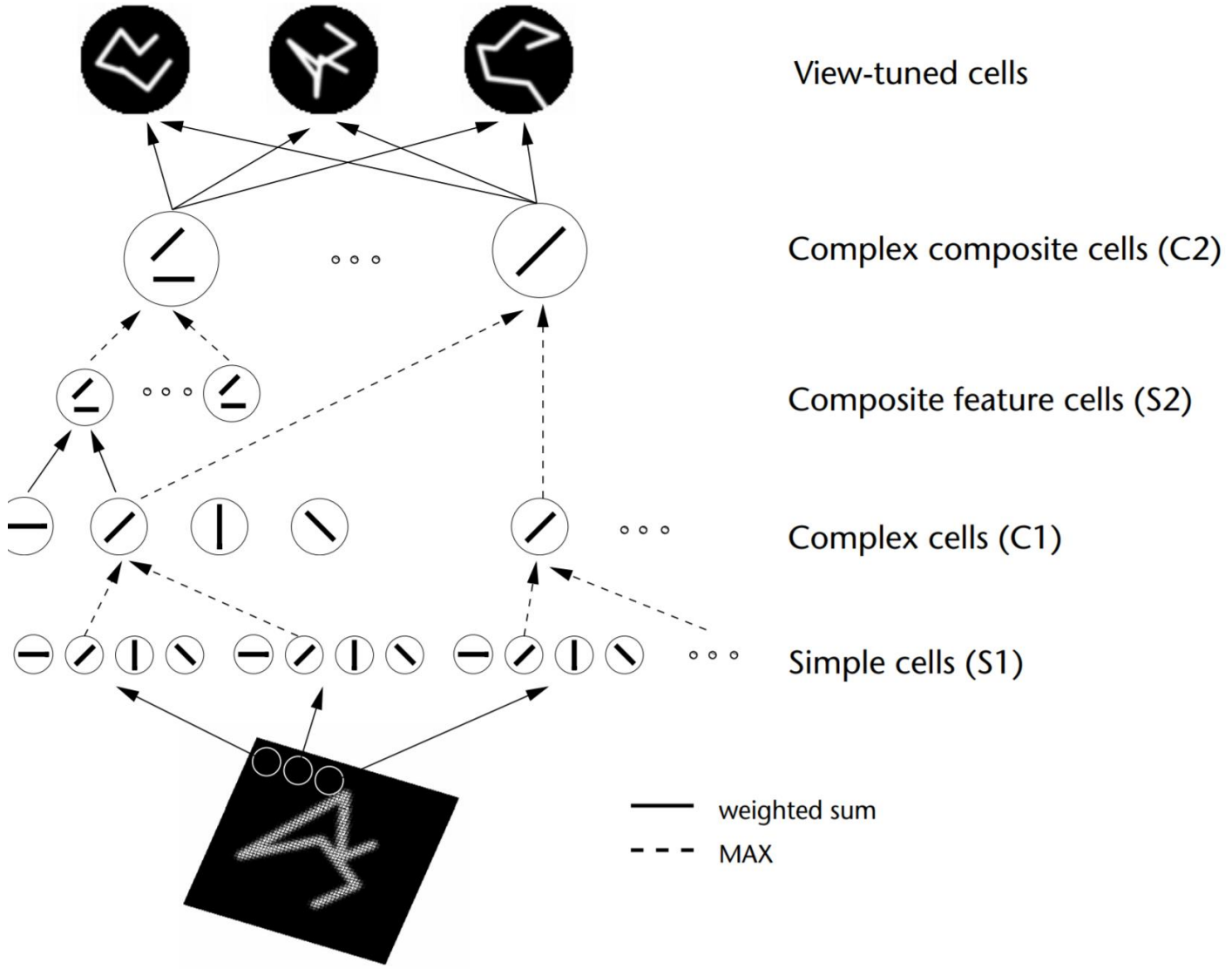
→ fewer parameters to characterize the image

A CNN compresses a fully connected network in three ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

H-Max model

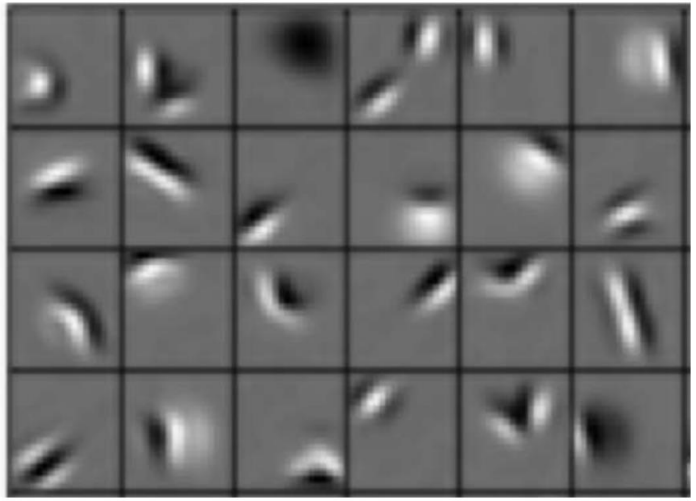
- Based on neuroscience studies, max pooling has replaced winner-take-all in pooling layers, thereby providing impressive performance gains
- Shunting inhibition



Riesenhuber, Maximilian, and Tomaso Poggio. "Hierarchical models of object recognition in cortex." *Nature neuroscience* 2.11 (1999): 1019.

Representation Learning in Deep CNNs

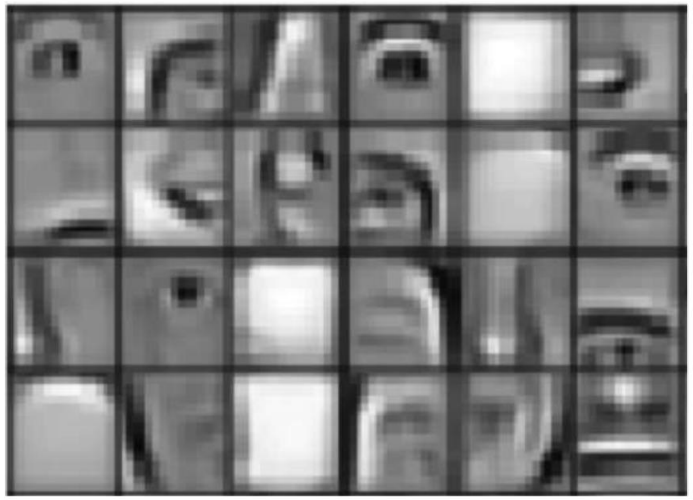
Low level features



Edges, dark spots

Conv Layer 1

Mid level features



Eyes, ears, nose

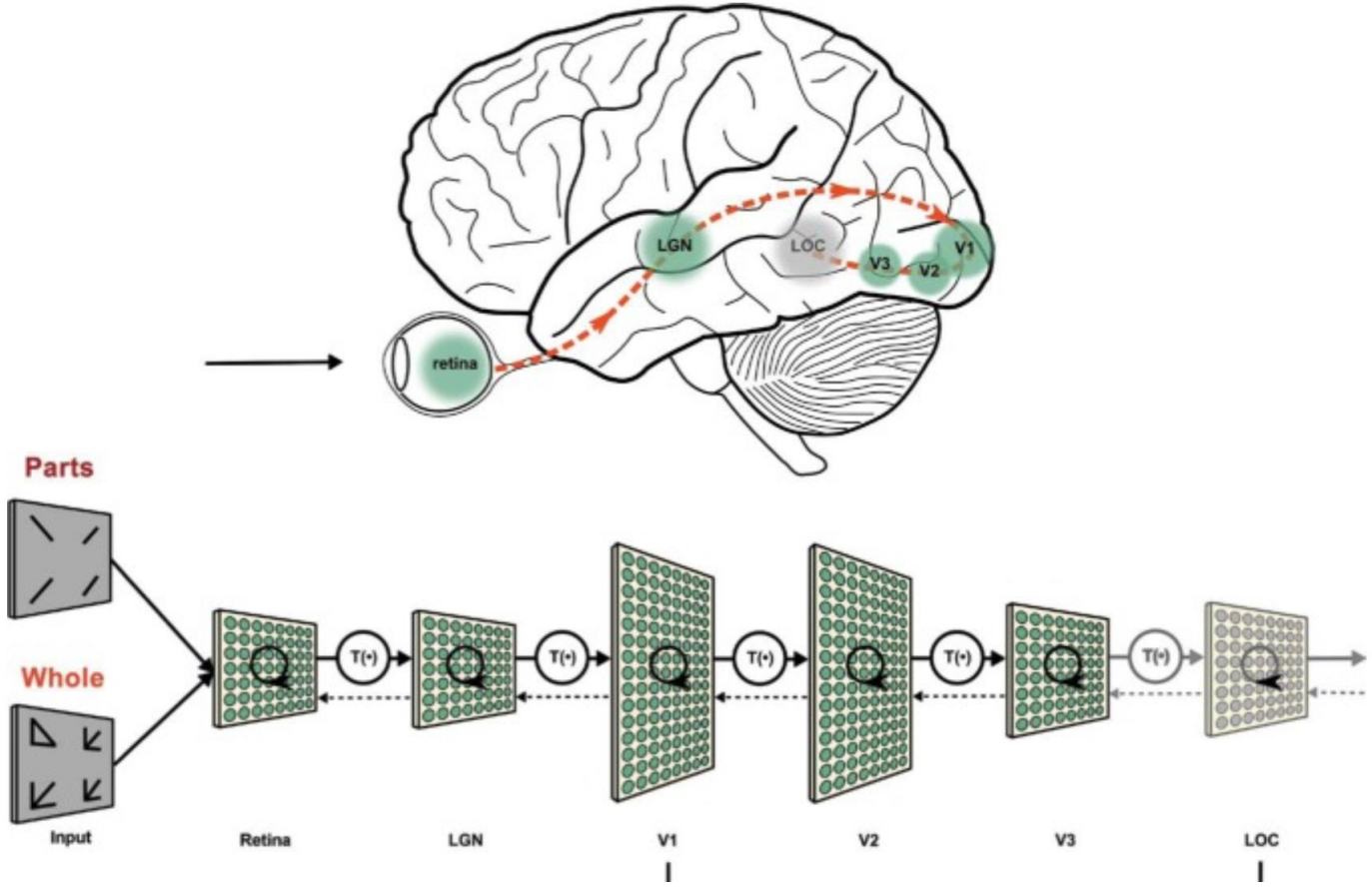
Conv Layer 2

High level features

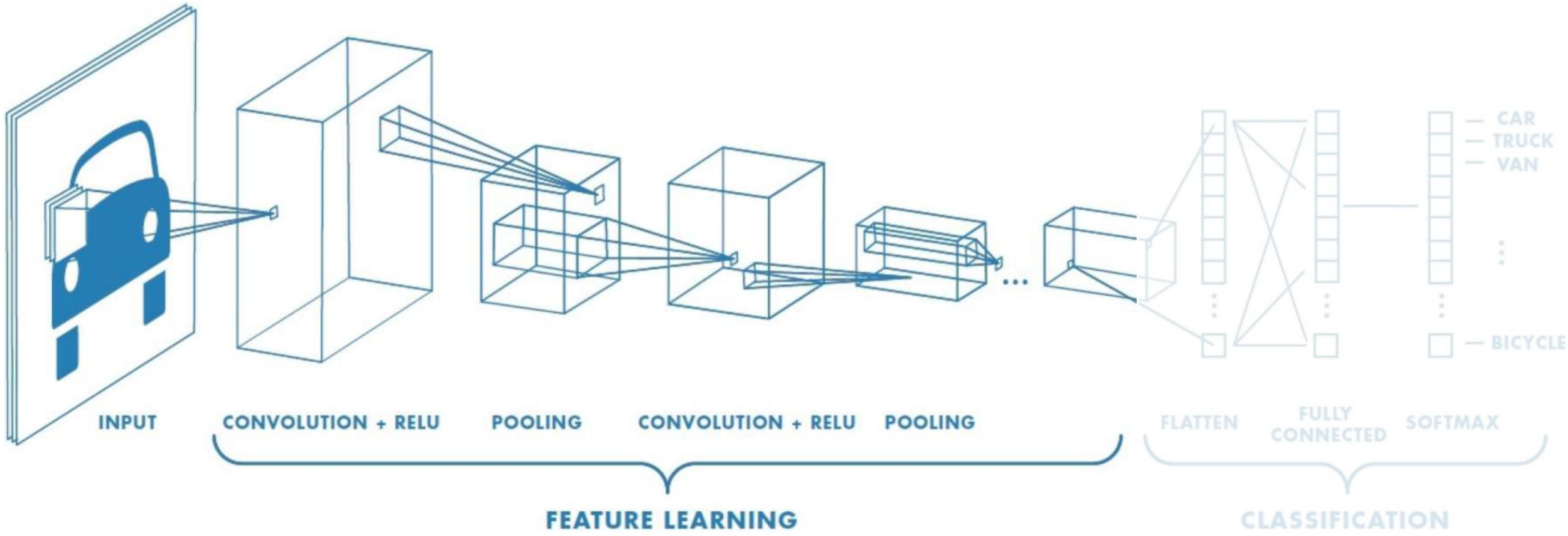
Facial structure

Conv Layer 3

Hierarchical visual processing to transform images into a better format across many layers

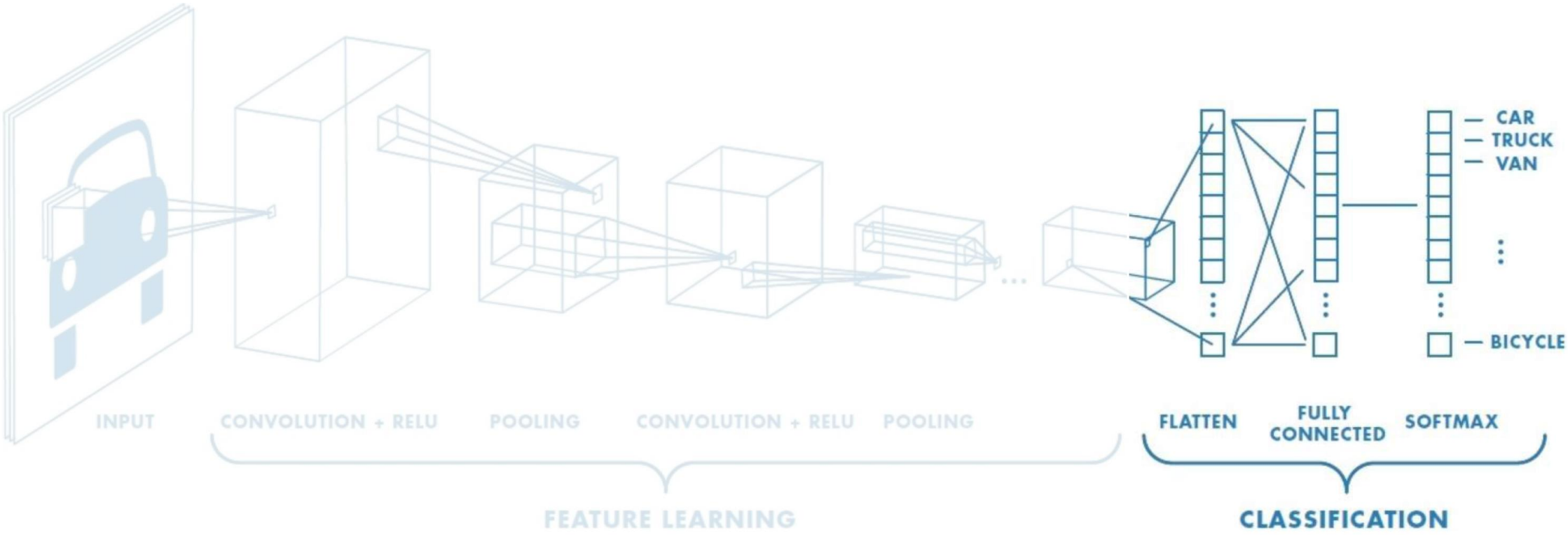


CNNs for Classification: Feature Learning



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

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

The whole CNN

Convolution

Max Pooling

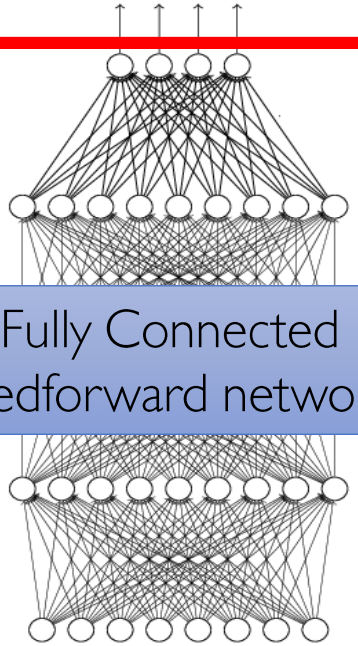
A new image

Convolution

Max Pooling

A new image

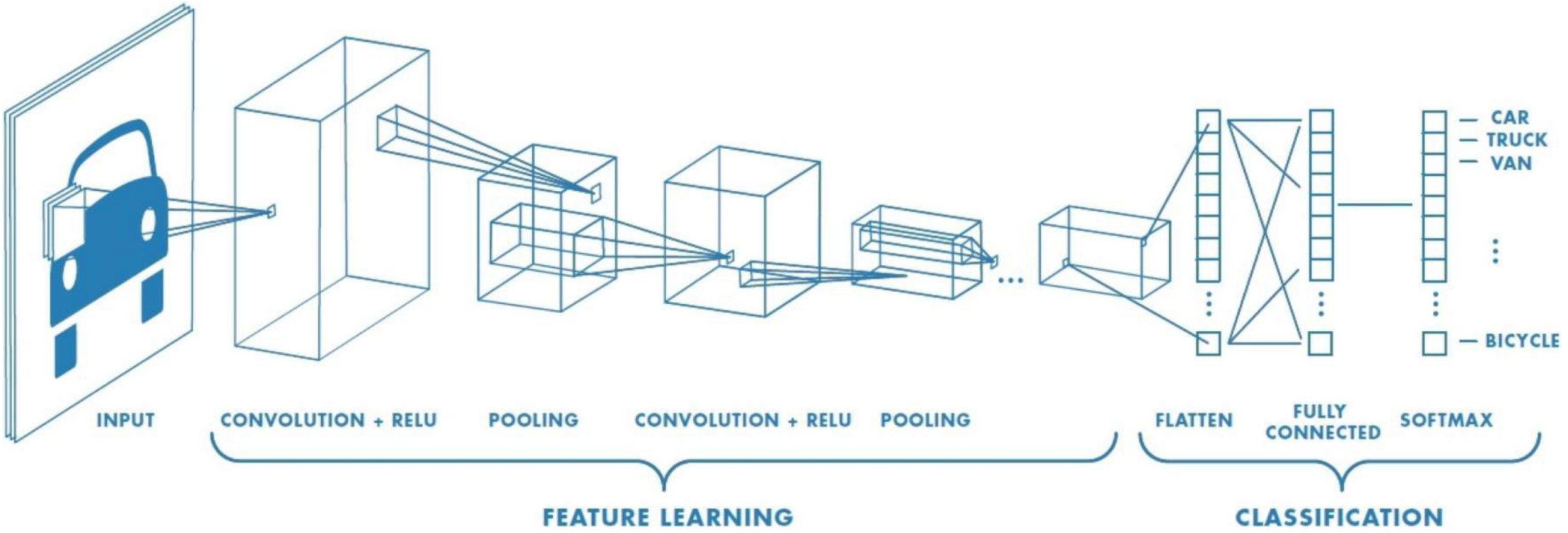
cat dog ……



Fully Connected
Feedforward network

Flattened

CNNs: Training with Backpropagation

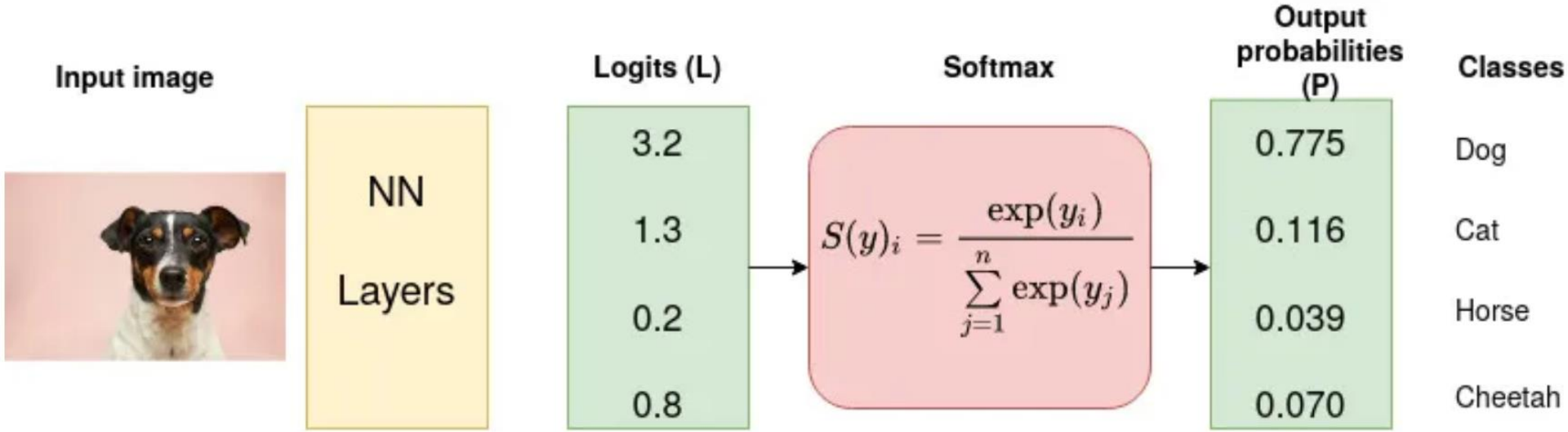


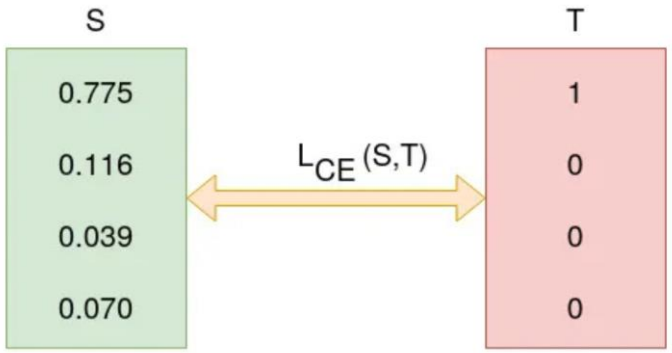
Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

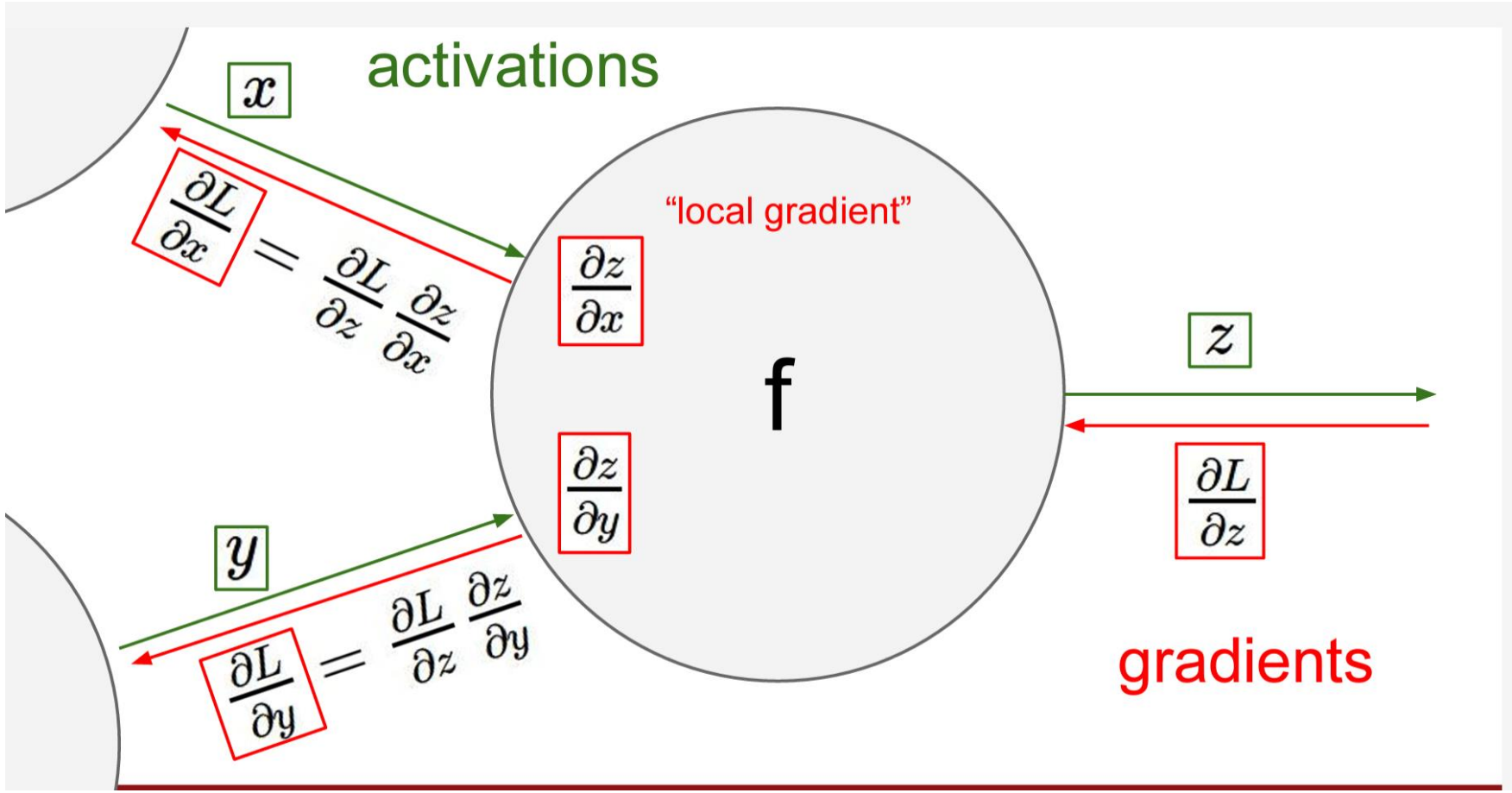
$$\boldsymbol{J}(\boldsymbol{\theta}) = \sum_i y^{(i)} \log\left(\hat{\boldsymbol{y}}^{(i)}\right)$$

Cross-entropy





Back propagation



Input and kernels

$$X = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0.5 & -0.3 \\ 0.8 & -0.6 \end{bmatrix}, \quad W_2 = \begin{bmatrix} -0.2 & 0.7 \\ 0.4 & -0.1 \end{bmatrix}$$

Convolution Operation

Stride = 1 (default). Padding = 0.

Feature Map 1 (using W_1):

$$S_1(i,j) = \sum_{m,n} X[i+m, j+n] \cdot W_1[m,n]$$

Compute sliding window:

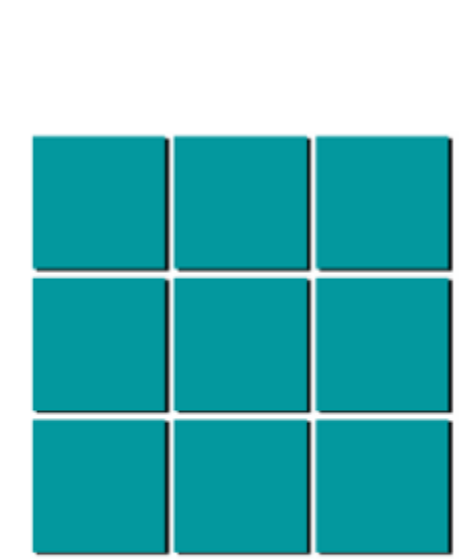
$$S_1(1,1) = (1)(0.5) + (2)(-0.3) + (0)(0.8) + (1)(-0.6) = 0.5 - 0.6 + 0 - 0.6 = -0.7$$
$$S_1(1,2) = (2)(0.5) + (1)(-0.3) + (1)(0.8) + (2)(-0.6) = 1.0 - 0.3 + 0.8 - 1.2 = 0.3$$

$$\vdots \quad \text{(Continue for all positions.)}$$

Complete:

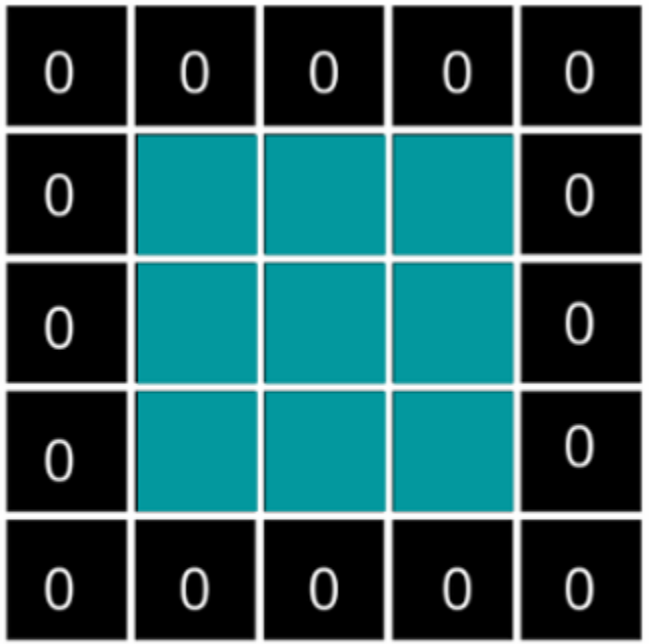
$$S_1 = \begin{bmatrix} -0.7 & 0.3 & -0.9 \\ 0.1 & -0.5 & 0.3 \\ -0.7 & 0.3 & -0.9 \end{bmatrix} \qquad S_2 = \begin{bmatrix} 1.1 & 0.7 & 1.3 \\ 0.9 & 1.2 & 0.8 \\ 1.1 & 0.7 & 1.3 \end{bmatrix}$$

Padding: keeps the input size consistent during convolutions, preventing data loss at the edges



Input Image

Applying padding of 1 on 3x3



Padded Image

MaxPooling (2x2 window, stride=2)

For S_1:

$$\text{MaxPooling}(S_1) = \begin{bmatrix} 0.3 & 0.3 \\ 0.3 & 0.3 \end{bmatrix}$$

For S_2:

$$\text{MaxPooling}(S_2) = \begin{bmatrix} 1.2 & 1.3 \\ 1.2 & 1.3 \end{bmatrix}$$

AveragePooling (2x2 window, stride=2)

For S_1:

$$\text{AveragePooling}(S_1) = \begin{bmatrix} -0.45 & -0.2 \\ -0.45 & -0.2 \end{bmatrix}$$

For S_2:

$$\text{AveragePooling}(S_2) = \begin{bmatrix} 0.975 & 0.875 \\ 0.975 & 0.875 \end{bmatrix}$$

Solving a baby example

Inputs:

$$X = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Convolution kernel (W):

$$W = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}, \quad \text{Bias} = 0.5$$

Fully Connected Weights:

$$W_{\text{fc}} = \begin{bmatrix} 0.2 & 0.3 & -0.5 & 0.7 \\ -0.6 & 0.1 & 0.4 & 0.5 \end{bmatrix}, \quad b_{\text{fc}} = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$$

Target:

$$\text{True Label (one-hot)} = [1, 0]$$

Forward Pass

1.1 Convolution:

Previously calculated:

$$Z = \begin{bmatrix} 4.5 & 5.5 & 3.5 \\ 6.5 & 8.5 & 6.5 \\ 9.5 & 11.5 & 9.5 \end{bmatrix}$$

1.2 ReLU Activation:

$$A = \begin{bmatrix} 4.5 & 5.5 & 3.5 \\ 6.5 & 8.5 & 6.5 \\ 9.5 & 11.5 & 9.5 \end{bmatrix}$$

1.3 Max Pooling:

$$M = \begin{bmatrix} 8.5 & 6.5 \\ 11.5 & 9.5 \end{bmatrix}$$

Flatten M:

$$\mathbf{M}_{\text{flat}} = [8.5, 6.5, 11.5, 9.5]$$

Fully connected

1.4 Fully Connected Layer:

Compute logits:

$$\mathbf{y} = W_{\text{fc}} \cdot \mathbf{M}_{\text{flat}} + b_{\text{fc}}$$

$$\mathbf{y} = \begin{bmatrix} 0.2 & 0.3 & -0.5 & 0.7 \\ -0.6 & 0.1 & 0.4 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 8.5 \\ 6.5 \\ 11.5 \\ 9.5 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$$

First row:

$$(0.2 \cdot 8.5) + (0.3 \cdot 6.5) + (-0.5 \cdot 11.5) + (0.7 \cdot 9.5) + 0.1 = 1.7 + 1.95 - 5.75 + 6.65 + 0.1 = 4.65$$

Second row:

$$(-0.6 \cdot 8.5) + (0.1 \cdot 6.5) + (0.4 \cdot 11.5) + (0.5 \cdot 9.5) - 0.2 = -5.1 + 0.65 + 4.6 + 4.75 - 0.2 = 4.7$$

Logits:

$$\mathbf{y} = \begin{bmatrix} 4.65 \\ 4.7 \end{bmatrix}$$

Soft max

1.5 Softmax Activation:

$$\text{Softmax}(\mathbf{y})_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Exponentials:

$$e^{4.65} \approx 104.7, \quad e^{4.7} \approx 109.9$$

Softmax:

$$\text{Softmax}(\mathbf{y}) = \begin{bmatrix} \frac{104.7}{104.7+109.9} \\ \frac{109.9}{104.7+109.9} \end{bmatrix} = \begin{bmatrix} 0.487 \\ 0.513 \end{bmatrix}$$

1.6 Cross-Entropy Loss:

$$\text{Loss} = -\sum_i \text{True Label}_i \cdot \log(\text{Predicted}_i)$$

$$\text{Loss} = -\log(0.487) \approx 0.719$$

Step 2: Backward Pass

2.1 Fully Connected Layer Gradients:

Softmax gradient:

$$\frac{\partial \text{Loss}}{\partial \mathbf{y}} = \text{Softmax}(\mathbf{y}) - \text{True Label}$$

$$\frac{\partial \text{Loss}}{\partial \mathbf{y}} = \begin{bmatrix} 0.487 - 1 \\ 0.513 - 0 \end{bmatrix} = \begin{bmatrix} -0.513 \\ 0.513 \end{bmatrix}$$

Weight gradient (W_{fc}):

$$\frac{\partial \text{Loss}}{\partial W_{\text{fc}}} = \frac{\partial \text{Loss}}{\partial \mathbf{y}} \cdot \mathbf{M}_{\text{flat}}^{\top}$$

Gradient:

$$\frac{\partial \text{Loss}}{\partial W_{\text{fc}}} = \begin{bmatrix} -0.513 \\ 0.513 \end{bmatrix} \cdot \begin{bmatrix} 8.5 & 6.5 & 11.5 & 9.5 \end{bmatrix}$$

Backpropagation Through the Fully Connected Layer

The gradient at the output of the fully connected layer, $\frac{\partial \text{Loss}}{\partial \mathbf{M}_{\text{flat}}}$, is computed as:

$$\frac{\partial \text{Loss}}{\partial \mathbf{M}_{\text{flat}}} = W_{\text{fc}}^{\top} \cdot \frac{\partial \text{Loss}}{\partial \mathbf{y}}$$

Given:

$$\frac{\partial \text{Loss}}{\partial \mathbf{y}} = \begin{bmatrix} -0.513 \\ 0.513 \end{bmatrix}, \quad W_{\text{fc}} = \begin{bmatrix} 0.2 & 0.3 & -0.5 & 0.7 \\ -0.6 & 0.1 & 0.4 & 0.5 \end{bmatrix}$$

Transpose:

$$W_{\text{fc}}^{\top} = \begin{bmatrix} 0.2 & -0.6 \\ 0.3 & 0.1 \\ -0.5 & 0.4 \\ 0.7 & 0.5 \end{bmatrix}$$

Matrix product:

$$\frac{\partial \text{Loss}}{\partial \mathbf{M}_{\text{flat}}} = \begin{bmatrix} 0.2 & -0.6 \\ 0.3 & 0.1 \\ -0.5 & 0.4 \\ 0.7 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} -0.513 \\ 0.513 \end{bmatrix}$$

Matrix product:

$$\frac{\partial \text{Loss}}{\partial \mathbf{M}_{\text{flat}}} = \begin{bmatrix} 0.2 & -0.6 \\ 0.3 & 0.1 \\ -0.5 & 0.4 \\ 0.7 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} -0.513 \\ 0.513 \end{bmatrix}$$

Element-wise computation:

$$\frac{\partial \text{Loss}}{\partial \mathbf{M}_{\text{flat}}} = \begin{bmatrix} (0.2 \cdot -0.513) + (-0.6 \cdot 0.513) \\ (0.3 \cdot -0.513) + (0.1 \cdot 0.513) \\ (-0.5 \cdot -0.513) + (0.4 \cdot 0.513) \\ (0.7 \cdot -0.513) + (0.5 \cdot 0.513) \end{bmatrix}$$

$$\frac{\partial \text{Loss}}{\partial \mathbf{M}_{\text{flat}}} = \begin{bmatrix} -0.1026 - 0.3078 \\ -0.1539 + 0.0513 \\ 0.2565 + 0.2052 \\ -0.3591 + 0.2565 \end{bmatrix} = \begin{bmatrix} -0.4104 \\ -0.1026 \\ 0.4617 \\ -0.1026 \end{bmatrix}$$

Backpropagate Through the Max Pooling Layer

The output of max pooling was:

$$M = \begin{bmatrix} 8.5 & 6.5 \\ 11.5 & 9.5 \end{bmatrix}$$

To backpropagate through the max pooling layer, assign the gradient values from $\frac{\partial \text{Loss}}{\partial \text{M}_{\text{flat}}}$ to the corresponding positions of the maximum values in the pooled matrix:

Original matrix A:

$$A = \begin{bmatrix} 4.5 & 5.5 & 3.5 \\ 6.5 & 8.5 & 6.5 \\ 9.5 & 11.5 & 9.5 \end{bmatrix}$$

Max pooling indices:

$$\begin{aligned} M_{1,1} &= 8.5 \quad (\text{from } A_{2,2}) \\ M_{1,2} &= 6.5 \quad (\text{from } A_{2,3}) \\ M_{2,1} &= 11.5 \quad (\text{from } A_{3,2}) \\ M_{2,2} &= 9.5 \quad (\text{from } A_{3,3}) \end{aligned}$$

Assign gradients:

$$\frac{\partial \text{Loss}}{\partial A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.4104 & -0.1026 \\ 0 & 0.4617 & -0.1026 \end{bmatrix}$$

Backpropagate Through the Convolution Layer

The **input matrix** X:

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

The **backpropagated gradient** from the max-pooling layer:

$$\frac{\partial \text{Loss}}{\partial A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.4104 & -0.1026 \\ 0 & 0.4617 & -0.1026 \end{bmatrix}$$

The **convolution kernel** W is of size 2×2:

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

Input Patches

For a 2×2 kernel, we extract overlapping patches from X, considering a stride of 1:

Patches of X:

1. Patch (1, 1):

$$X_{1,1} = \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix}$$

2. Patch (1, 2):

$$X_{1,2} = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$$

3. Patch (2, 1):

$$X_{2,1} = \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$$

4. Patch (2, 2):

$$X_{2,2} = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$$

Computing ∂Loss/∂W: by summing the element-wise product of the input patches and the backpropagated gradients corresponding to the patches

The gradient of the kernel is computed by summing the element-wise product of the input patches and the backpropagated gradients corresponding to the patches:

$$\frac{\partial \text{Loss}}{\partial W} = \sum_{\text{all patches}} X_{\text{patch}} \cdot \frac{\partial \text{Loss}}{\partial A}_{\text{patch position}}$$

For Each Patch:

1. Patch (1, 1): Gradient from $\frac{\partial \text{Loss}}{\partial A}$:

$$\frac{\partial \text{Loss}}{\partial A}_{1,1} = 0$$

Contribution:

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \cdot 0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

2. Patch (1, 2): Gradient from $\frac{\partial \text{Loss}}{\partial A}$:

$$\frac{\partial \text{Loss}}{\partial A}_{1,2} = 0$$

Contribution:

$$\begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix} \cdot 0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

3. Patch (2, 1): Gradient from $\frac{\partial \text{Loss}}{\partial A}$:

$$\frac{\partial \text{Loss}}{\partial A}_{2,1} = -0.4104$$

Contribution:

$$\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \cdot (-0.4104) = \begin{bmatrix} -1.6416 & -2.052 \\ -2.8728 & -3.2832 \end{bmatrix}$$

4. Patch (2, 2): Gradient from $\frac{\partial \text{Loss}}{\partial A}$:

$$\frac{\partial \text{Loss}}{\partial A}_{2,2} = -0.1026$$

Contribution:

$$\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix} \cdot (-0.1026) = \begin{bmatrix} -0.513 & -0.6156 \\ -0.8208 & -0.9234 \end{bmatrix}$$

Summing Contributions

$$\frac{\partial \text{Loss}}{\partial W} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -1.6416 & -2.052 \\ -2.8728 & -3.2832 \end{bmatrix} + \begin{bmatrix} -0.513 & -0.6156 \\ -0.8208 & -0.9234 \end{bmatrix}$$

Element-wise sum:

$$\frac{\partial \text{Loss}}{\partial W} = \begin{bmatrix} -1.6416 - 0.513 & -2.052 - 0.6156 \\ -2.8728 - 0.8208 & -3.2832 - 0.9234 \end{bmatrix} = \begin{bmatrix} -2.1546 & -2.6676 \\ -3.6936 & -4.2066 \end{bmatrix}$$

Final Gradient of the Kernel

$$\frac{\partial \text{Loss}}{\partial W} = \begin{bmatrix} -2.1546 & -2.6676 \\ -3.6936 & -4.2066 \end{bmatrix}$$

Gradient Descent Formula

For each weight in the kernel, the update rule is:

$$W^{(t+1)} = W^{(t)} - \eta \cdot \frac{\partial \text{Loss}}{\partial W}$$

Where:

- $W^{(t)}$ is the kernel at the current iteration.
- $W^{(t+1)}$ is the updated kernel.

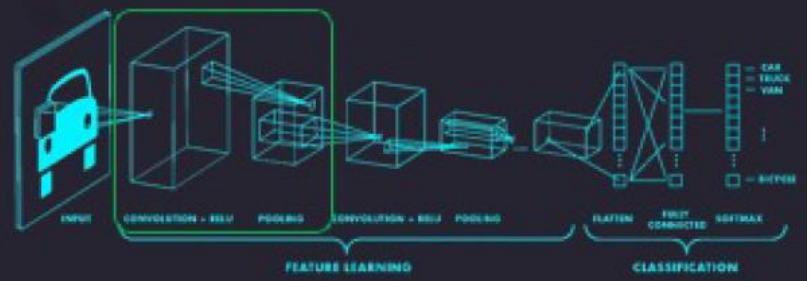
Tensorflow codes

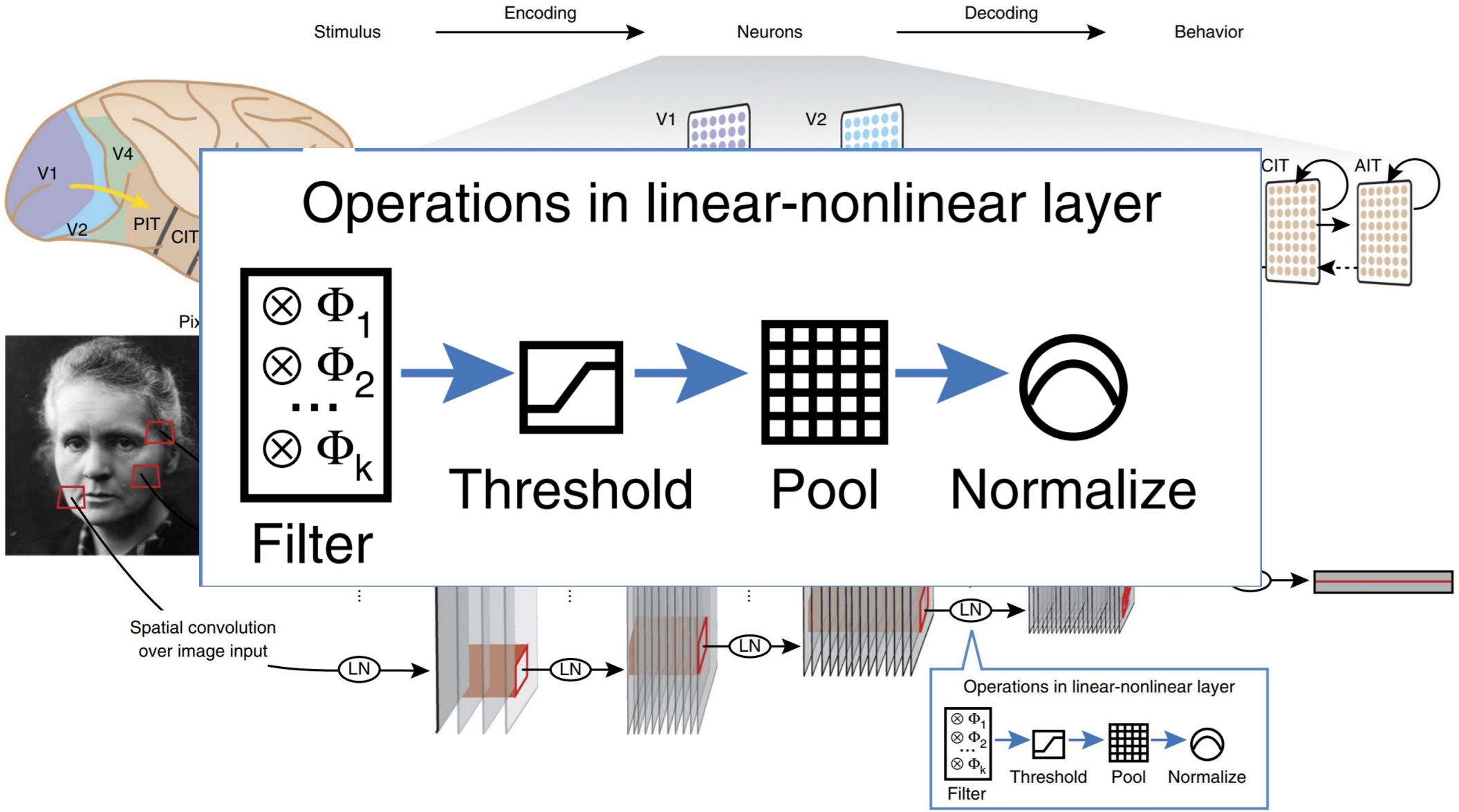
```python
import tensorflow as tf

def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # second convolutional layer
        tf.keras.layers.Conv2D(64, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # fully connected classifier
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')   # 10 outputs
    ])
    return model
```



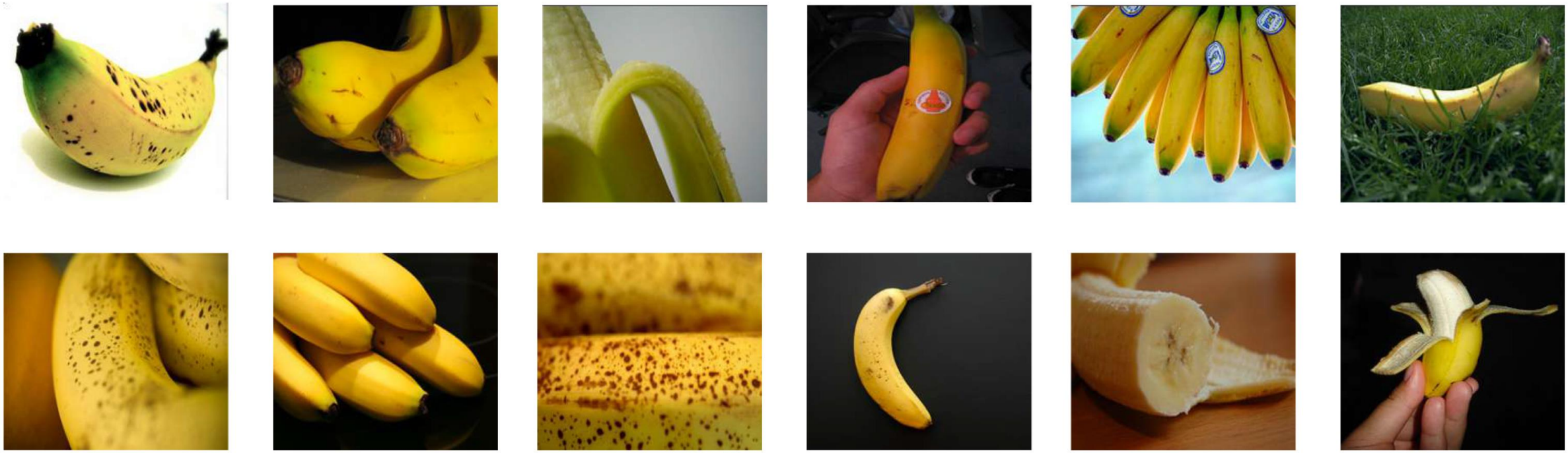


Yamins, Daniel LK, and James J. DiCarlo. "Using goal-driven deep learning models to understand sensory cortex." *Nature neuroscience* 19.3 (2016): 356.

ImageNet Dataset

Dataset of over 14 million images across 21,841 categories

"Elongated crescent-shaped yellow fruit with soft sweet flesh"

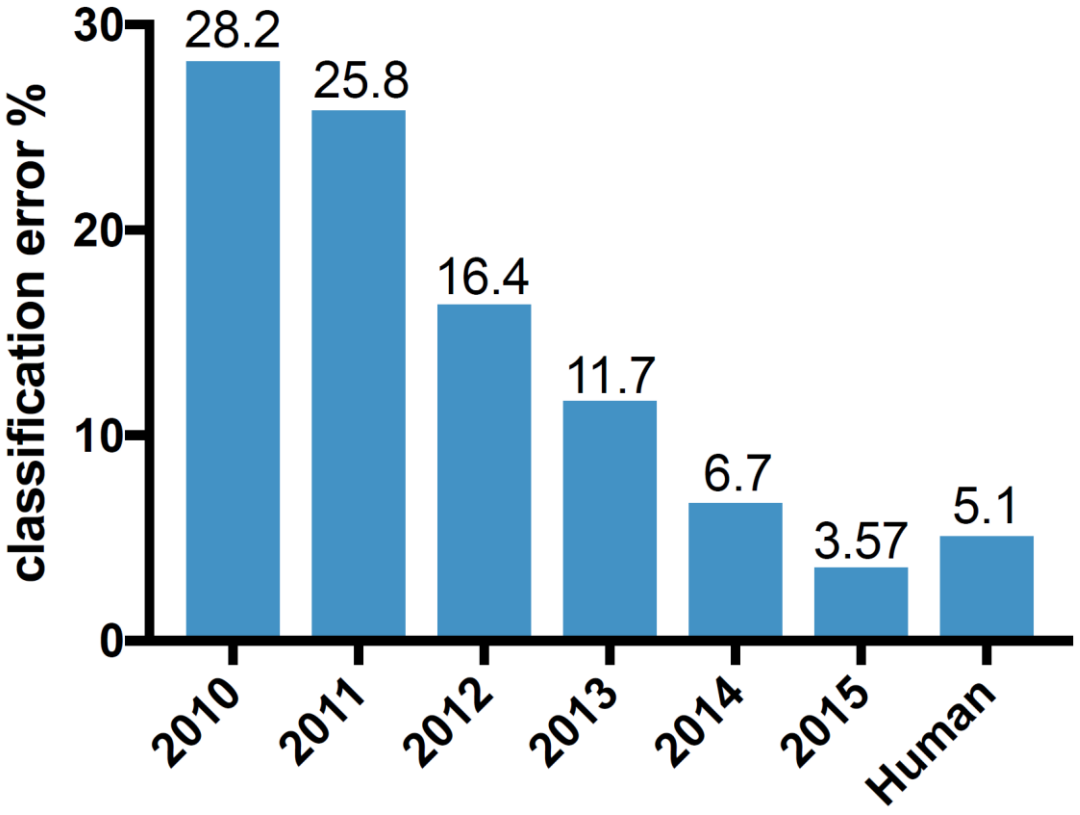


1409 pictures of bananas.

ImageNet Challenge: Classification Task

Classification task:

produce a list of object categories present in image. 1000 categories.



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

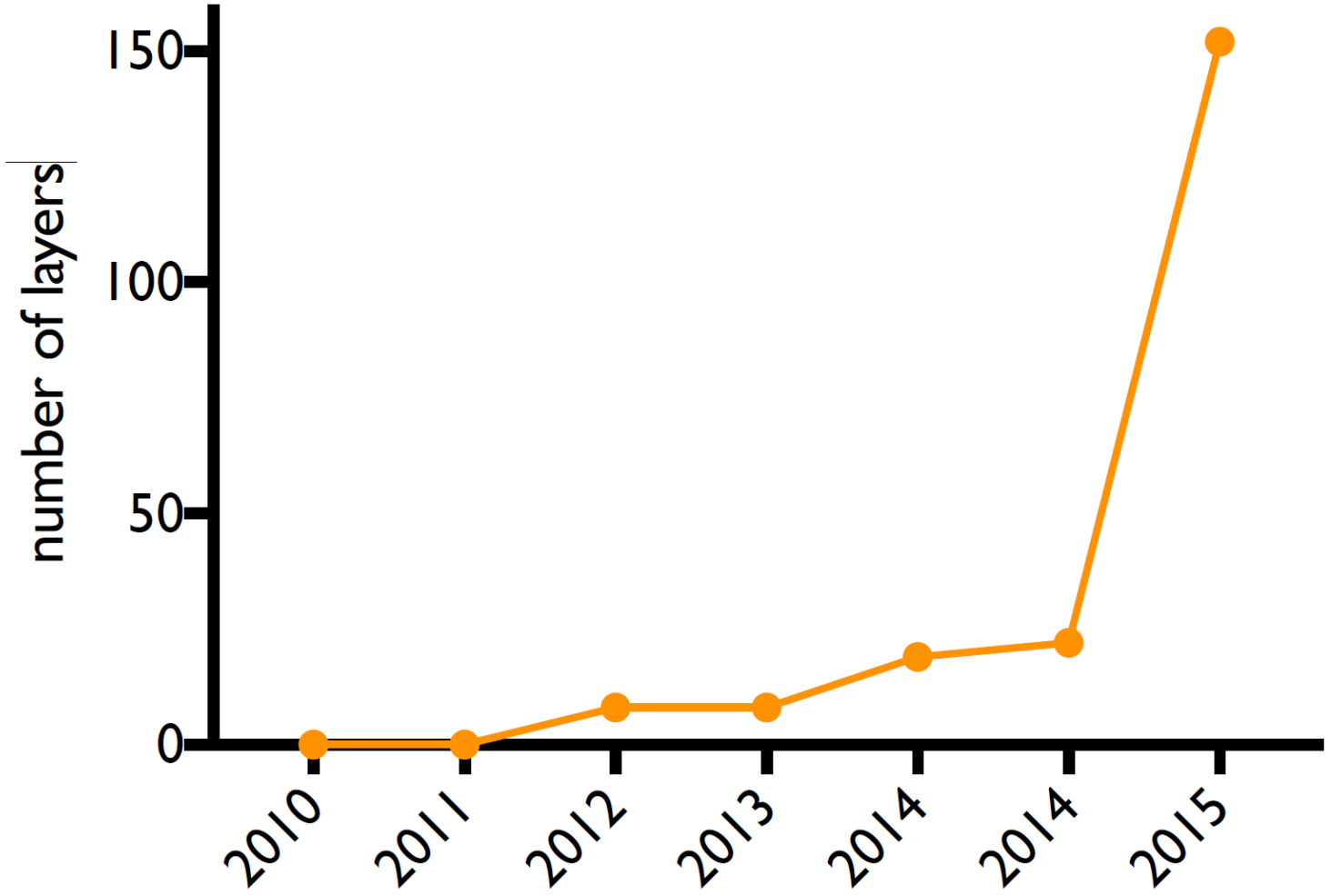
2013: **ZFNet**
- 8 layers, more filters

2014: **VGG**
- 19 layers

2014: **GoogLeNet**
- "Inception" modules
- 22 layers, 5million parameters

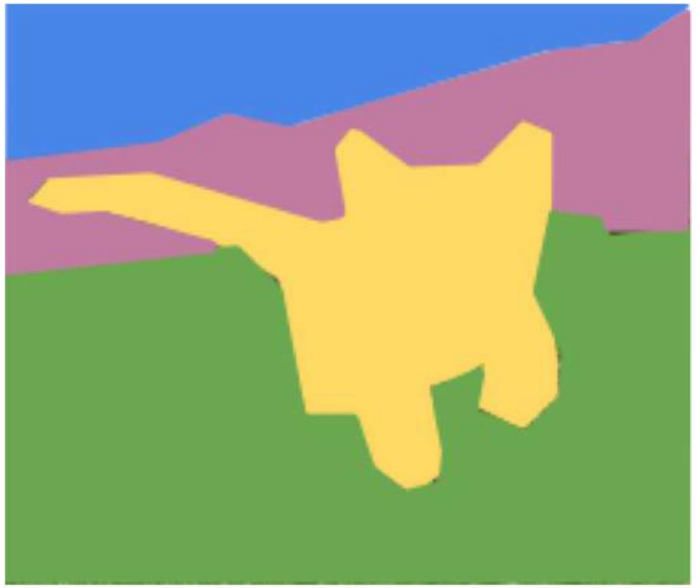
2015: **ResNet**
- 152 layers

Number of layers



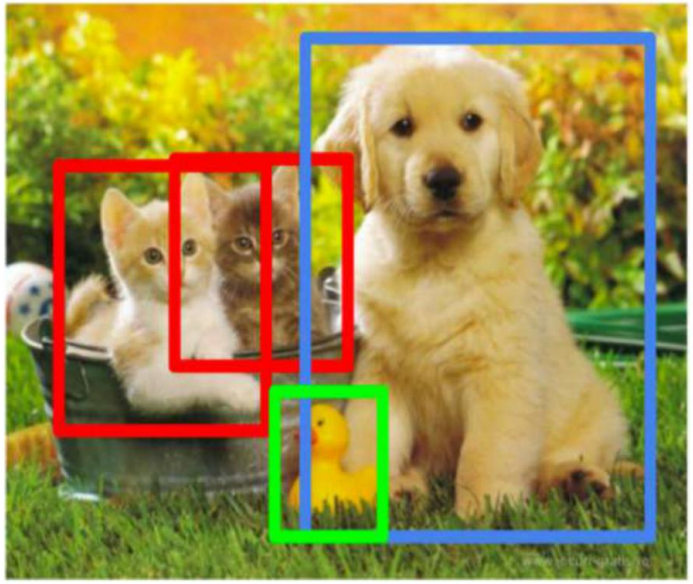
An Architecture for Many Applications; Beyond Classification

Semantic Segmentation



CAT

Object Detection



CAT, DOG, DUCK

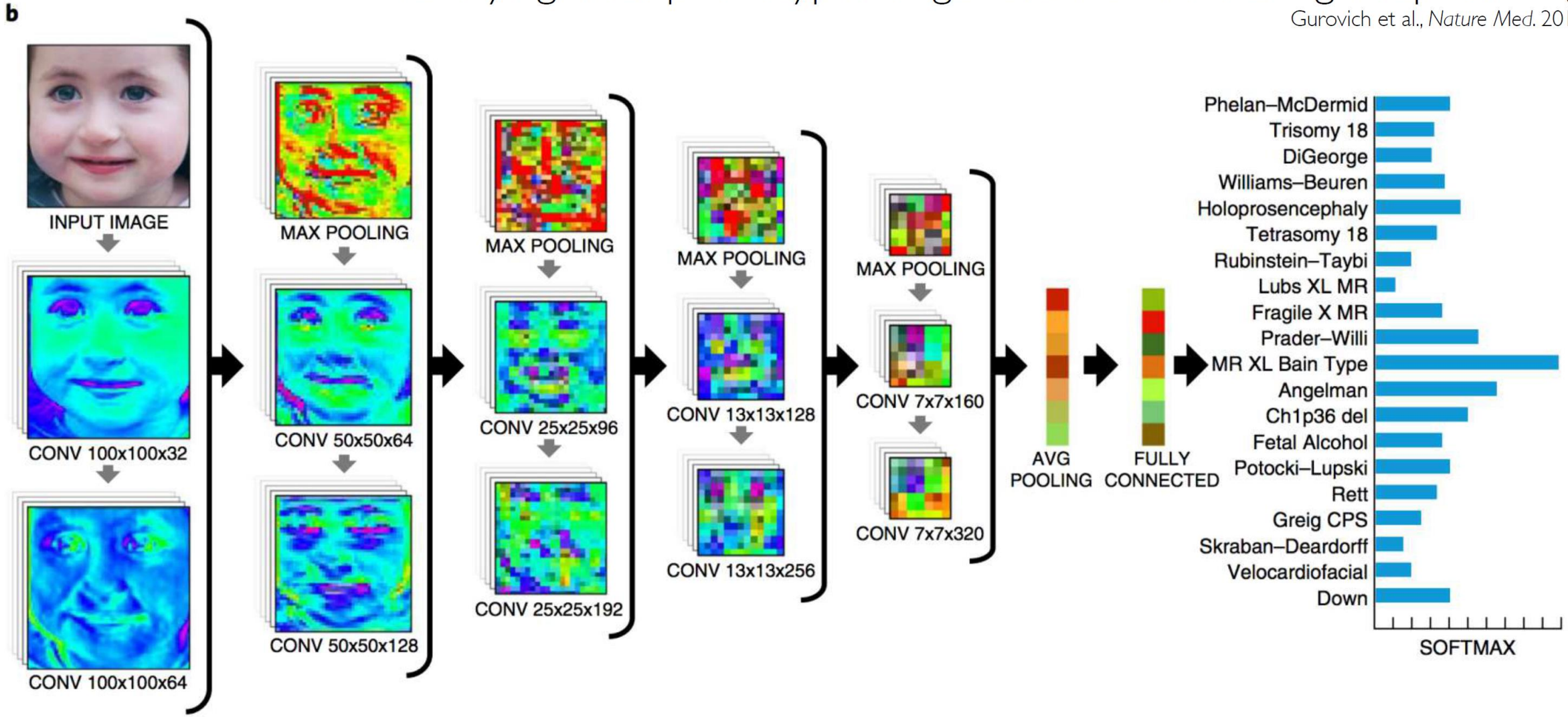
Image Captioning

The cat is in the grass.

Impact: Healthcare

Identifying facial phenotypes of genetic disorders using deep learning

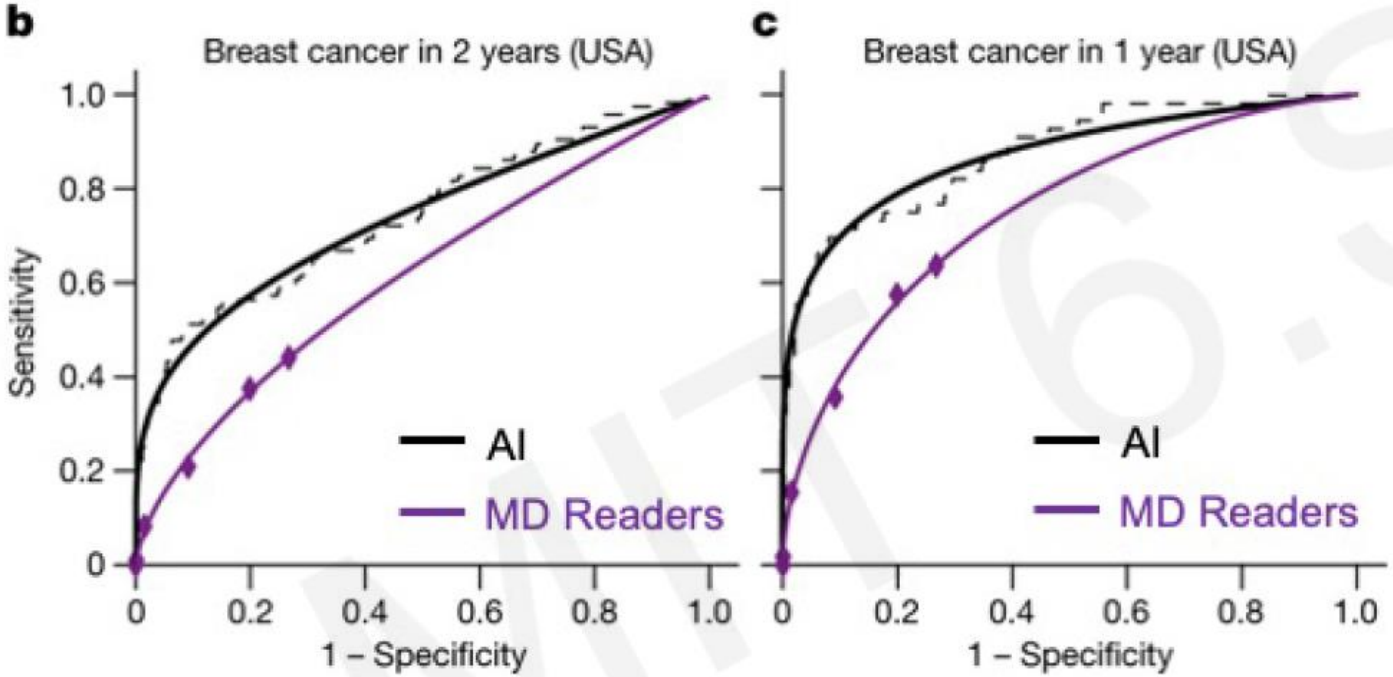
Gurovich et al., *Nature Med.* 2019



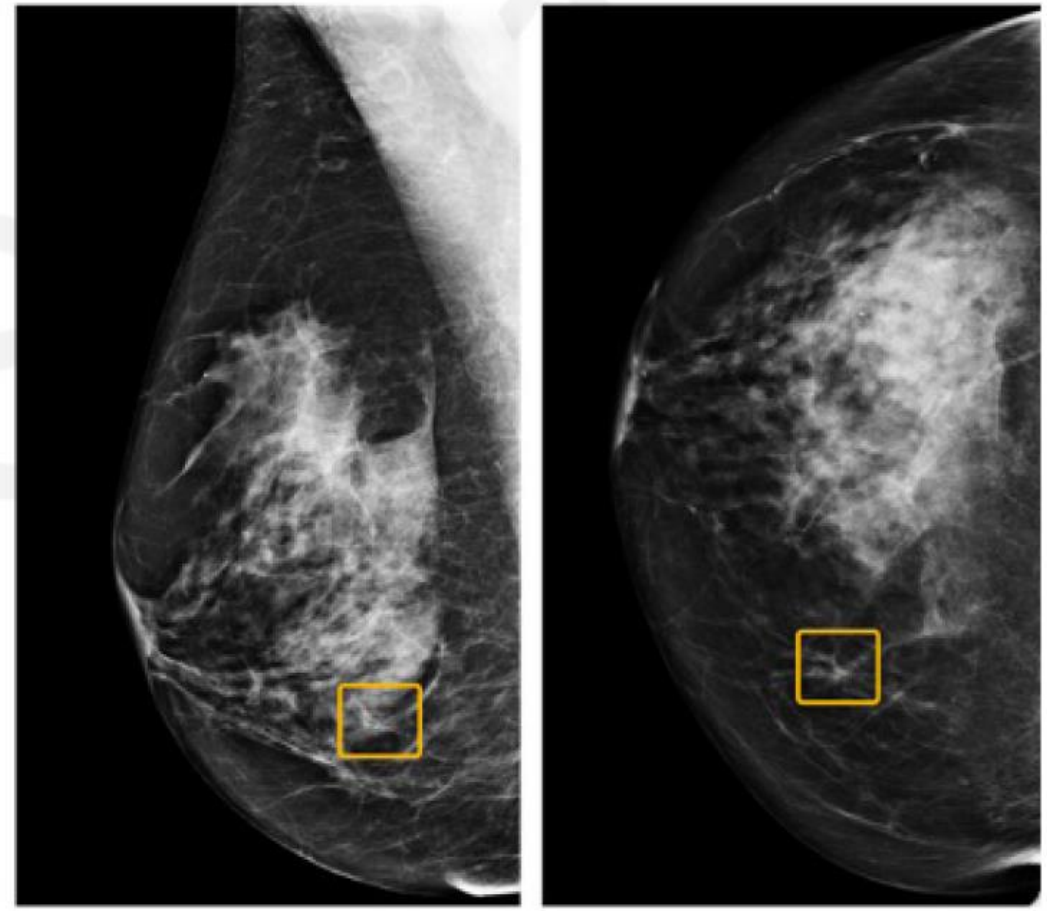
Classification: Breast Cancer Screening

International evaluation of an AI system for breast cancer screening

nature



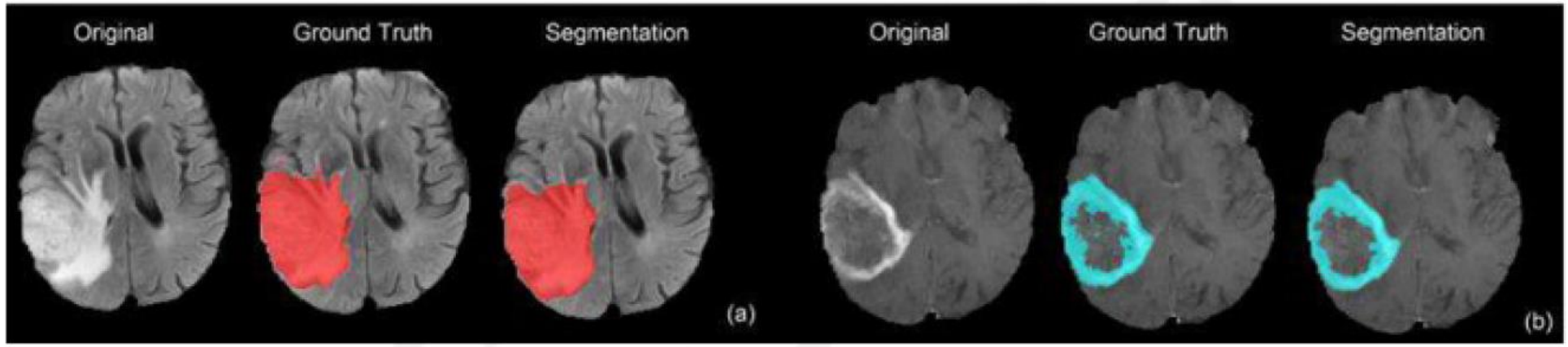
CNN-based system outperformed expert radiologists at detecting breast cancer from mammograms



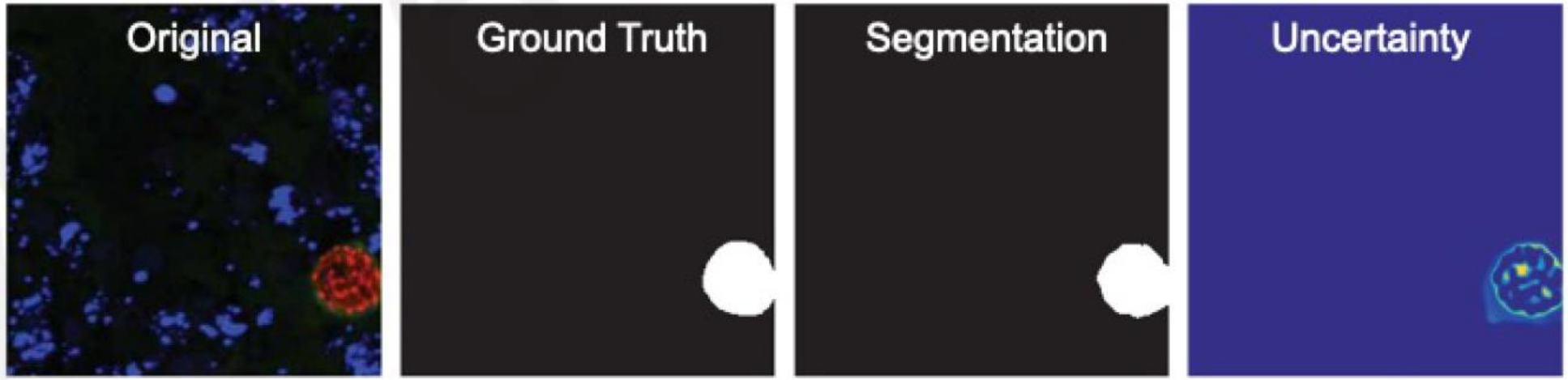
Breast cancer case missed by radiologist but detected by AI

Semantic Segmentation: Biomedical Image Analysis

Brain Tumors
Dong+ *MIUA* 2017.



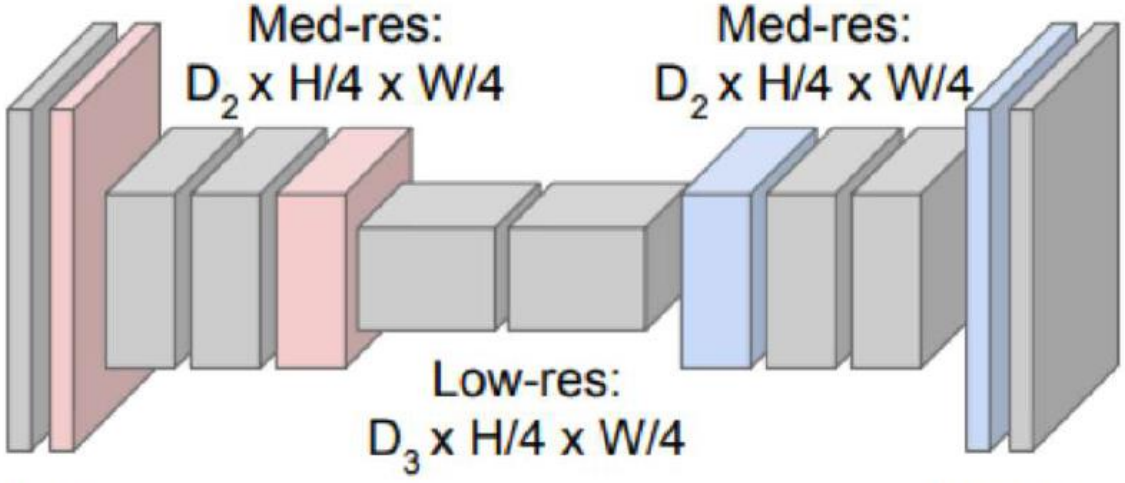
Malaria Infection
Soleimany+ *arXiv* 2019.



Semantic Segmentation: FCNs

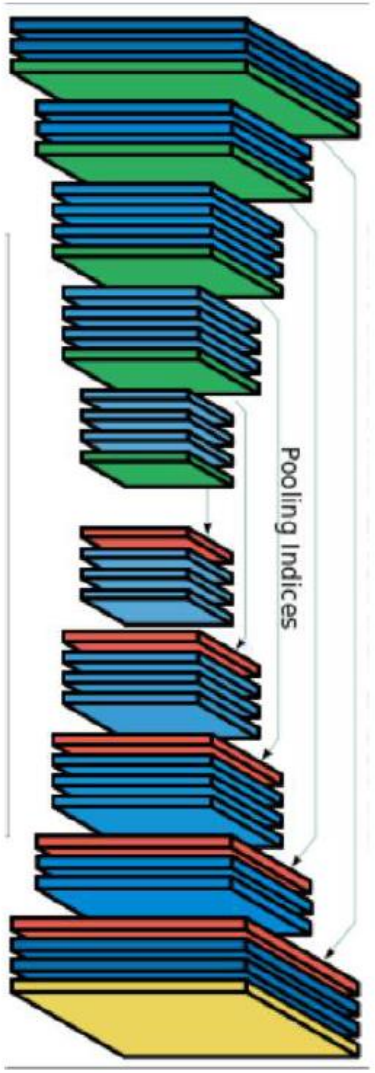
FCN: Fully Convolutional Network.
Network designed with all convolutional layers,
with **downsampling** and **upsampling** operations

Input:
3 x H x W



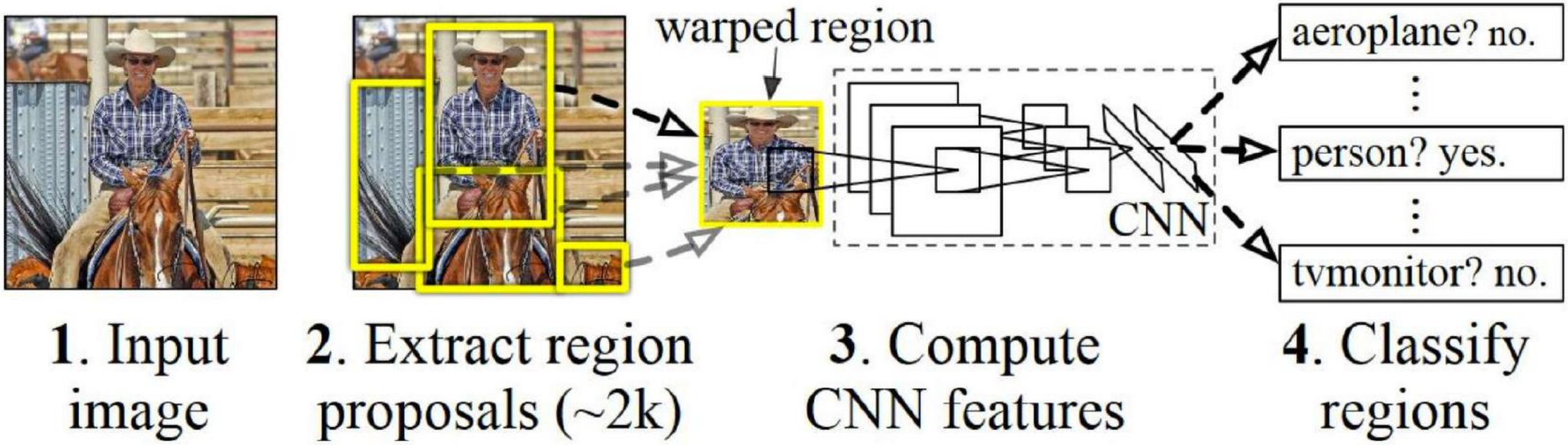
Predictions:
H x W

Driving Scene Segmentation



Object Detection with R-CNNs

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



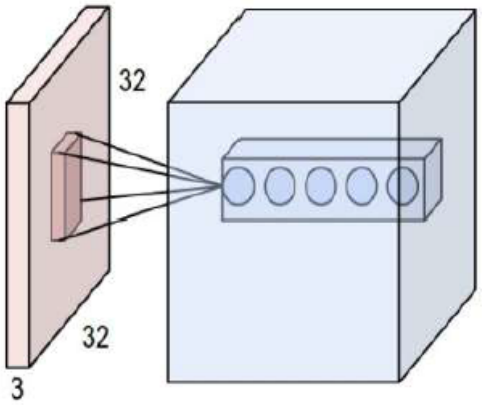
Deep Learning for Computer Vision: Summary

Foundations

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

CNNs

- CNN architecture
- Application to classification
- ImageNet



Applications

- Segmentation, object detection, image captioning
- Visualization