

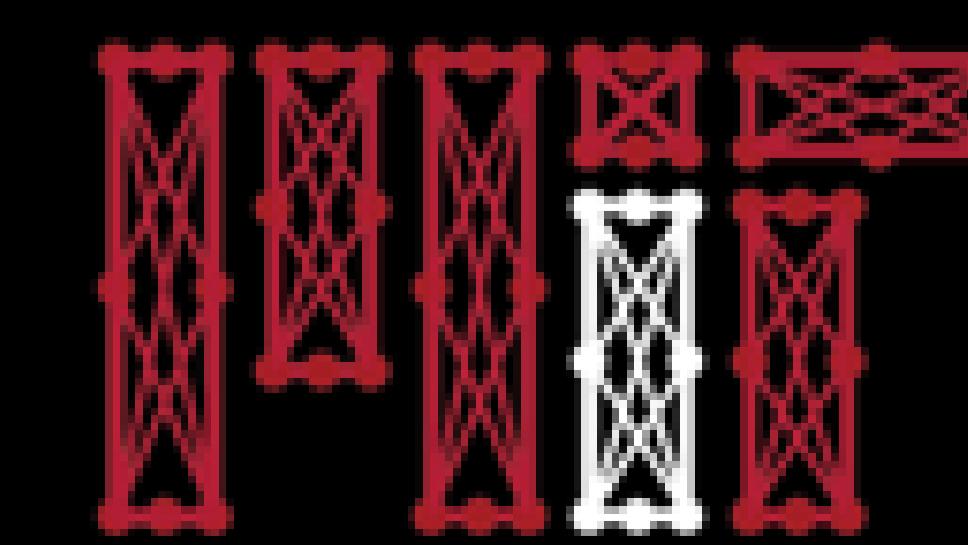


# Deep Computer Vision

Alexander Amini

MIT 6.S191

January 28, 2020



6.S191 Introduction to Deep Learning

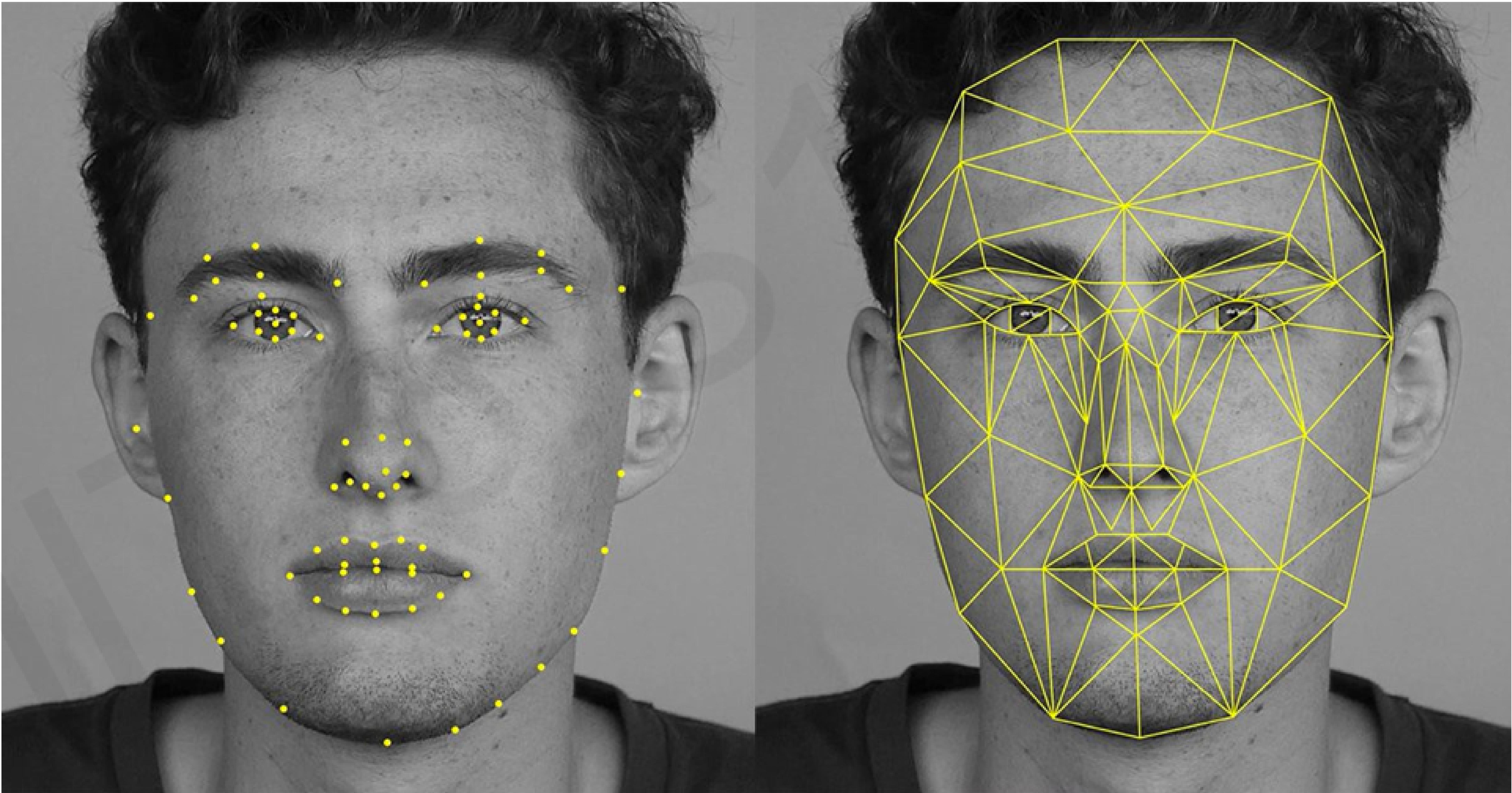
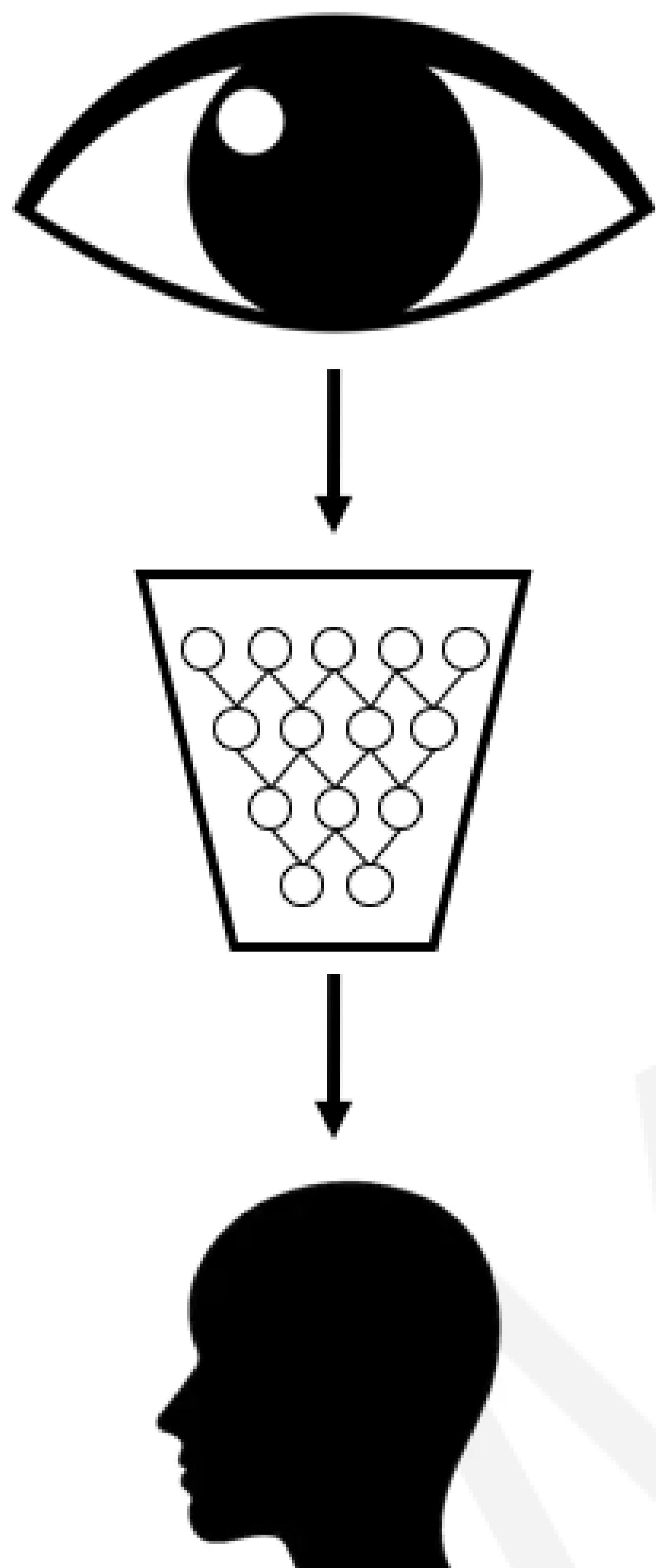
[introtodeeplearning.com](http://introtodeeplearning.com) @MITDeepLearning



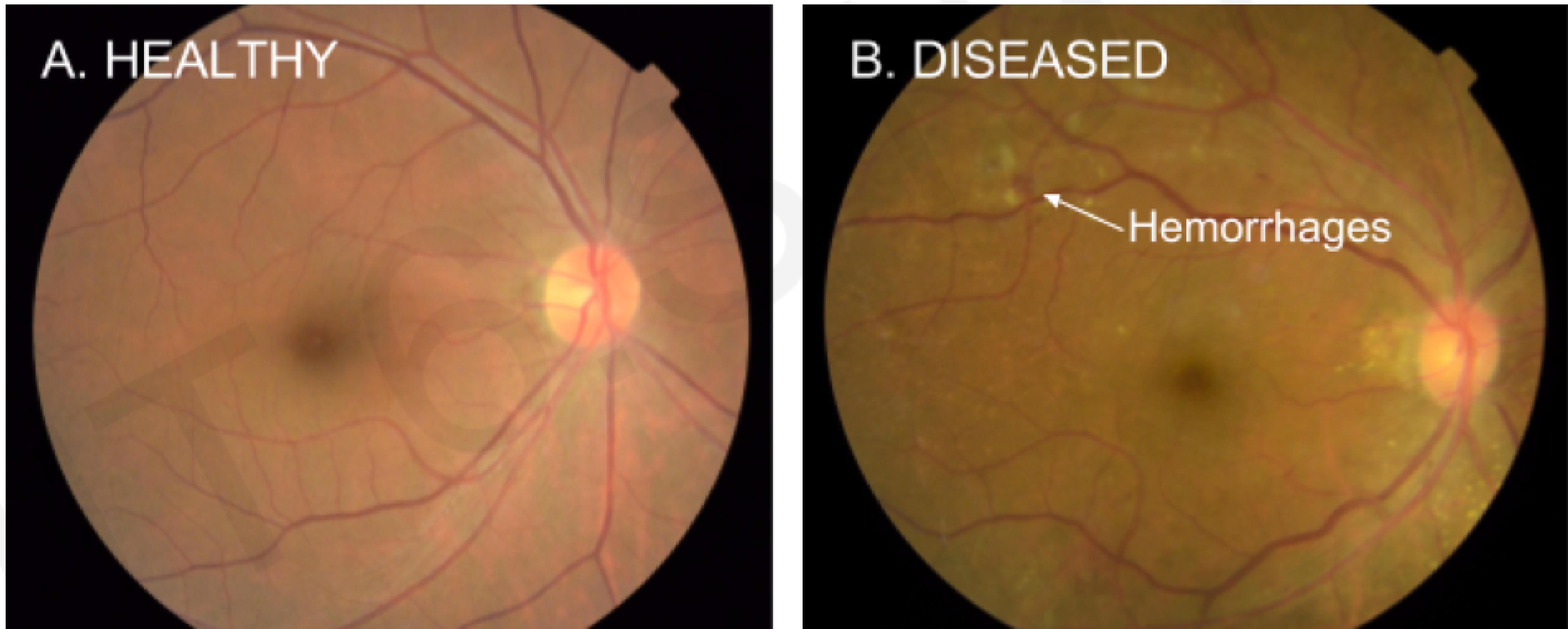
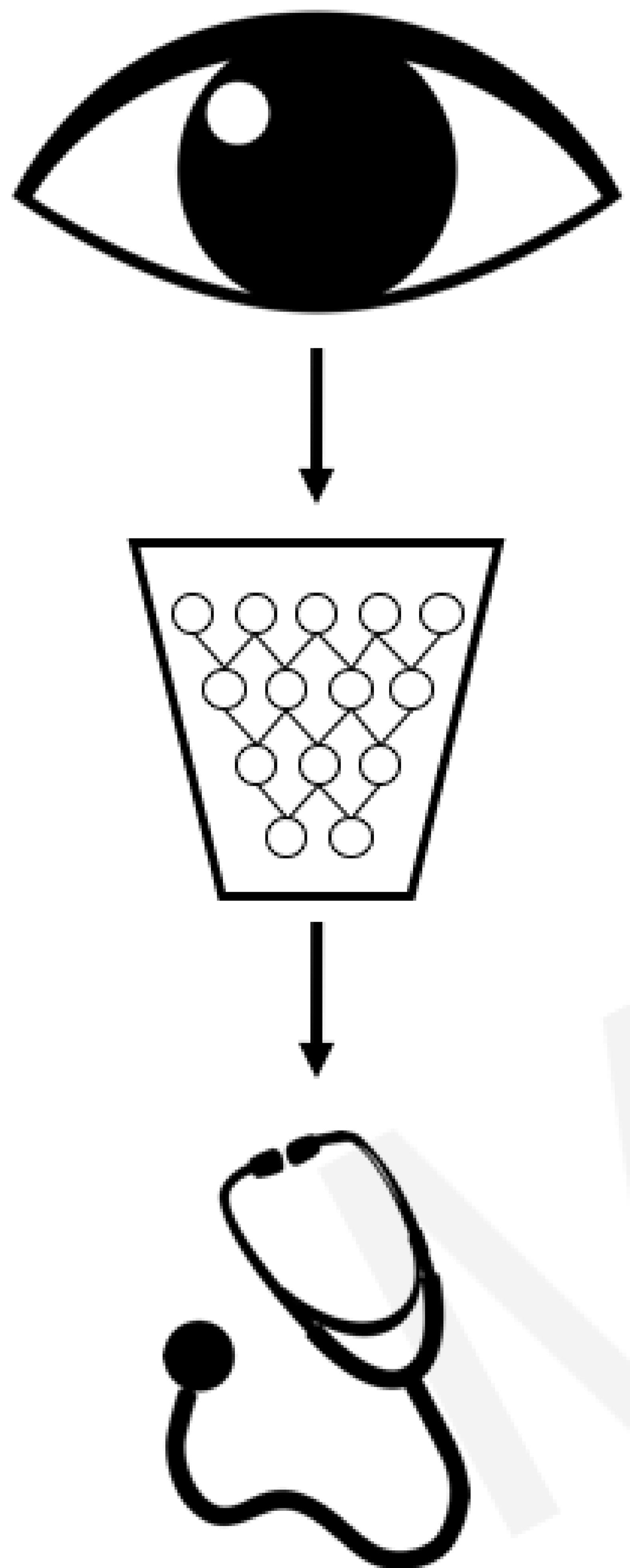
MIT



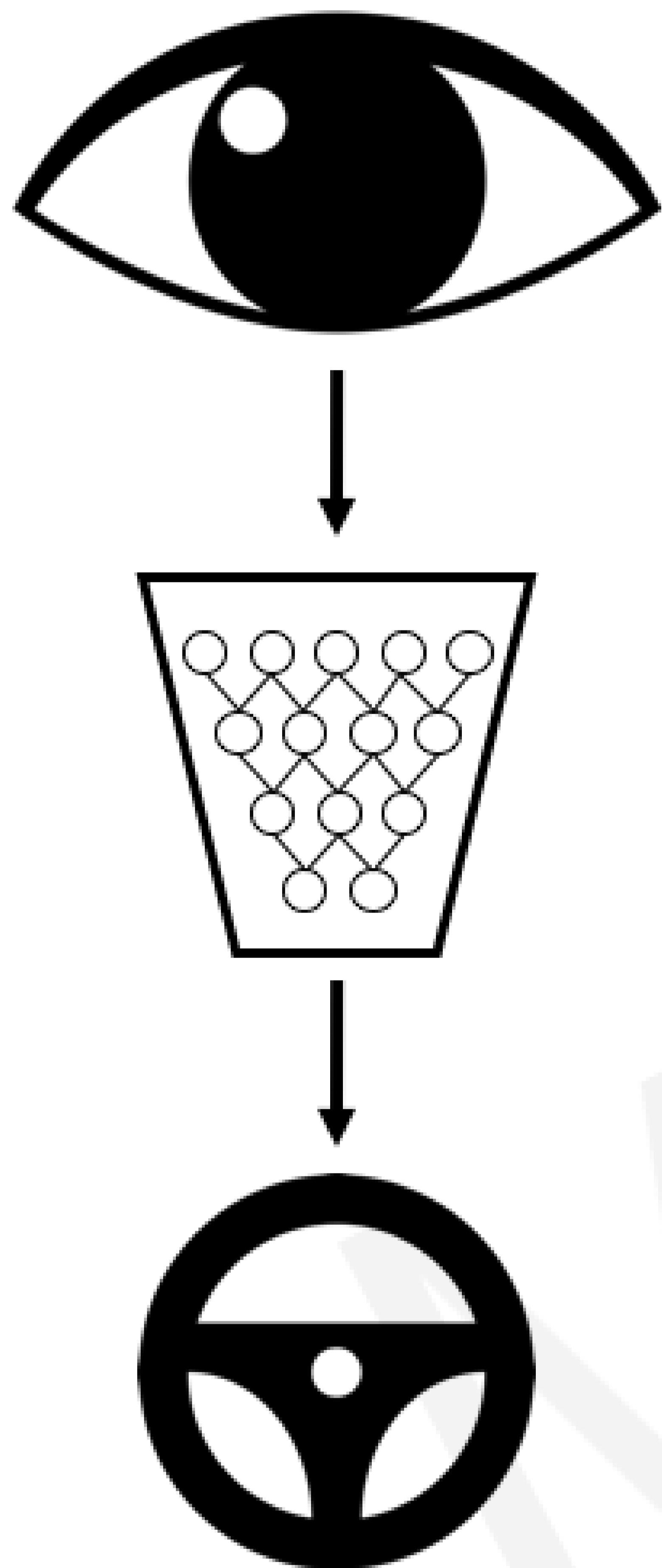
# Impact: Facial Detection & Recognition



# Impact: Medicine, Biology, Healthcare

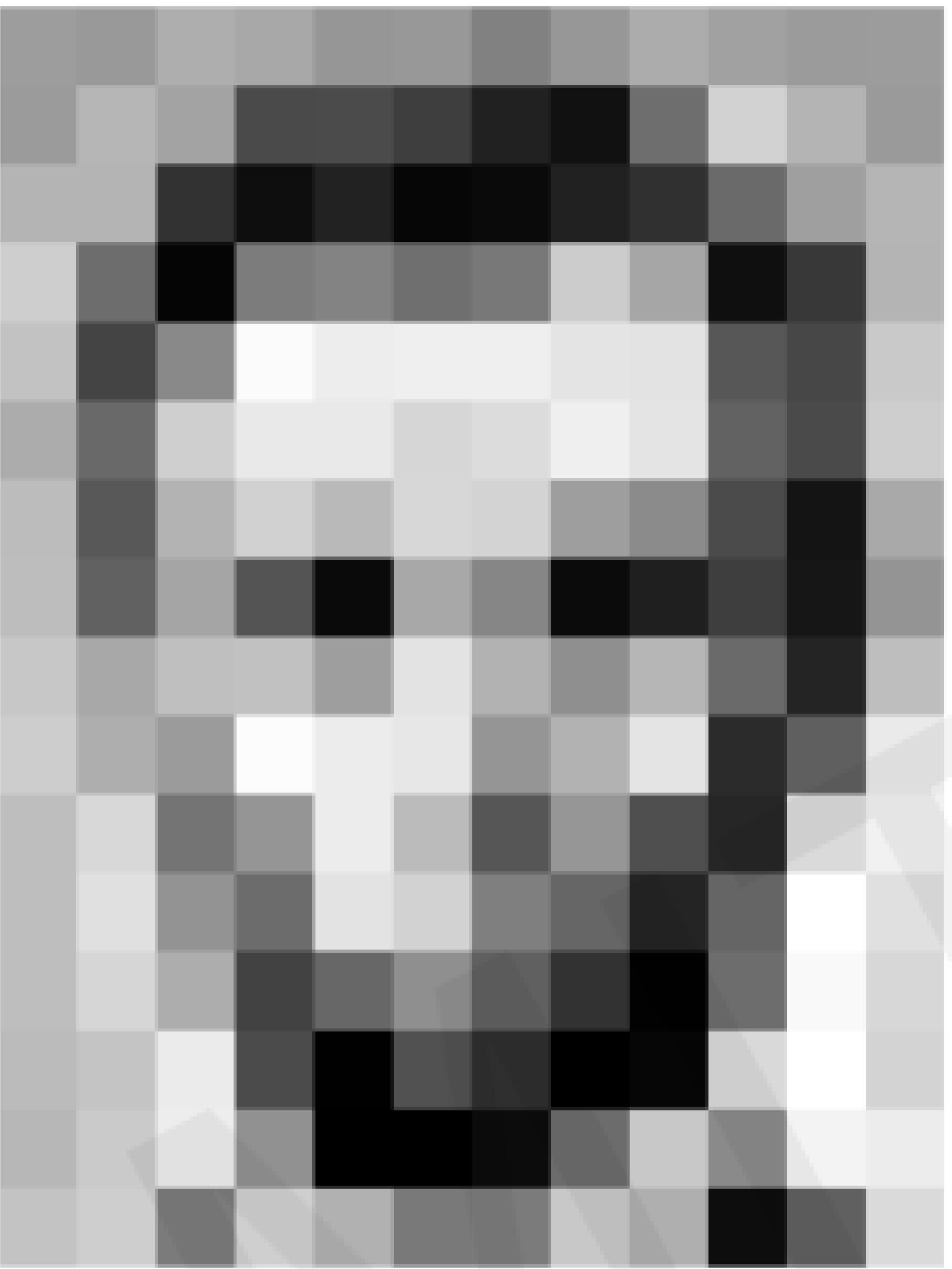


# Impact: Self-Driving Cars



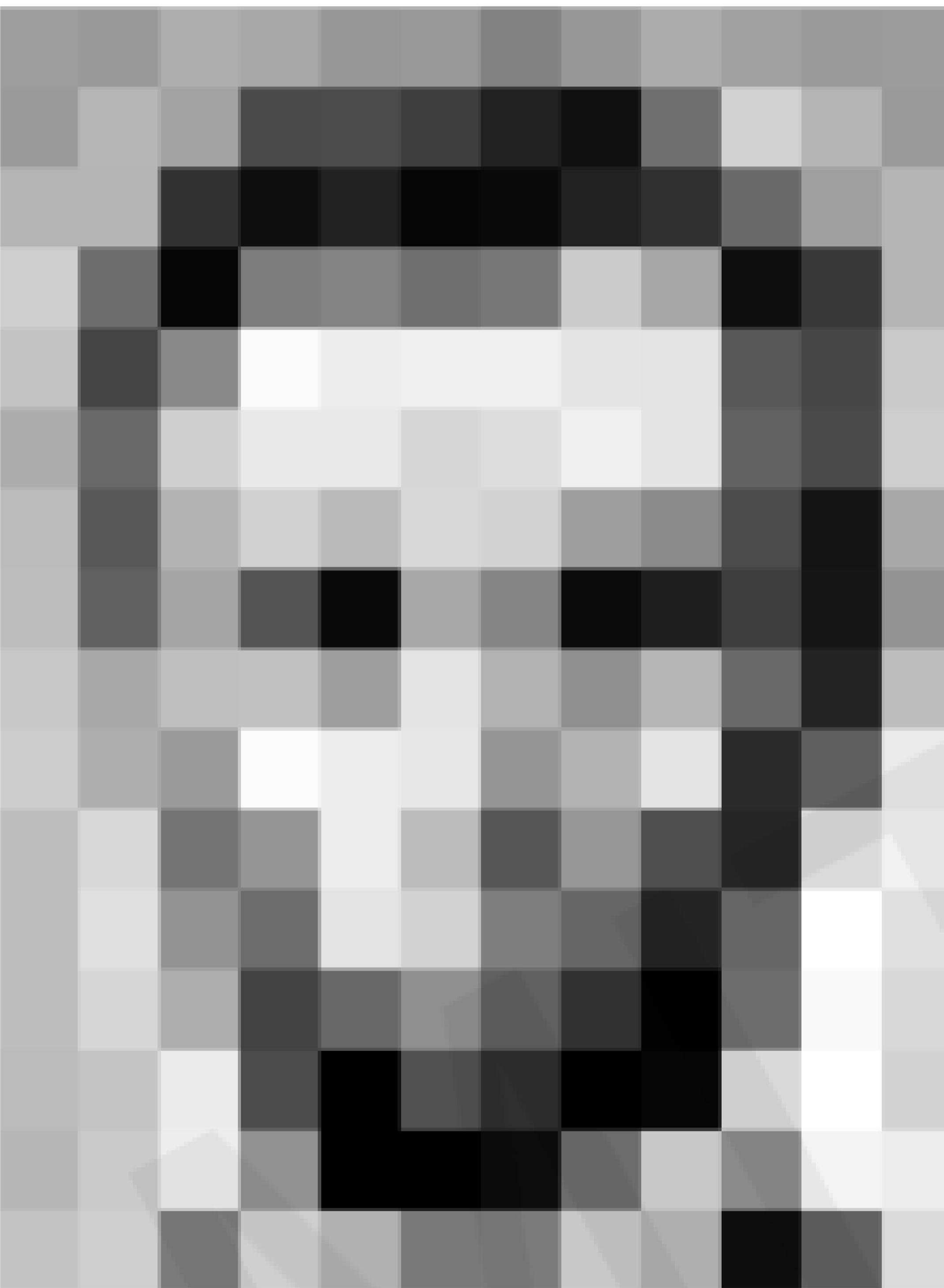
# What Computers “See”

# Images are Numbers



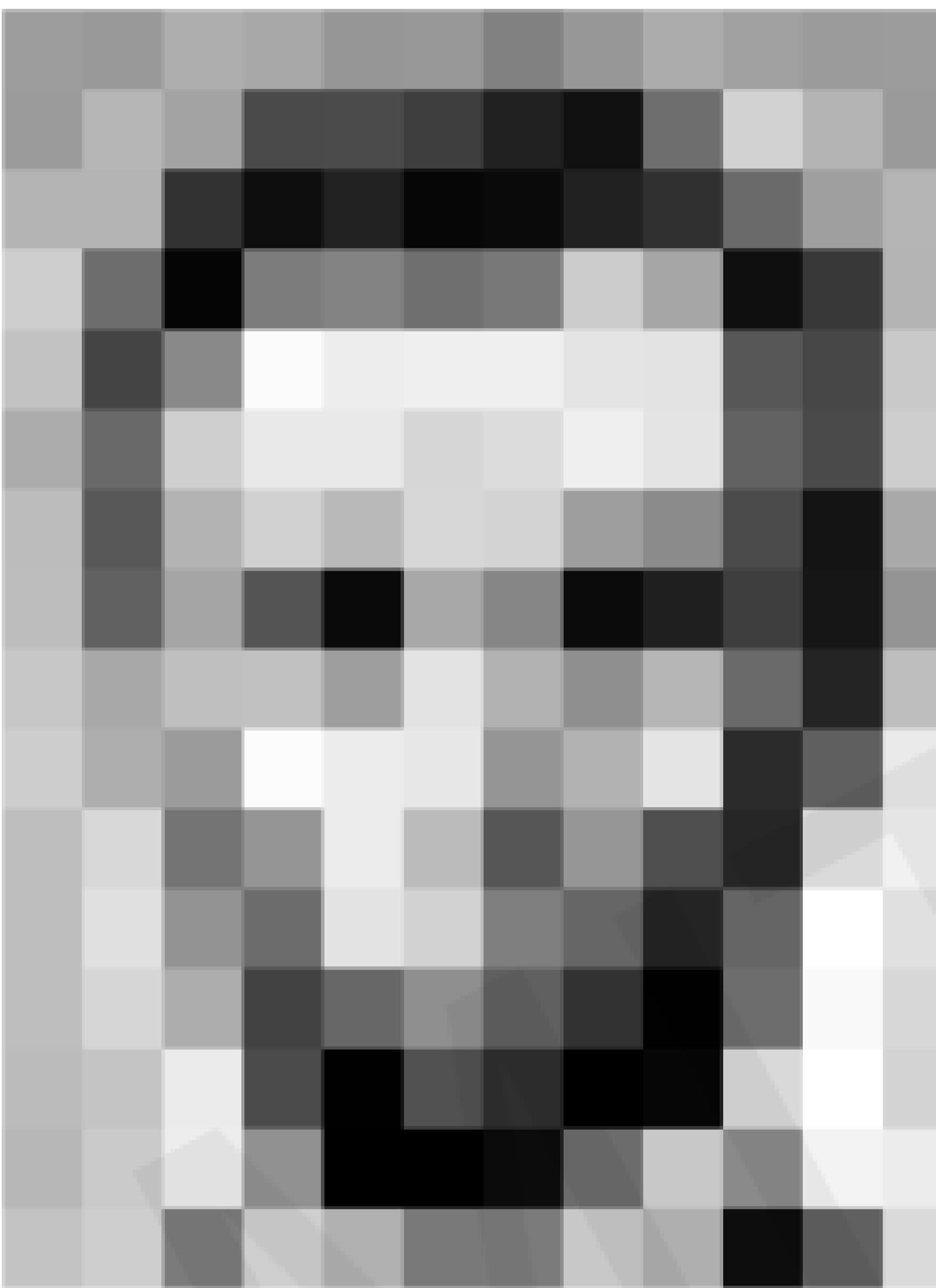
6.S191

# Images are Numbers



157	153	174	168	150	152	129	151	172	161	155	166
155	162	169	74	75	62	83	17	110	210	180	164
180	180	50	14	84	6	10	33	45	106	159	181
206	169	5	124	191	111	120	204	166	15	56	180
194	63	197	251	257	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	63	179	209	185	215	211	198	199	75	20	169
189	97	165	24	10	168	154	11	31	62	22	148
190	168	161	163	158	227	178	143	182	106	36	190
206	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	36	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	55	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
180	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	209	175	13	96	218

# Images are Numbers



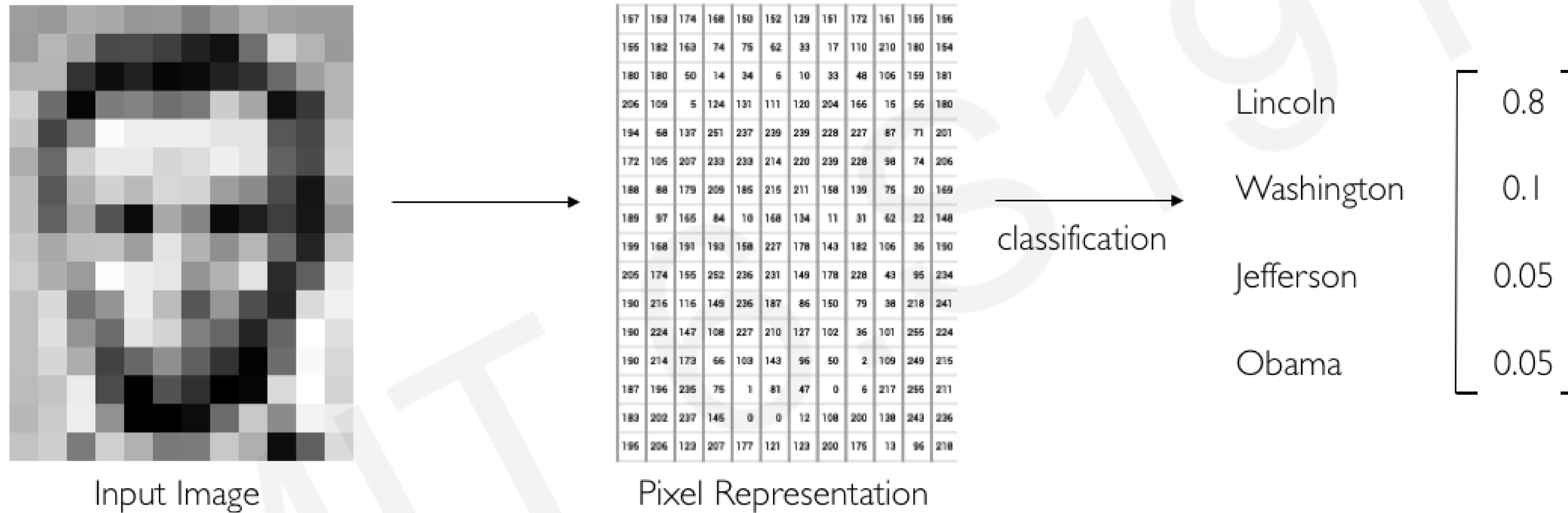
157	153	174	168	150	152	129	151	172	161	155	166
155	162	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	45	106	159	181
206	169	5	124	131	111	120	204	166	15	56	180
194	68	137	251	257	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	68	179	209	185	215	211	198	199	75	20	169
189	97	165	24	10	168	134	11	31	62	22	148
190	168	191	163	158	227	178	143	182	106	36	190
206	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	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
180	202	237	145	0	0	12	108	200	138	243	236
196	206	123	207	177	121	123	209	175	13	96	218

What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
156	162	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	169	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	68	179	209	185	215	211	198	199	75	20	169
189	97	165	24	10	168	134	11	31	62	22	148
190	168	191	163	158	227	178	143	182	106	36	190
206	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	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
180	202	237	145	0	0	12	108	200	138	243	236
196	206	123	207	177	121	123	209	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



- **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

Domain knowledge

Define features

Detect features  
to classify

Problems?

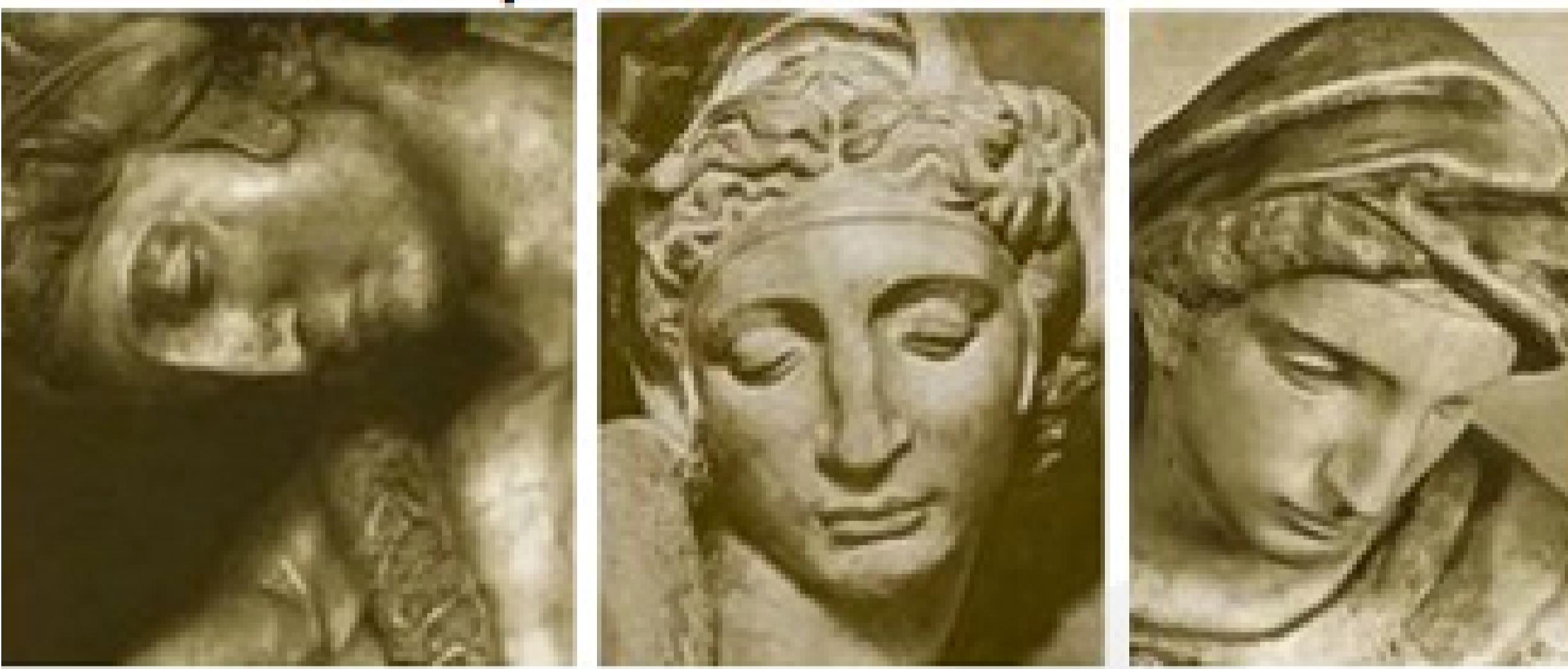
# Manual Feature Extraction

Domain knowledge

Define features

Detect features  
to classify

Viewpoint variation



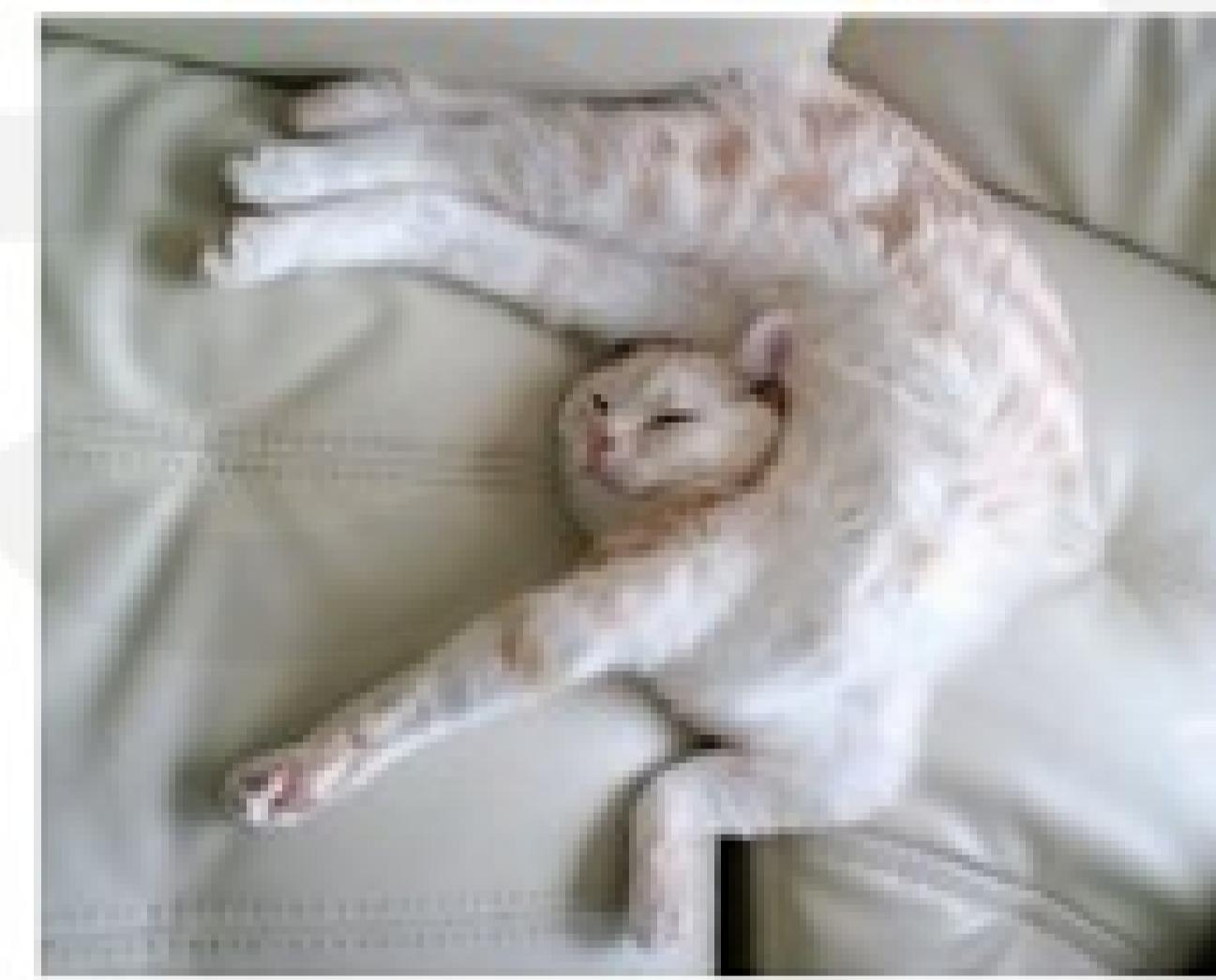
Illumination conditions



Scale variation



Deformation



Background clutter



Occlusion



Intra-class variation



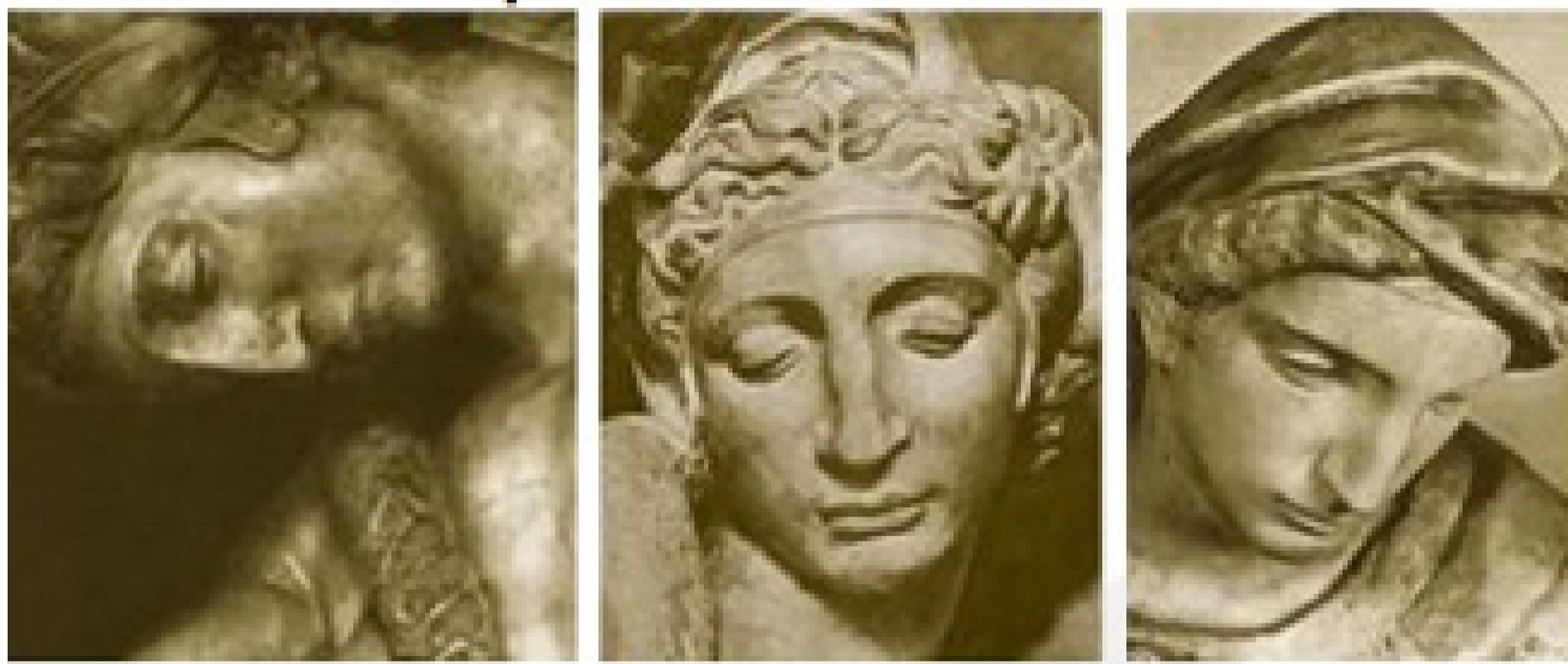
# Manual Feature Extraction

Domain knowledge

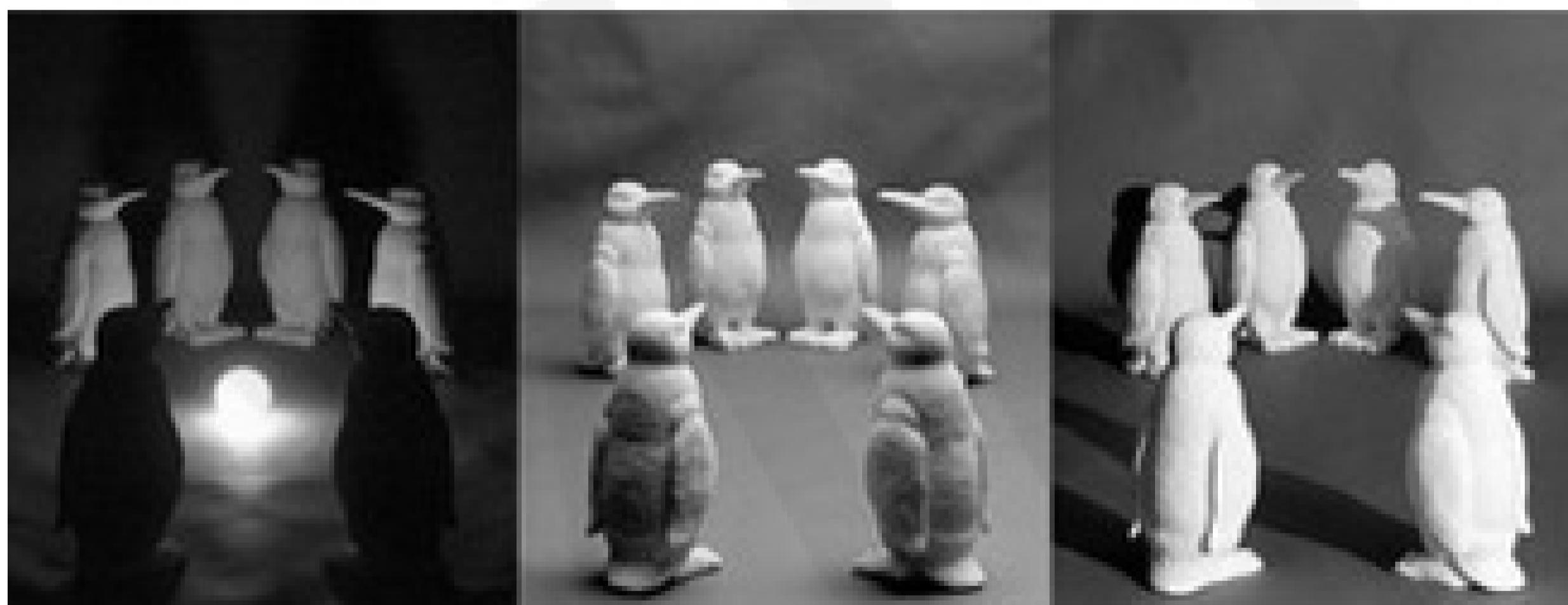
Define features

Detect features  
to classify

Viewpoint variation



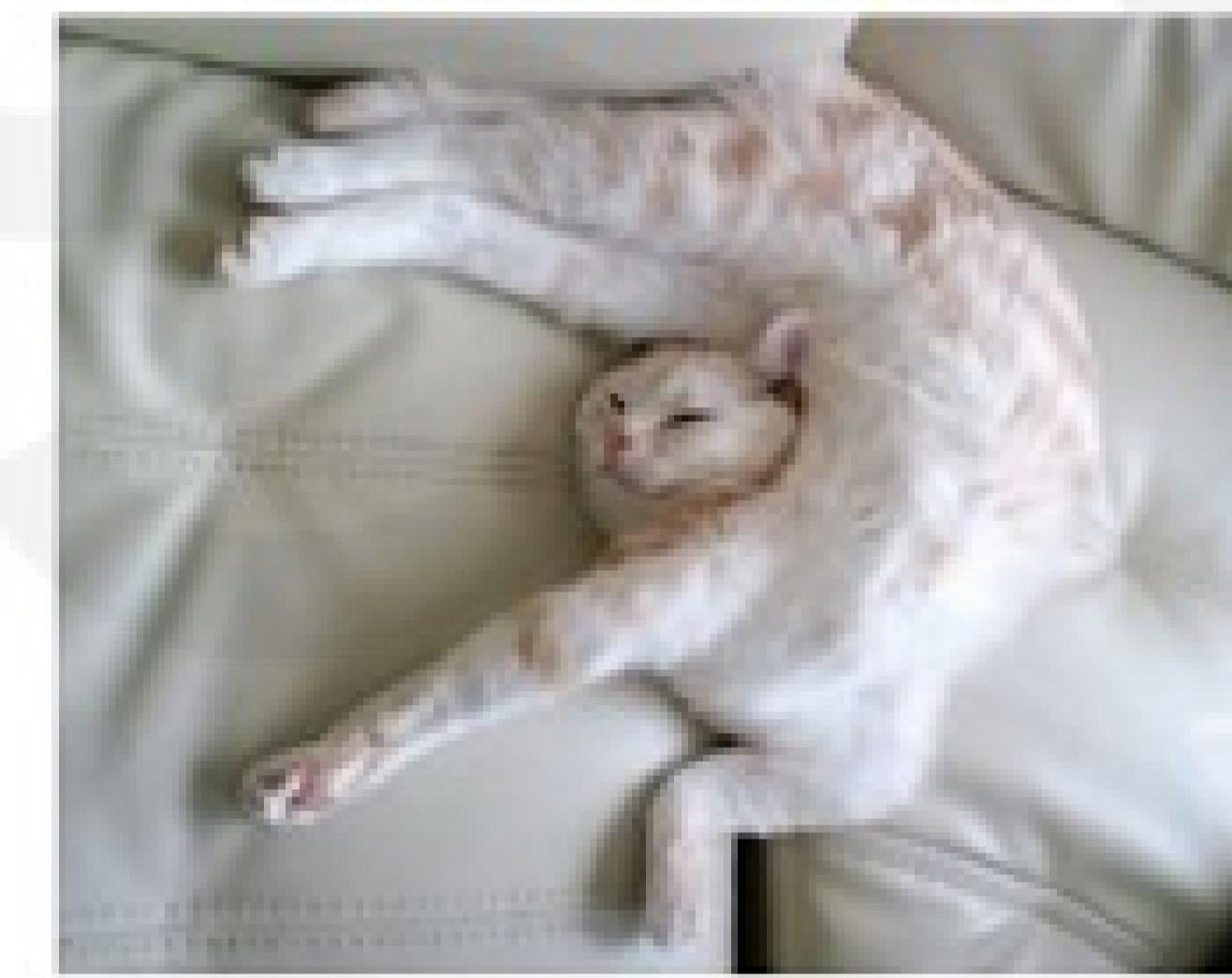
Illumination conditions



Scale variation



Deformation



Background clutter



Occlusion



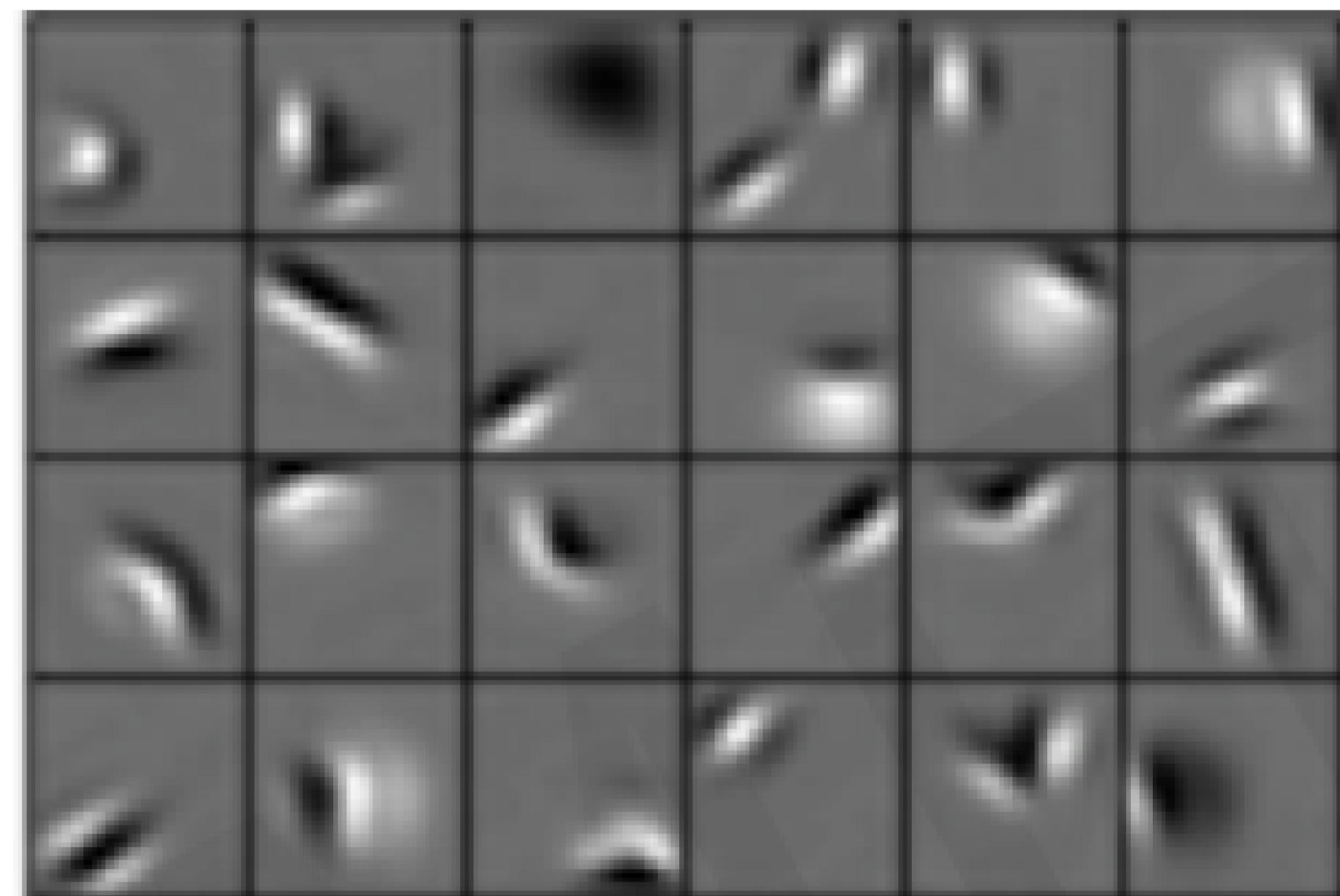
Intra-class variation



# Learning Feature Representations

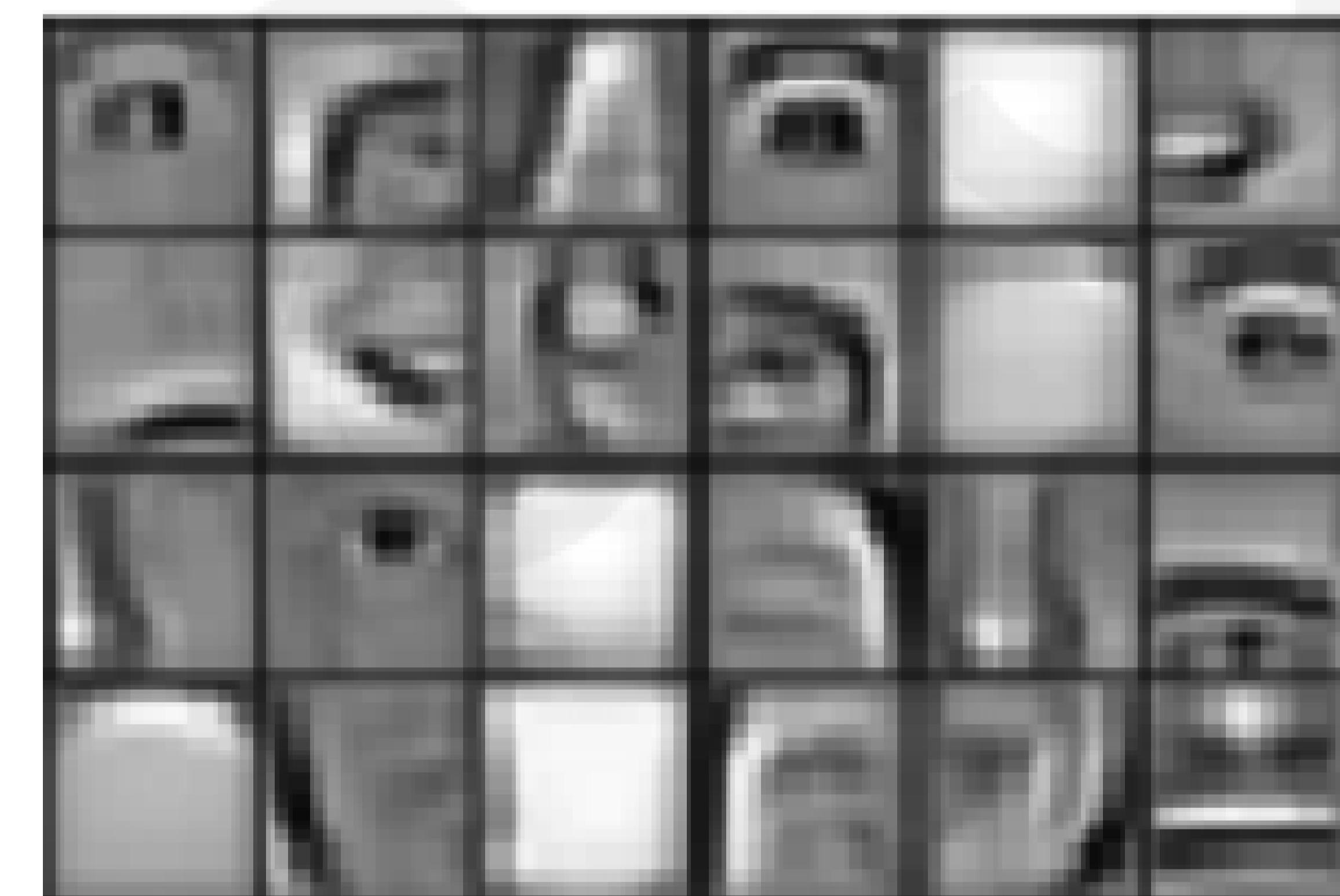
Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



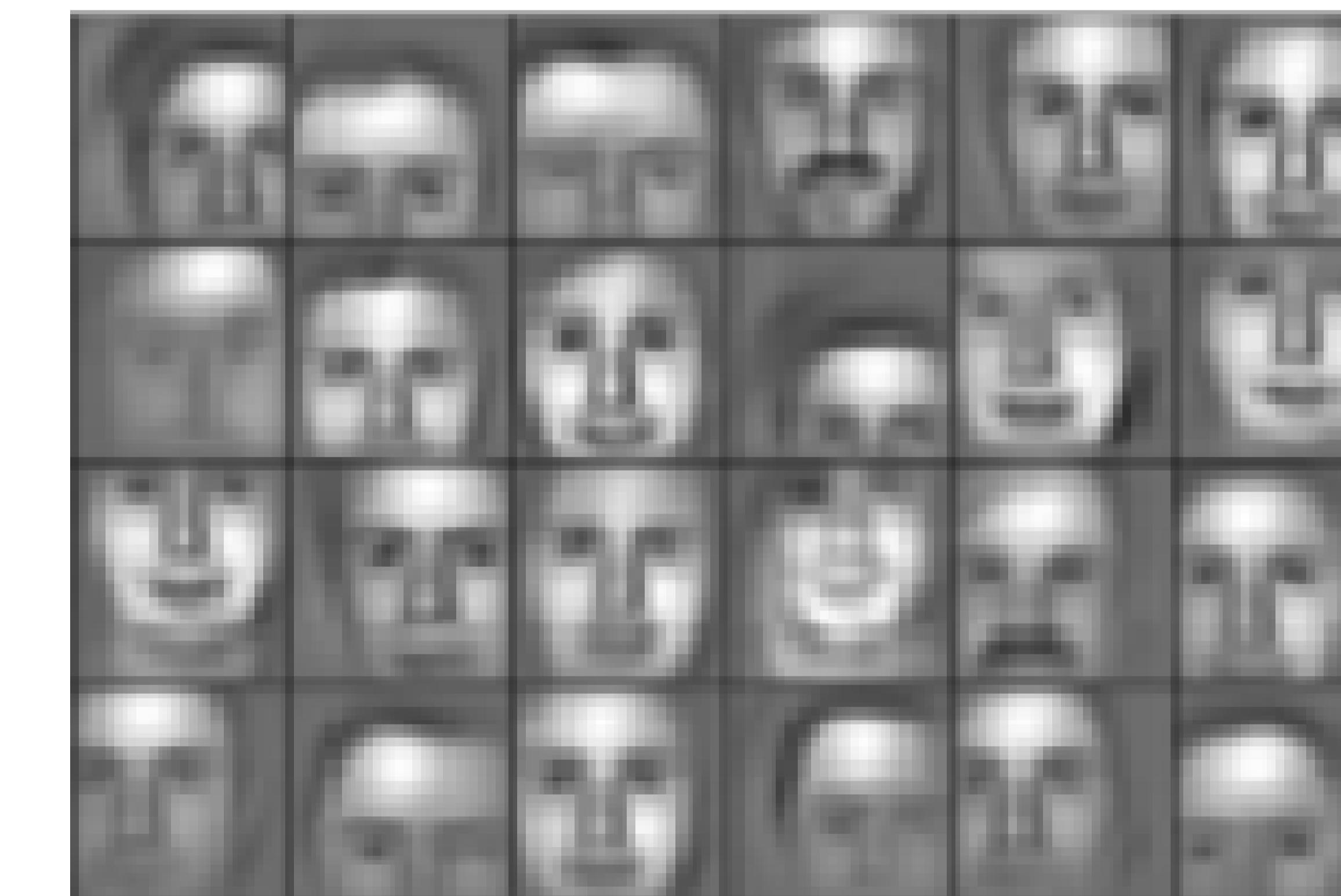
Edges, dark spots

Mid level features



Eyes, ears, nose

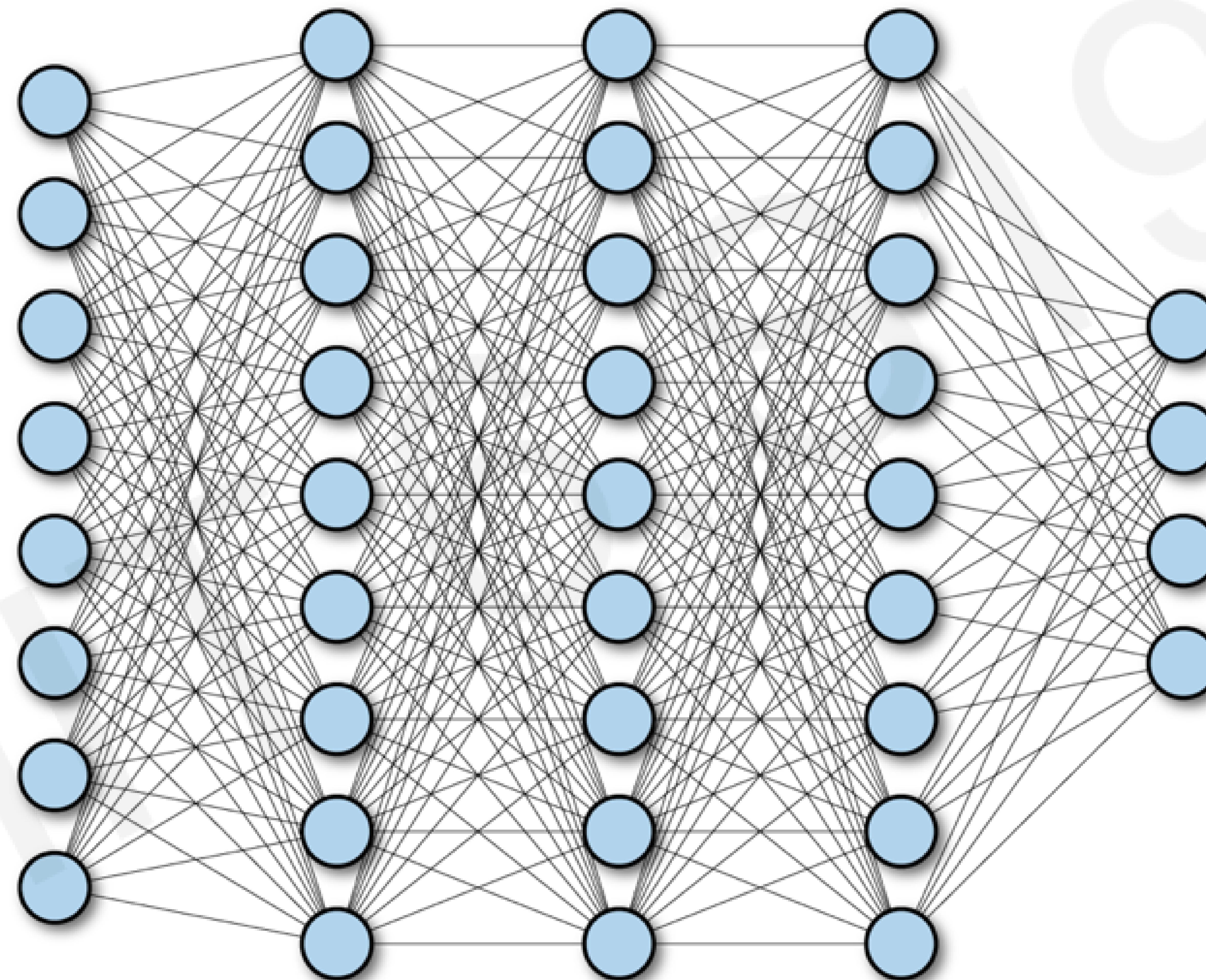
High level features



Facial structure

# Learning Visual Features

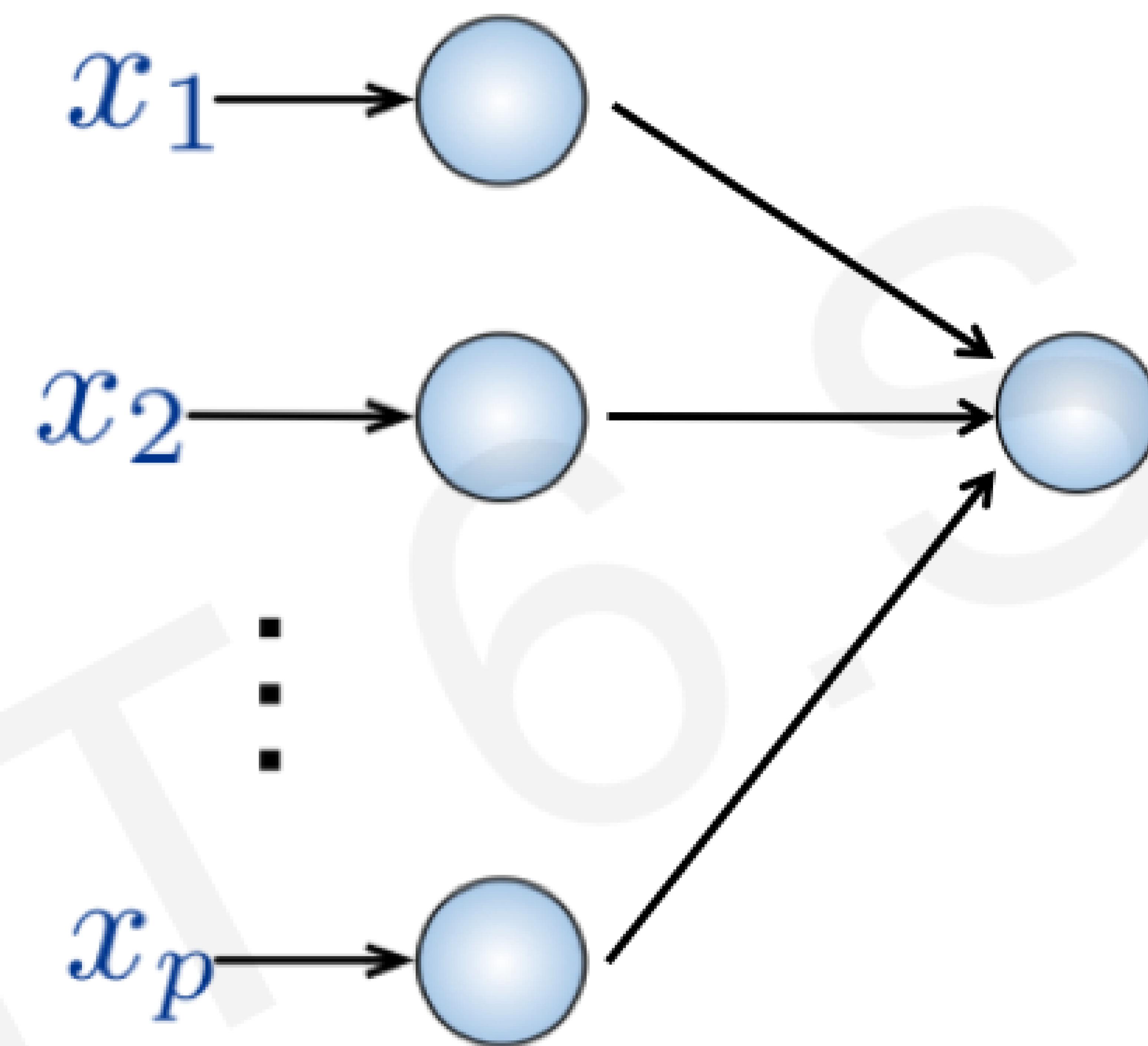
# Fully Connected Neural Network



# Fully Connected Neural Network

## Input:

- 2D image
- Vector of pixel values



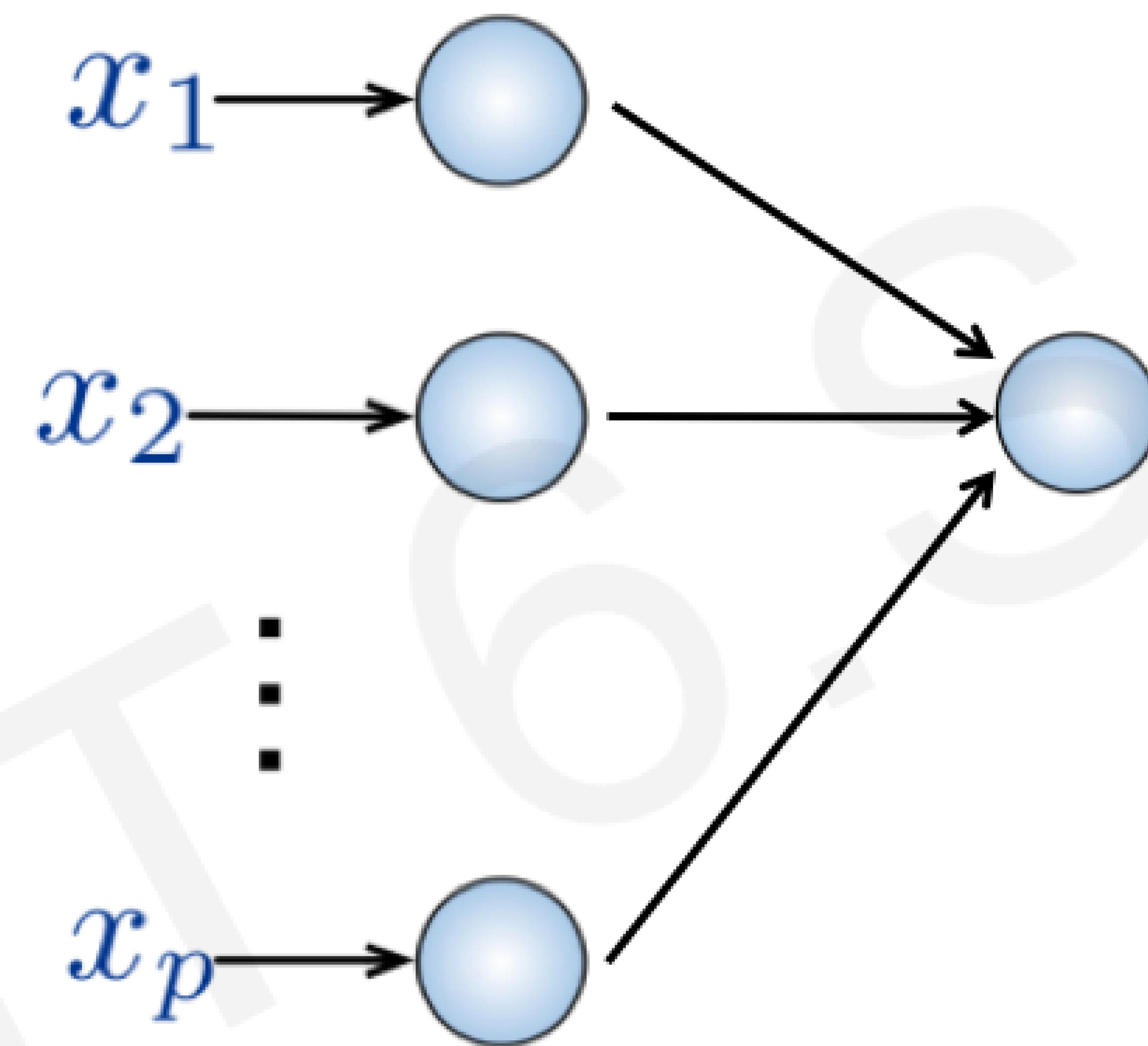
## Fully Connected:

- 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



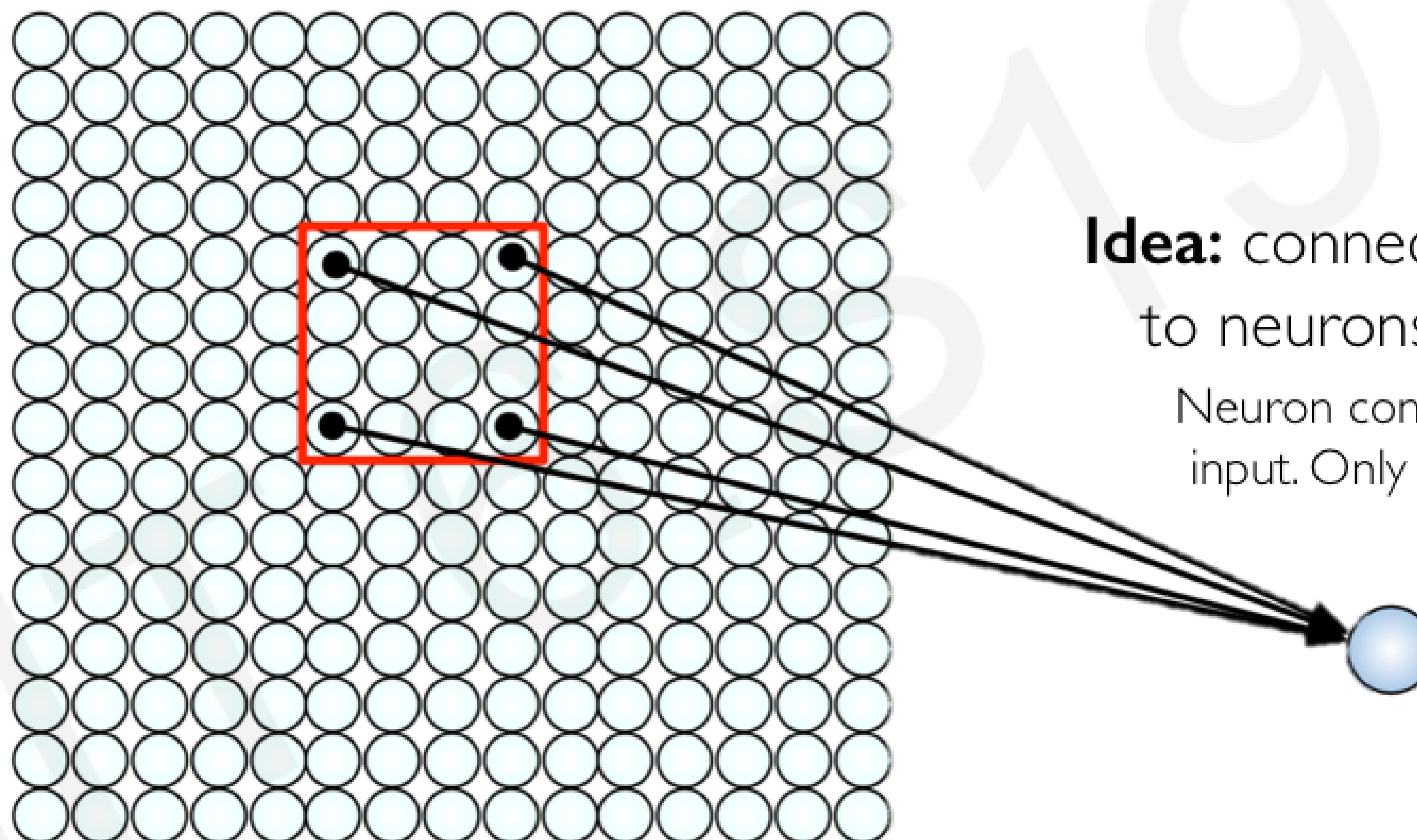
## Fully Connected:

- 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?

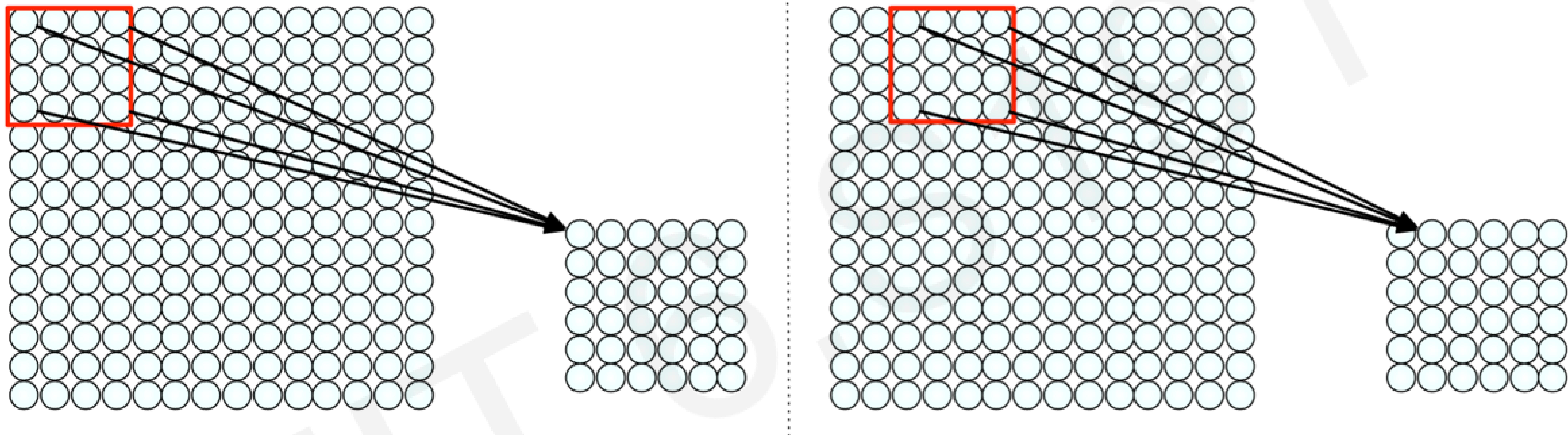
# Using Spatial Structure

**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

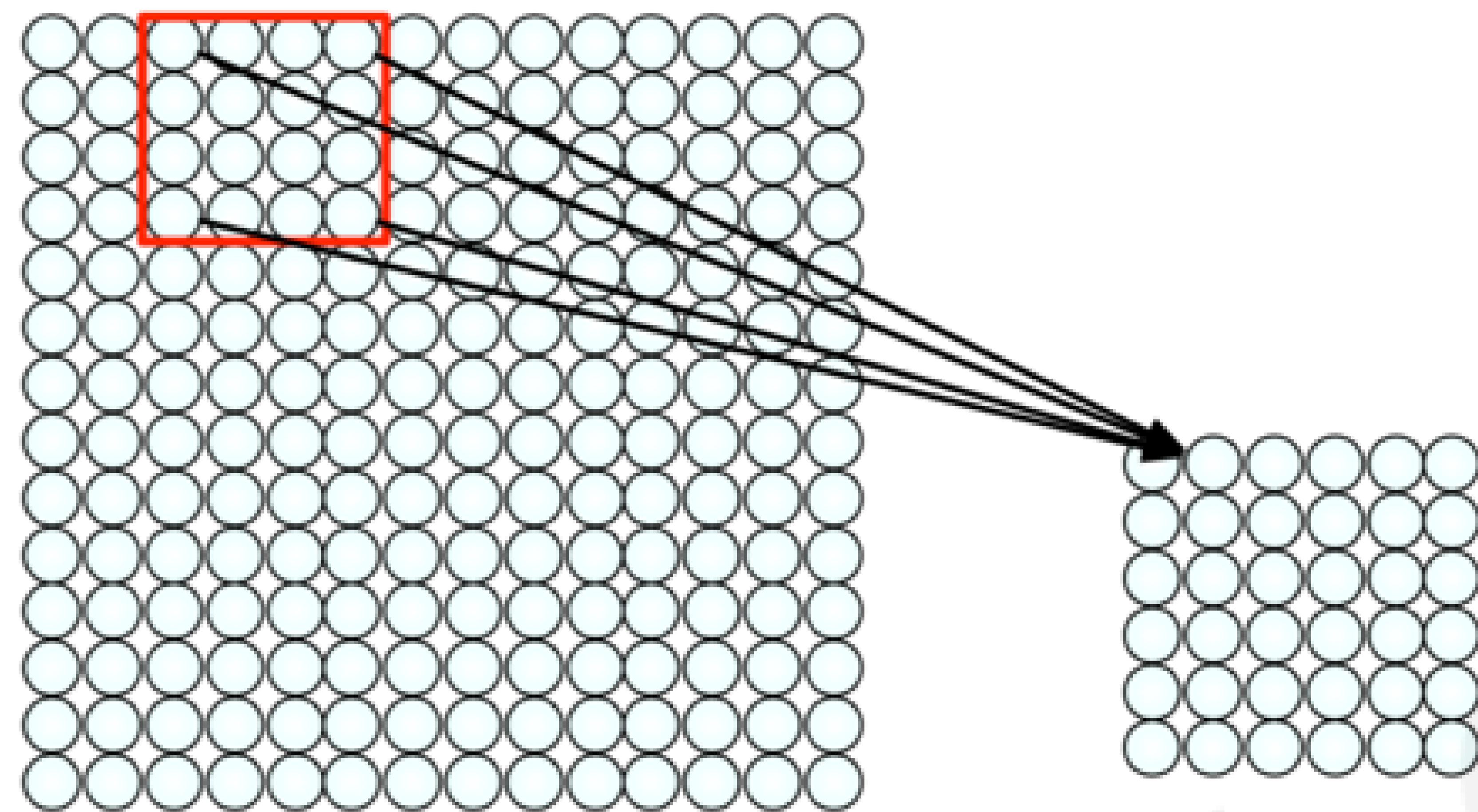


Connect patch in input layer to a single neuron in subsequent layer.  
Use a sliding window to define connections.  
How can we **weight** the patch to detect particular features?

# Applying Filters to Extract Features

- 1) Apply a set of weights – a filter – to extract **local features**
- 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  $4 \times 4$  : 16 different weights
- Apply this same filter to  $4 \times 4$  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?

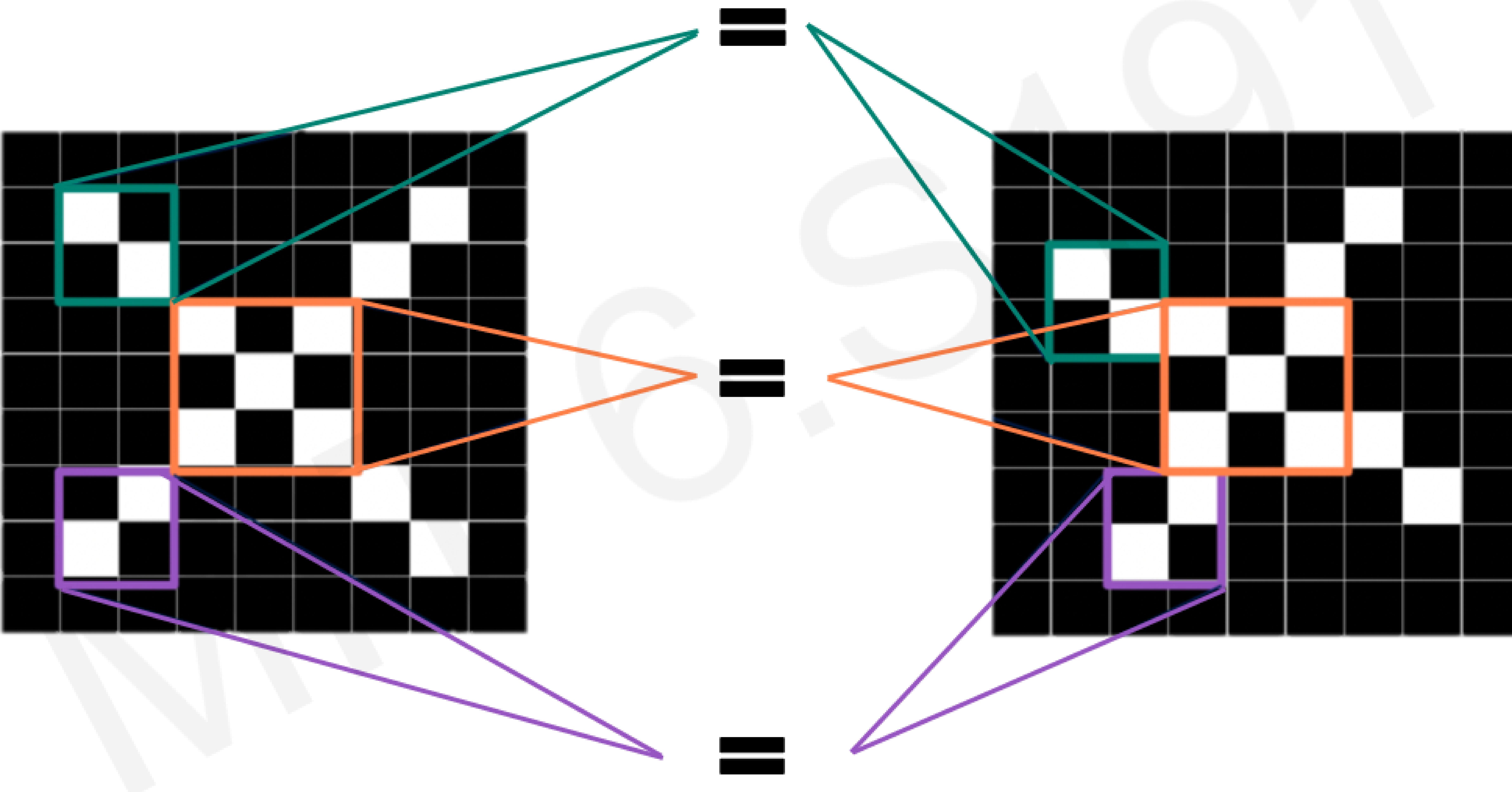


-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-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
-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



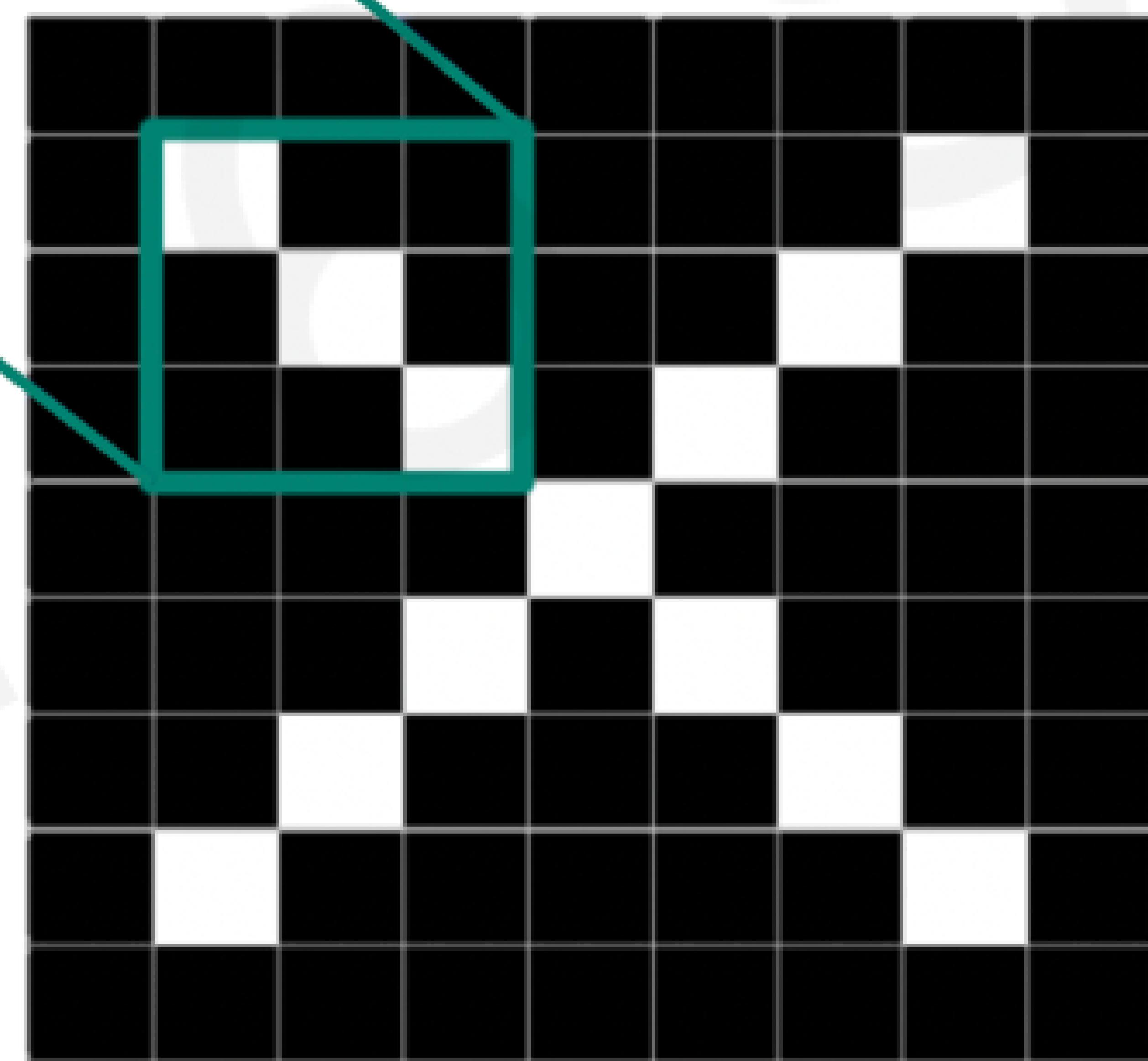
# Filters to Detect X Features

filters

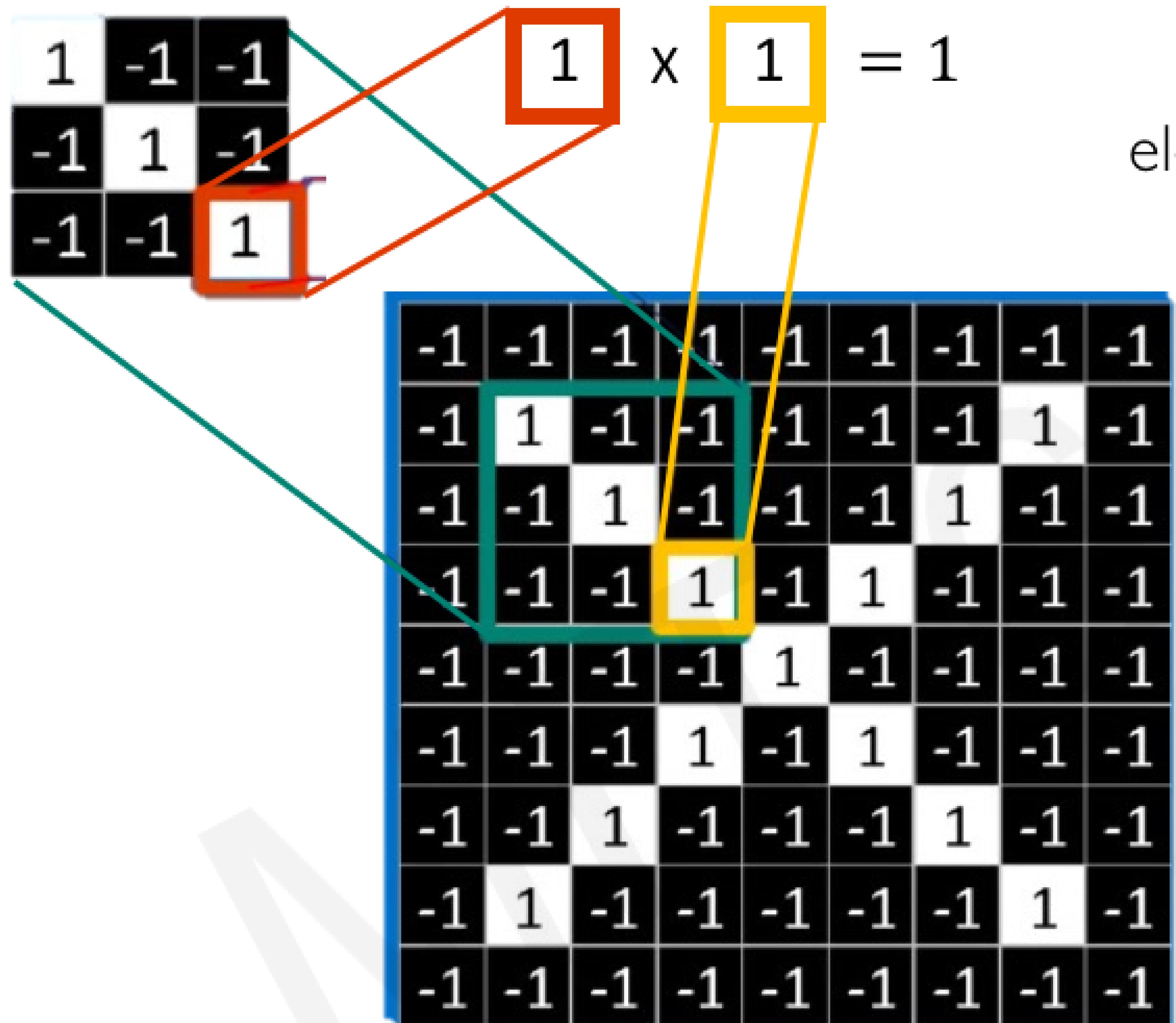
$$\begin{matrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{matrix}$$

$$\begin{matrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{matrix}$$

$$\begin{matrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{matrix}$$



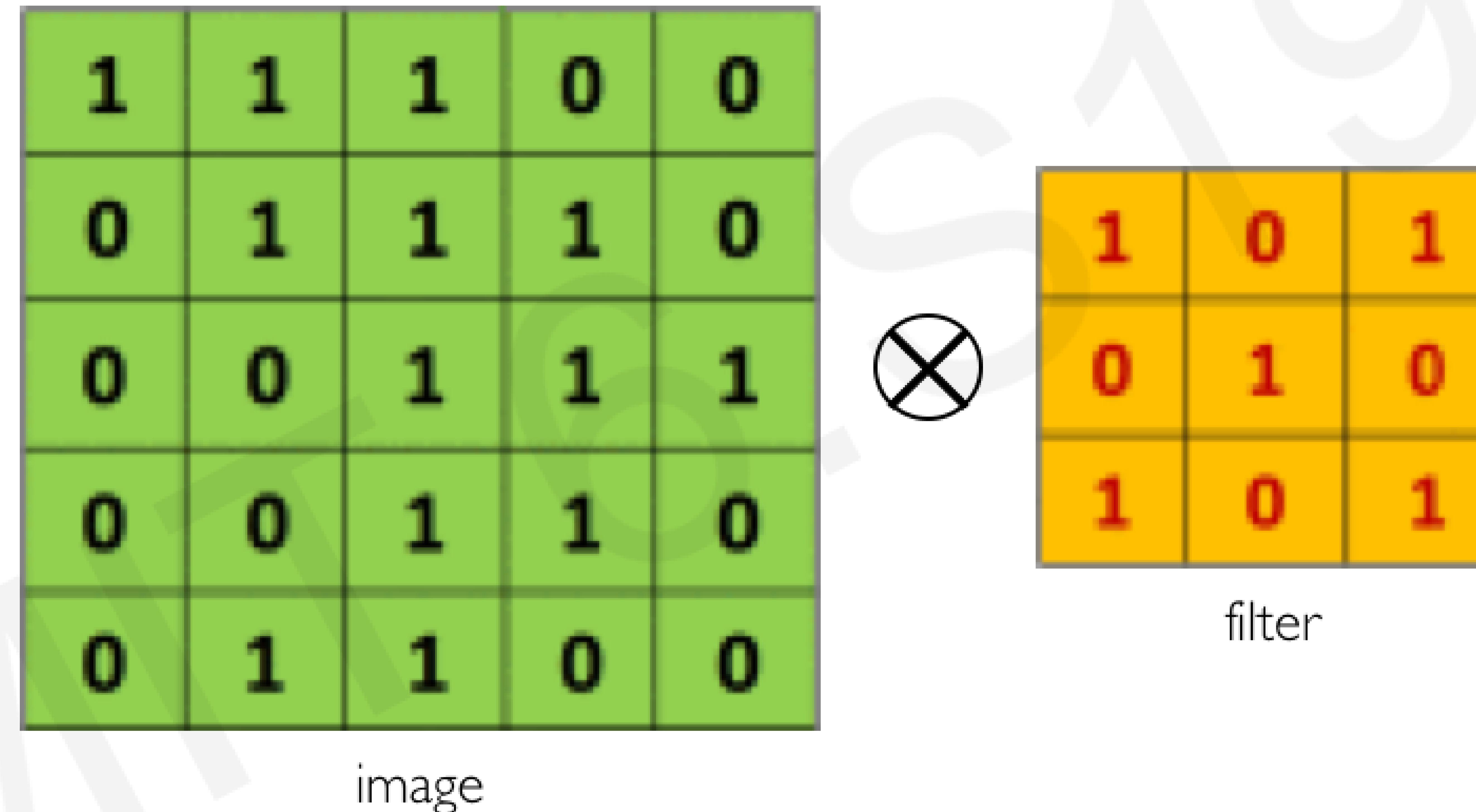
# The Convolution Operation



$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

# The Convolution Operation

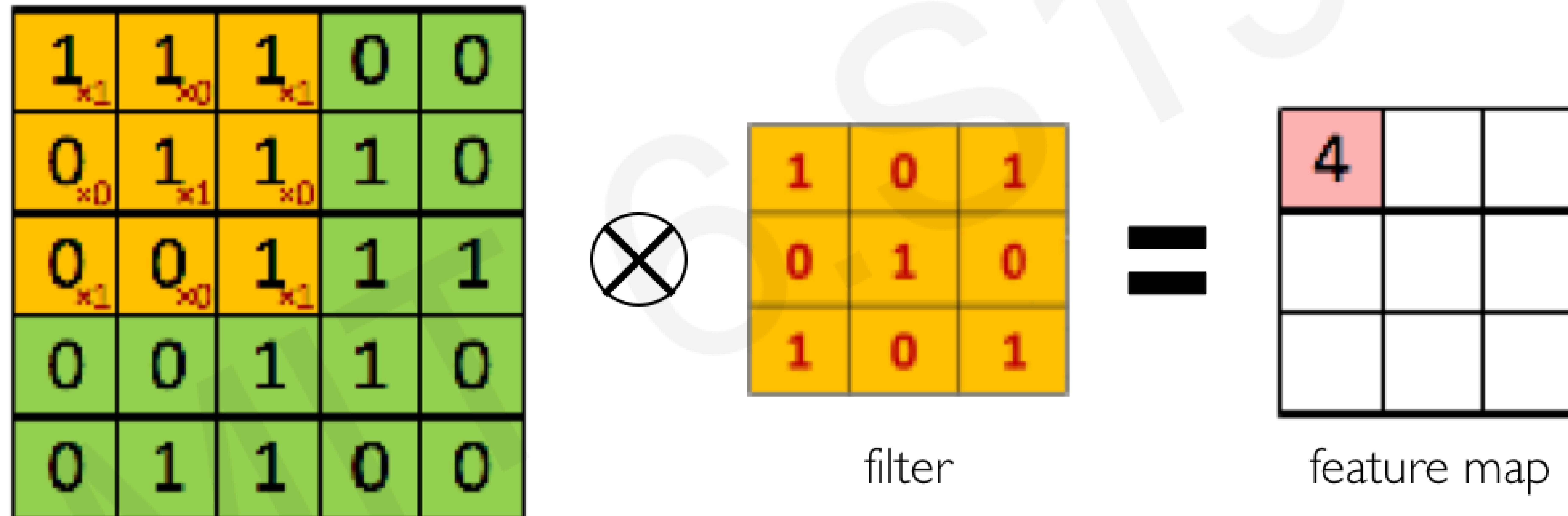
Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:



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

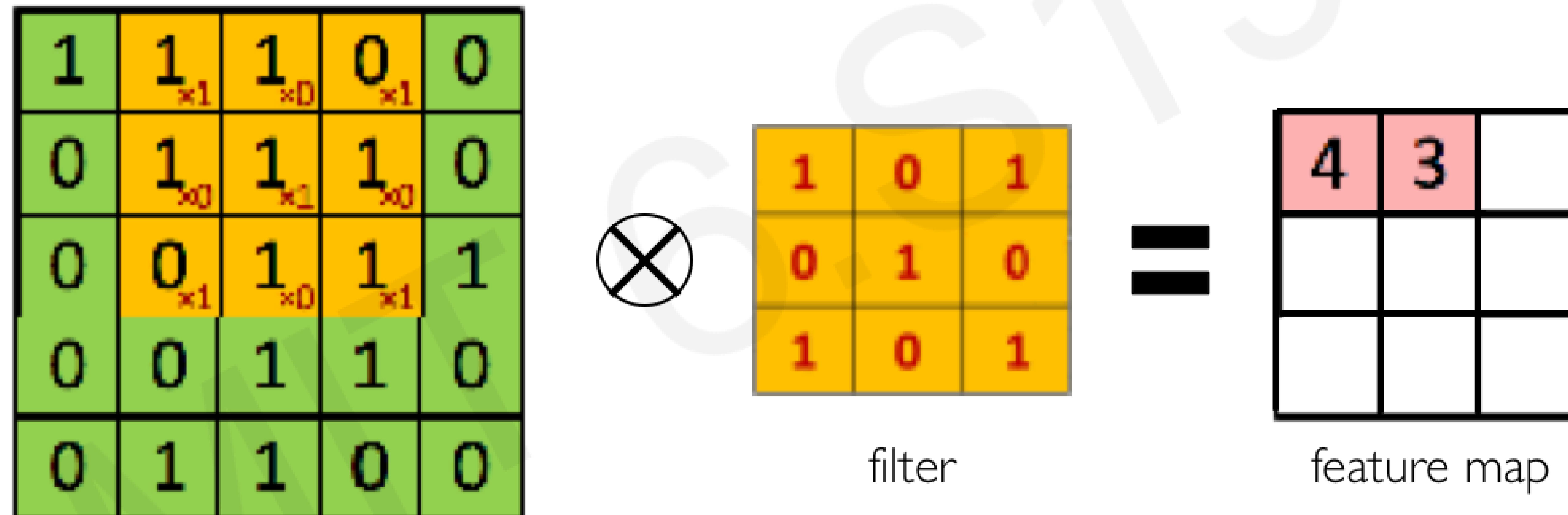
# The Convolution Operation

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



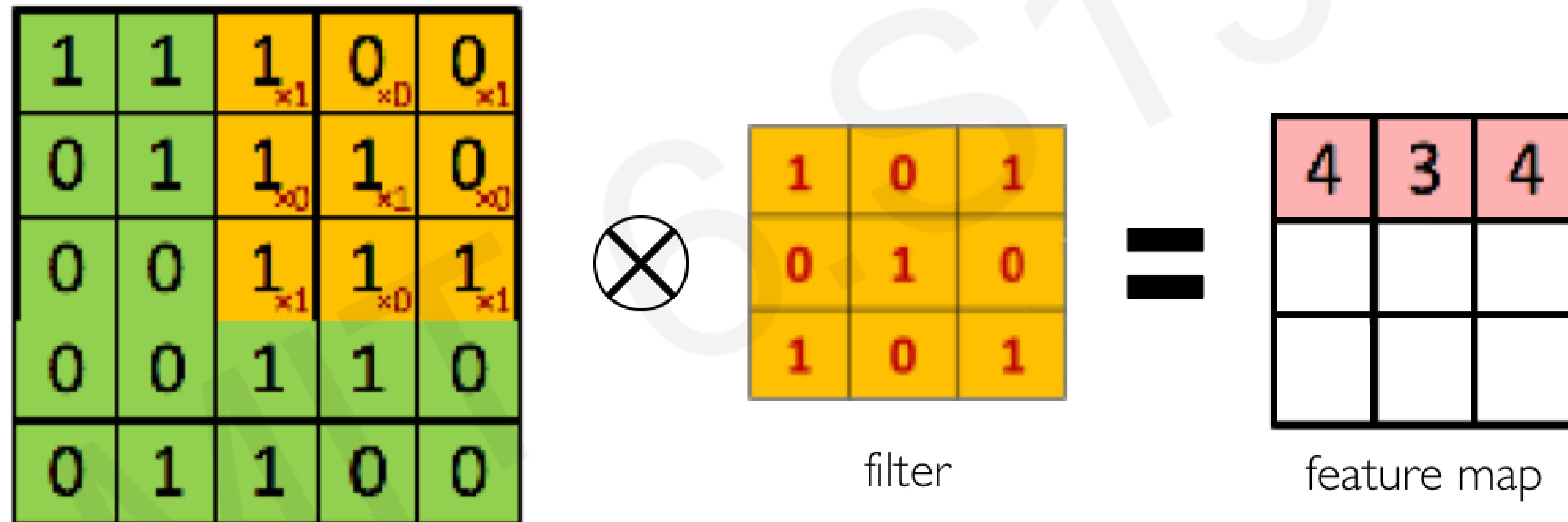
# The Convolution Operation

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



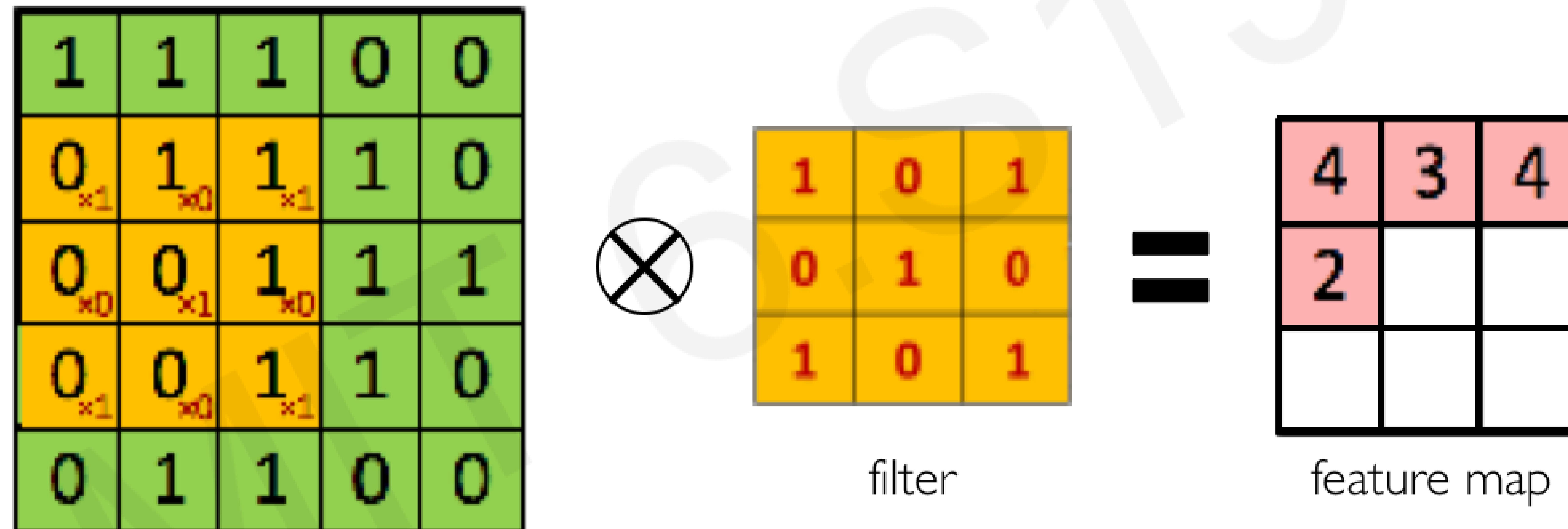
# The Convolution Operation

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



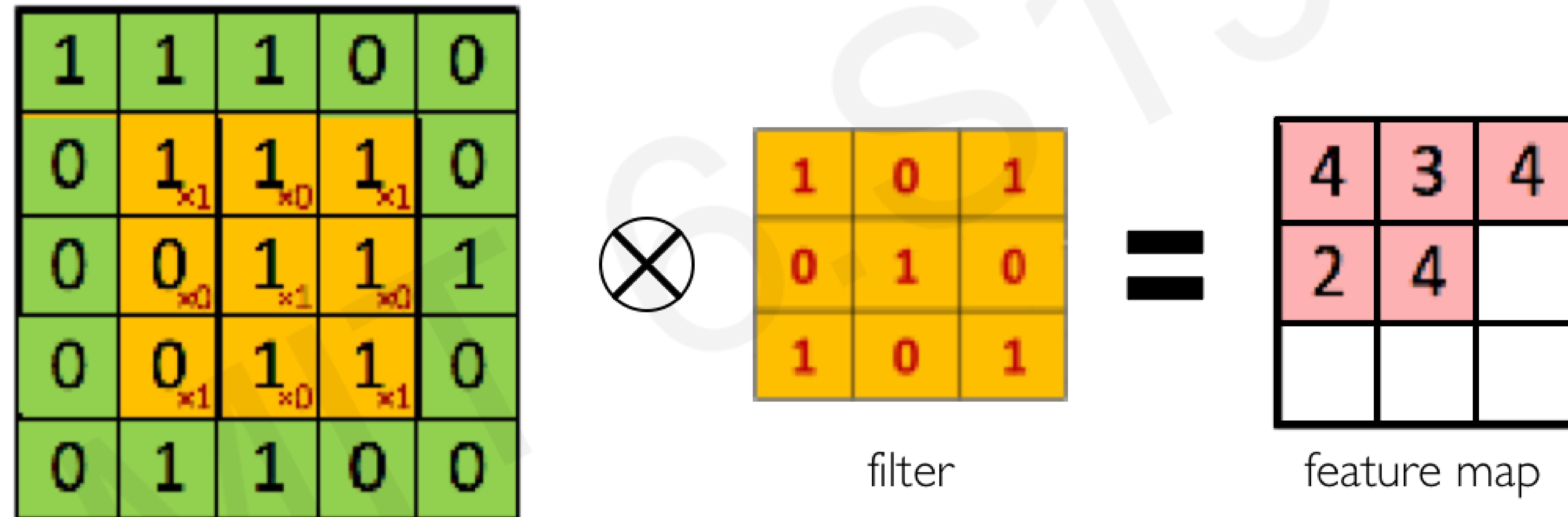
# The Convolution Operation

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



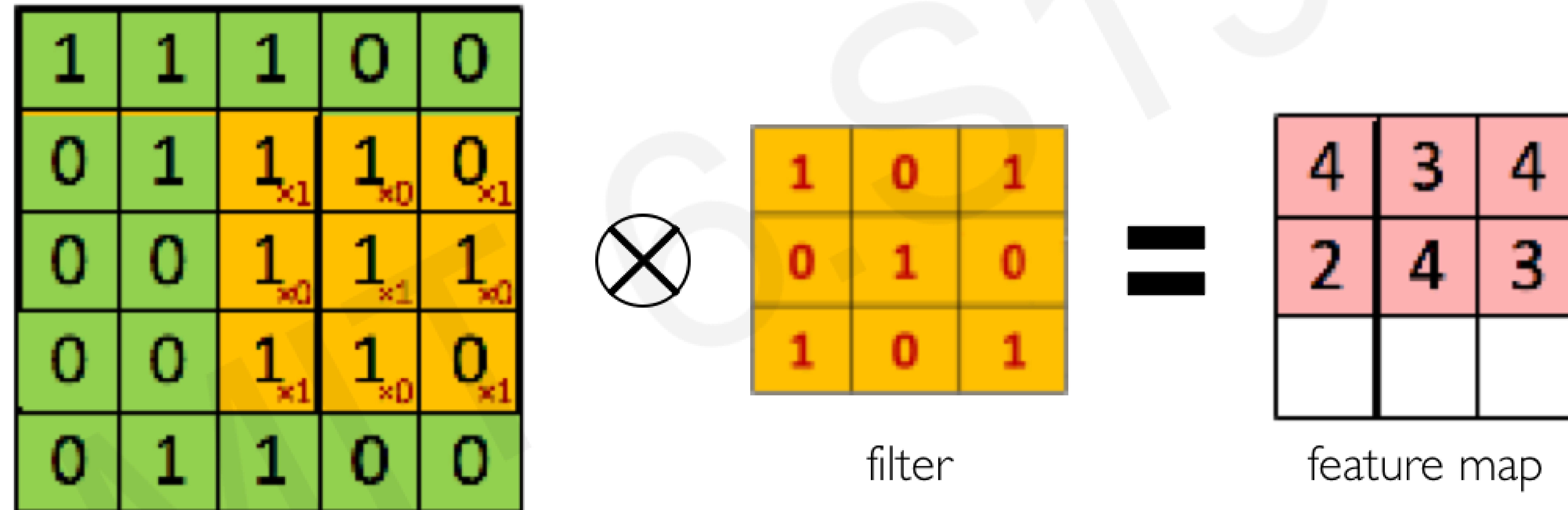
# The Convolution Operation

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



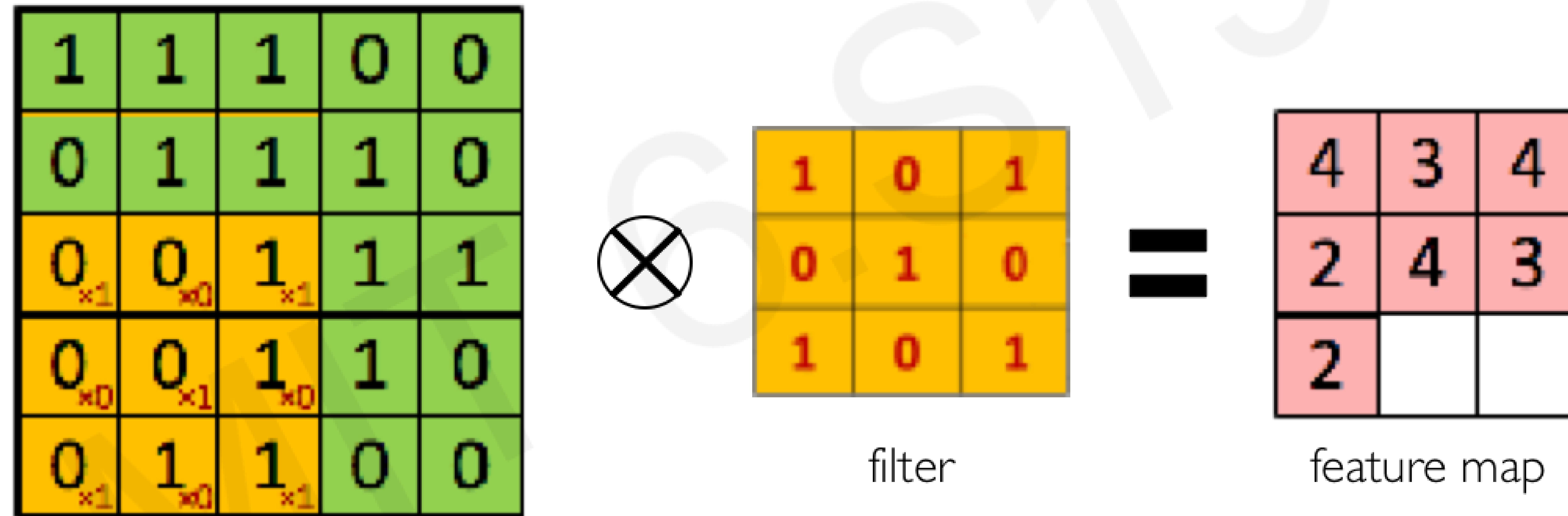
# The Convolution Operation

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



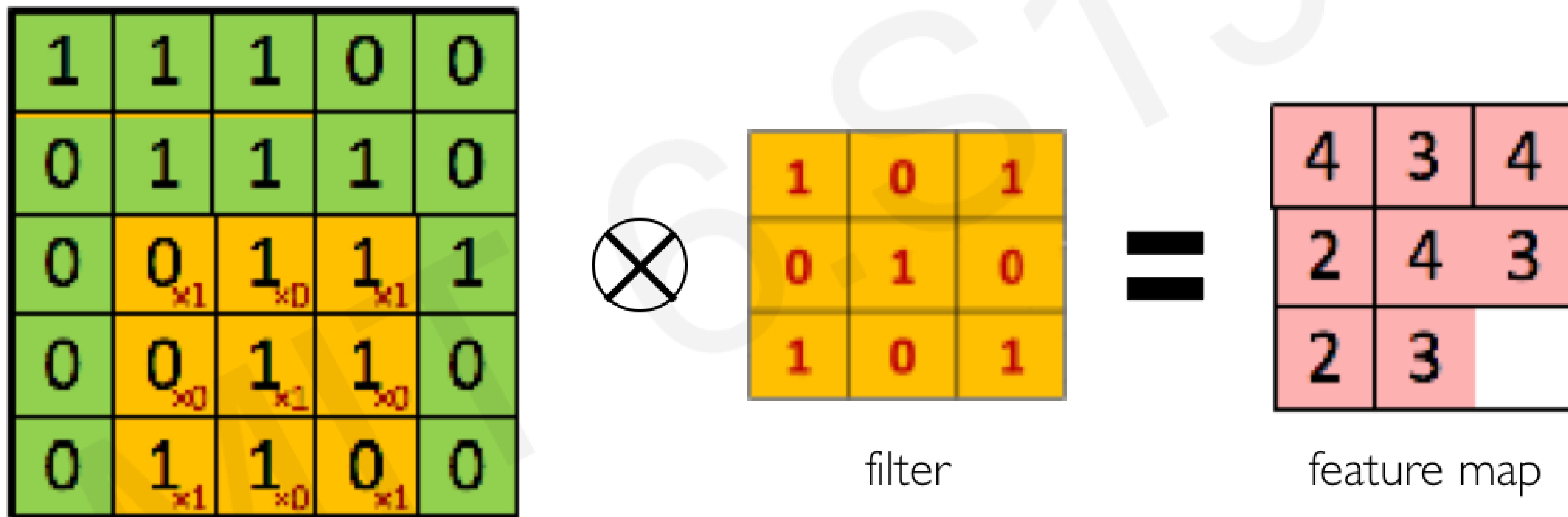
# The Convolution Operation

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



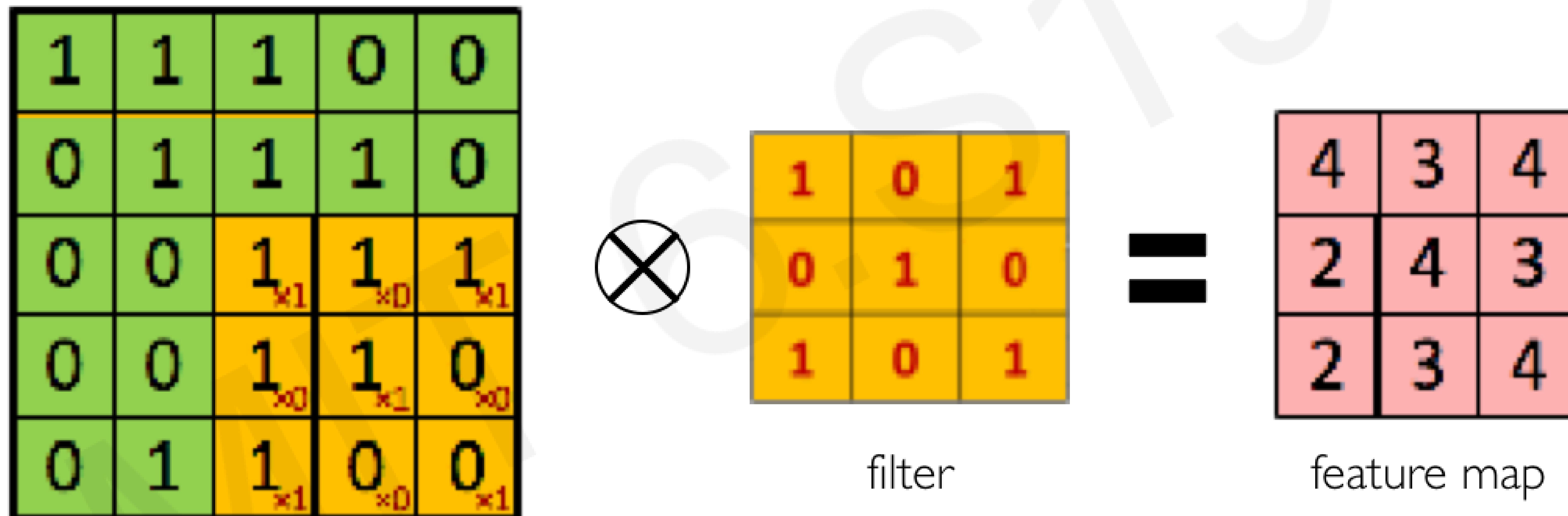
# The Convolution Operation

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

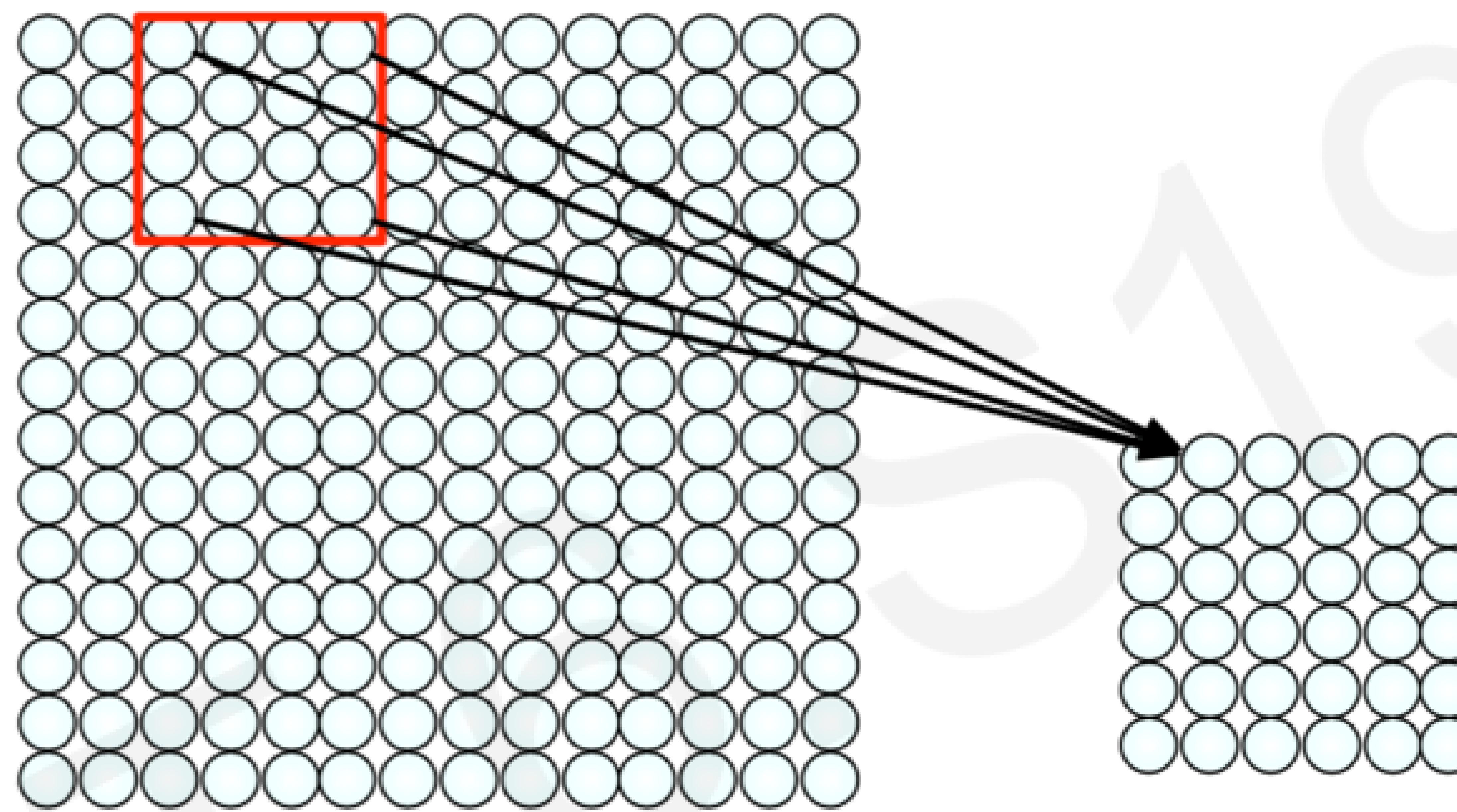


Edge Detect



“Strong” Edge  
Detect

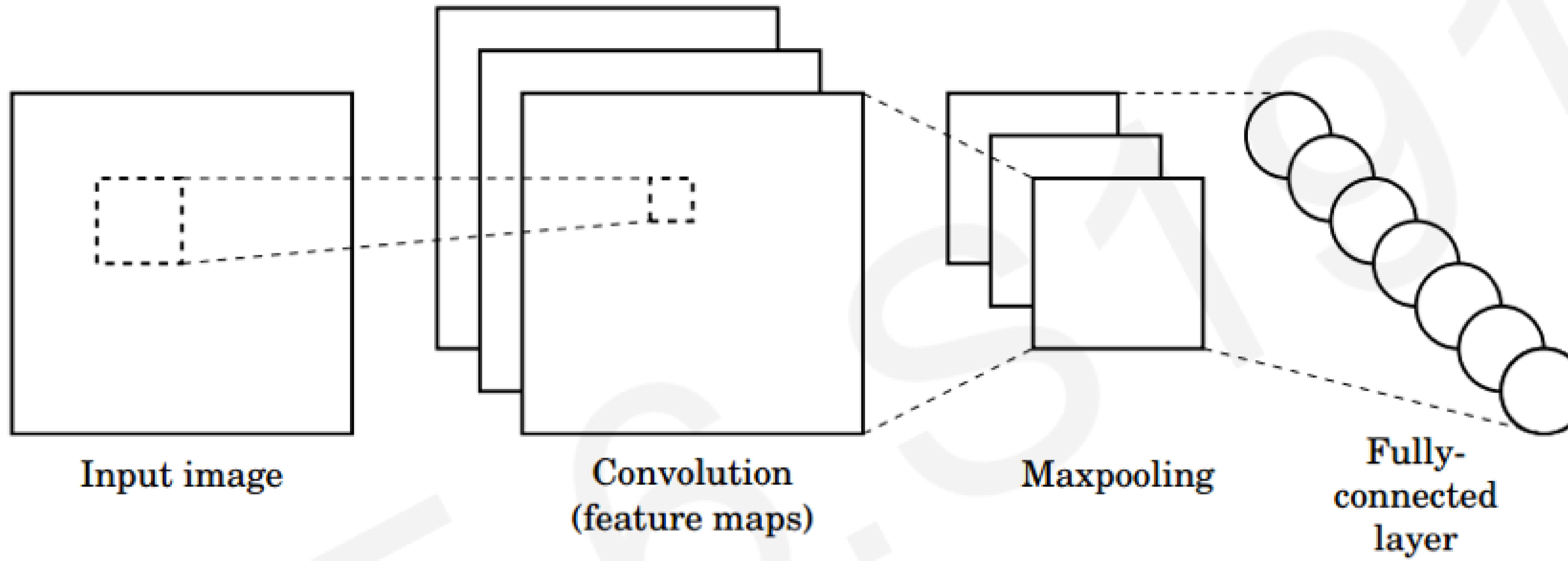
# Feature Extraction with Convolution



- 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 to generate feature maps.

 `tf.keras.layers.Conv2D`

**2. Non-linearity:** Often ReLU.

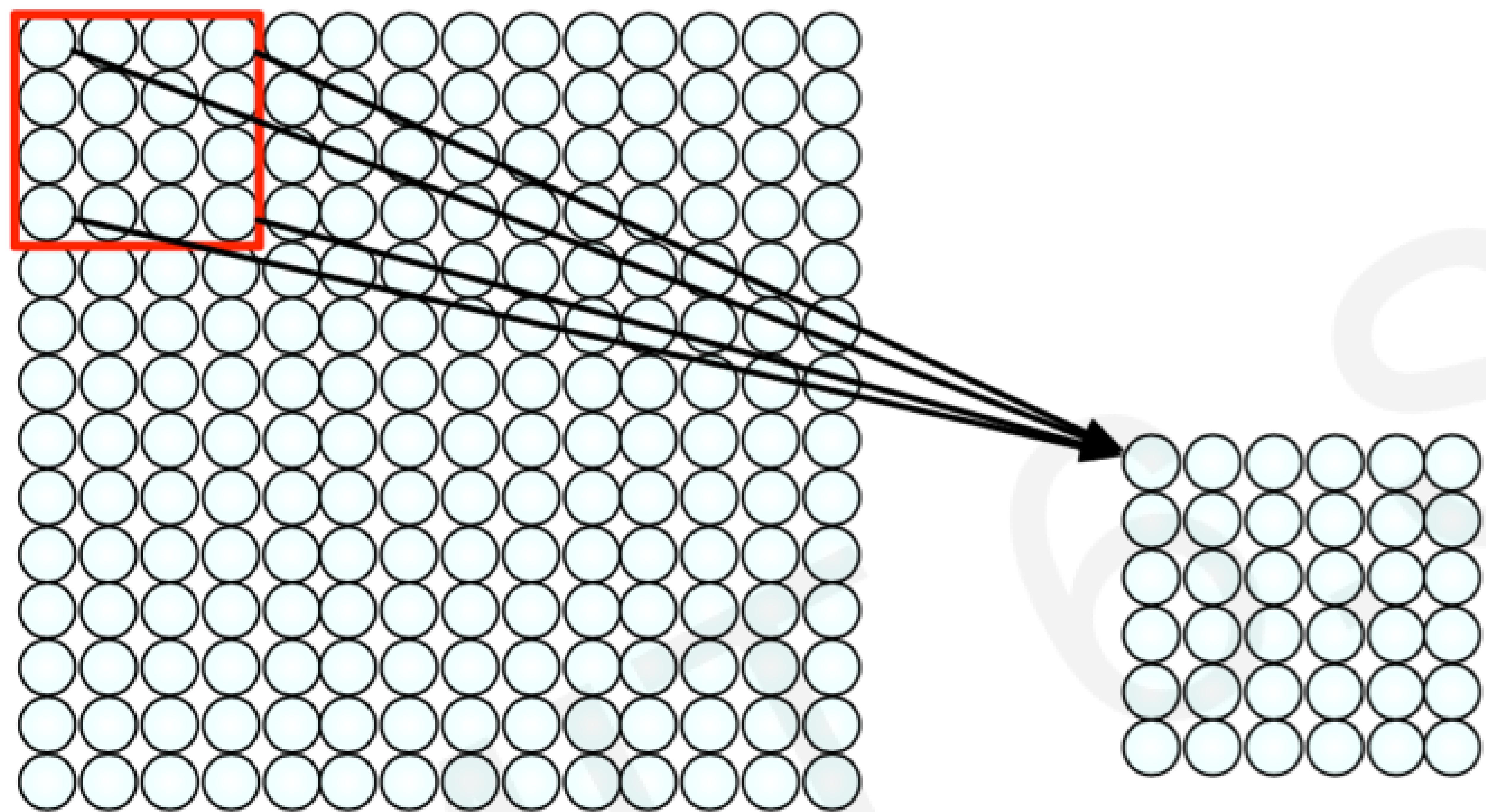
 `tf.keras.activations.*`

**3. Pooling:** Downsampling operation on each feature map.

 `tf.keras.layers.MaxPool2D`

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

# Convolutional Layers: Local Connectivity

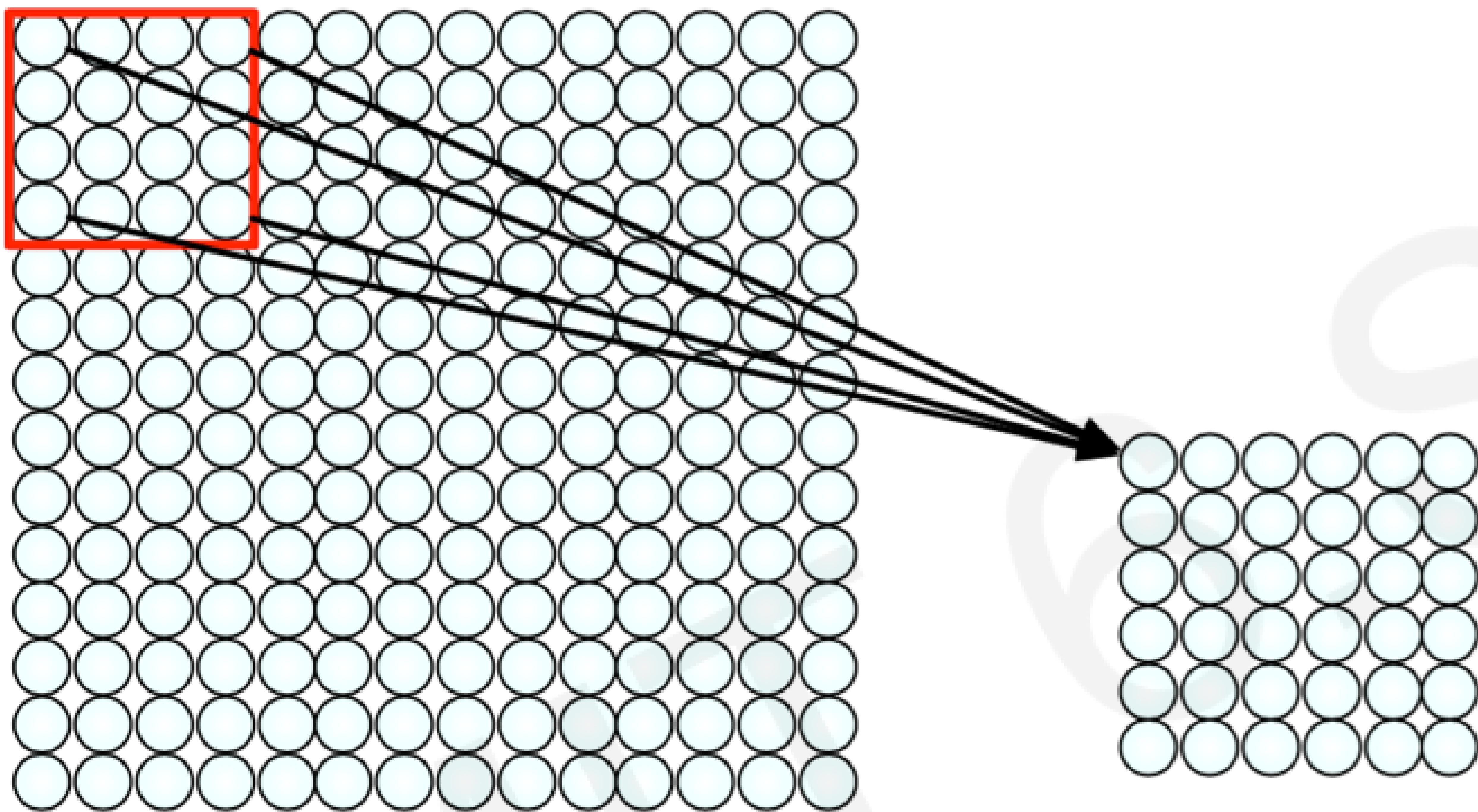


 `tf.keras.layers.Conv2D`

**For a neuron in hidden layer:**

- Take inputs from patch
- Compute weighted sum
- Apply bias

# Convolutional Layers: Local Connectivity



4x4 filter: matrix  
of weights  $w_{ij}$

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

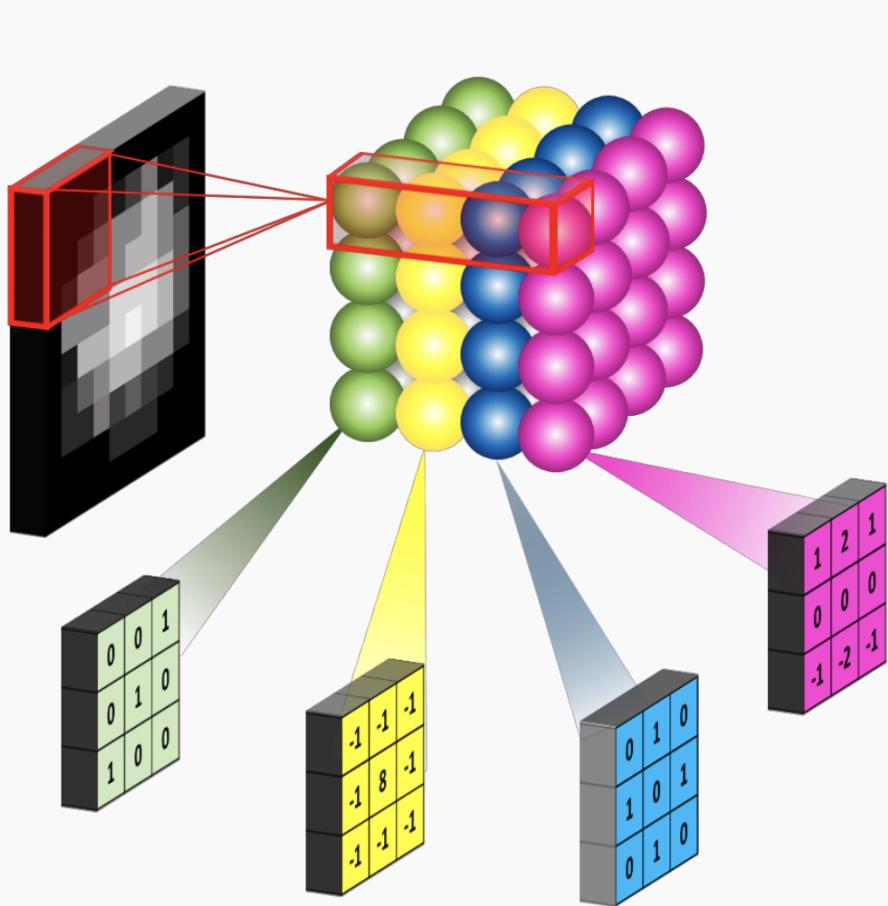
for neuron  $(p,q)$  in hidden layer

`tf.keras.layers.Conv2D`

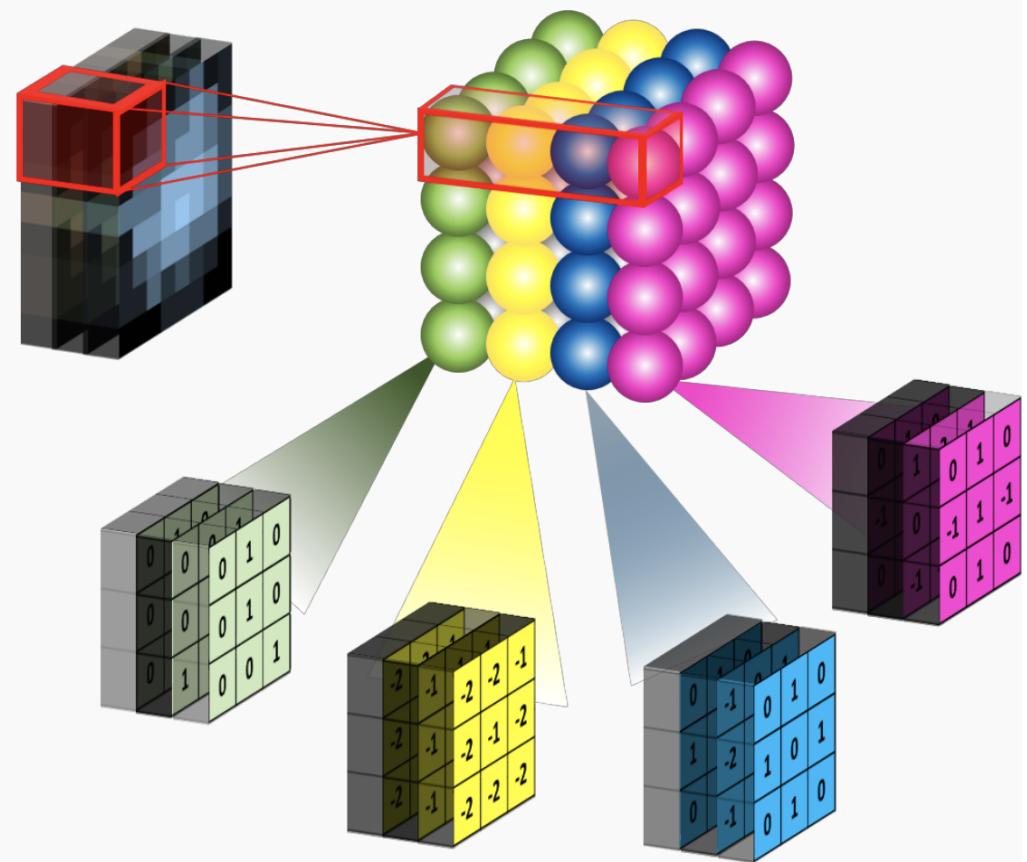
**For a neuron in hidden layer:**

- 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

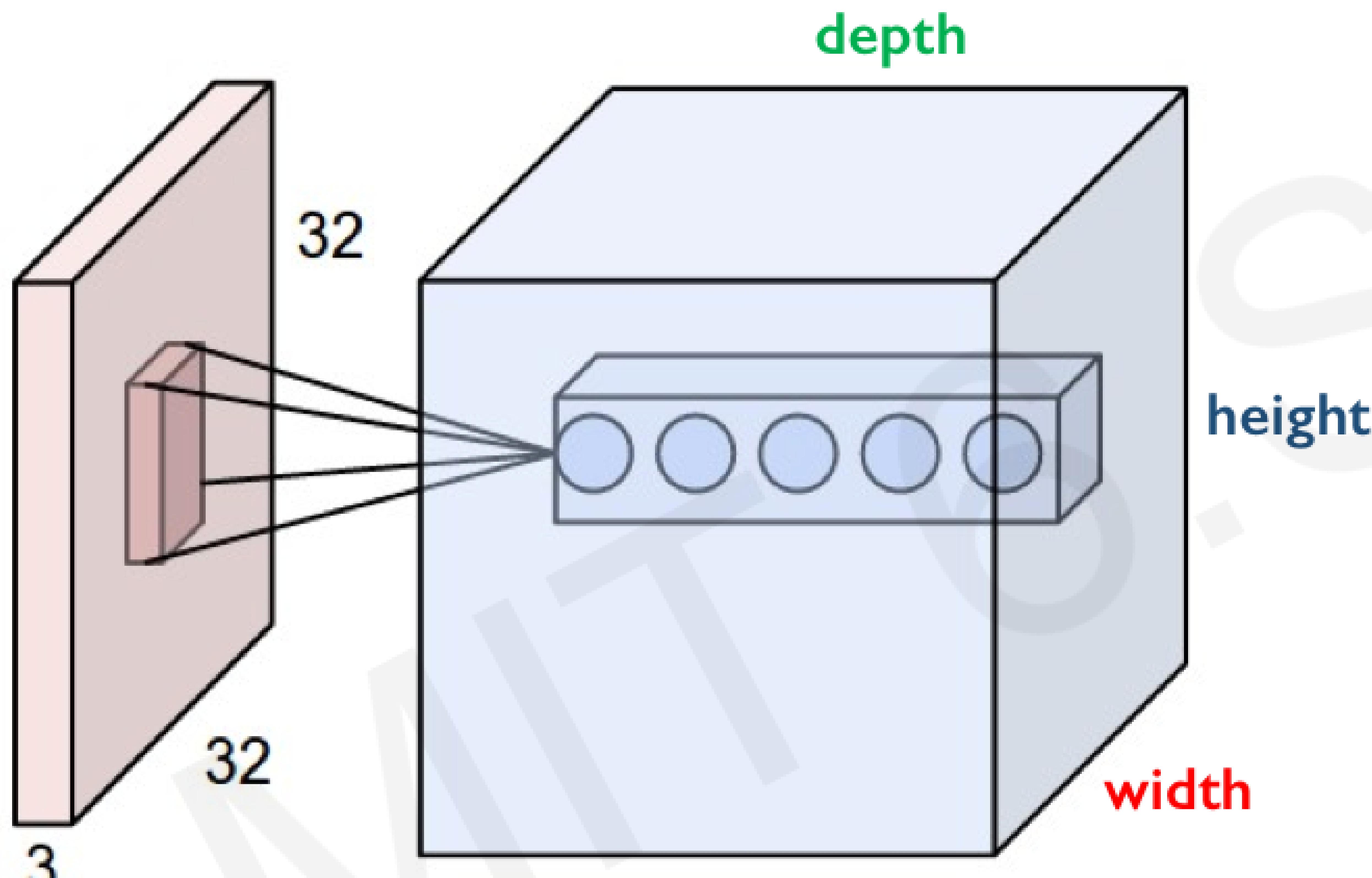


Convolutional layer with four 3x3 filters on a **black and white image** (just one channel)



Convolutional layer with four 3x3 filters on an **RGB image**. As you can see, the filters are now cubes, and they are applied on the full depth of the image..

# 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:**

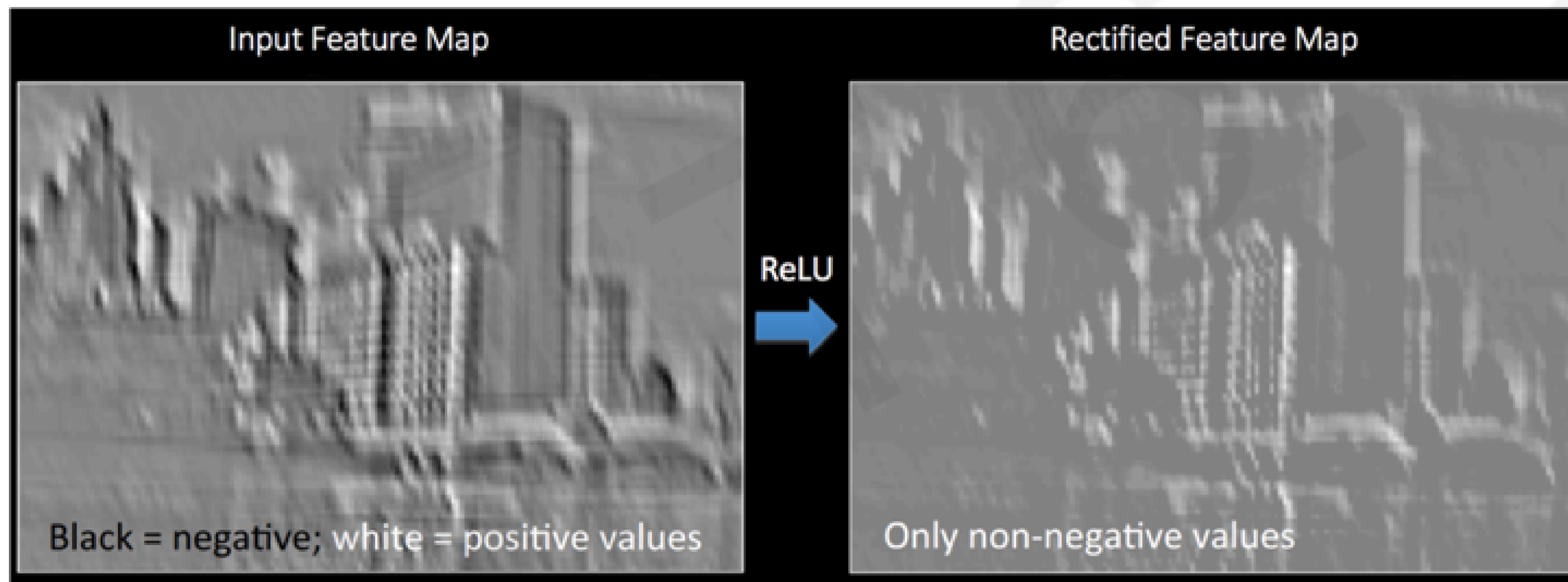
Locations in input image that  
a node is path connected to



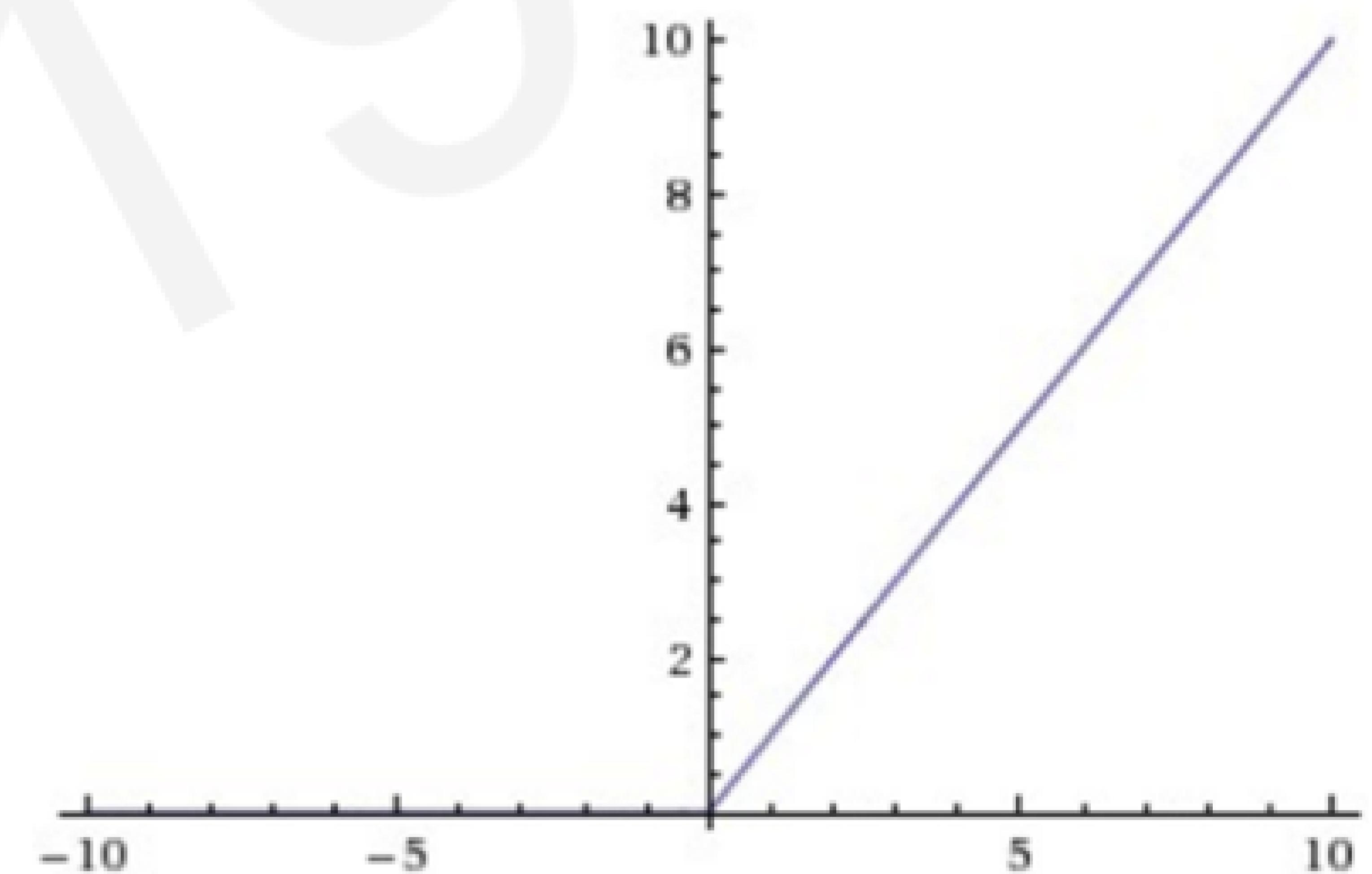
```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

# 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!**

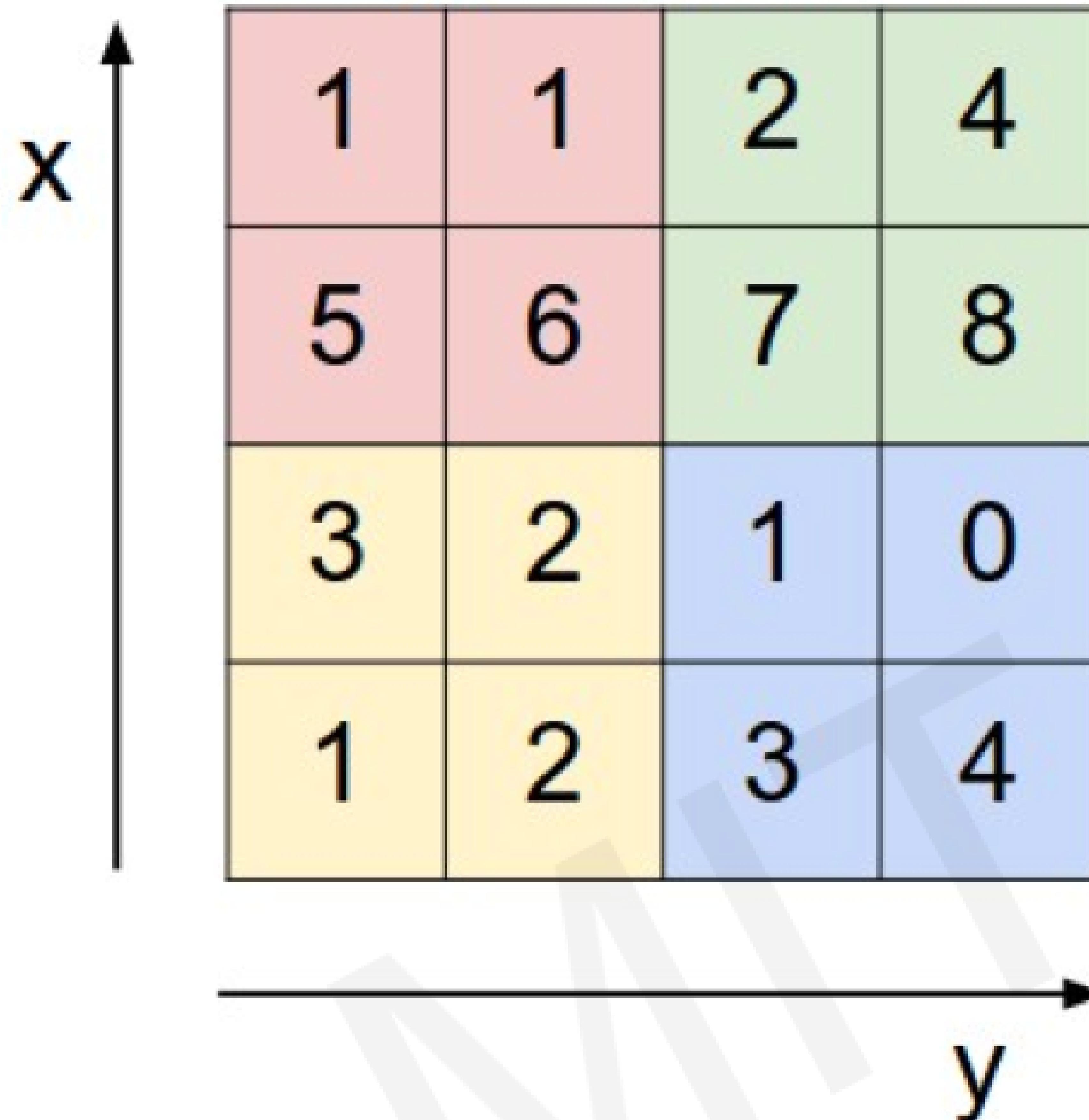


## Rectified Linear Unit (ReLU)



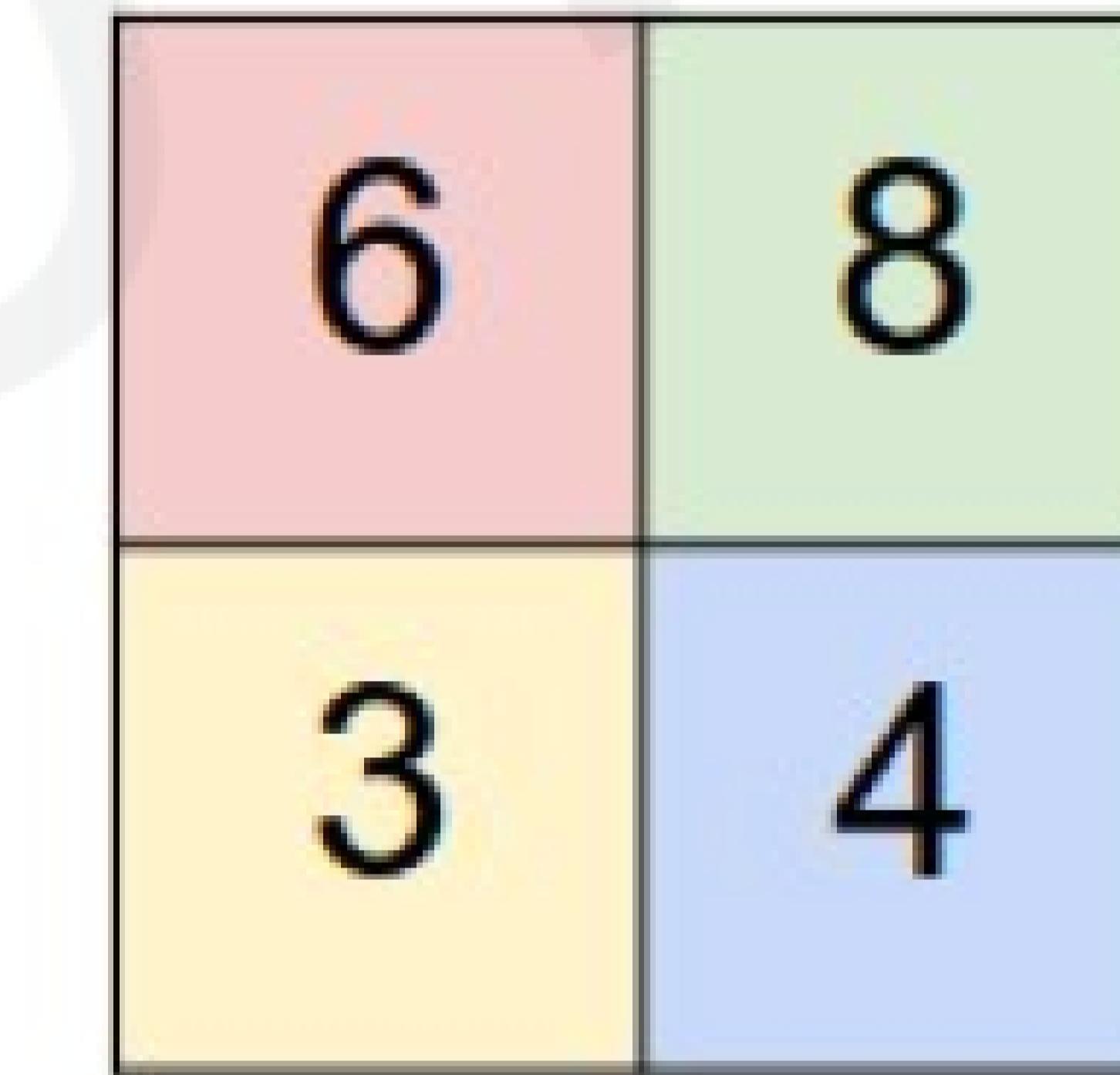
 `tf.keras.layers.ReLU`

# Pooling



max pool with 2x2 filters  
and stride 2

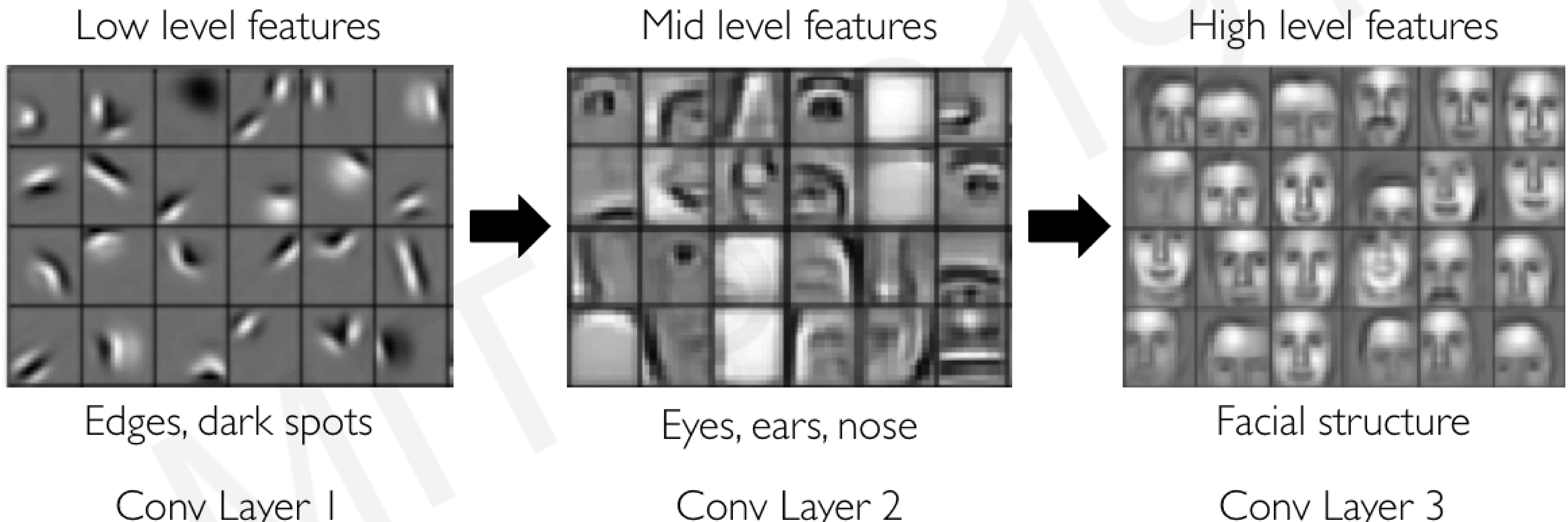
```
tf.keras.layers.MaxPool2D(  
    pool_size=(2,2),  
    strides=2  
)
```



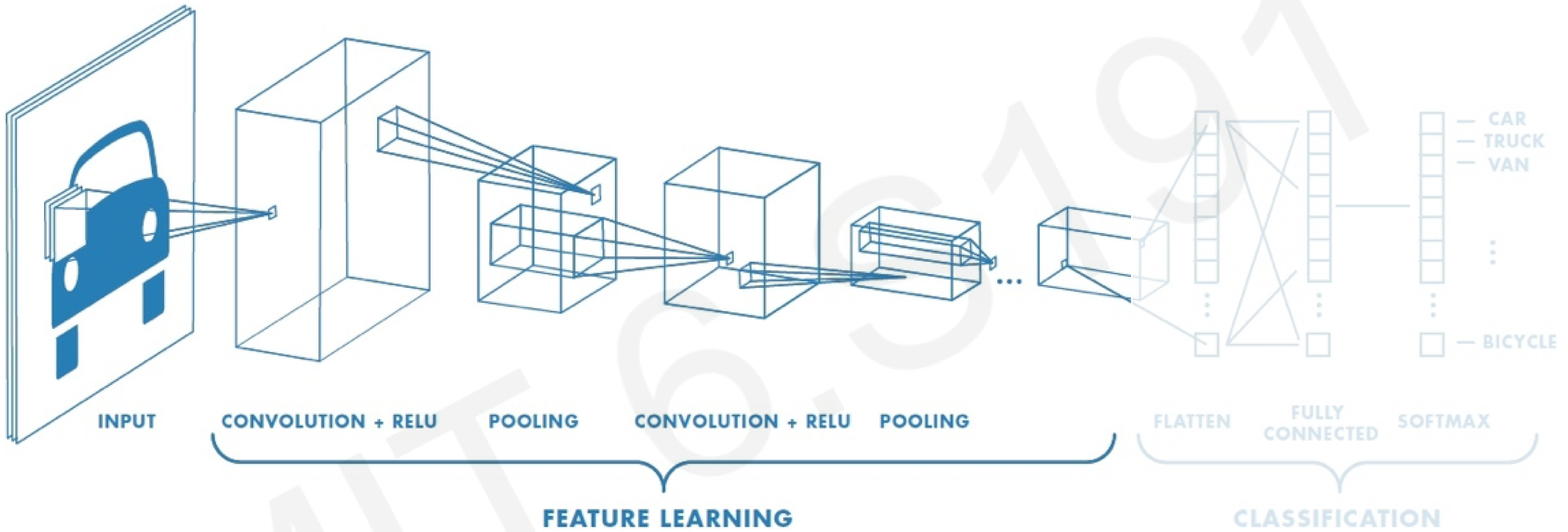
- 1) Reduced dimensionality
- 2) Spatial invariance

How else can we downsample and preserve spatial invariance?

# Representation Learning in Deep CNNs

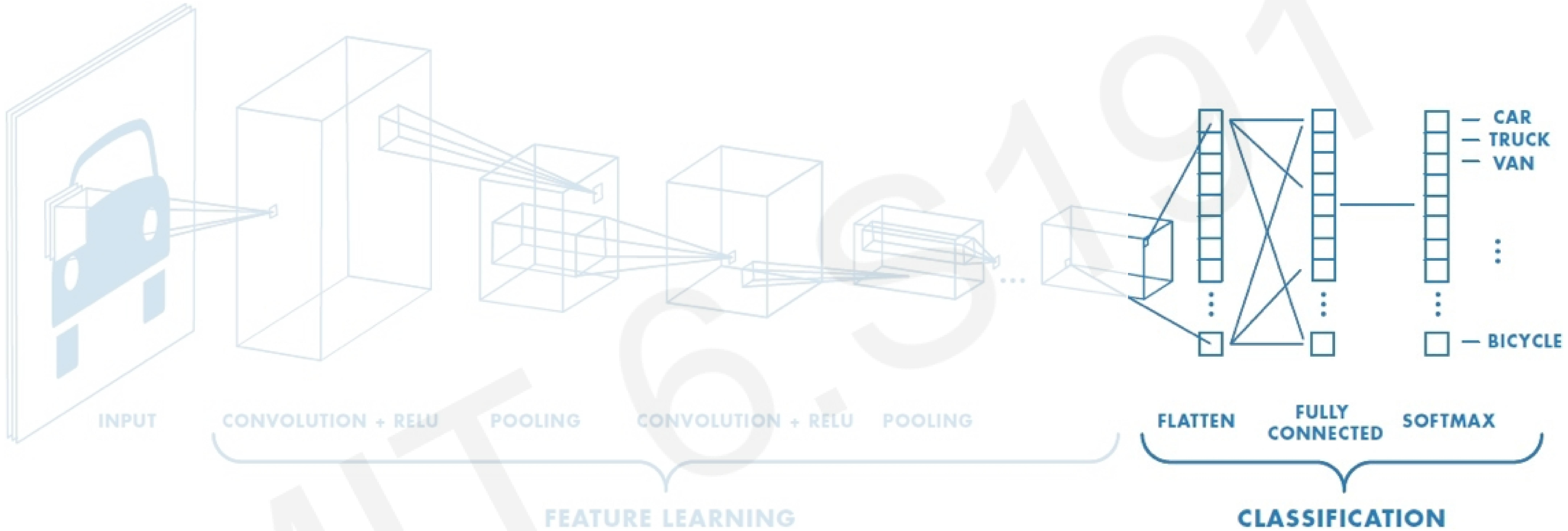


# CNNs for Classification: Feature Learning



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



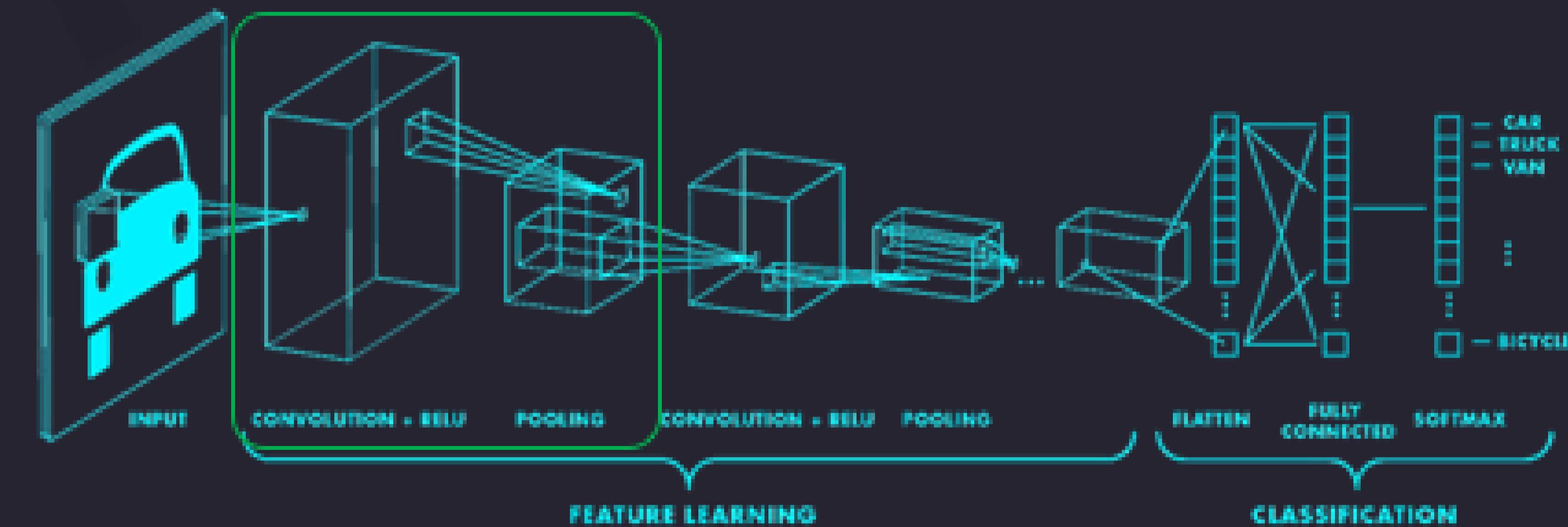
- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

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

# Putting it all together

```
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
```



# 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



## ImageNet Large Scale Visual Recognition Challenges



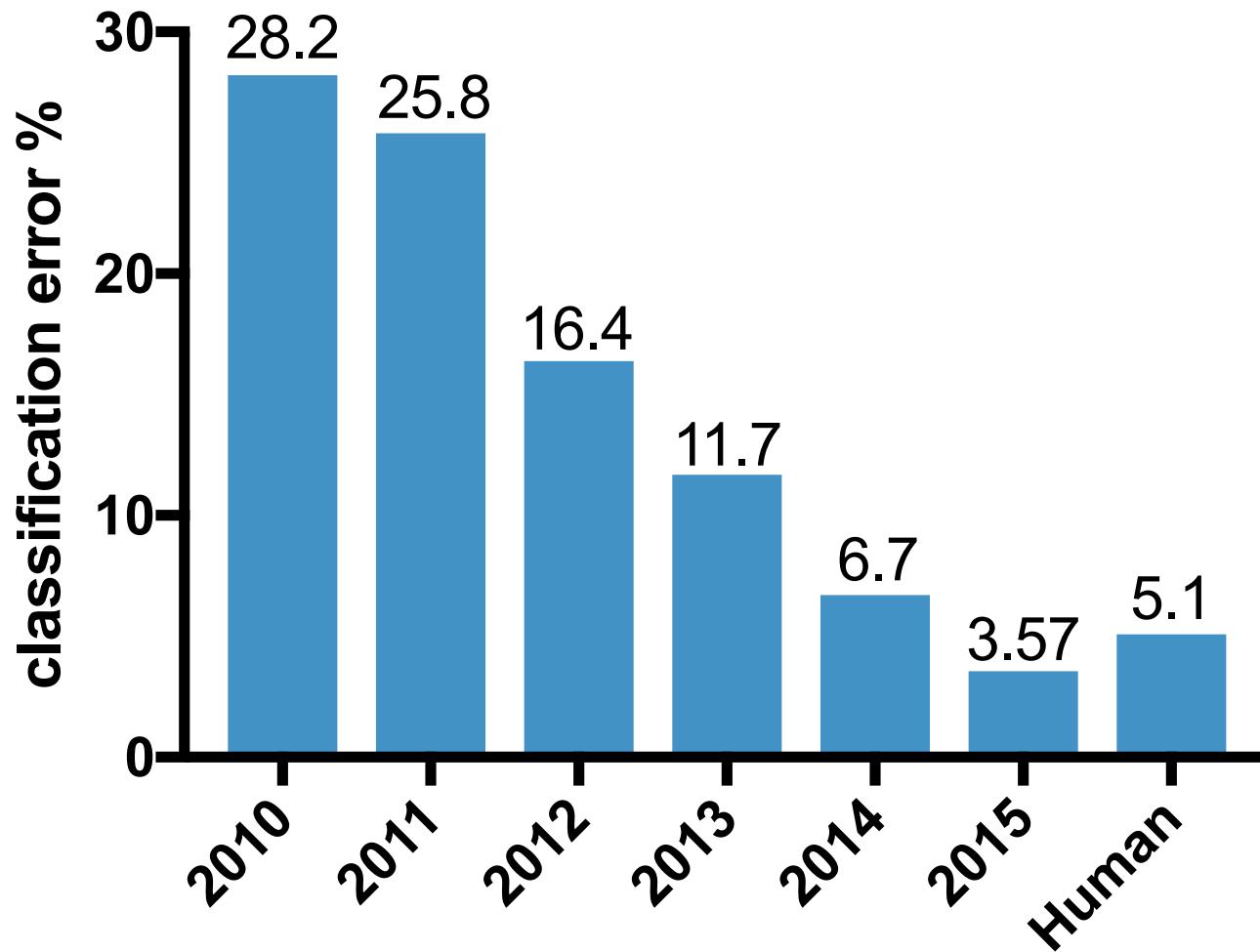
**Classification task:** produce a list of object categories present in image. 1000 categories.

“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**

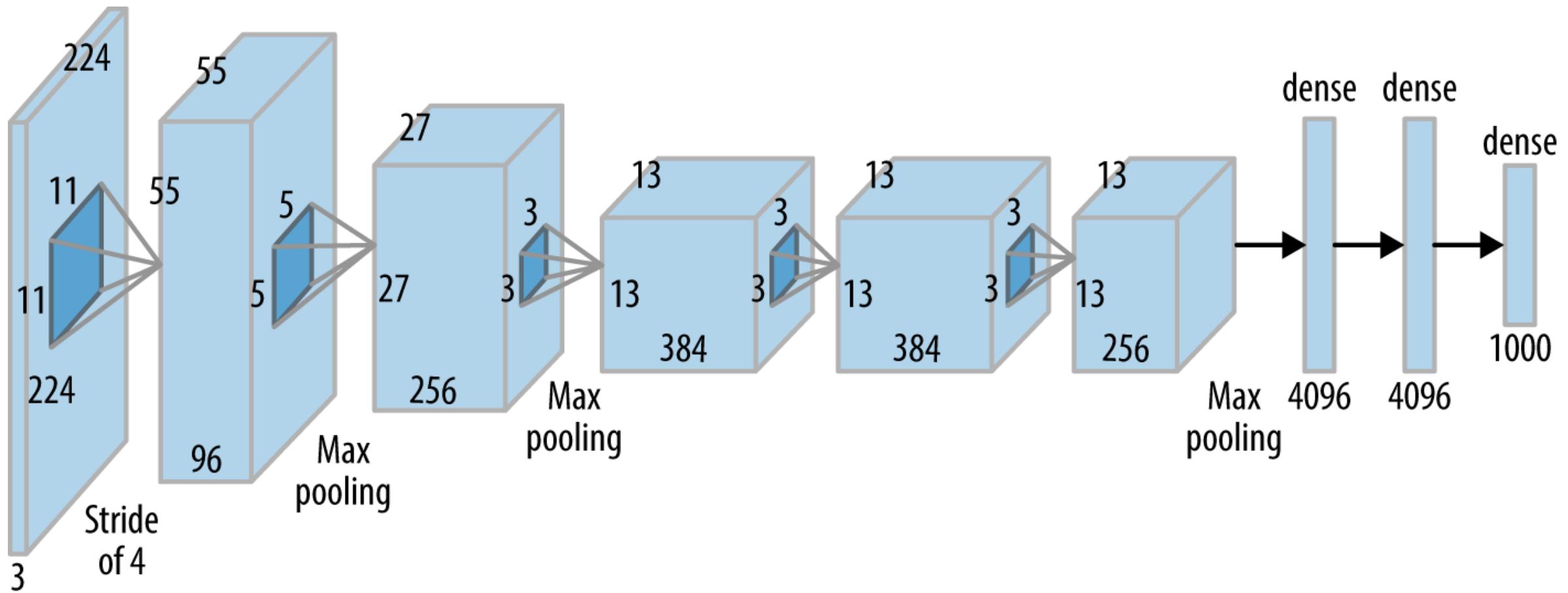
- 19 layers

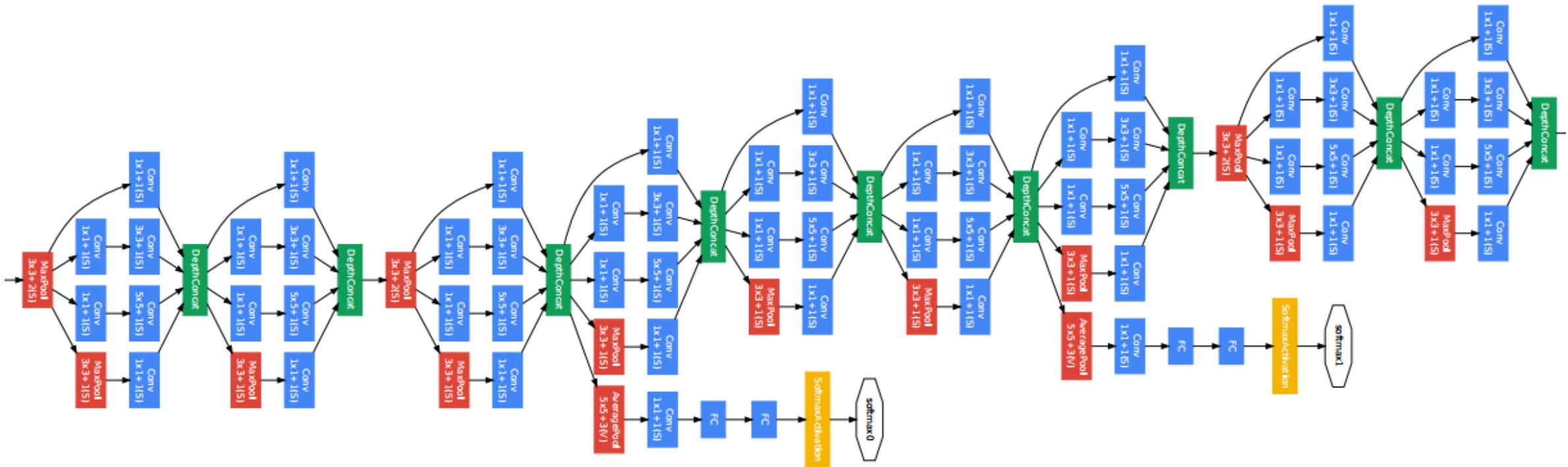
**2014: GoogLeNet**

- “Inception” modules
- 22 layers, 5 million parameters

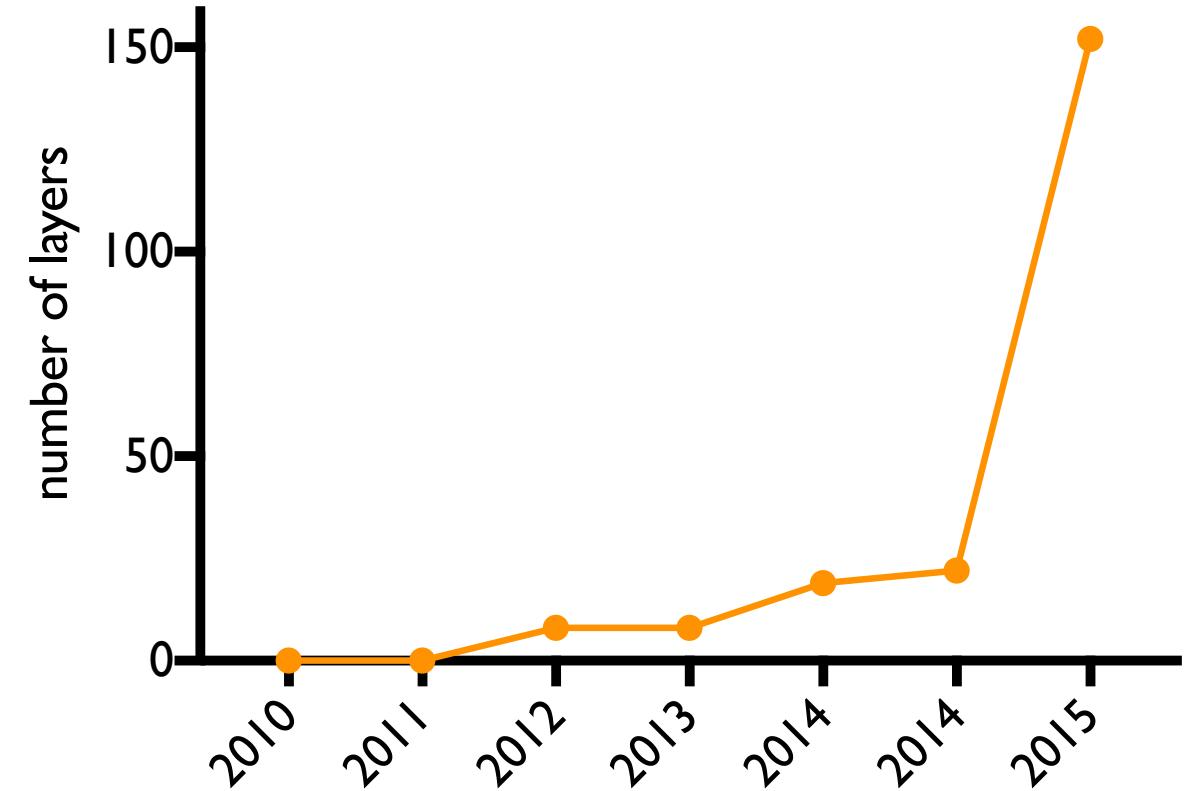
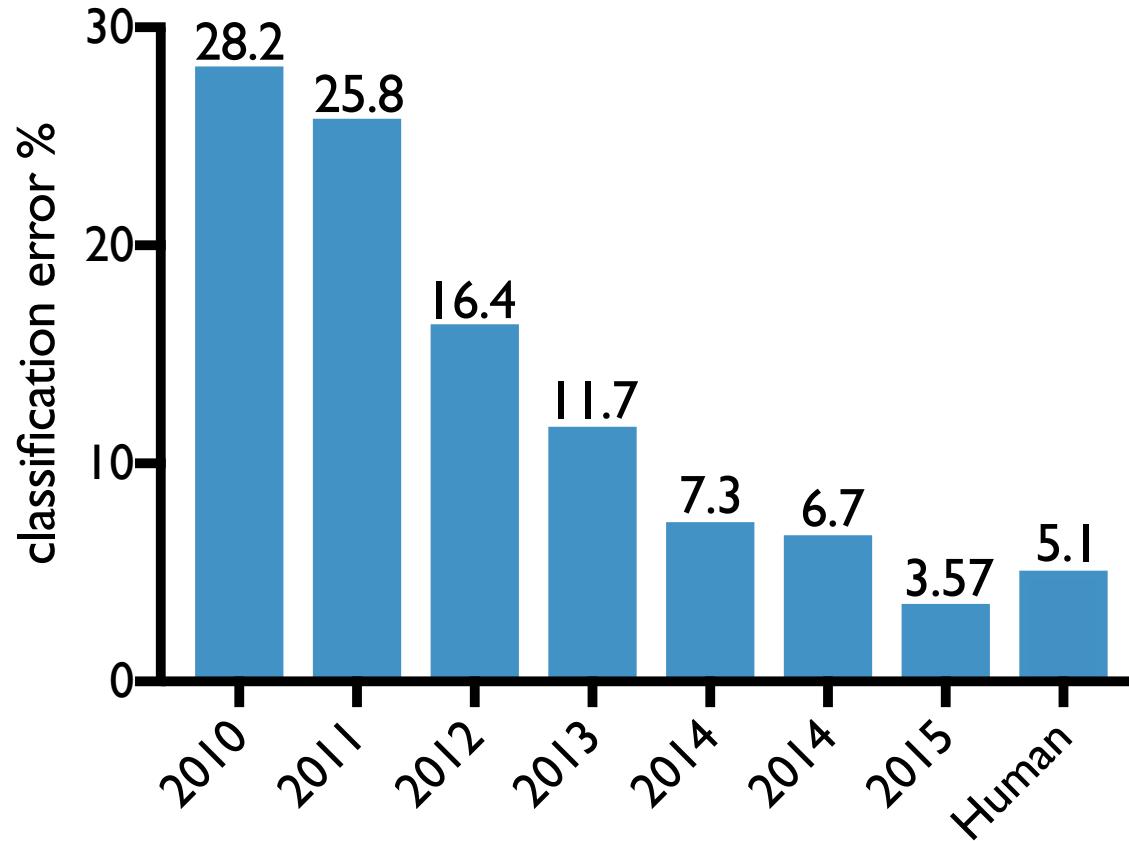
**2015: ResNet**

- 152 layers



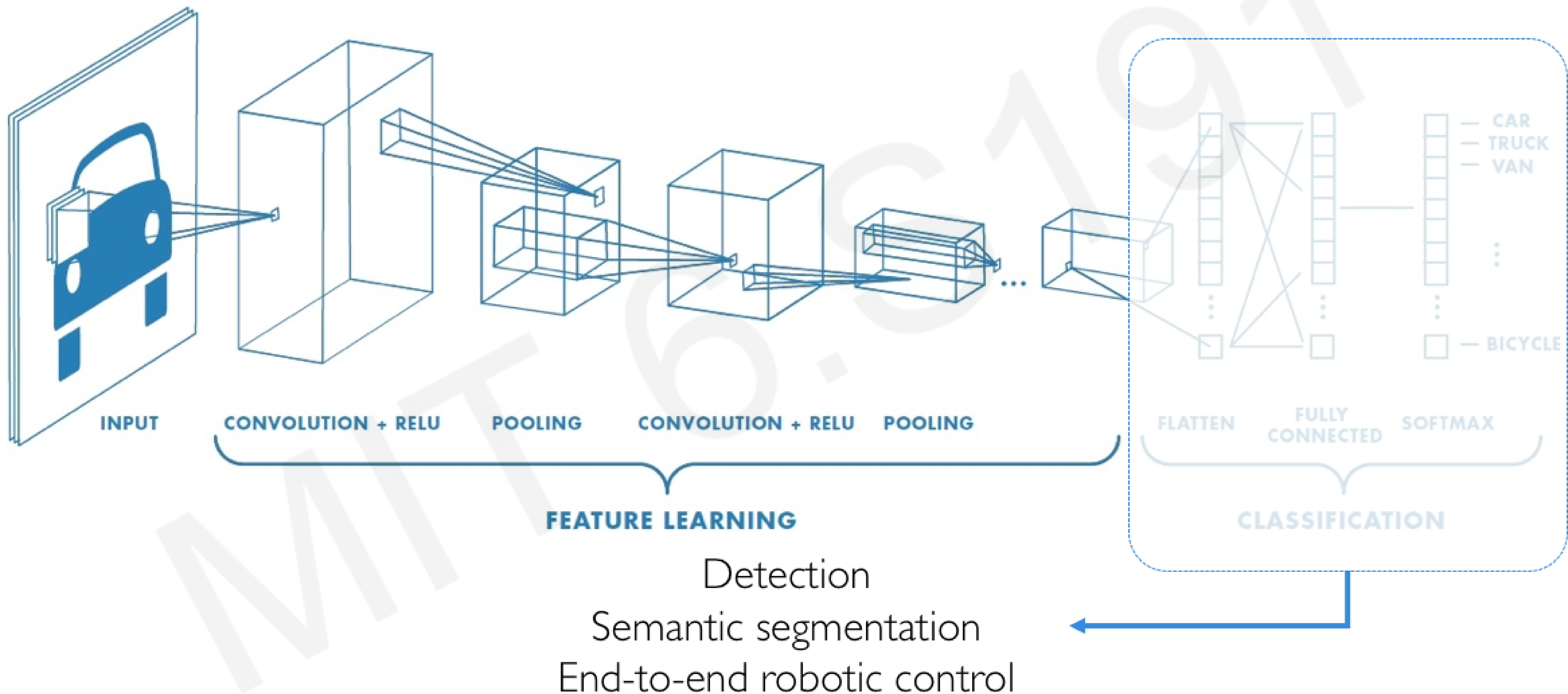


# ImageNet Challenge: Classification Task



# An Architecture for Many Applications

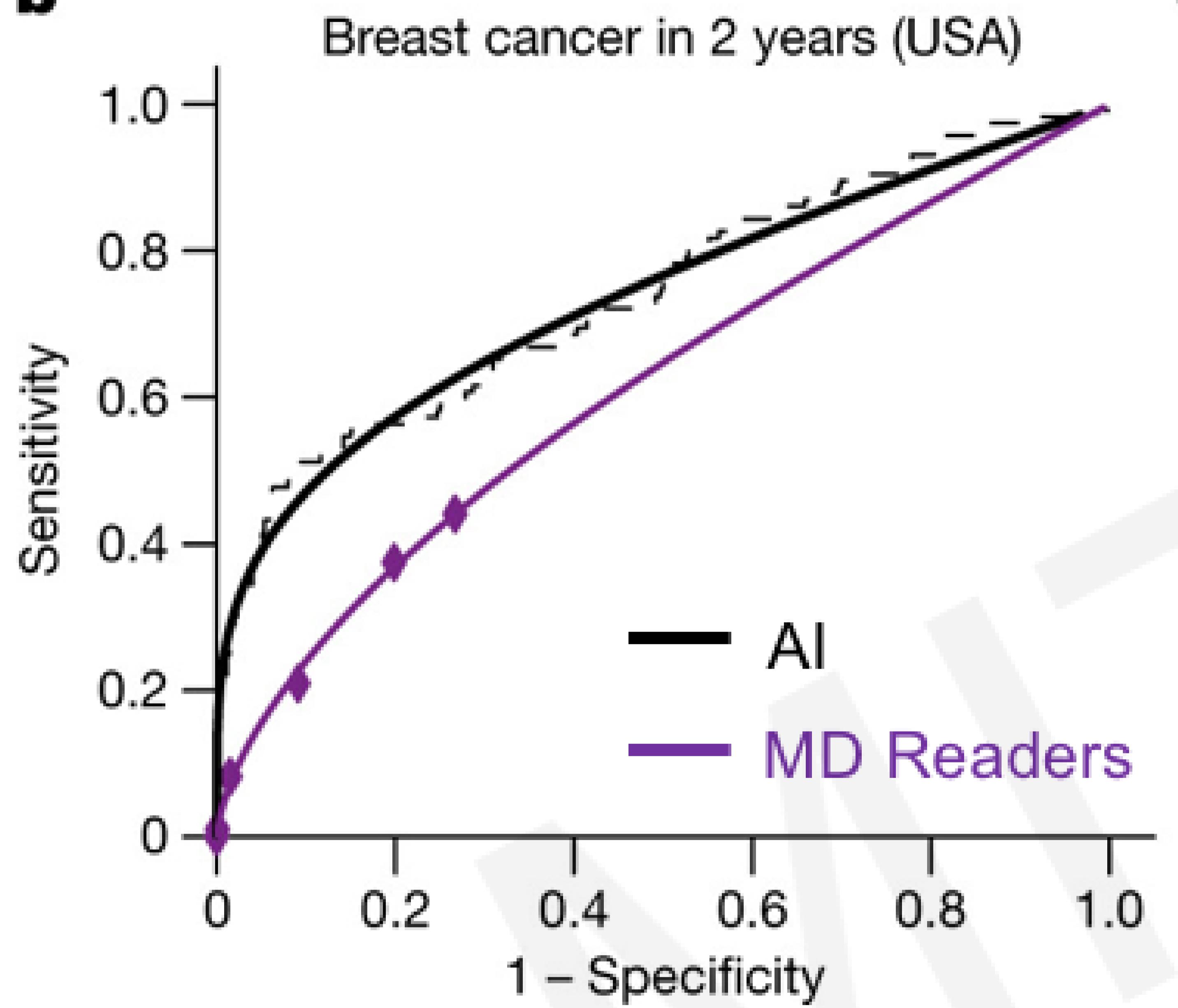
# An Architecture for Many Applications



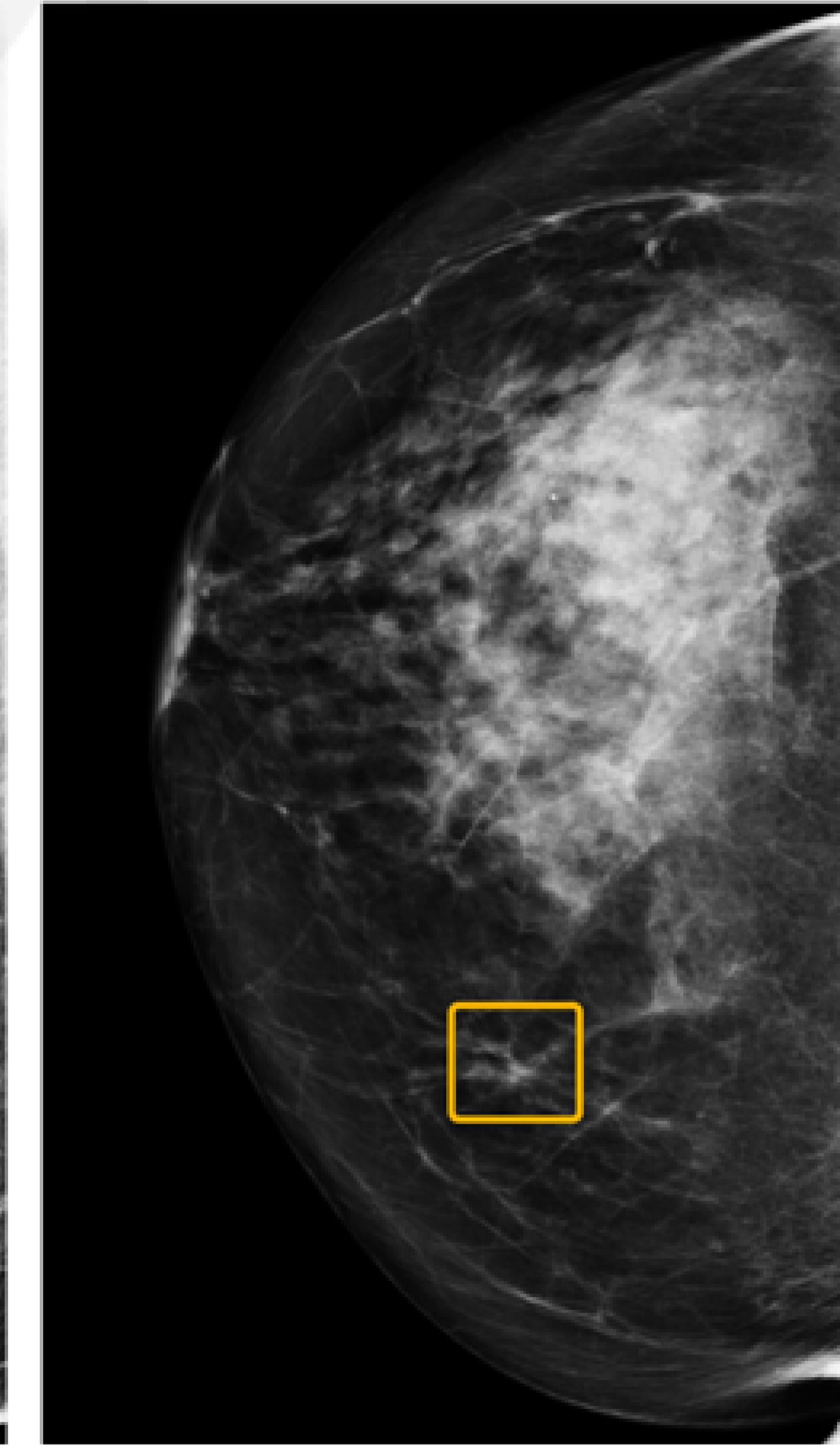
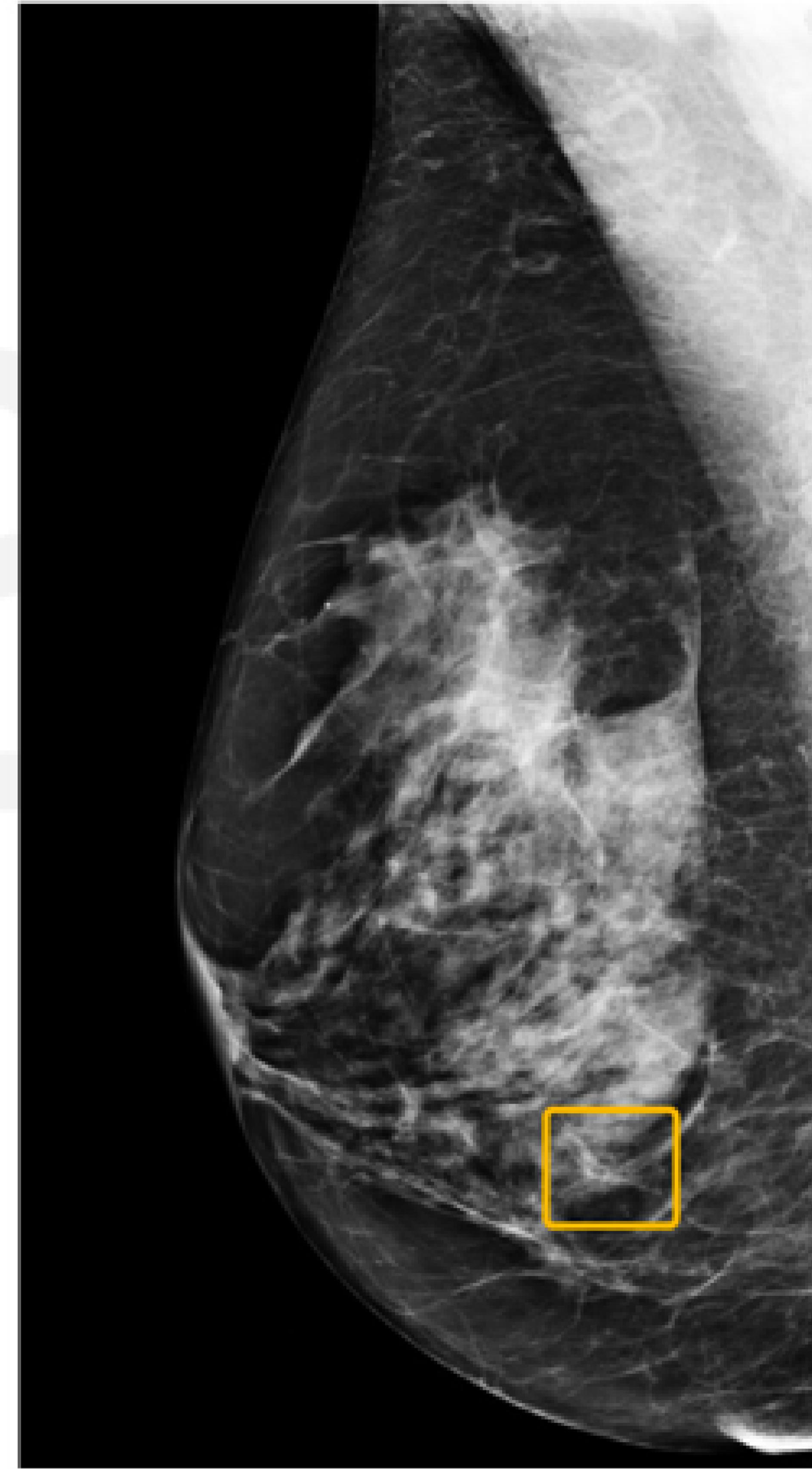
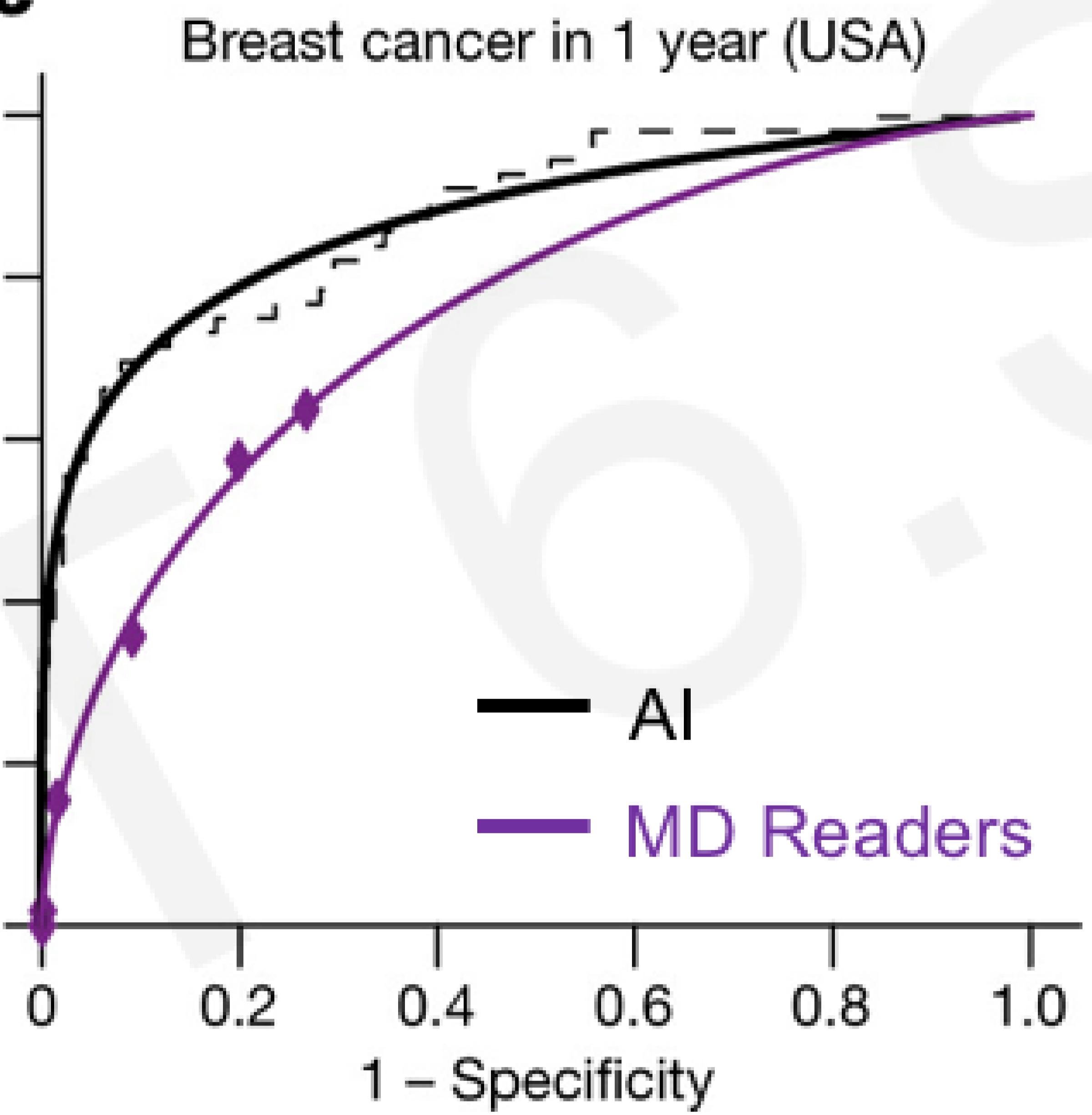
# Detection: Breast Cancer Screening

## International evaluation of an AI system for breast cancer screening

b



c



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

Breast cancer case missed by radiologist but detected by AI

# Semantic Segmentation: Fully Convolutional Networks

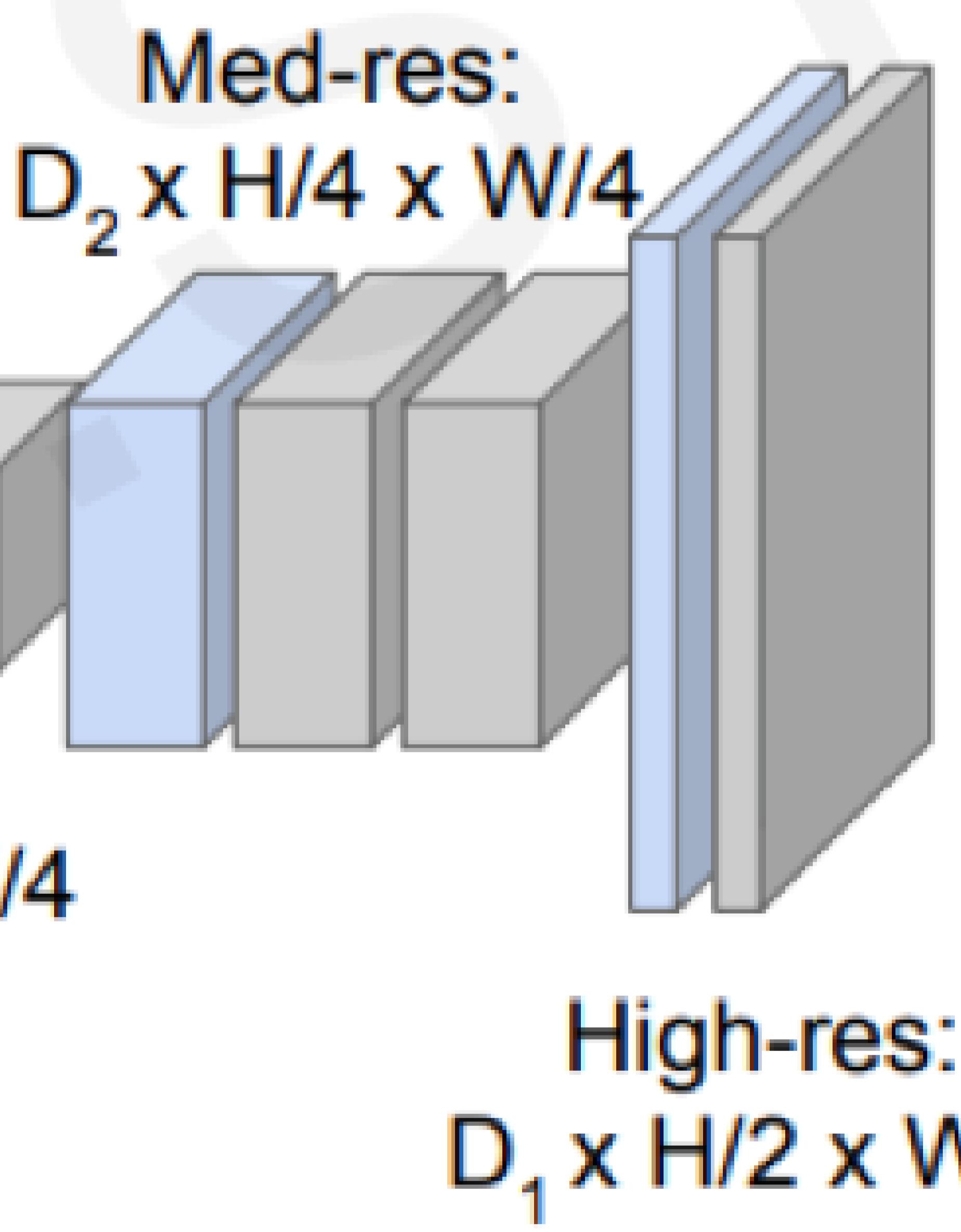
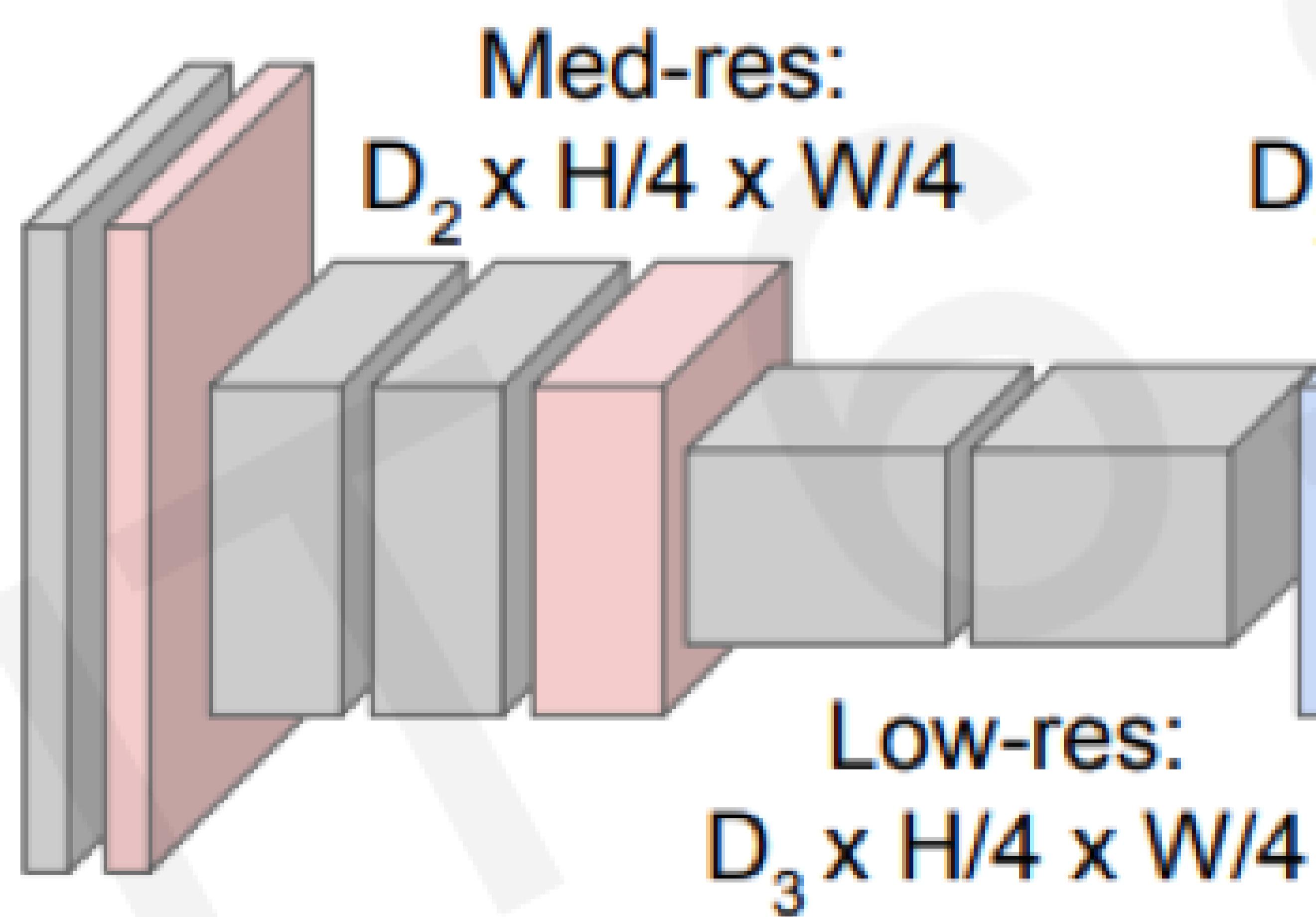
FCN: Fully Convolutional Network.

Network designed with all convolutional layers,  
with **downsampling** and **upsampling** operations



Input:  
 $3 \times H \times W$

High-res:  
 $D_1 \times H/2 \times W/2$

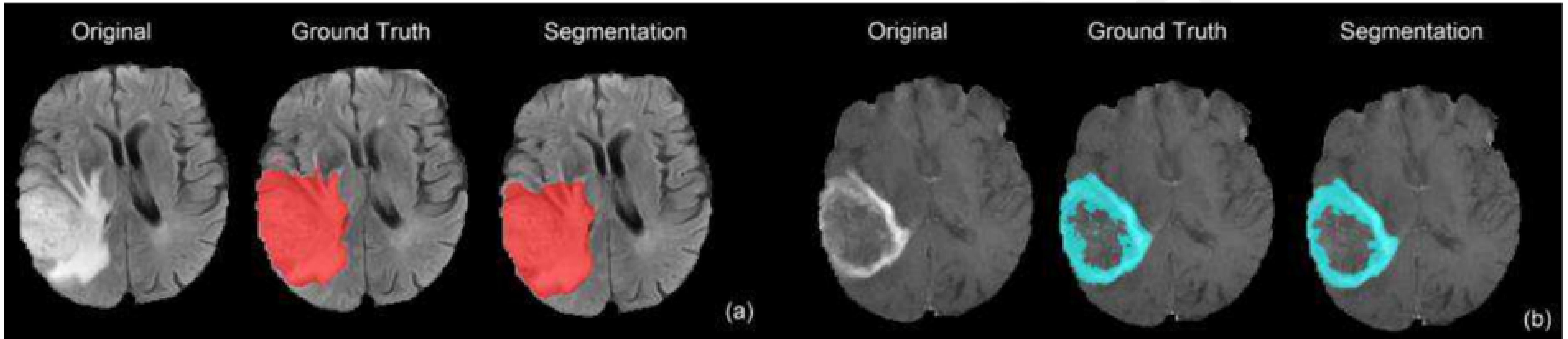


Predictions:  
 $H \times W$

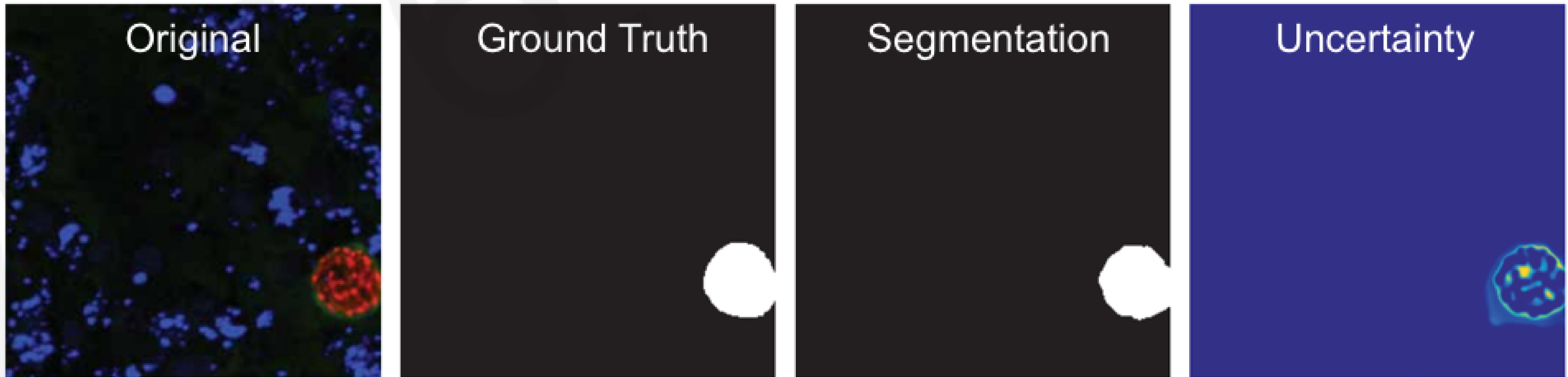
 `tf.keras.layers.Conv2DTranspose`

# Semantic Segmentation: Biomedical Image Analysis

Brain Tumors  
Dong+ MIUA 2017.



Malaria Infection  
Soleimany+ arXiv 2019.



# Self-Driving Cars: Navigation from Visual Perception

Raw Perception

$I$

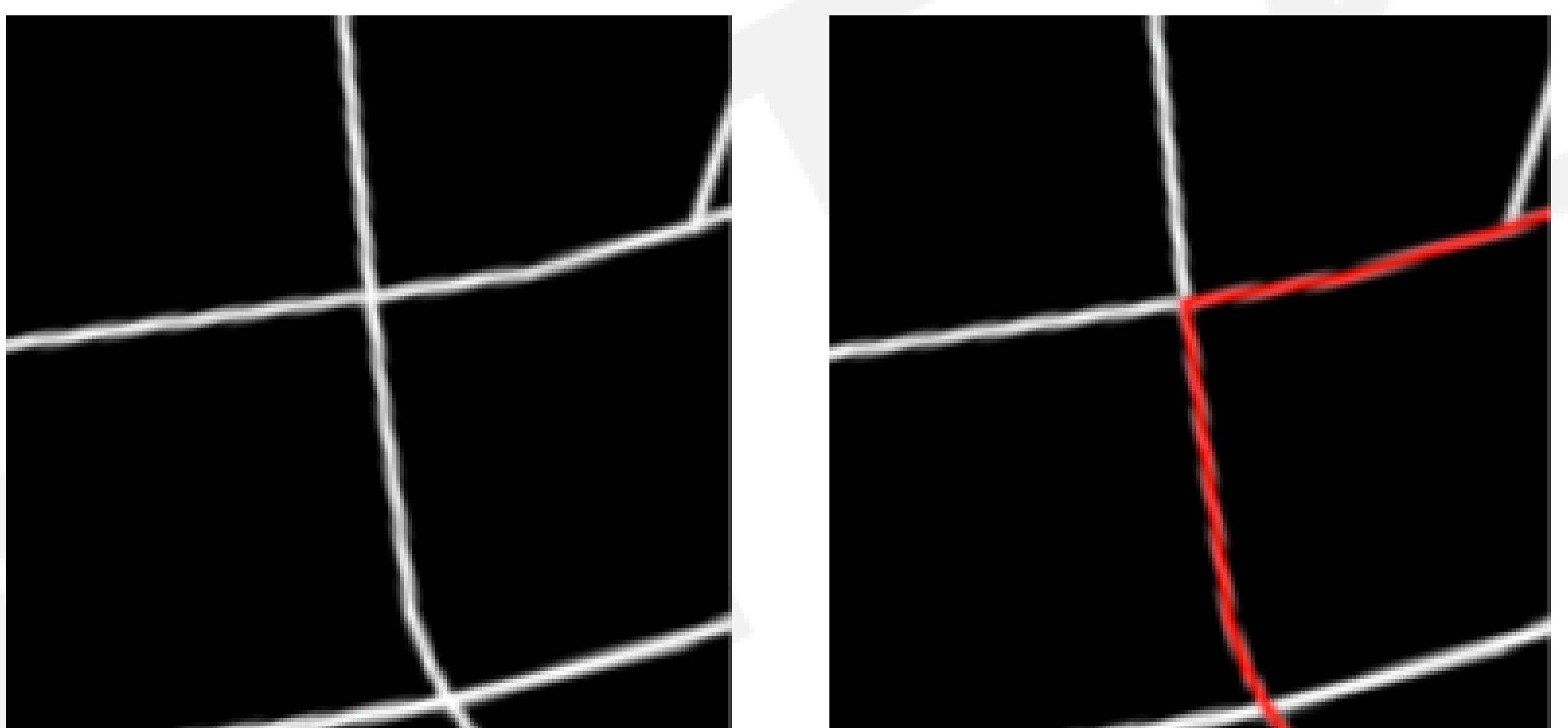
(ex. camera)



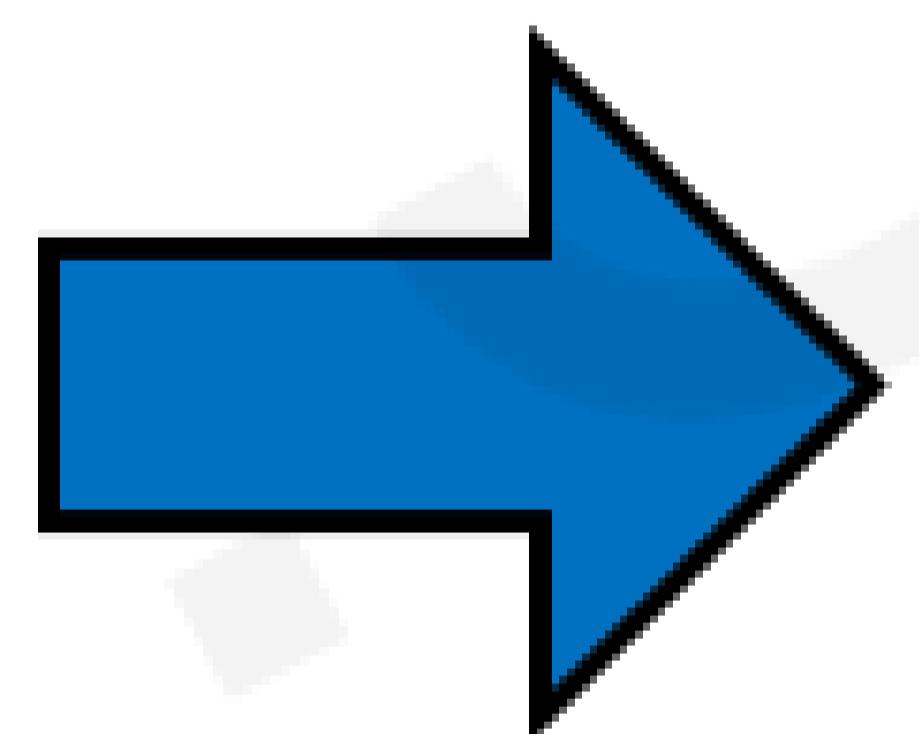
Coarse Maps

$M$

(ex. GPS)

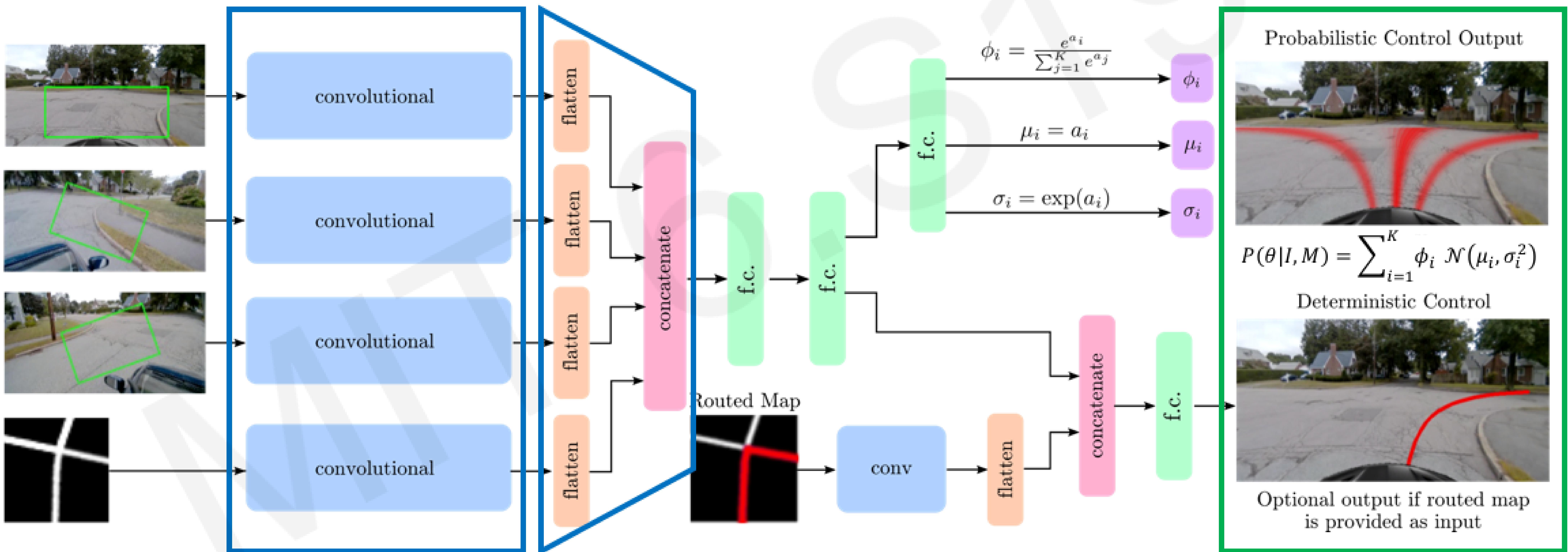


Possible Control Commands



# End-to-End Framework for Autonomous Navigation

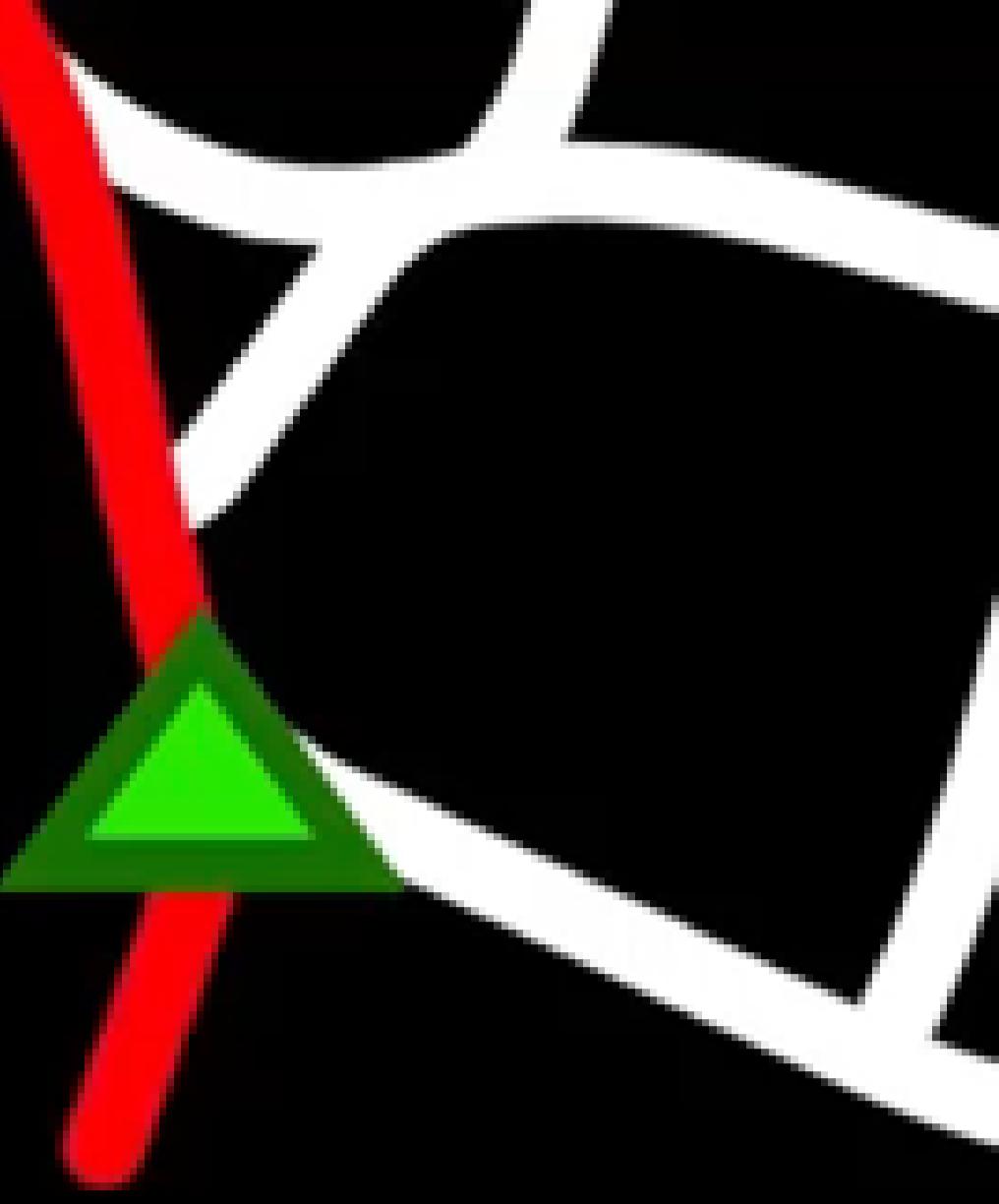
Entire model is trained end-to-end **without any human labelling or annotations**



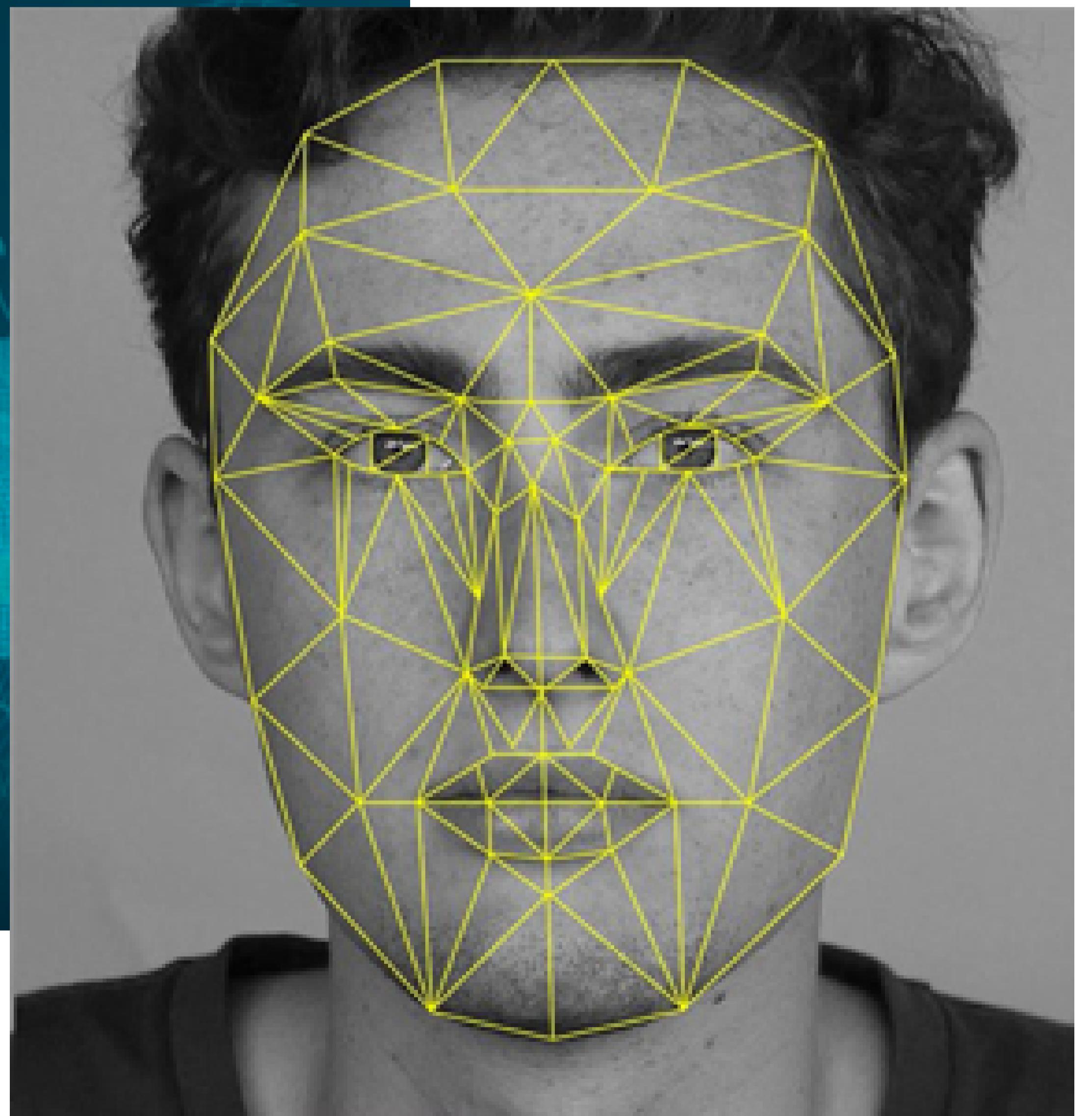
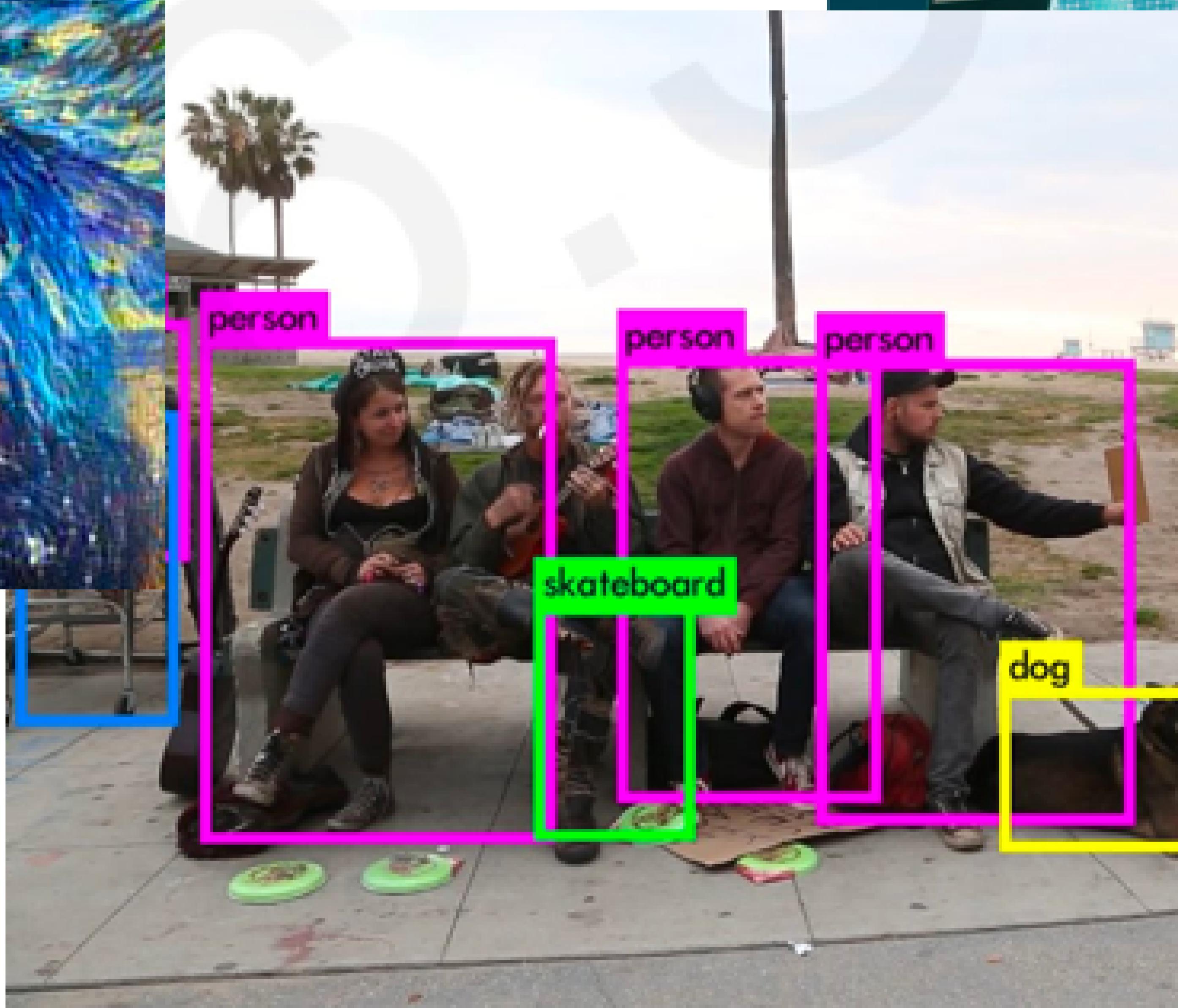
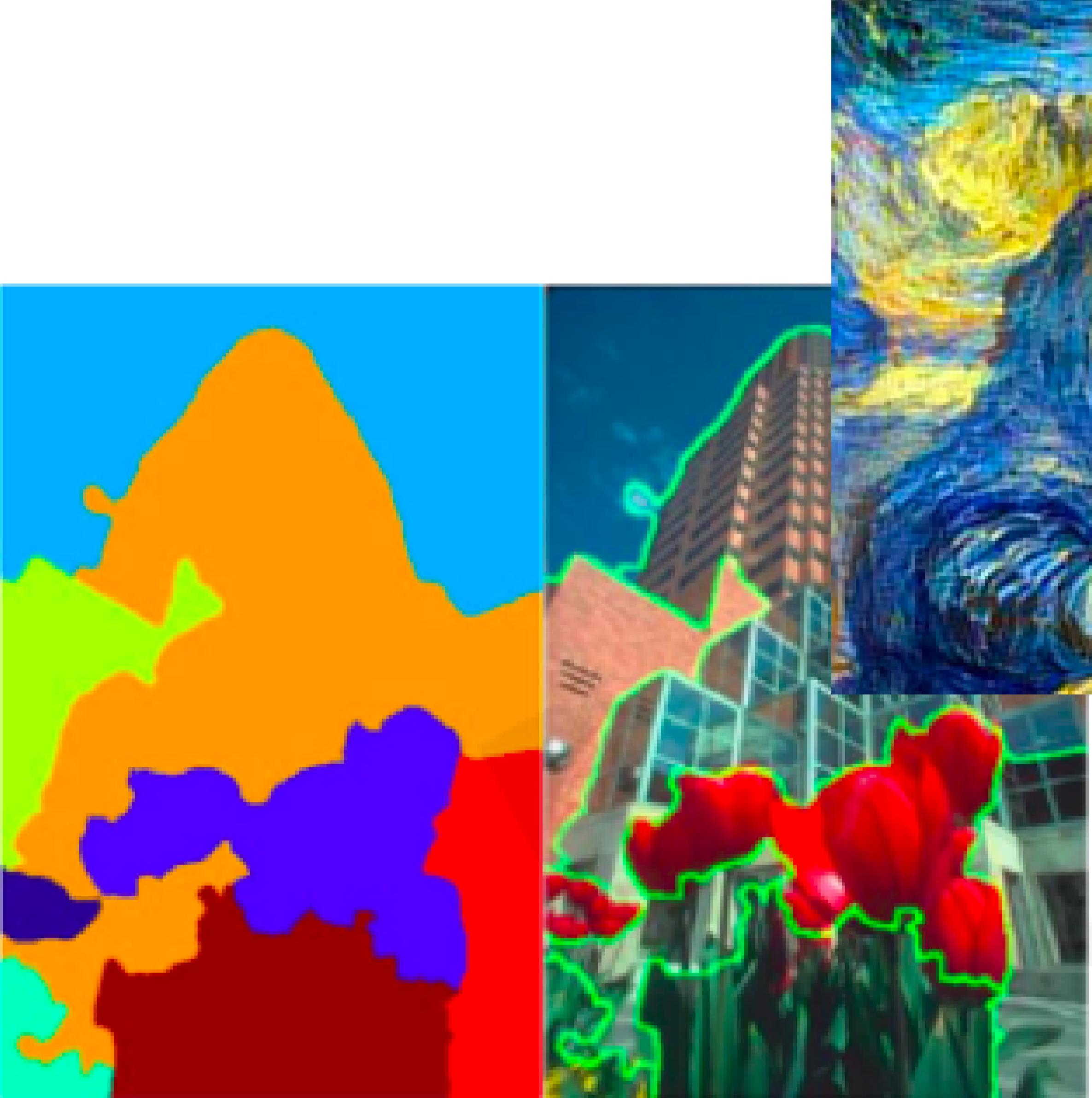
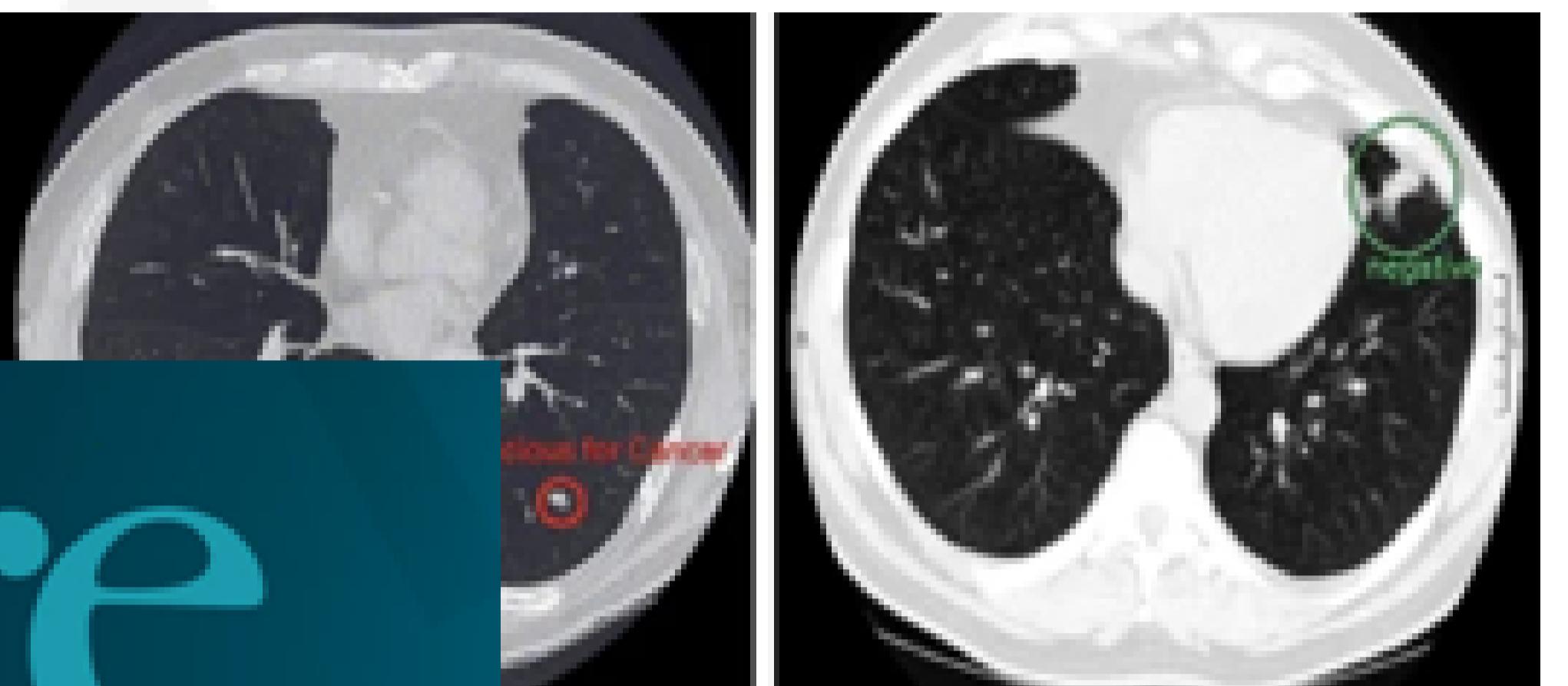
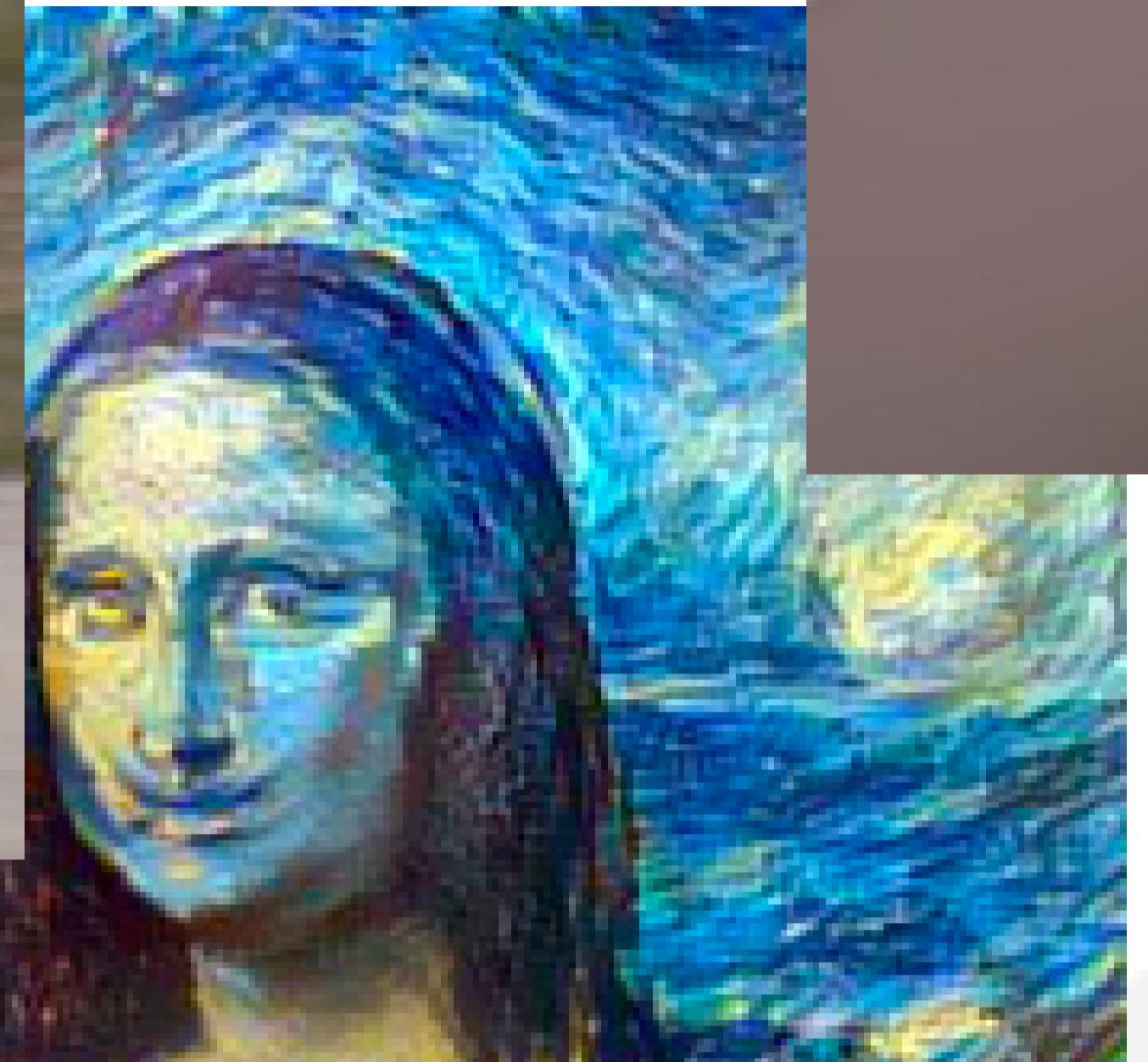


# Auto ON

## Navigation and Localization



# Deep Learning for Computer Vision: Impact

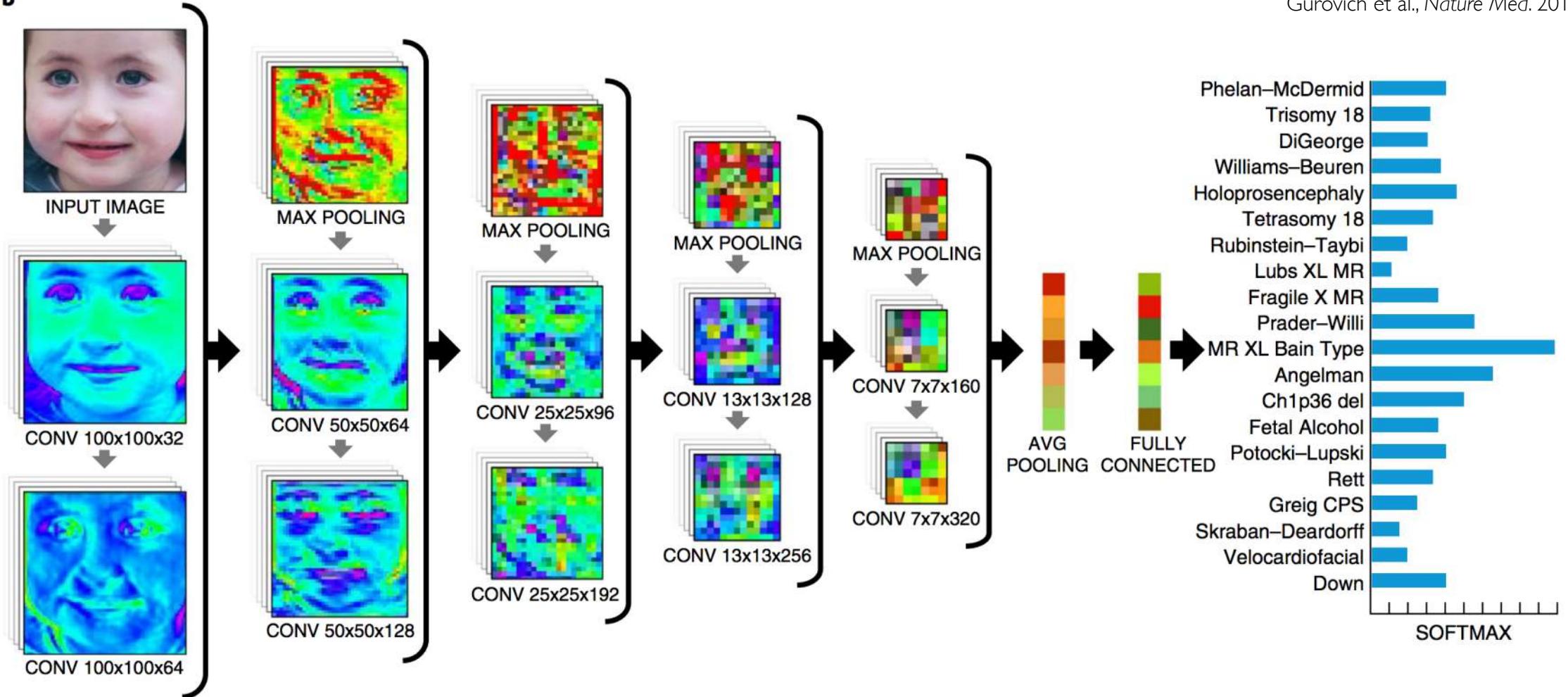


# Impact: Healthcare

Identifying facial phenotypes of genetic disorders using deep learning

Gurovich et al., *Nature Med.* 2019

b



# Deep Learning for Computer Vision: Summary

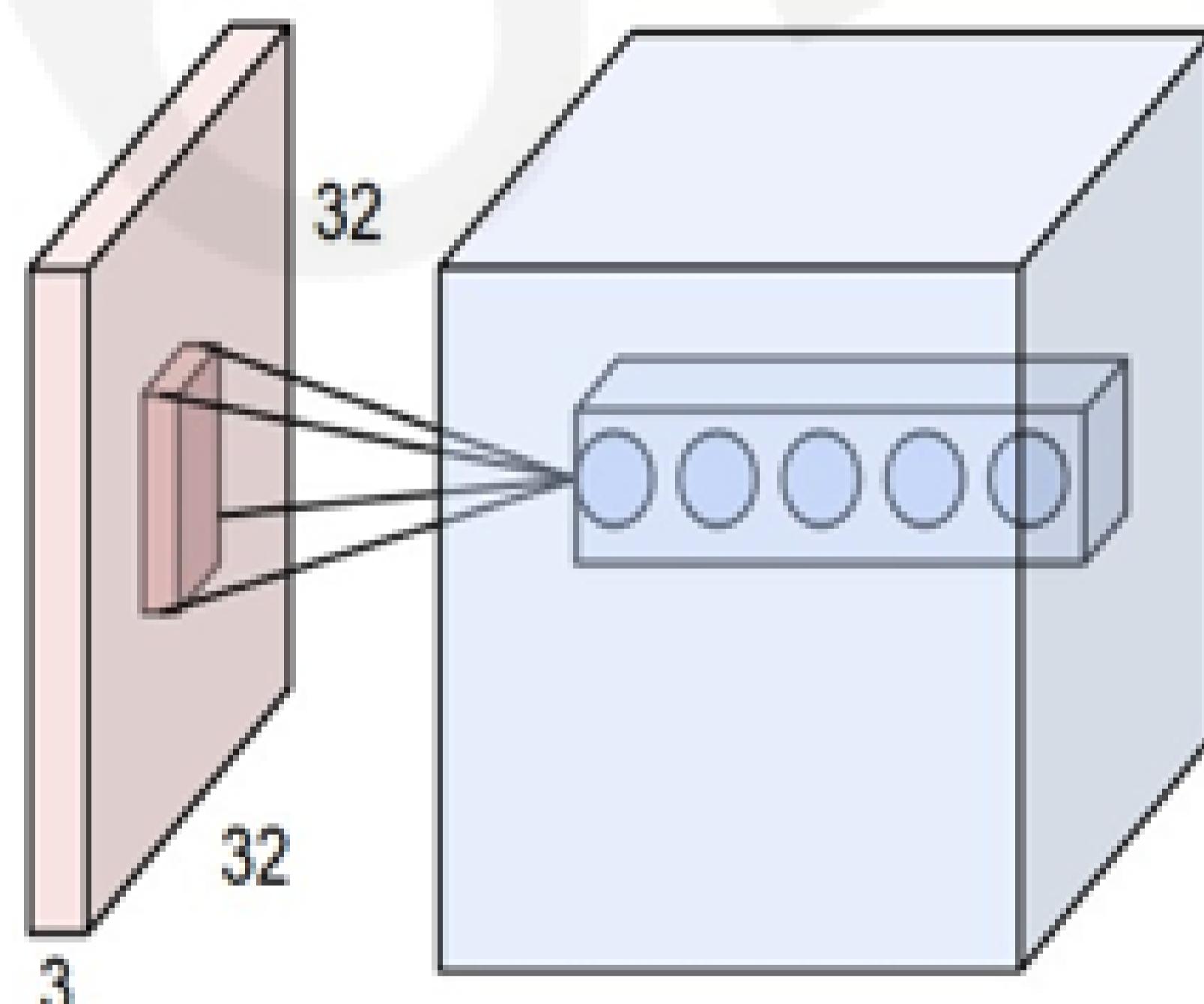
## Foundations

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



## CNNs

- CNN architecture
- Application to classification
- ImageNet



## Applications

- Segmentation, image captioning, control
- Security, medicine, robotics

