PERCEPTRON & CNN

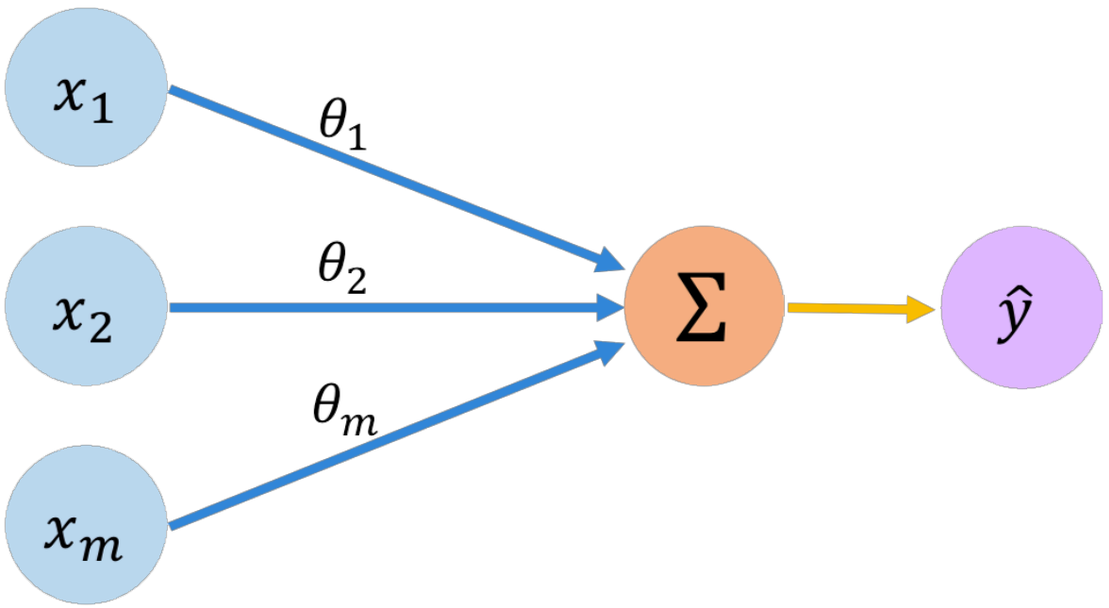
INTRODUCTION TO NEURAL NETWORK

AGENDA

- Introduction to Perceptron
- Activation function
- Convolution Layer
- Pooling Layer
- Convolutional Neural Network (CNN) and Softmax Output
- Object Detection
- *You Only Look Once* (YOLO)

INTRODUCTION TO PERCEPTRON

- A single node with multiple inputs but one output



$$\hat{y} = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_m x_m = \sum_{i=1}^{m} \theta_i x_i$$

$$\vec{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix}, \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

$$\hat{y} = \vec{\theta} \cdot \vec{x} = \Theta^T \mathbf{x}$$

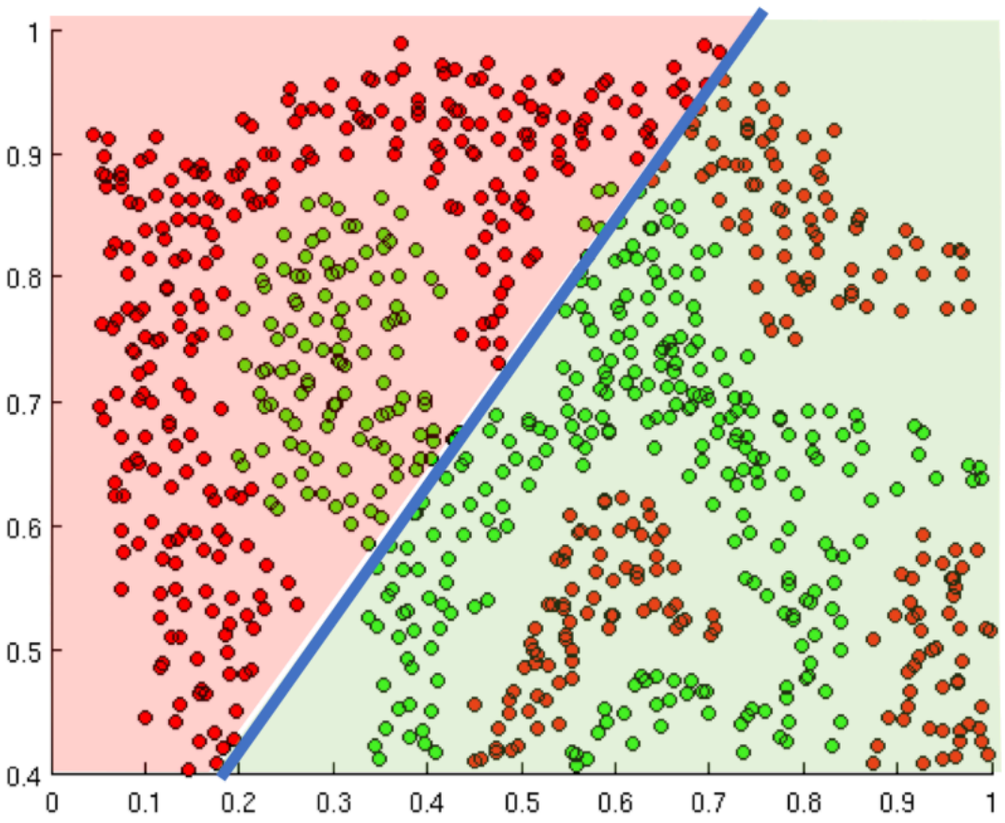
With bias: $\hat{y} = b + \sum_{i=1}^{m} \theta_i x_i = \Theta^T \mathbf{x} + b$

The the sum is a linear function (in multi-dimension) of the input. Problem?

WHAT IS THE PROBLEM WITH LINEAR?

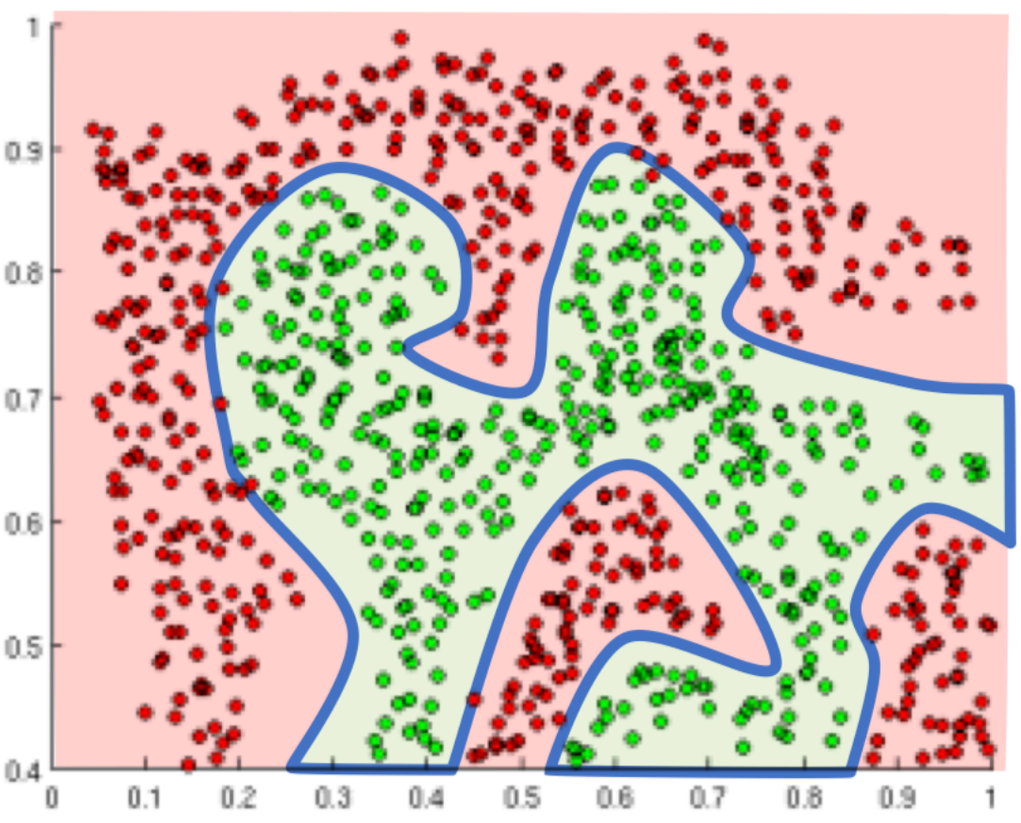
Given a model like this, the task is try to separate the green dots and the red dots.

With linear function:



A linear function produces linear decision no matter what the network size is.

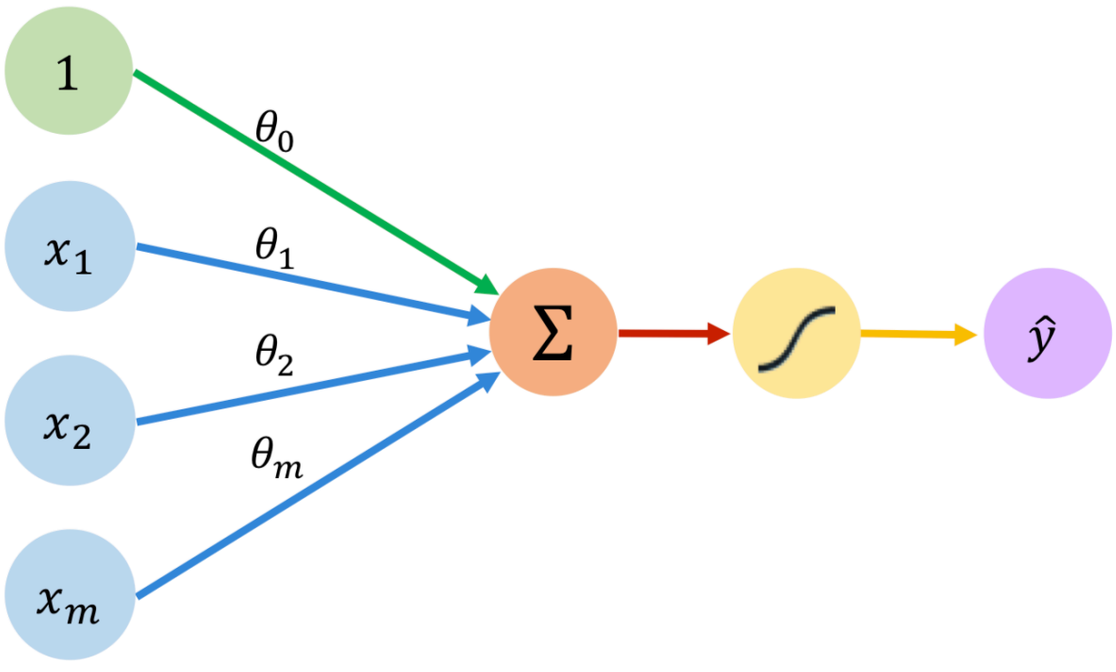
If it is non-linear:



Non-linearities allow us to approximate arbitrarily complex functions.

ADD NON-LINEARITY TO THE MODEL

- Add non-linearity by adding a function g to the equation



$$\hat{y} = g\left(b + \sum_{i=1}^{m} \theta_i x_i\right) = g\left(\Theta^T \mathbf{x} + b\right)$$

$$b = \theta_0$$

$$\overrightarrow{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \end{bmatrix}, \overrightarrow{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

$$\hat{y} = g\left(\sum_{i=0}^{m} \theta_i x_i\right) = g\left(\Theta^T \mathbf{x}\right)$$

NON-LEARN FUNCTION

Linear combination of inputs

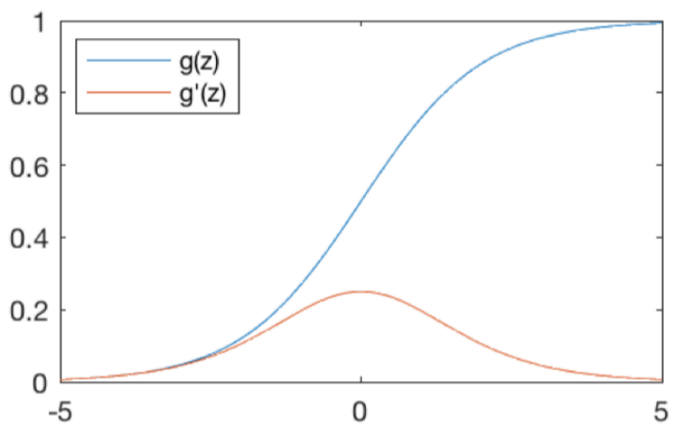
Output

$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i \, \theta_i\right)$$

Non-linear activation function

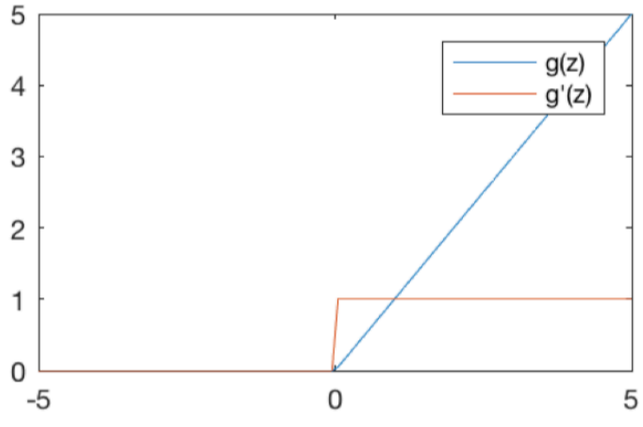
Bias

Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

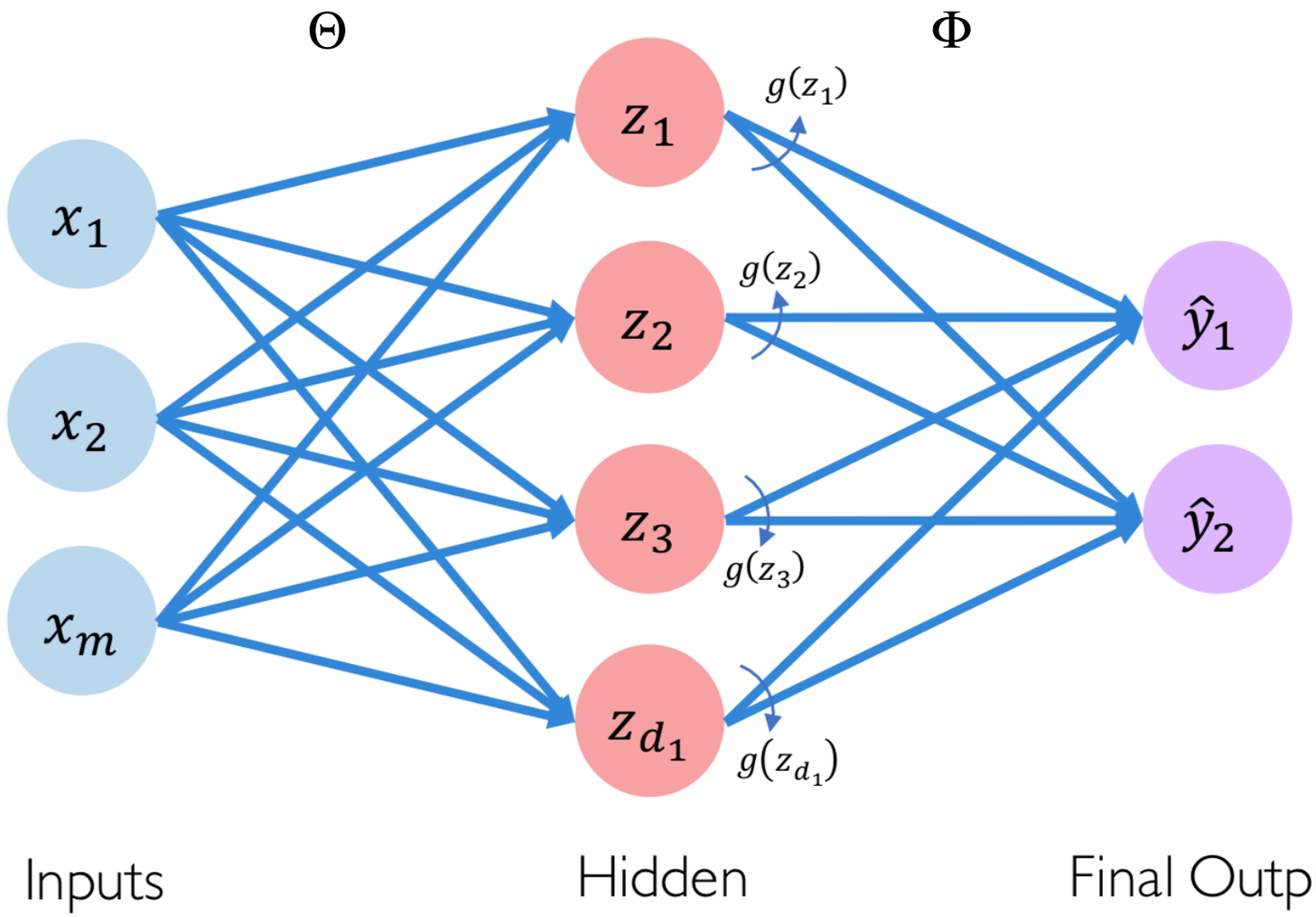
Rectified Linear Unit (ReLU)



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

PERCEPTRON NEURAL NETWORK (FULLY CONNECTED LAYER)

- We can put many of those perceptrons into a single layer, forming a simple neural network.



$$\Theta = \begin{bmatrix} \theta_{0,0} & \theta_{0,1} & \cdots & \theta_{0,m} \\ \theta_{1,0} & \theta_{1,1} & \cdots & \theta_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{d,0} & \theta_{d,1} & \cdots & \theta_{d,m} \end{bmatrix}$$

Where each row i represents the weights connecting inputs to z_i.

$$z_i = \Theta_i^T \mathbf{x}$$

$$\hat{y}_i = \Phi_i^T g(\mathbf{z})$$

IMAGE AS INPUT

- 800x600 RGB image as input
- Input layer : first hidden layer = 4:1
- 518,400,000,000 parameters
- 3862.38 GB of memory

CONVOLUTION

- Each neurone in the hidden layer is only connected to a n-by-n patch from the previous layer
- Achieve this by ***convolution***
- A ***filter*** is applied to each n-by-n patch of input

CONVOLUTION

0	0	1	0	0	1	1	0	0
1	1	1	0	1	1	0	0	0
0	1	0	1	0	1	1	0	0
1	1	0	0	1	0	1	1	1
0	0	1	1	0	0	0	1	0
1	0	1	1	1	1	0 *0	0 *0	1 *0
0	1	1	1	1	1	1 *1	0 *0	1 *1
1	1	1	1	0	0	1 *0	0 *1	1 *-1

Input

0	0	0
1	0	1
0	1	-1

Filter

3	0	3	0	1	2	0
1	2	-1	3	0	1	1
0	1	2	0	2	0	3
0	1	1	1	1	1	-1
2	1	2	2	1	2	0
1	2	3	2	1	2	1

Output

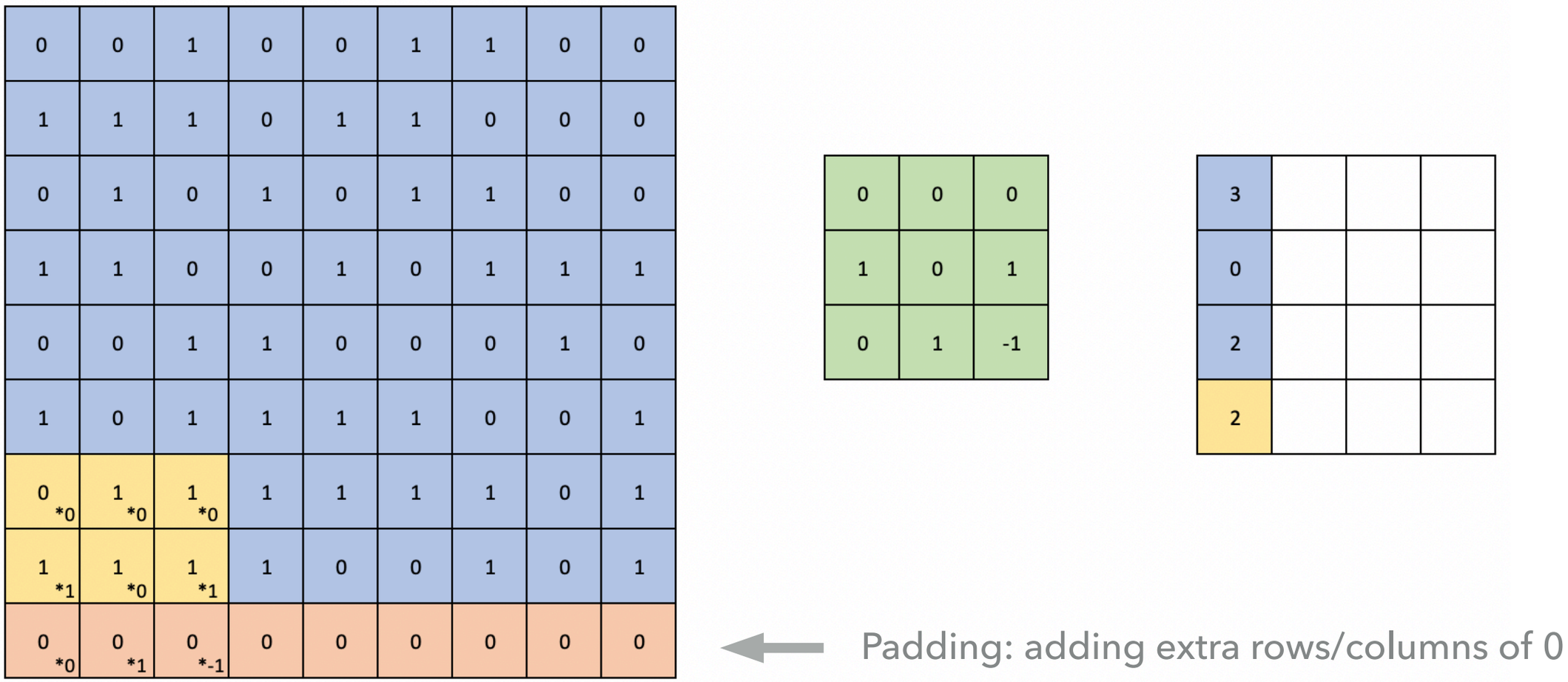
This process is also called feature extraction.

What if your filter is very big and you want to slide it by more than 1 unit each time?

CONVOLUTION – STRIDE

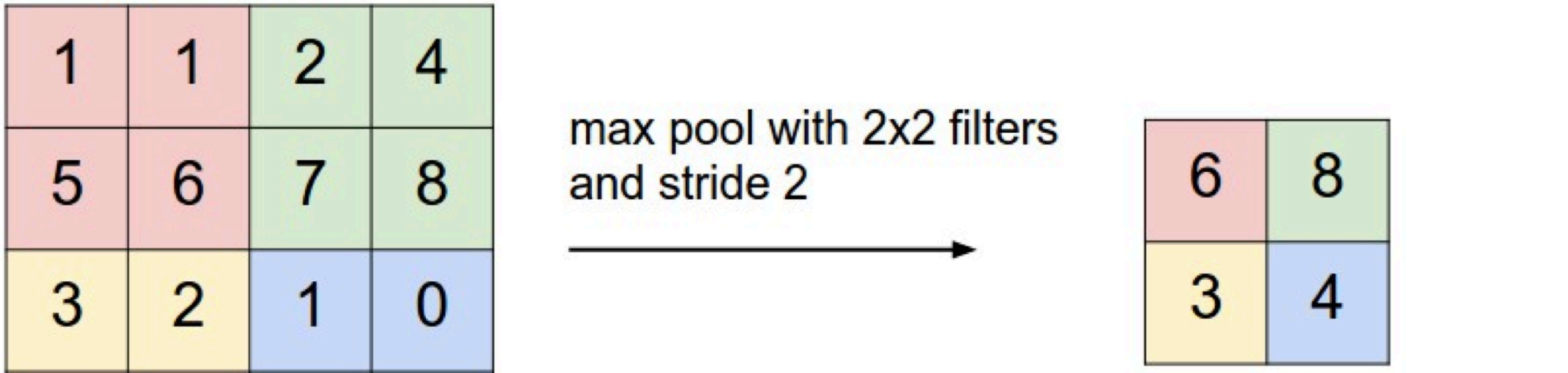
- Stride is how much you want to move the filter each time

  (stride of 2 means you want to move the filter by 2 units every time instead of 1)



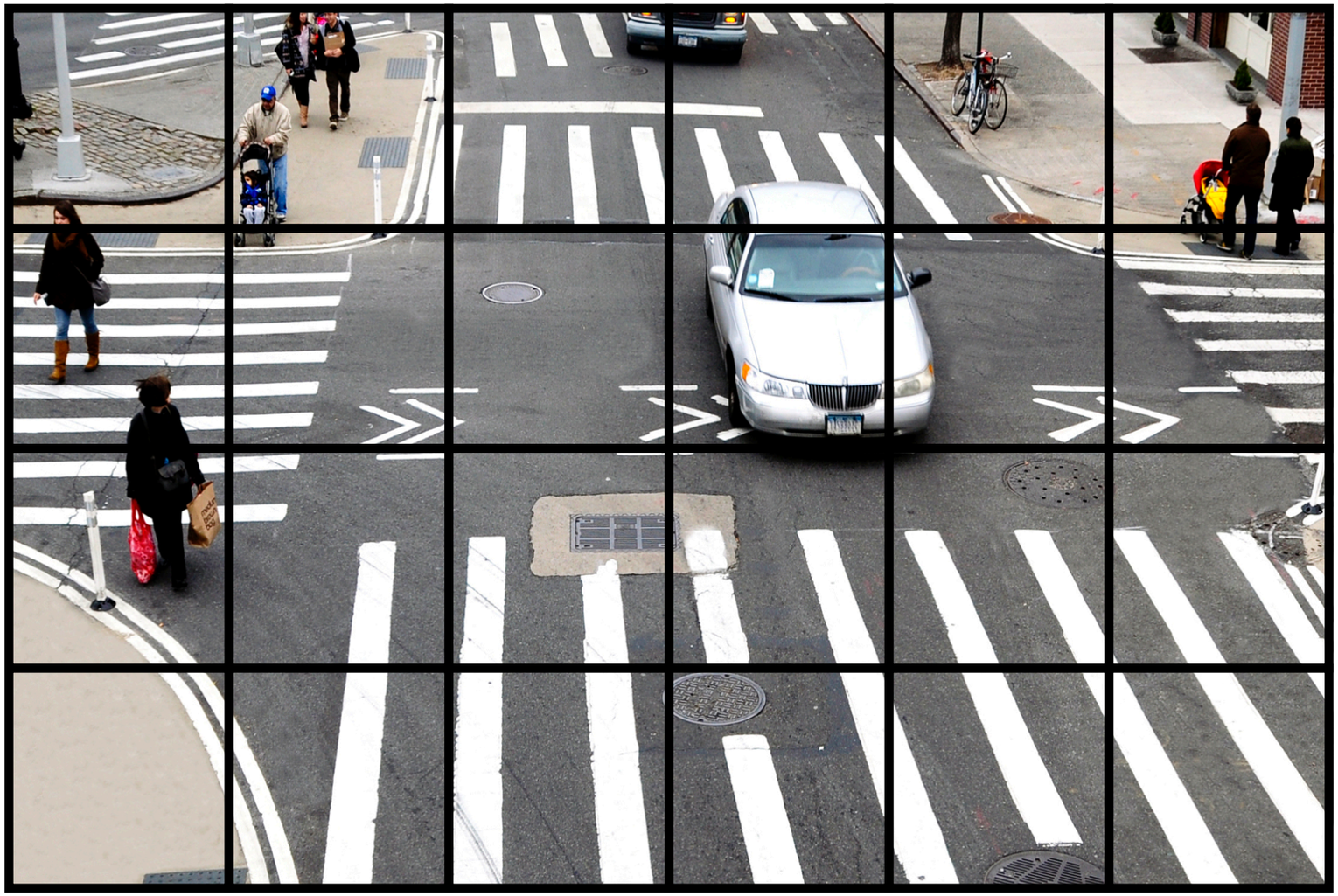
POOLING (MAXPOOL)

- A pooling layer is usually added after the convolutional layer to reduce the dimension while keep “as much information as possible”



1) Reduced dimensionality
2) Spatial invariance

WHY POOLING?



Extracting feature: car

How much “car” is in each grid?

0.1	0.0	0.3	1.6	0.0	0.0
0.0	0.0	0.1	5.2	1.1	0.0
0.0	0.0	0.1	0.2	0.0	0.1
0.0	0.1	0.0	0.0	0.0	0.0

After Maxpool

0.1	5.2	1.1
0.1	0.2	0.1

POOLING LAYER

- Features in the adjacent area “usually” belong to the same bigger feature (object)
- Only the more complete one needs to be saved (since convolution can analyze same “pixel” multiple times)

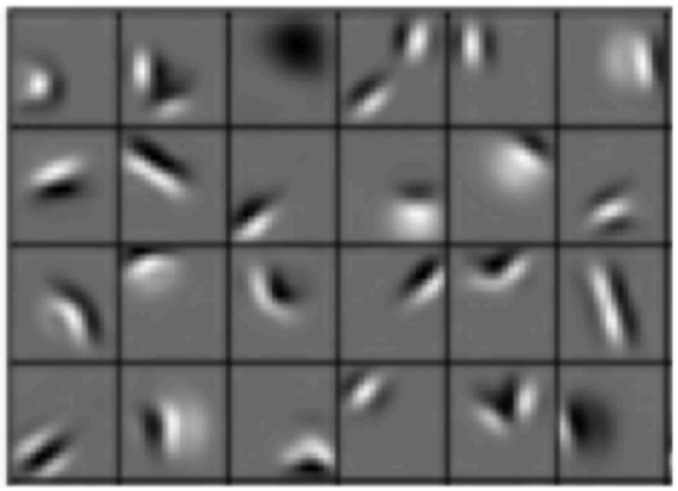
WHY FEATURE EXTRACTION? WHAT DOES IT DO?

Original

"Strong" Edge Detect

FEATURES

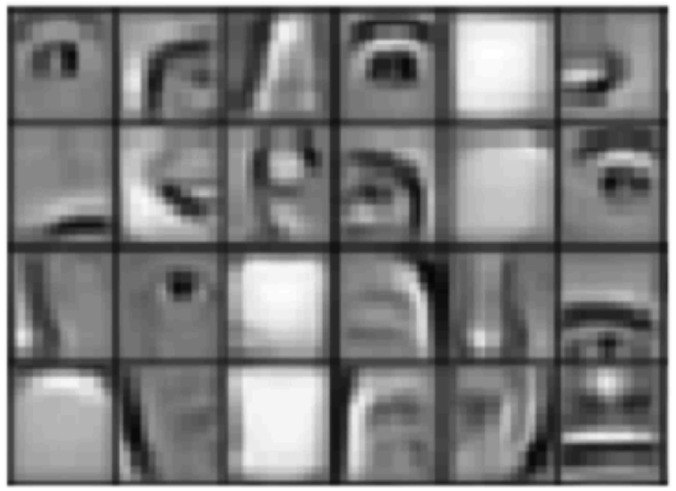
Low level features



Edges, dark spots

Conv Layer 1

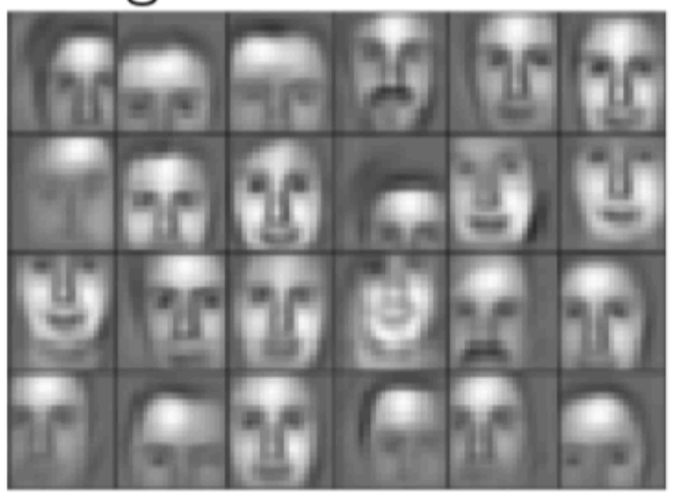
Mid level features



Eyes, ears, nose

Conv Layer 2

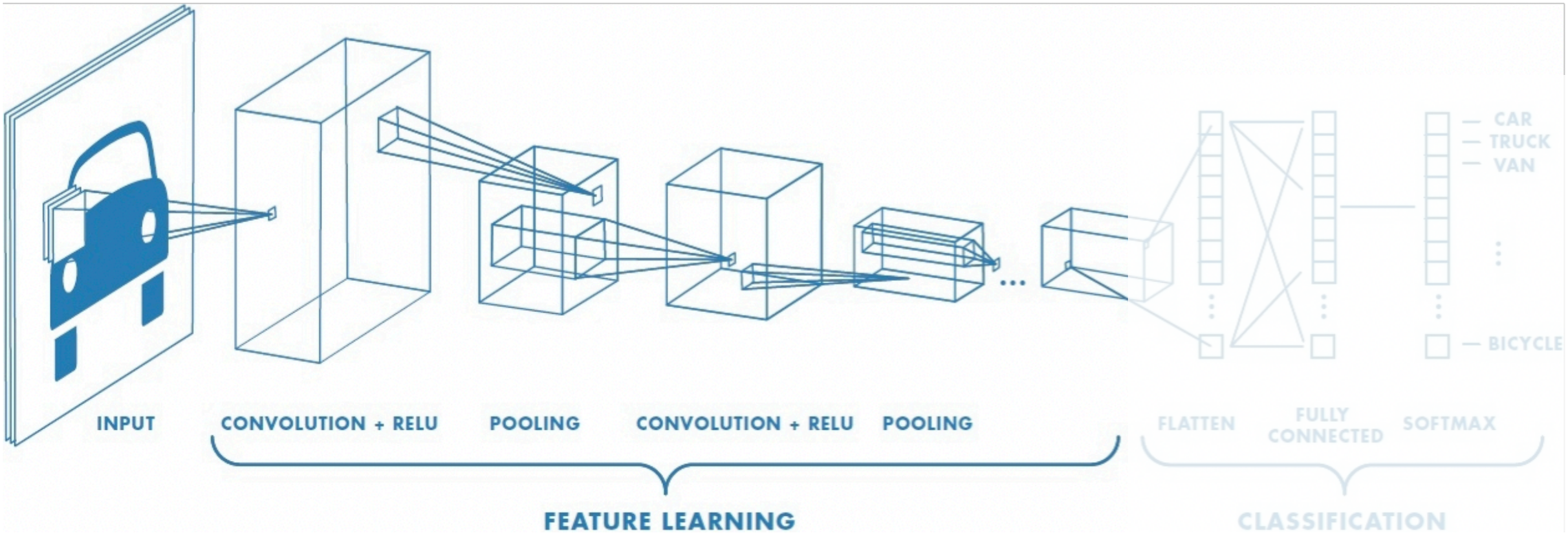
High level features



Facial structure

Conv Layer 3

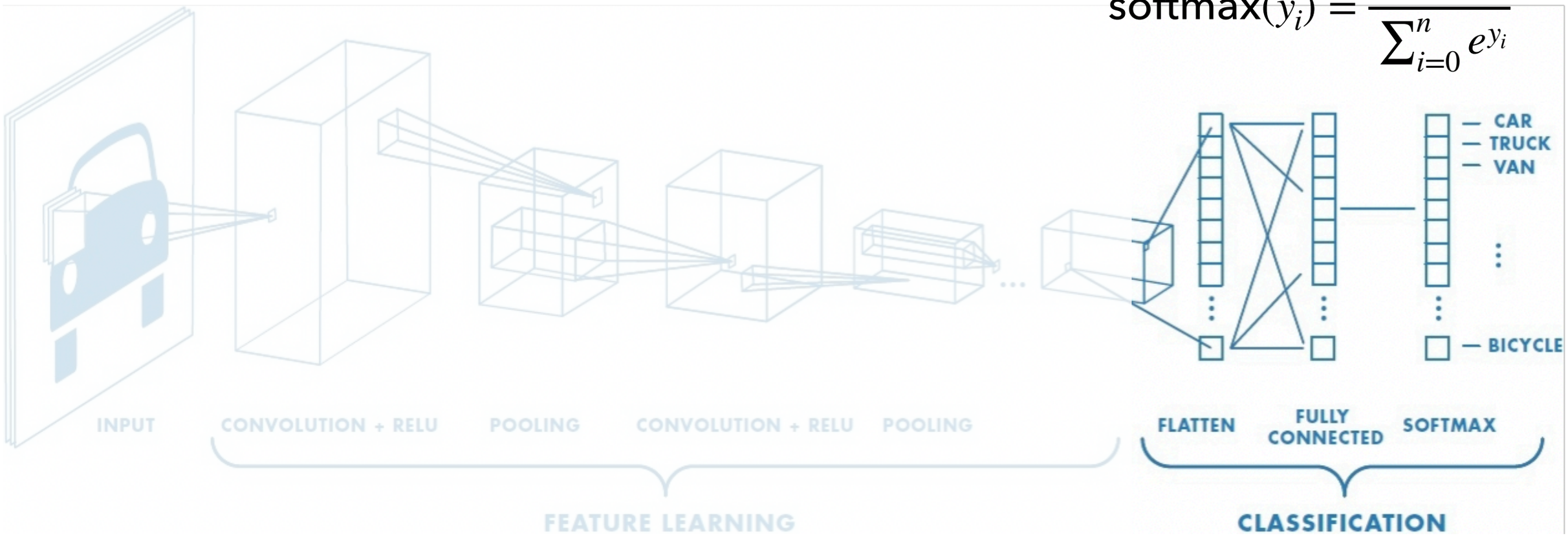
CONVOLUTIONAL NEURAL NETWORK



1. Learn features in input through **convolution**
2. Introduce **non-linearity** through activation function
3. Reduce dimensionality and preserve spatial invariance with pooling

CONVOLUTIONAL NEURAL NETWORK

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_{i=0}^{n} e^{y_i}}$$

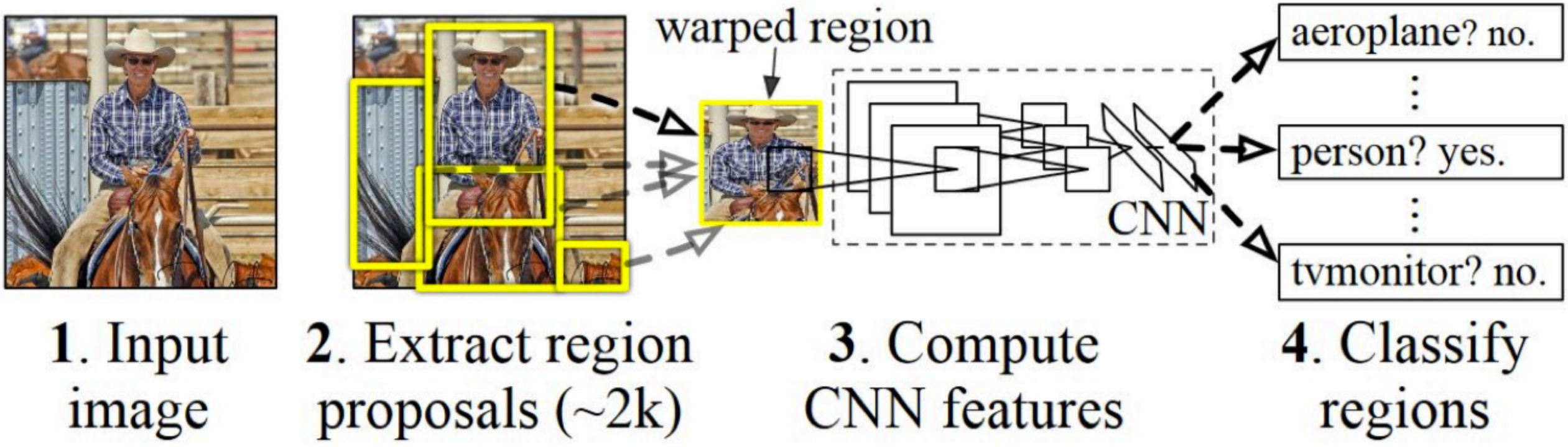


1. Use high-level feature outputted from CONV and POOL layers as input

2. Use fully connected layers to classify input image

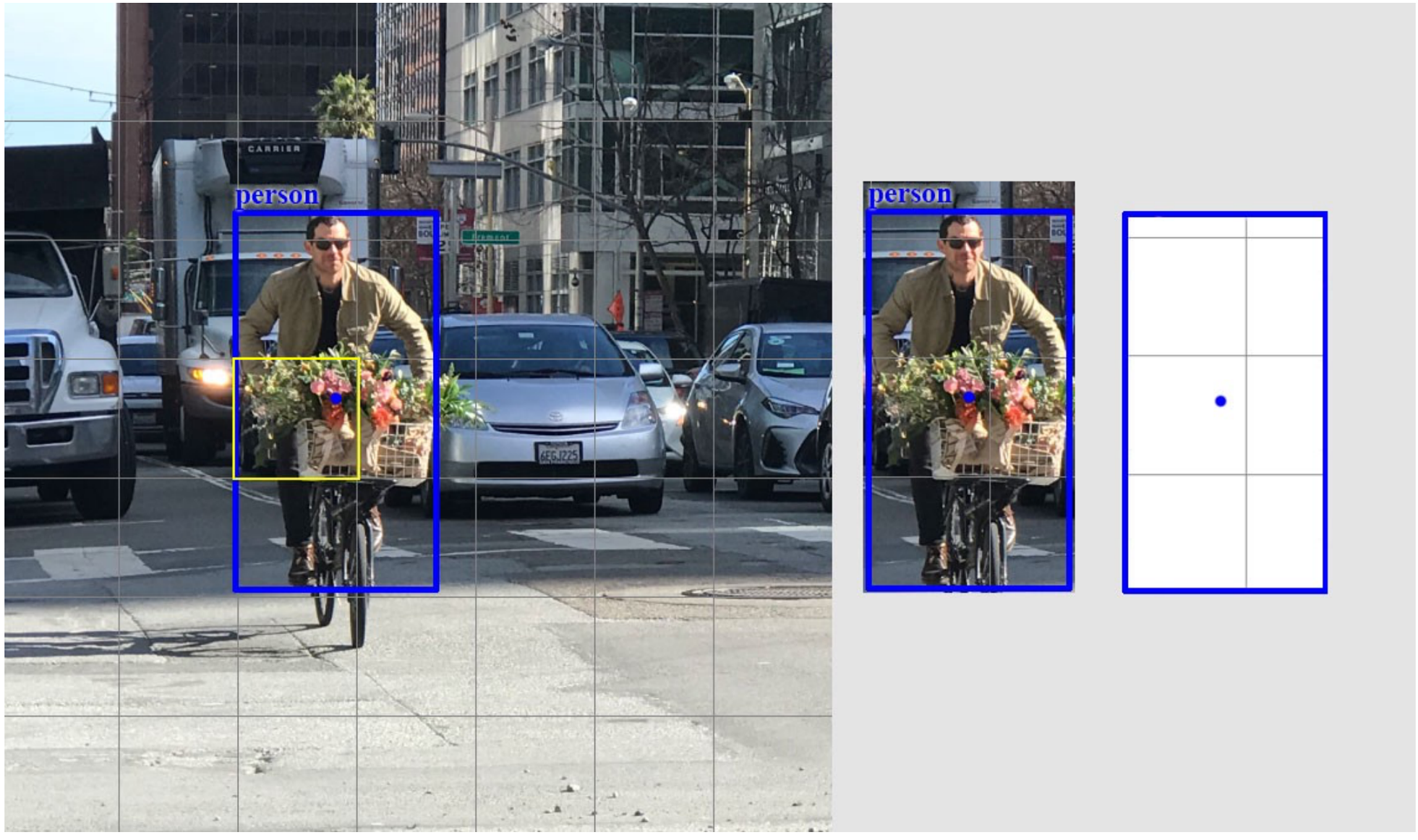
3. Express outputs as probability of image belonging to a particular class out of n classes

OBJECT DETECTION



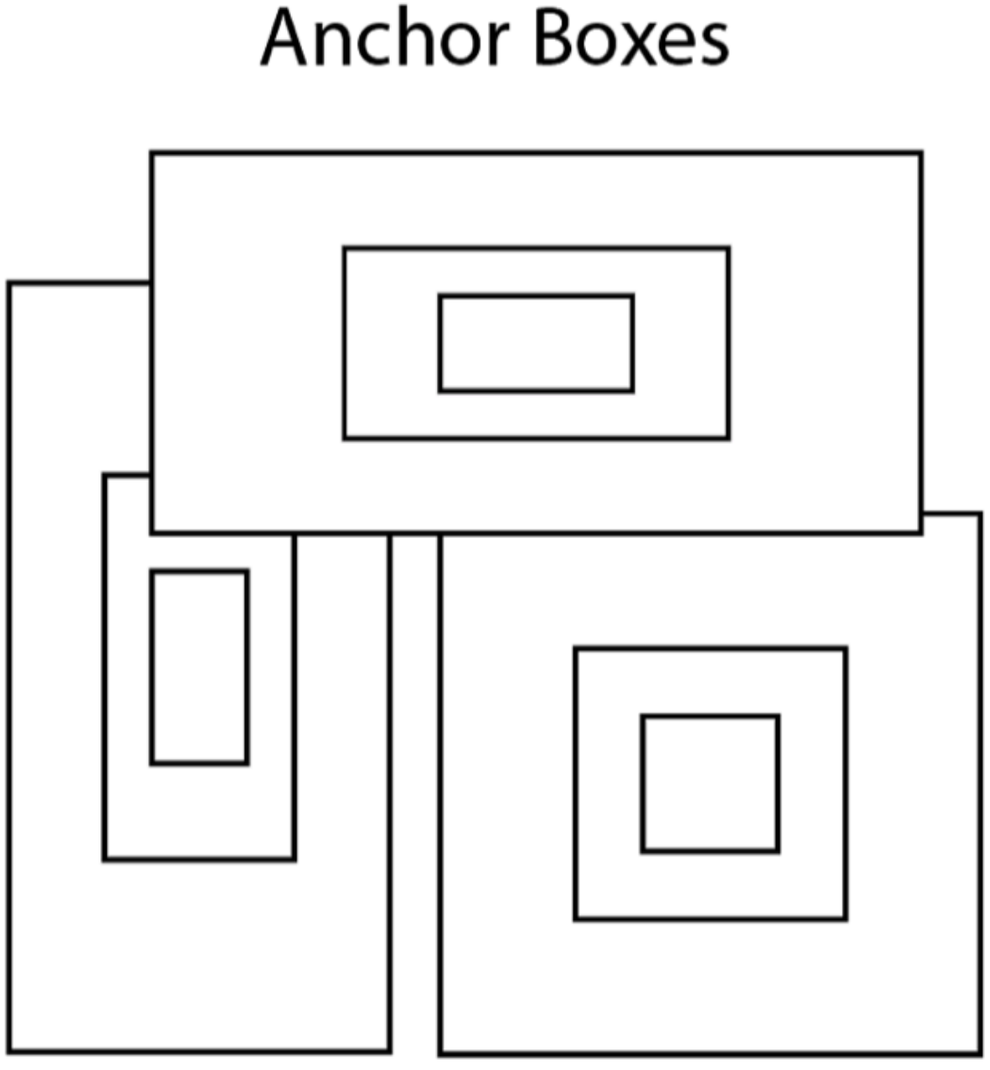
R-CNN: Find the region that we think have objects. Use CNN to classify.

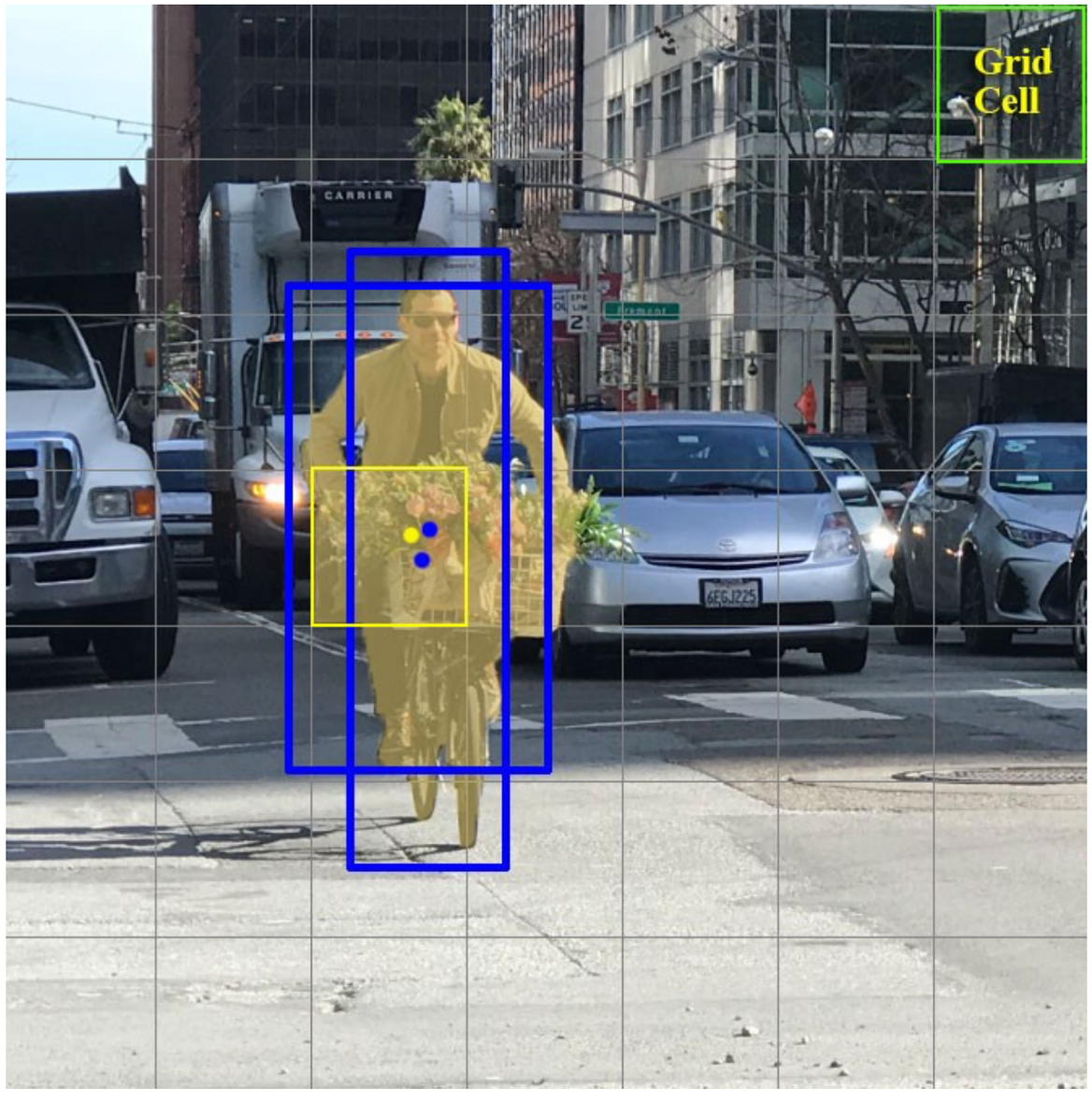
YOU ONLY LOOK ONCE (YOLO)



1. Divide into S by S grid, each grid predicts one object

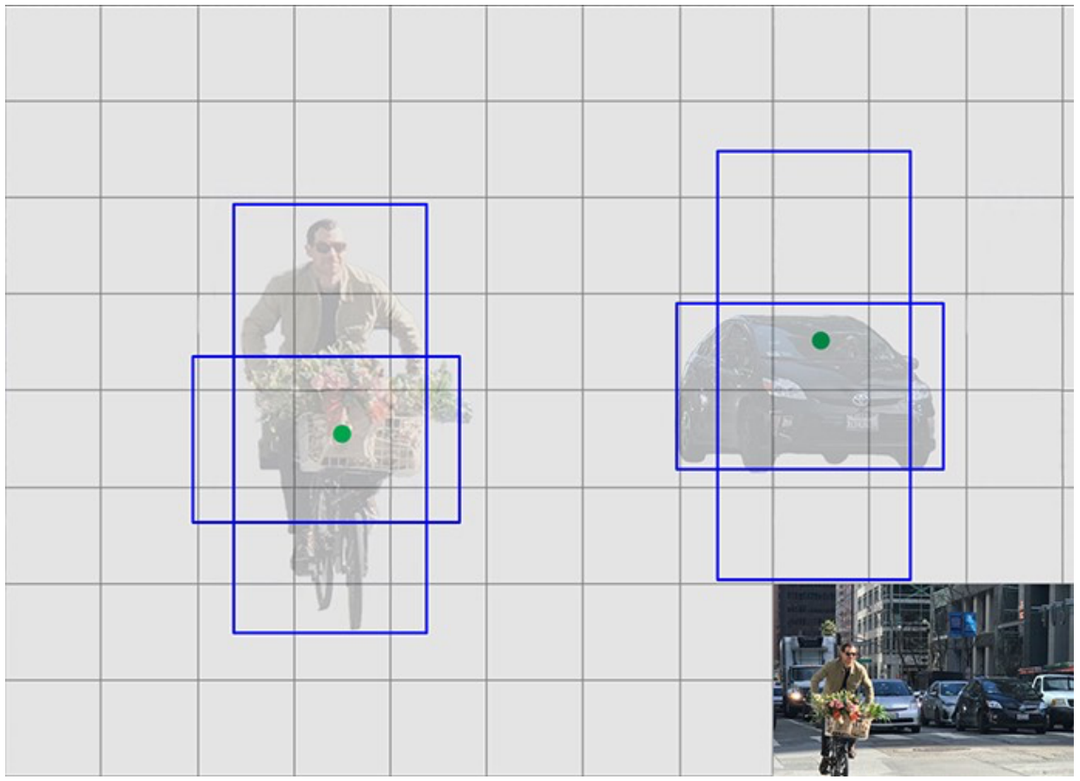
YOLO





Apply Anchor boxes to see if there is an object.

YOLO



1. What is the probability that there is something in this box?

$$\text{box confidence score} \equiv P_r(object) \cdot IoU$$

2. Given that there is an object in the box, what is the probability of a particular class?

$$\text{conditional class probability} \equiv P_r(class_i \mid object)$$

3. Combine 1 and 2: what is the confidence that an object belong to a particular class is here?

$$\text{class confidence score} \equiv P_r(class_i) \cdot IoU$$