



# RECONOCIMIENTO DE PATRONES EN IMÁGENES TICS 585

FACULTAD DE INGENIERÍA Y CIENCIAS  
UNIVERSIDAD ADOLFO IBÁÑEZ

SEGUNDO SEMESTRE 2021

PROFESOR: MIGUEL CARRASCO

INTRODUCCIÓN IMÁGENES

## ■ Supervisados

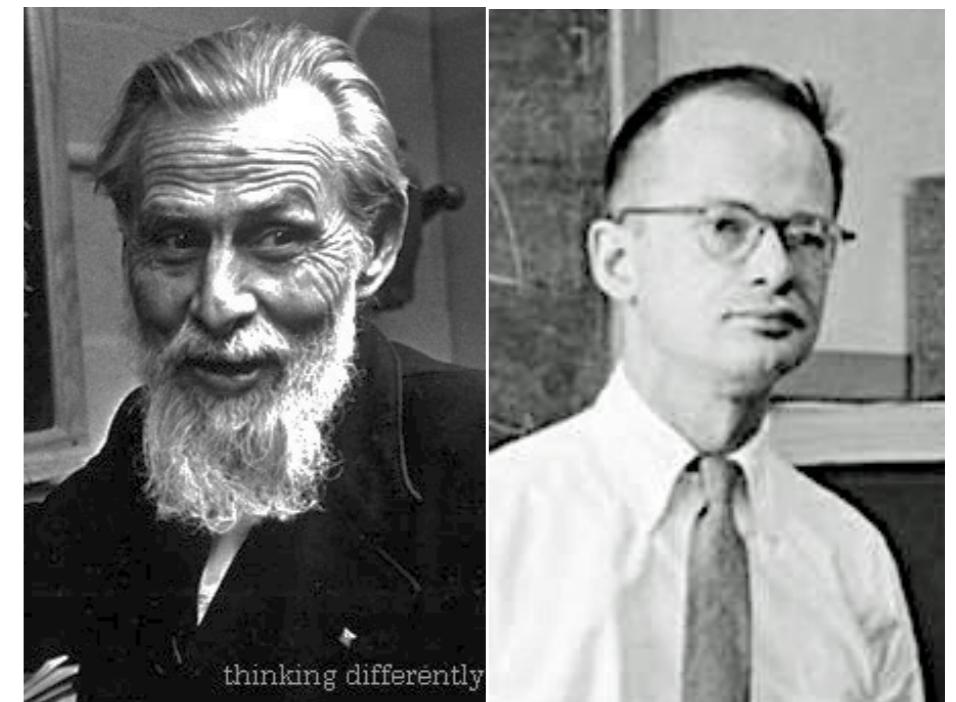


### ■ Redes Neuronales (Neural Networks)

El sistema neuronal humano está compuesto por miles de millones de neuronas interconectadas, las cuales ante un estímulo producen una salida.

Gracias al mecanismo de comunicación presente en las neuronas, somos capaces de aprender, reaccionar frente estímulos, sentir, lo cual es algo propio en todos los seres animales.

- El estudio sobre el funcionamiento de las neuronas comenzó desde hacia muchas décadas, sin embargo, no fue hasta el año 1943 cuando el neuropsicólogo *Warren McCulloch* y el matemático *Walter Pitts* fueron los primeros en proponer y construir una neurona eléctrica.

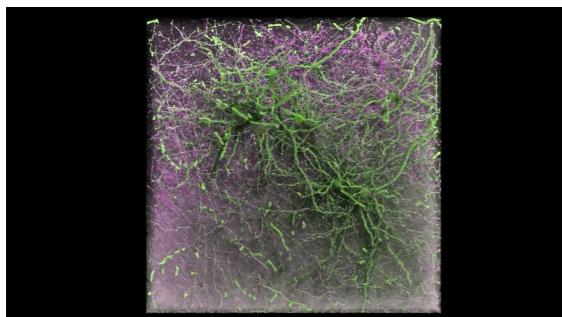


Warren McCulloch

Walter Pitts

## ■ Supervisados

- kNN
- Mahalanobis
- LDA
- Árboles de decisión
- Bayesiano
- Redes Neuronales



### ■ Redes Neuronales (Neural Networks)

El sistema neuronal humano está compuesto por miles de millones de neuronas interconectadas, las cuales ante un estímulo producen una salida.



*fuente:* <https://www.youtube.com/watch?v=ZWACm6BkDVo>

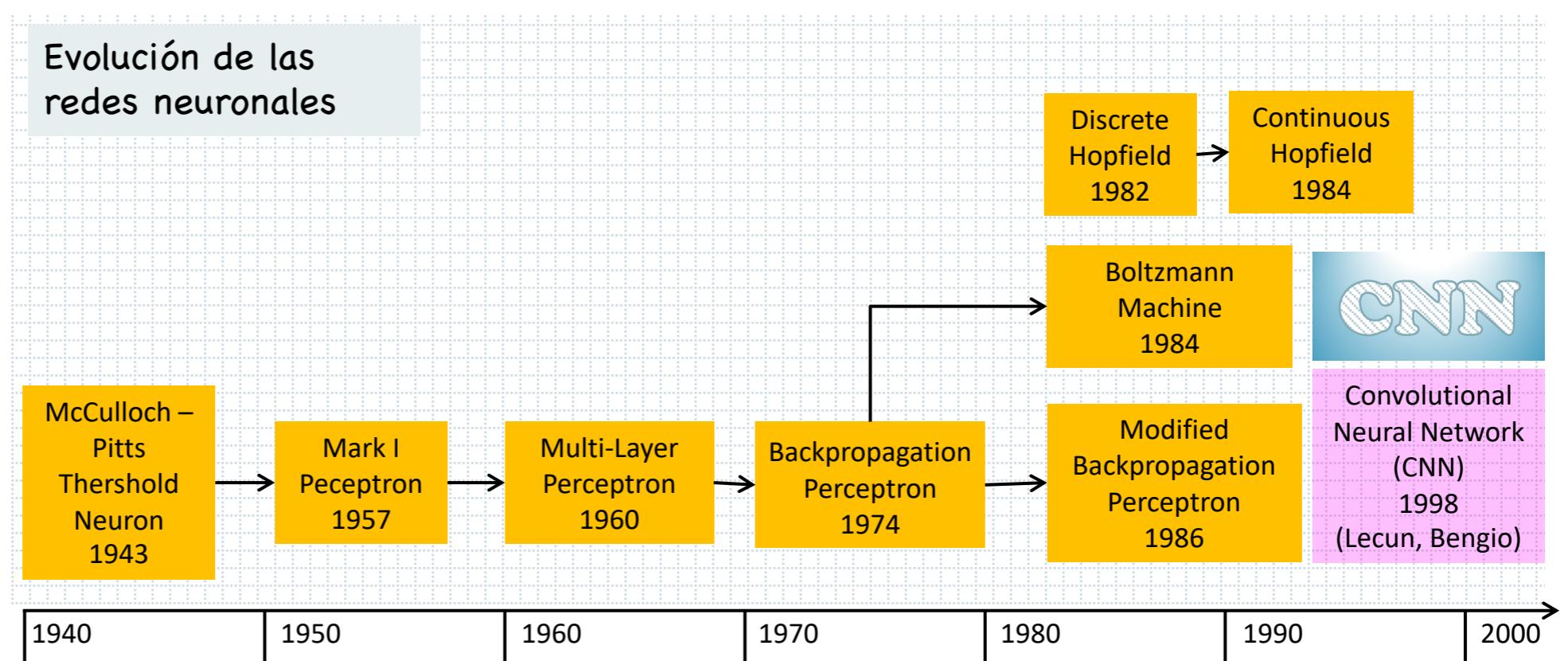
*fuente:* [https://www.youtube.com/watch?v=morHZ\\_RDdyQ](https://www.youtube.com/watch?v=morHZ_RDdyQ)

## ■ Supervisados



### ■ Redes Neuronales (Artificial Neural Networks, ANN)

En 1959 *Rosenblatt* presentó la primera red neuronal (*perceptrón*) la cual constituyó el primer gran impulso de las ANN en utilizar sistemas computacionales. No fue hasta 1986 cuando *Rumelhart, Hinton y Williams* presentan un algoritmo eficiente de aprendizaje *Backpropagation perceptron* el cual posibilitó el creciente desarrollo de las redes neuronales en la actualidad.



*fuente:* <http://koti.mbnet.fi/phodju/nenet/NeuralNetworks>

## ■ Supervisados



### ■ **Redes Neuronales ¿Cómo, cuando, donde?**

*¿Cuando ocuparlas?*

Las redes neuronales pueden ser empleadas para aprender funciones continuas, discretas, vectoriales en problemas de aprendizaje supervisado (con clases conocidas). Las relaciones internas de la red no es comprensible por el ser humano, por ello decimos que funcionan como **caja negra**.

*¿Donde se aplican?*

Han sido aplicadas en un gran número de áreas como el reconocimiento de patrones, aproximación de funciones, control y predicción. Entre las aplicaciones desarrolladas se mencionan algunas tales como:

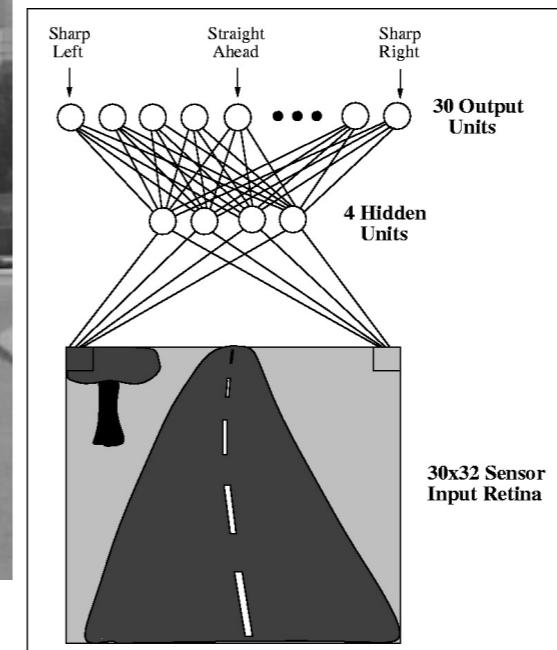
- Reconocimiento de caracteres
- Reconocimiento de la voz
- Reconocimiento de objetos
- Reconocimiento de situaciones anómalas, fraude, galaxias.
- Reconocimiento de fallas en objetos
- Reconocimiento de gestos, posturas, movimientos.
- Reconocimiento del camino

## ■ Supervisados



### ■ Redes Neuronales ¿Ejemplos?

Reconocimiento de autos y personas



The Autonomous Land Vehicle In a Neural Network (ALVINN), MIT



Ejemplos de aplicaciones con redes neuronales son muchos

Reconocimiento de posturas



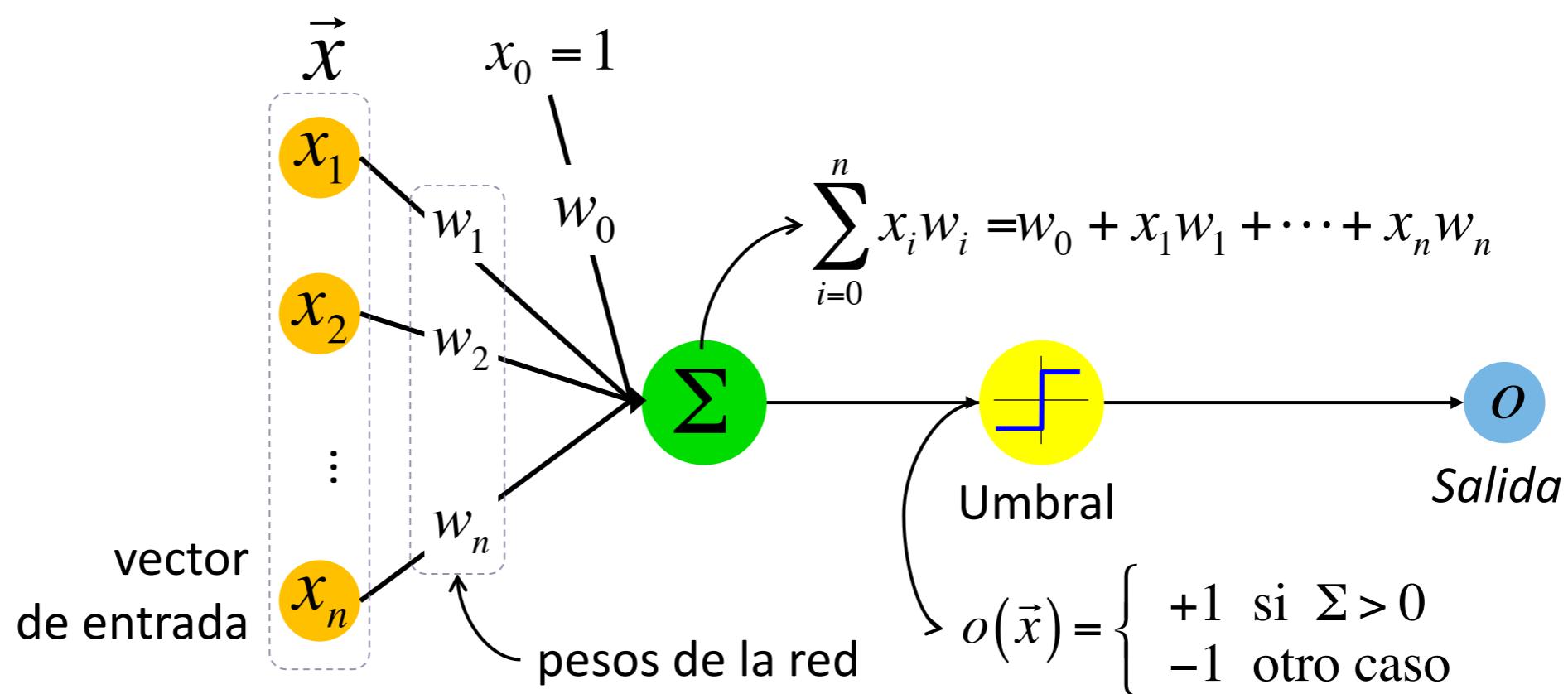
## ■ Supervisados



### ■ Perceptrón lineal: Analicemos el caso más simple, el perceptrón

#### Definiciones

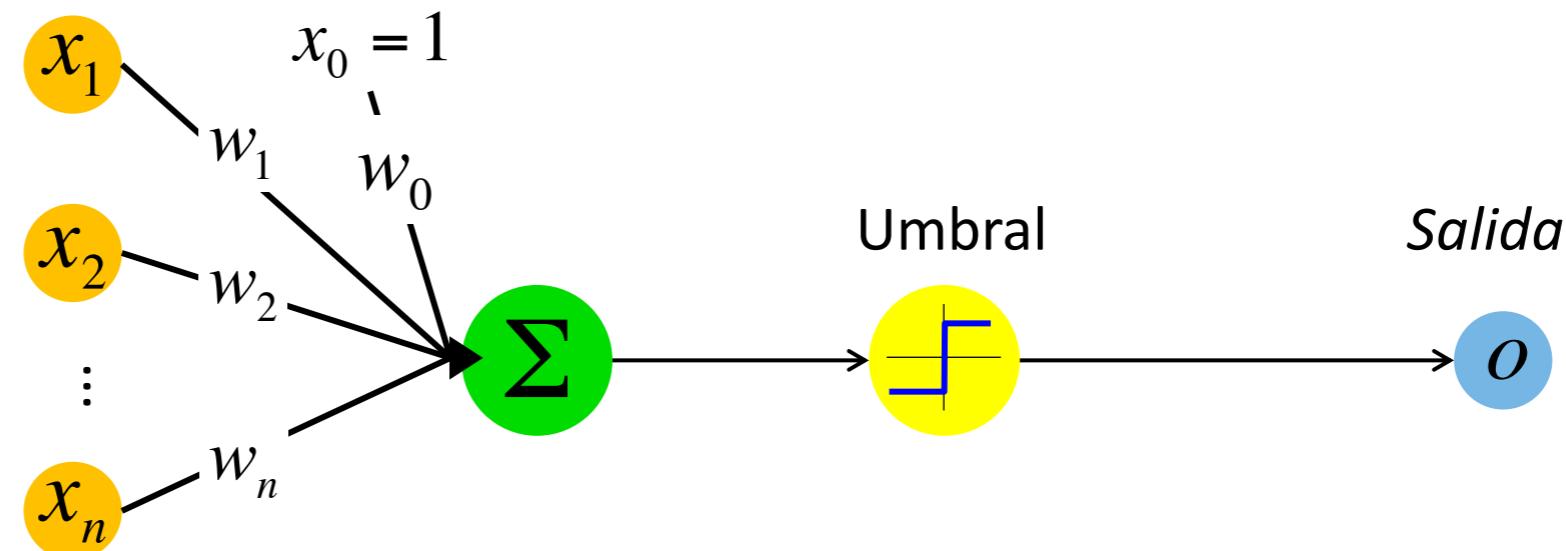
- $\vec{x}$  : conjunto de valores de entrada (características)
- $\vec{w}$  : conjunto de pesos de la red
- $\Sigma$  : sumatoria
- $\text{U}$  : umbral, threshold o función de activación



## ■ Supervisados



- Perceptrón lineal: Analicemos el caso más simple



- Objetivo:

Encontrar el conjunto de pesos ( $\vec{w}$ ) que minimice el error cuadrático sobre el set de entrenamiento,

Set de entrenamiento  $\langle \vec{x}, t \rangle$

$\vec{x}$  : valores de entrada (características)

$t$  : clase del vector de características  $\vec{x}$

$$\text{Error}[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

## ■ Supervisados



### ■ Perceptrón lineal:

Encontrar el conjunto de pesos ( $\vec{w}$ ) que minimice el error cuadrático sobre el set de entrenamiento

$$\text{Error}[\vec{w}] = E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

### ■ Problema

*¿Cómo encontramos los pesos?*

La solución consiste **en mover los pesos en la dirección opuesta al gradiente**. Para ello debemos actualizar cada peso según el error obtenido

$$w_i \leftarrow w_i + \nabla w_i$$

$$\nabla w_i = -\eta \frac{\partial E}{\partial w_i}$$

tasa de aprendizaje regula la velocidad de ajuste de los pesos (menor a uno)

## ■ Supervisados



### ■ Perceptrón lineal: algoritmo gradiente descendiente

① Inicialice cada  $w_i$  con un valor aleatorio pequeño entre -1 y +1  
Inicialice cada gradiente  $\nabla w_i$  en cero

① Hasta que el error sea mínimo, haga:

② Para cada  $\langle \vec{x}, t \rangle$  en el conjunto de entrenamiento, haga

a. Ingrese el vector  $\vec{x}$  y calcule la salida  $o$

b. Para cada gradiente  $\nabla w_i$  calcule:

$$\nabla w_i \leftarrow \nabla w_i + \eta(t - o)x_i$$

③ Para cada peso  $w_i$  calcule:

$$w_i \leftarrow w_i + \nabla w_i$$

① Calcule el error y verifique la condición inicial

$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

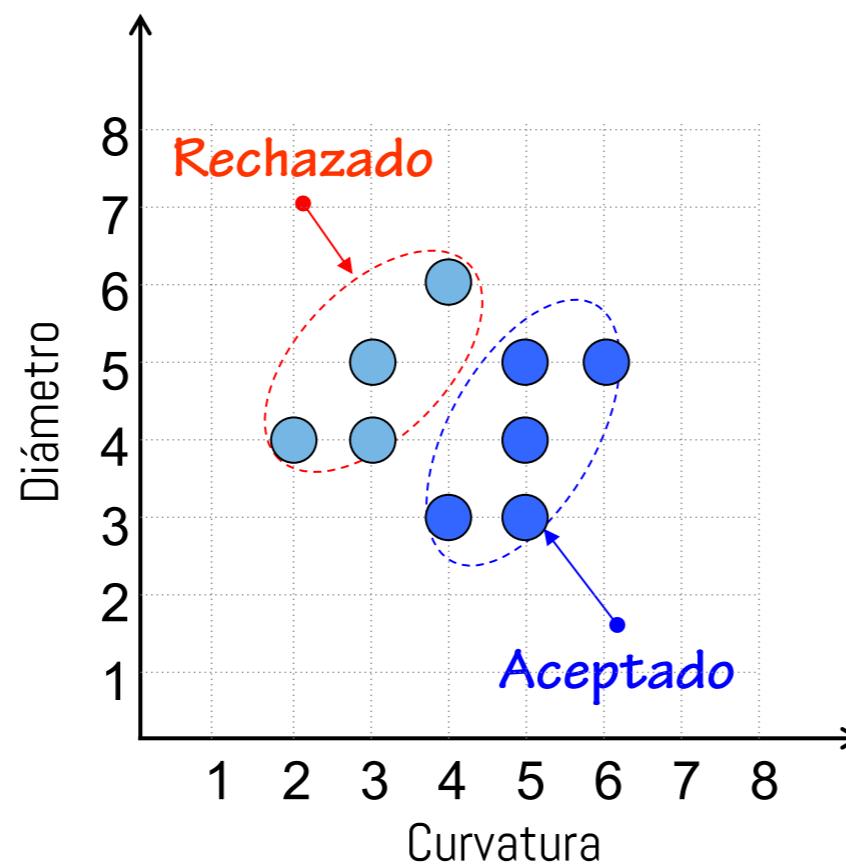
## ■ Supervisados



### ■ Ejemplo:

Supongamos que la empresa *McKoy* produce galletas de muy alta calidad, por ello tiene un muy exhaustivo control de calidad.

La empresa desea automatizar su sistema para que automáticamente determine el rango de aceptación según las características extraídas.



Curvatura	Diámetro	Control Calidad
4	3	Aceptado
5	3	Aceptado
5	4	Aceptado
5	5	Aceptado
6	5	Aceptado
2	4	Rechazado
3	4	Rechazado
3	5	Rechazado
4	6	Rechazado

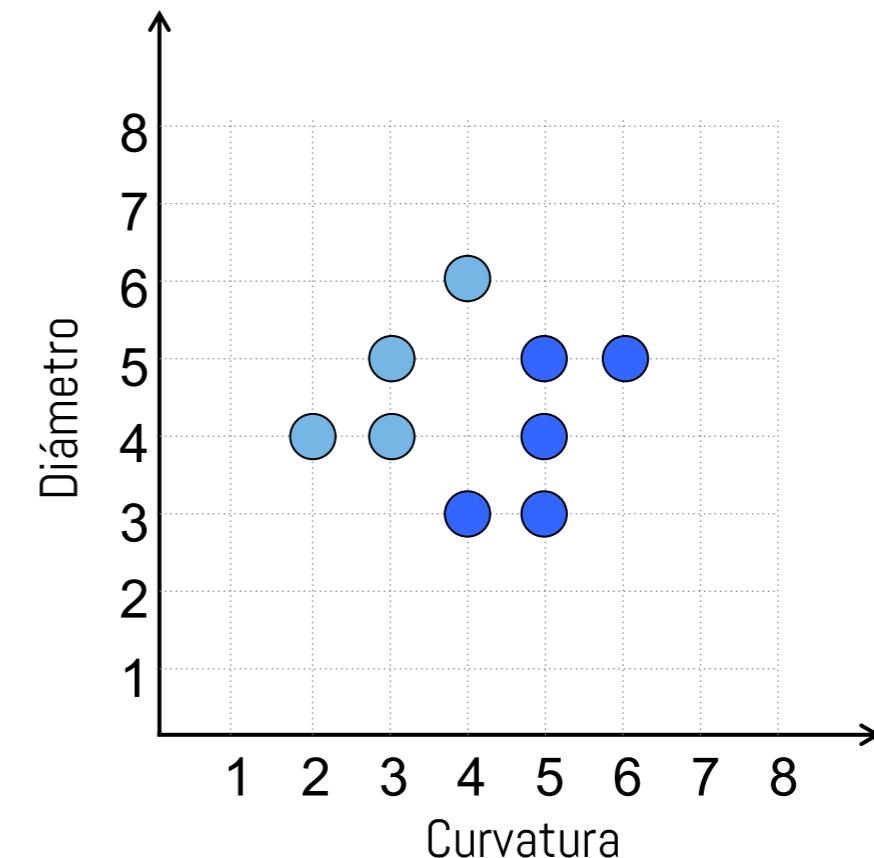
## ■ Supervisados



### ■ Preliminar:

Separemos los datos en sus correspondientes clases. Dado que estamos empleando sólo un perceptrón lineal, podemos clasificar dos clases [+1, -1]

Curvatura	Diámetro	Clase
4	3	1
5	3	1
5	4	1
5	5	1
6	5	1
2	4	-1
3	4	-1
3	5	-1
4	6	-1

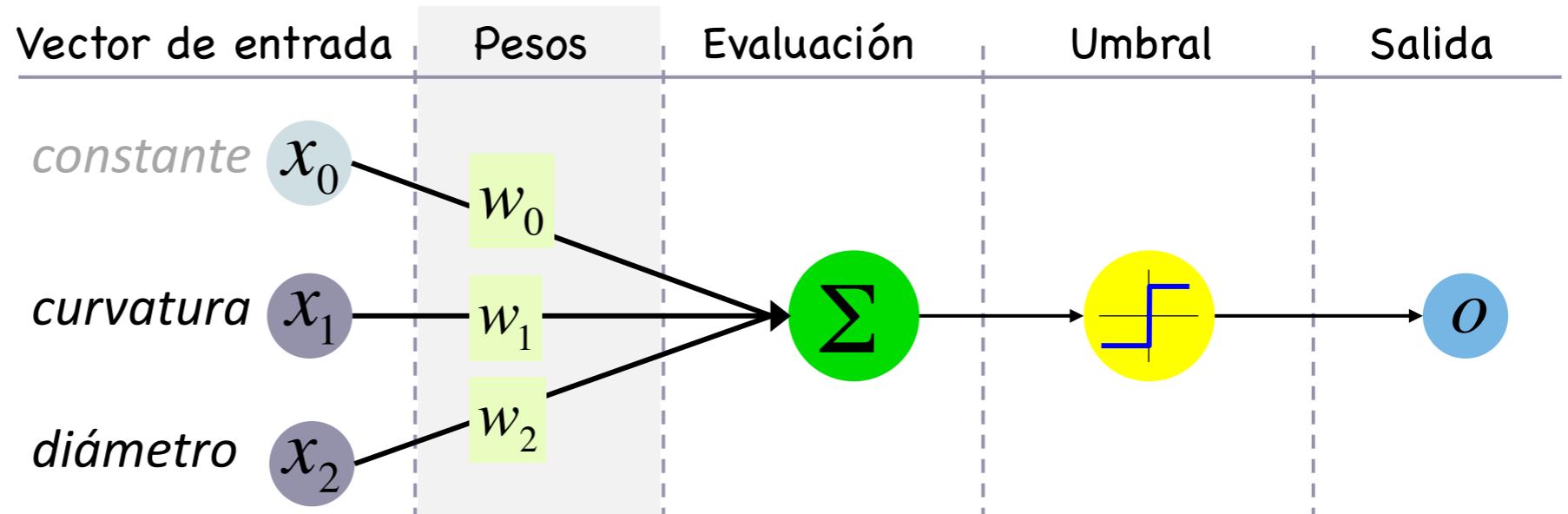


## ■ Supervisados



### ■ PRIMERO:

Inicialicemos los pesos del perceptrón en forma aleatoria. Estos valores deben ser muy pequeños para no perder estabilidad en el algoritmo. Además asignemos cero a los gradientes de los pesos.



#### Pesos iniciales

$$w_0 = -0.1632$$

$$w_1 = 0.1988$$

$$w_2 = -0.0415$$

Estos valores fueron elegidos en forma aleatoria.  
Recuerde inicializar con valores pequeños

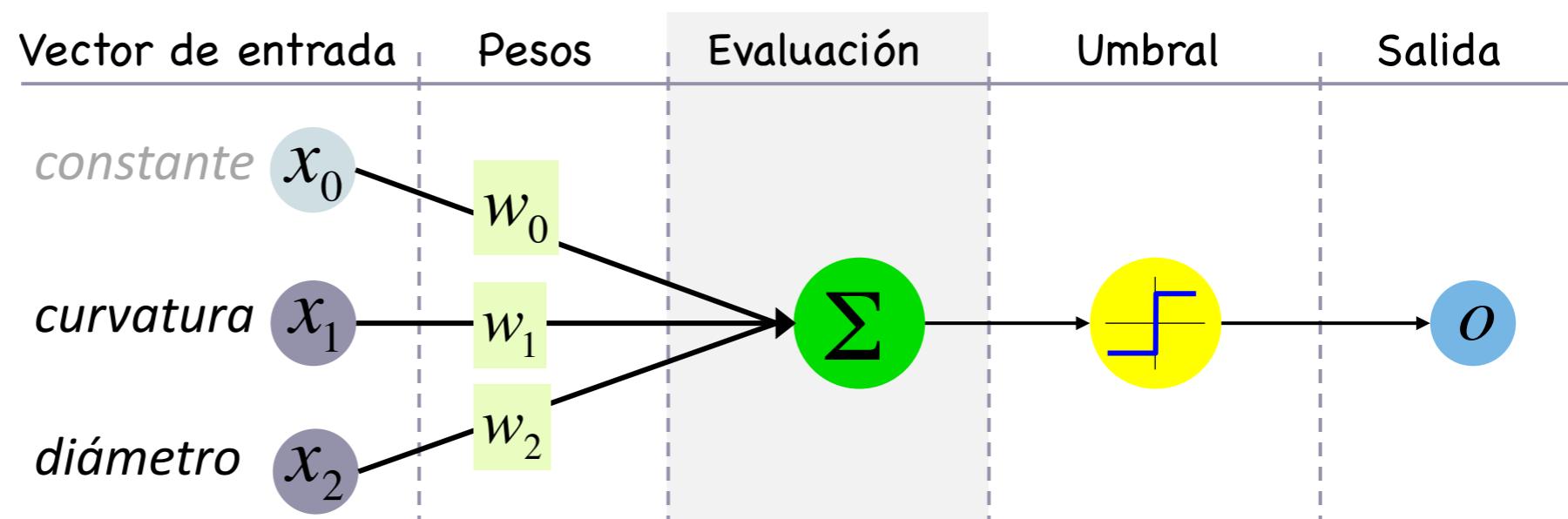


## ■ Supervisados



### ■ SEGUNDO:

Para cada vector del conjunto de entrenamiento calculemos la salida.



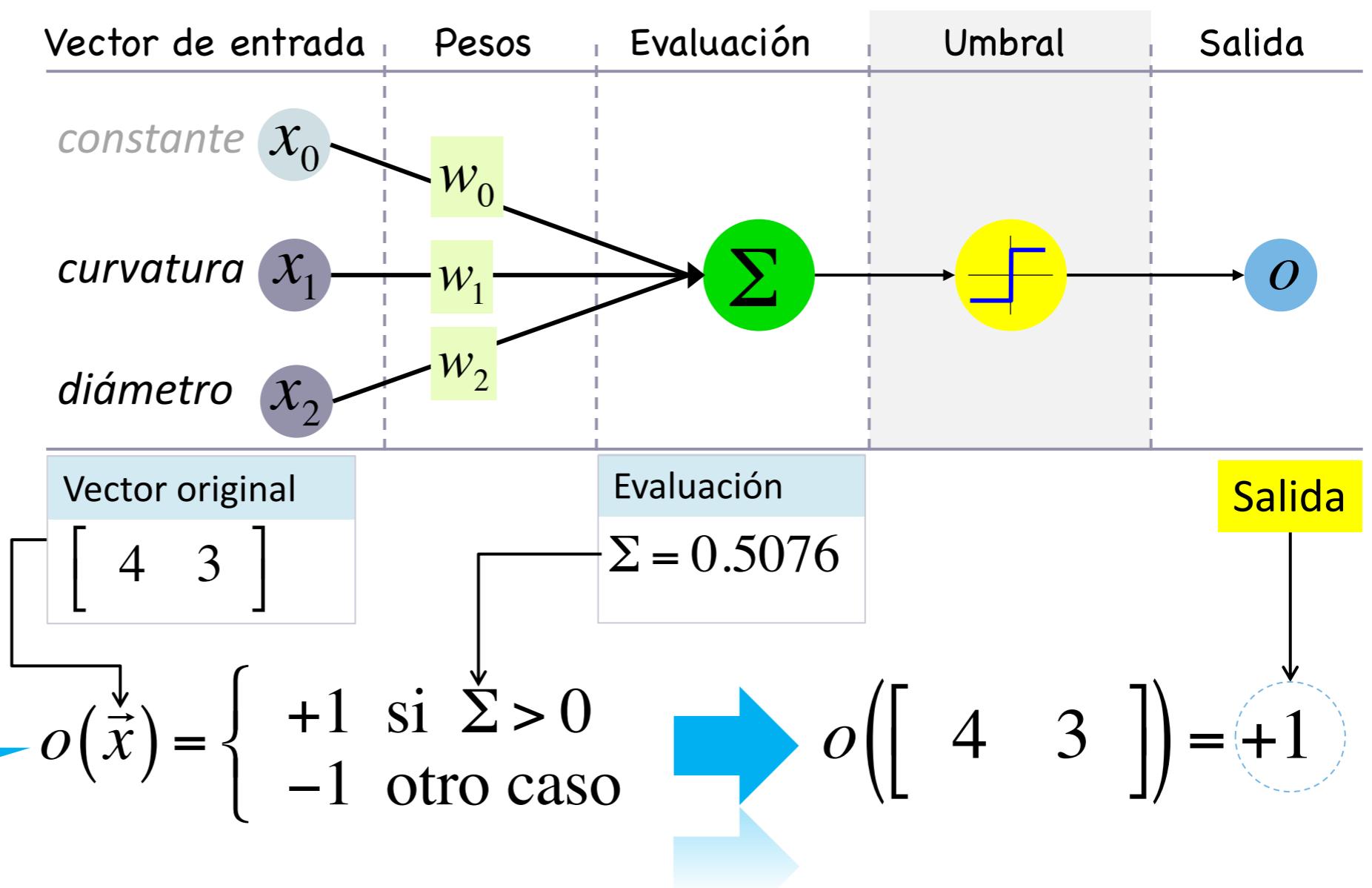
$$\begin{array}{cccc} & \text{constante} & \text{curvatura} & \text{diámetro} \\ w_0 & \downarrow & \downarrow & \downarrow \\ \Sigma = (-0.1632) \times 1 + (0.1988) \times 4 + (-0.0415) \times 3 & & & \end{array}$$

## ■ Supervisados



### ■ SEGUNDO:

Para cada vector del conjunto de entrenamiento calculemos la salida.



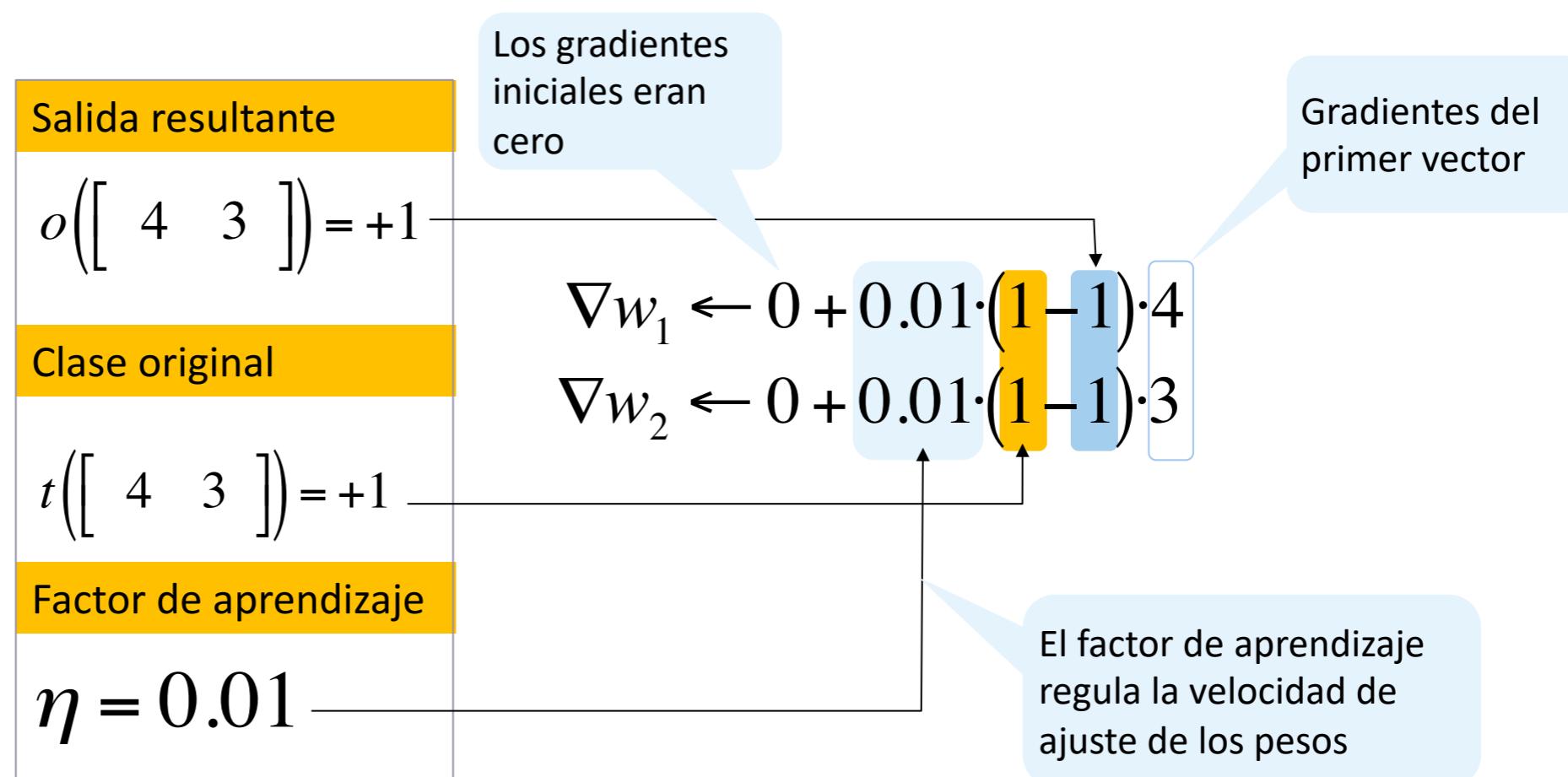
## ■ Supervisados



### ■ TERCERO:

Para cada respuesta, calculamos el gradiente de los pesos. Para ello ocupamos la salida de la neurona y la clase original, además del **factor de aprendizaje**.

$$\nabla w_i \leftarrow \nabla w_i + \eta(t - o)x_i$$



## ■ Supervisados



### ■ TERCERO:

Para cada respuesta, calculamos el gradiente de los pesos. Para ello ocupamos la salida de la neurona y la clase original, además del factor de aprendizaje.

$$\nabla w_i \leftarrow \nabla w_i + \eta(t - o)x_i$$

Curvatura	Diámetro	Clase	Salida	$\nabla w_1$	$\nabla w_2$
4	3	1	1	0	0
5	3	1	1	0	0
5	4	1	1	0	0
5	5	1	1	0	0
6	5	1	1	0	0
2	4	-1	1	-0.04	-0.08
3	4	-1	1	-0.10	-0.16
3	5	-1	1	-0.16	-0.26
4	6	-1	1	-0.24	-0.38

Finalmente el  
gradiente  
acumulado es

:

$$\nabla w_1 = -0.24$$

$$\nabla w_2 = -0.38$$

## ■ Supervisados



### ■ CUARTO:

Actualizamos los pesos de la neurona empleando el valor del gradiente calculado en el tercer paso.

$$w_i \leftarrow w_i + \nabla w_i$$

#### Pesos originales

$$\begin{aligned}w_0 &= -0.1632 \\w_1 &= 0.1988 \\w_2 &= -0.0415\end{aligned}$$

#### Gradientes estimados

$$\begin{aligned}\nabla w_1 &= -0.24 \\\nabla w_2 &= -0.38\end{aligned}$$

#### Pesos re-estimados

$$\begin{aligned}w_0 &= -0.1632 \\w_1 &= -0.0412 \\w_2 &= -0.4215\end{aligned}$$

Reemplazando los pesos y los gradientes

$$w_1 \leftarrow w_1 + \nabla w_1 = 0.1988 + (-0.24)$$

$$w_2 \leftarrow w_2 + \nabla w_2 = -0.0415 + (-0.38)$$

## ■ Supervisados



### Pesos re-estimados

$$w_0 = -0.1632$$

$$w_1 = -0.0412$$

$$w_2 = -0.4215$$

### ■ QUINTO:

Calculamos el error cuadrático de la salida de la neurona. Para ellos debemos re-estimar la salida por cada vector del conjunto de entrenamiento

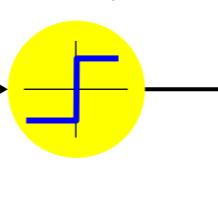
$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$\Sigma = (-0.1632) \times 1 + (-0.0412) \times 4 + (-0.4215) \times 3$$

$$\Sigma = -1.5925$$

constante                          curvatura                          diámetro

$w_0$                            $w_1$                            $w_2$



### Clase original

$$t\left(\begin{bmatrix} 4 & 3 \end{bmatrix}\right) = 1$$

### Salida resultante

$$o\left(\begin{bmatrix} 4 & 3 \end{bmatrix}\right) = -1$$

### Estimación del error

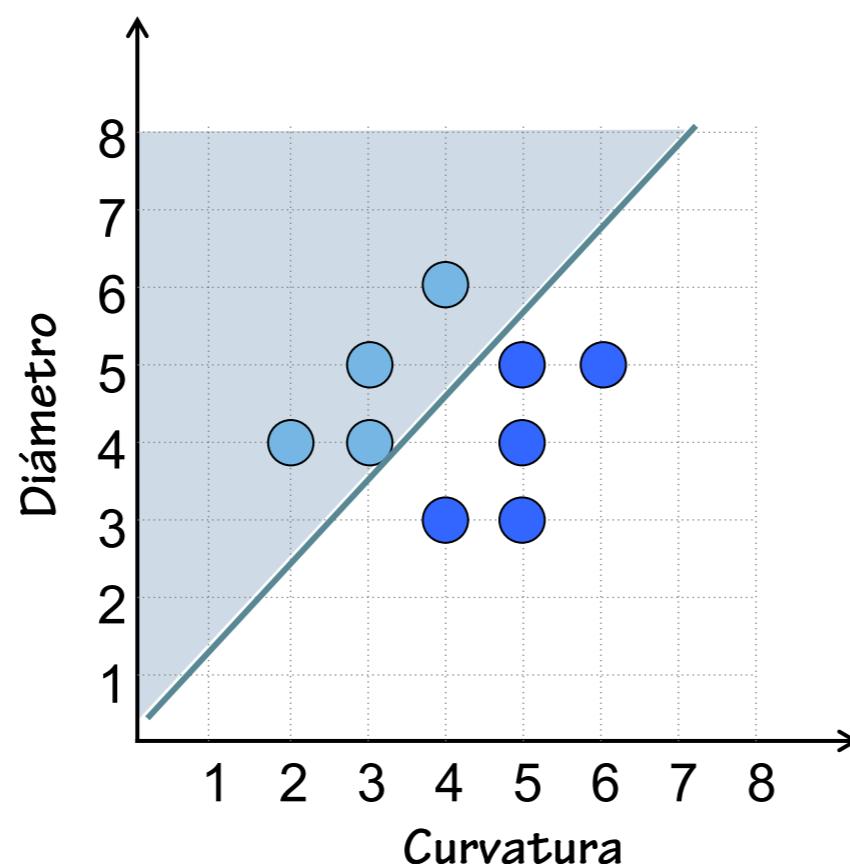
$$E[\vec{w}] = \frac{1}{2}(1 - 1)^2 + \dots$$

## ■ Supervisados



### ■ Finalmente:

El proceso continúa desde el segundo paso al quinto paso. De esta forma el error va disminuyendo. Es importante tener en consideración que el proceso de refinamiento puede verse afectado por problemas de overfitting o sobreentrenamiento.



- Luego de algunas iteraciones, el algoritmo estimó los siguientes pesos de la neurona:

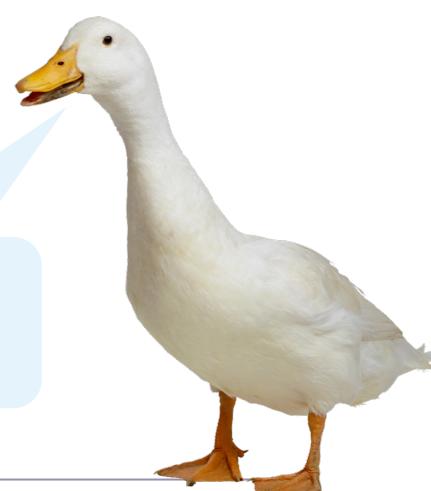
$$w_0 = -0.1632$$

$$w_1 = 4.2988$$

$$w_2 = -3.3215$$

Ecuación lineal  
de la recta

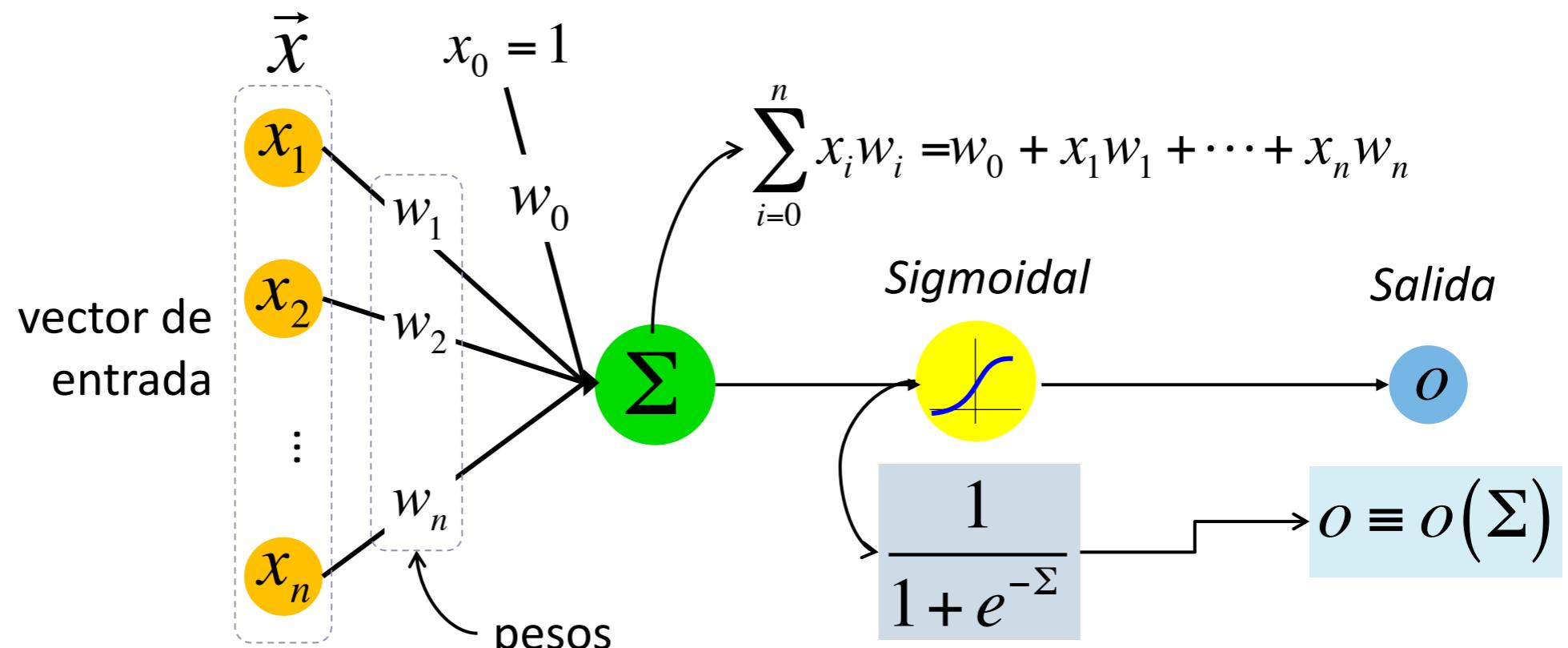
$$(-0.1632) \times x_0 + (4.2988) \times x_1 + (-3.3215) \times x_2 = 0$$



## ■ Supervisados



### ■ Perceptrón con activación sigmoidal.



### ■ Regla de entrenamiento

- Ingrese el vector  $\vec{x}$  y calcule la salida  $o$
- Para cada gradiente  $\nabla w_i$  calcule:

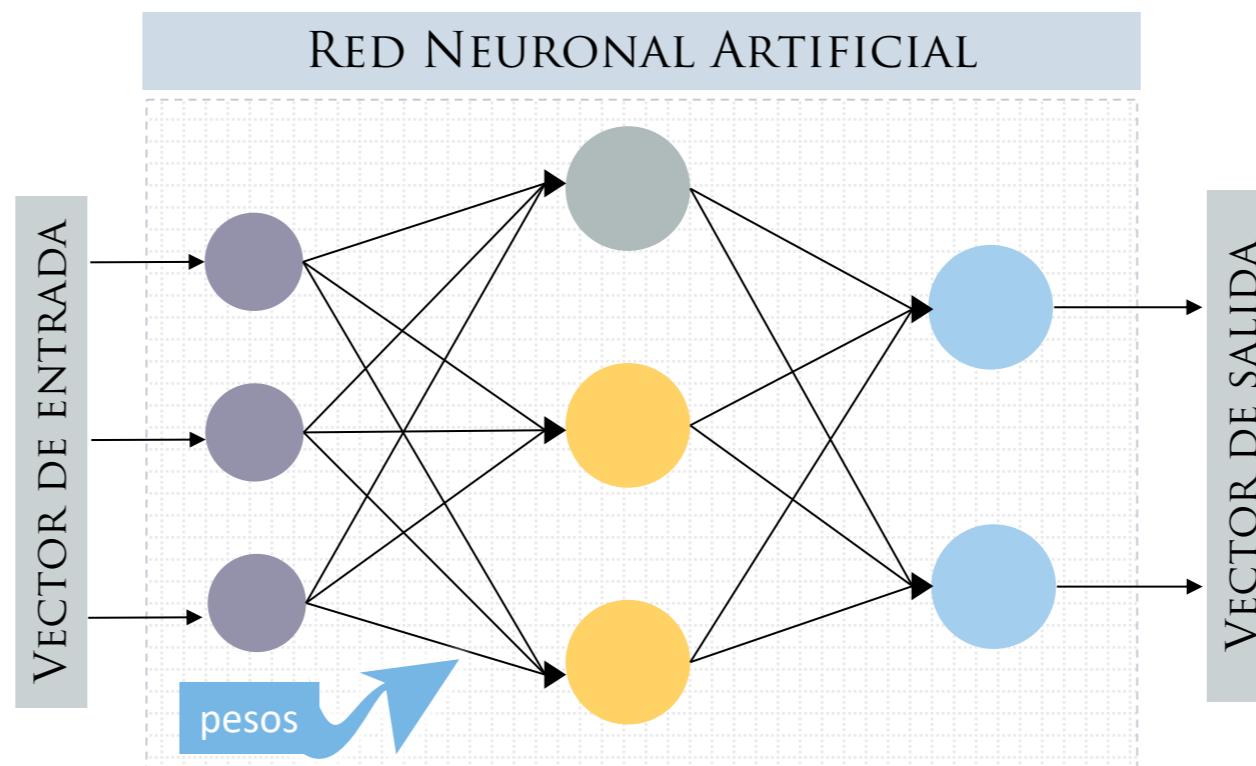
$$\nabla w_i \leftarrow \nabla w_i + \eta(t - o)o(1 - o)x_i$$

## ■ Supervisados



### ■ Redes Neuronales (en resumen)

El mayor problema de la red neuronal consiste en cómo ajustar los pesos de las conexiones internas para obtener la salida deseada.



Toda red neuronal se caracteriza por su:

1. Estructura interna
2. Métodos para determinar los pesos
3. Función de activación

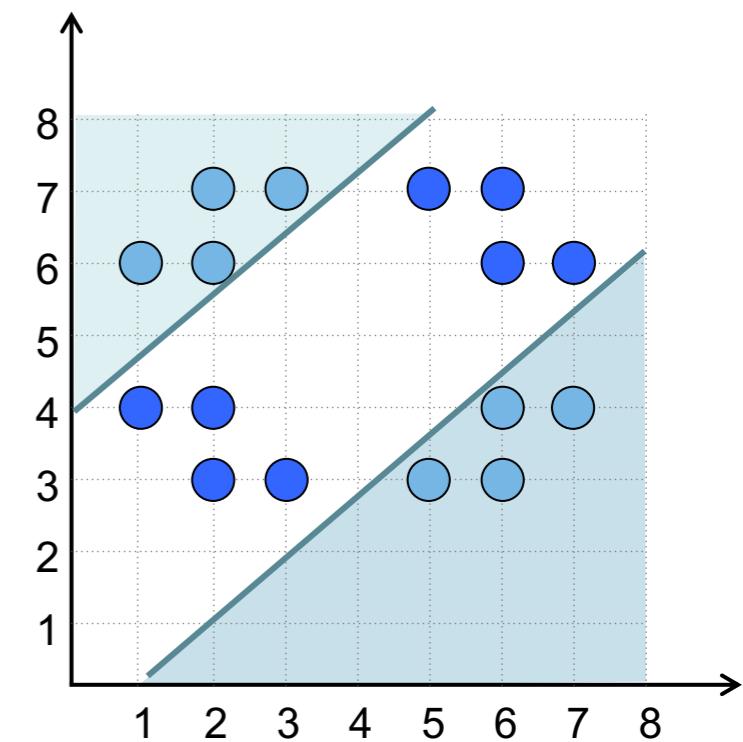
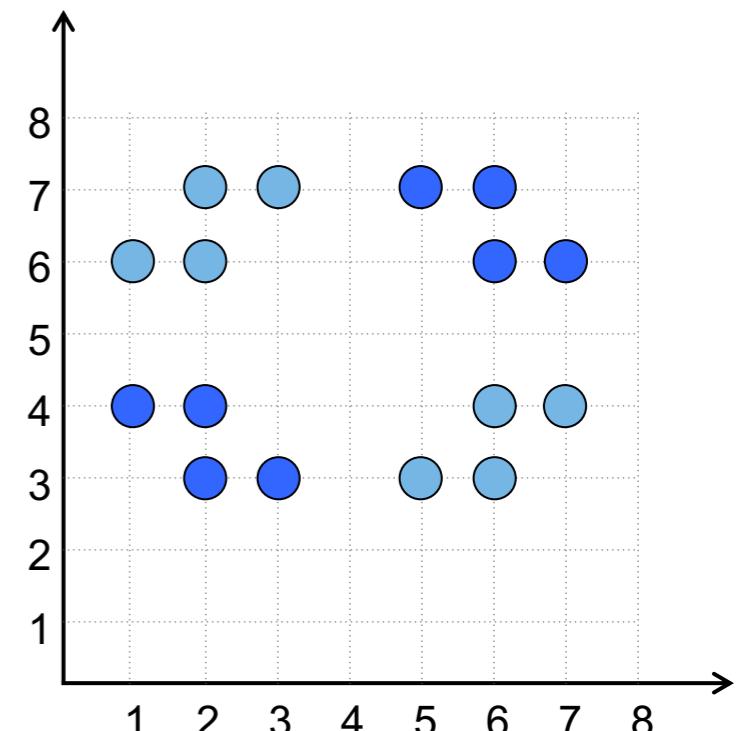
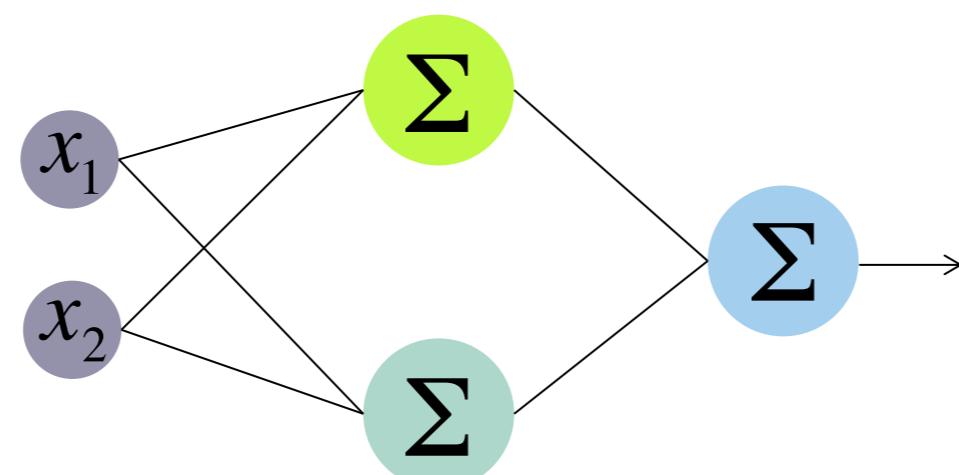


## ■ Supervisados



- **Perceptrón (y qué pasa cuando?)**  
En situaciones como la presente en la figura, ¿cómo generamos el perceptrón para separar las clases?
- **SOLUCIÓN:**  
Empleando la combinación de dos perceptrones a través de un or-exclusivo

$$x_1 \otimes x_2 = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$$



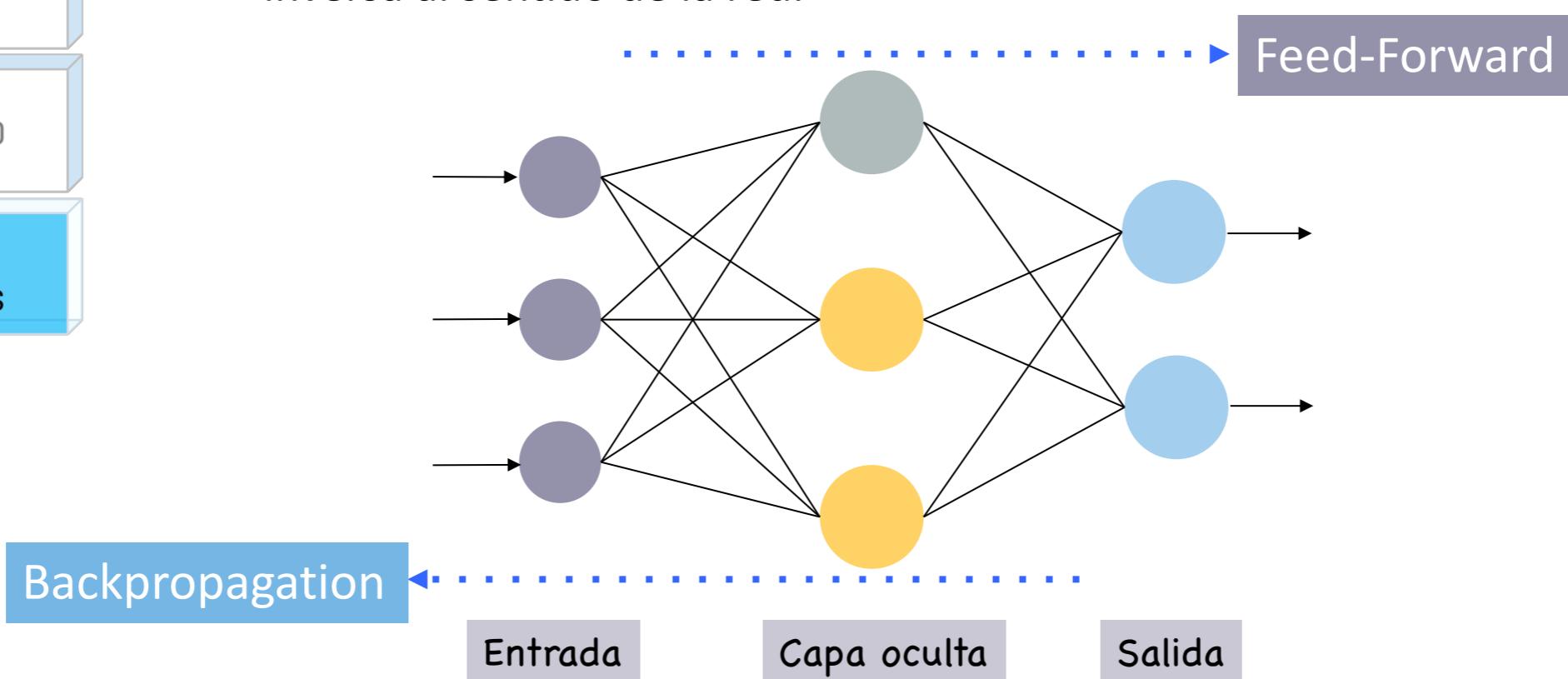
## ■ Supervisados



### ■ Backpropagation algorithm

El algoritmo backpropagation permite entrenar y determinar los pesos de una red con múltiples capas ocultas.

De la misma forma que determinamos los pesos en el perceptrón lineal, el **algoritmo backpropagation utiliza las etiquetas de salida para ajustar los pesos internos de la red**. La principal ventaja consiste en determinar los pesos o el error de forma tal que retroceden o actualizan en forma inversa al sentido de la red.

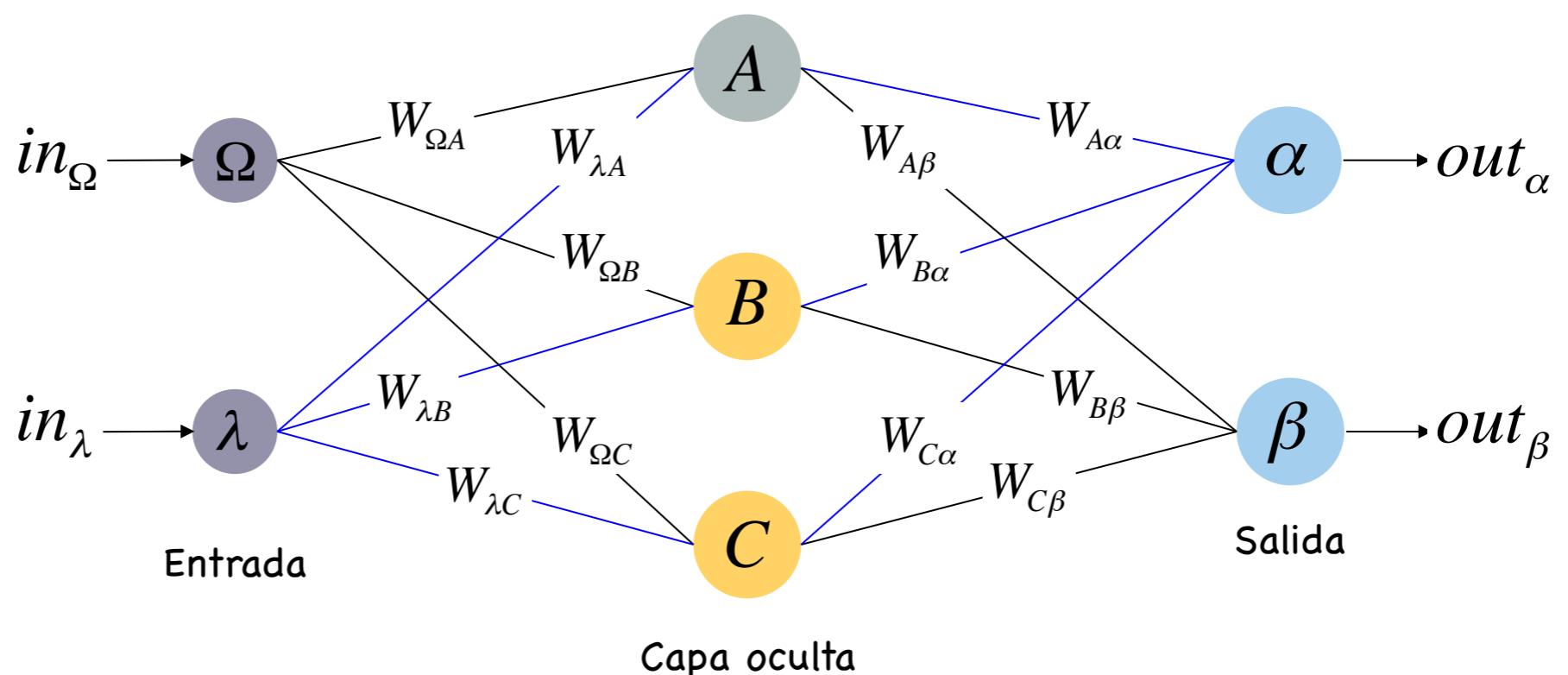


## ■ Supervisados



### ■ Backpropagation algorithm

Veamos un ejemplo paso a paso para comprender el algoritmo. Suponga una red en la cual tenemos dos entradas, tres capas ocultas y dos salidas



1

Calcular el error de las neuronas de salida

$$\delta_{\alpha} = out_{\alpha} \cdot (1 - out_{\alpha}) \cdot (goal_{\alpha} - out_{\alpha})$$

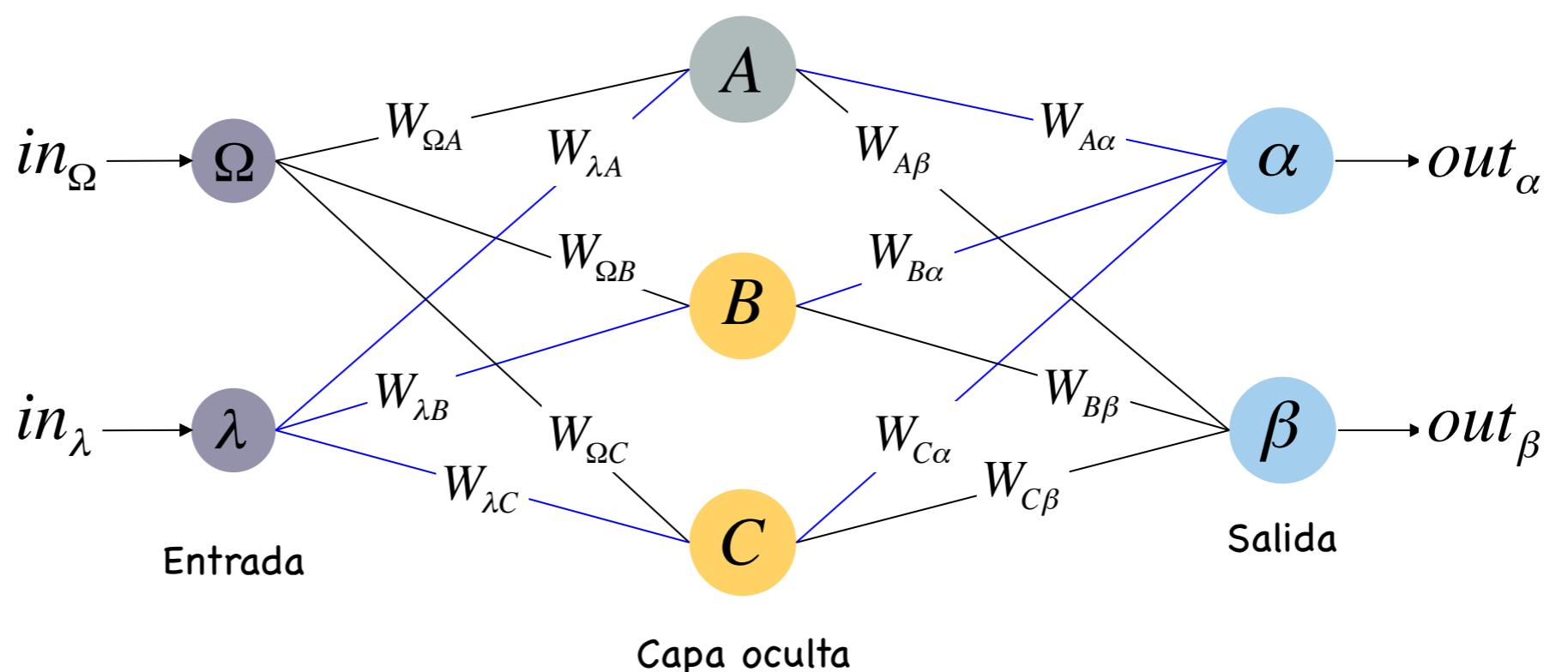
$$\delta_{\beta} = out_{\beta} \cdot (1 - out_{\beta}) \cdot (goal_{\beta} - out_{\beta})$$

## ■ Supervisados



### ■ Backpropagation algorithm

Veamos un ejemplo paso a paso para comprender el algoritmo. Suponga una red en la cual tenemos dos entradas, tres capas ocultas y dos salidas



2

Actualizar los pesos de la capa de salida

$$W_{A\alpha}^+ = W_{A\alpha} + \eta \cdot \delta_{\alpha} \cdot out_A$$

$$W_{B\alpha}^+ = W_{B\alpha} + \eta \cdot \delta_{\alpha} \cdot out_B$$

$$W_{C\alpha}^+ = W_{C\alpha} + \eta \cdot \delta_{\alpha} \cdot out_C$$

$$W_{A\beta}^+ = W_{A\beta} + \eta \cdot \delta_{\beta} \cdot out_A$$

$$W_{B\beta}^+ = W_{B\beta} + \eta \cdot \delta_{\beta} \cdot out_B$$

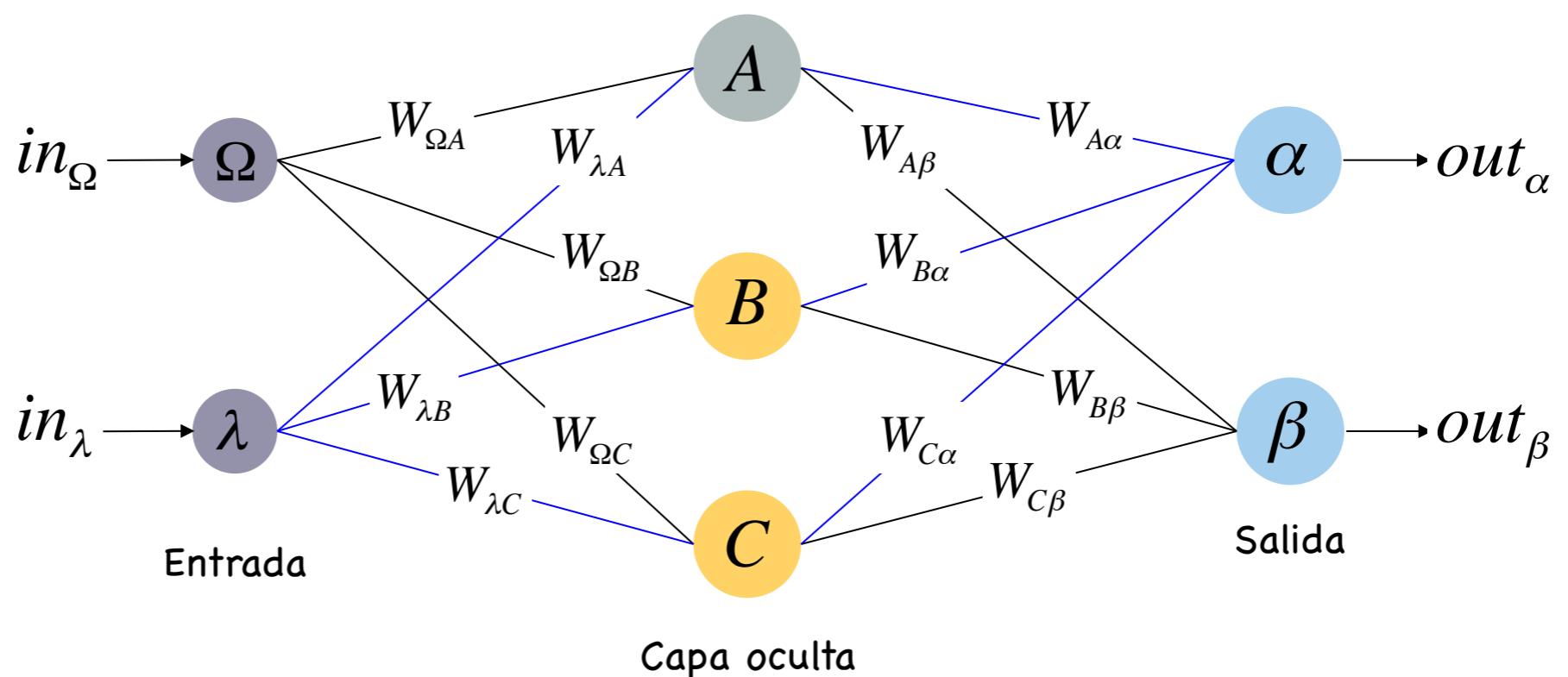
$$W_{C\beta}^+ = W_{C\beta} + \eta \cdot \delta_{\beta} \cdot out_C$$

## ■ Supervisados



### ■ Backpropagation algorithm

Veamos un ejemplo paso a paso para comprender el algoritmo. Suponga una red en la cual tenemos dos entradas, tres capas ocultas y dos salidas



3

Backpropagate los errores de la capa oculta

$$\delta_A = out_A \cdot (1 - out_A) \cdot (\delta_\alpha \cdot W_{A\alpha} + \delta_\beta \cdot W_{A\beta})$$

$$\delta_B = out_B \cdot (1 - out_B) \cdot (\delta_\alpha \cdot W_{B\alpha} + \delta_\beta \cdot W_{B\beta})$$

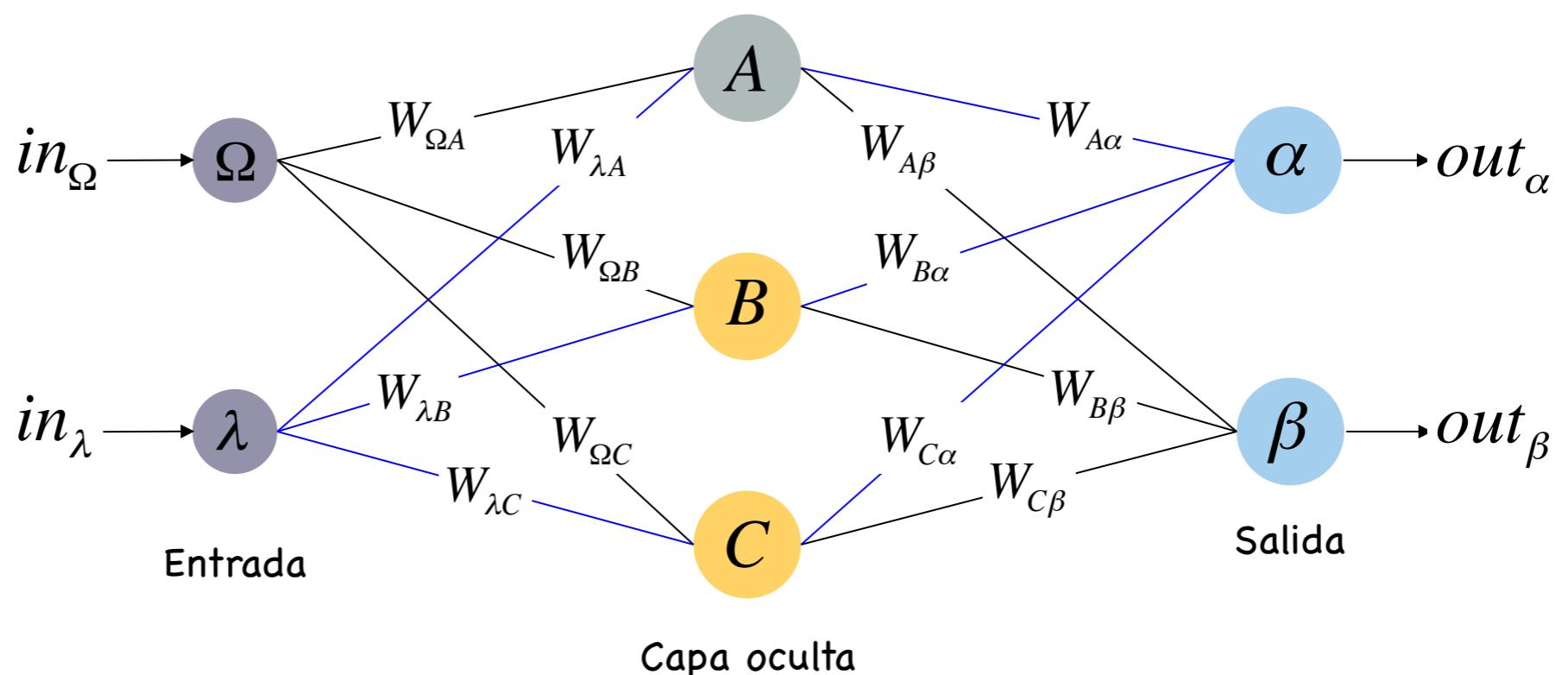
$$\delta_C = out_C \cdot (1 - out_C) \cdot (\delta_\alpha \cdot W_{C\alpha} + \delta_\beta \cdot W_{C\beta})$$

## ■ Supervisados



### ■ Backpropagation algorithm

Veamos un ejemplo paso a paso para comprender el algoritmo. Suponga una red en la cual tenemos dos entradas, tres capas ocultas y dos salidas



4

Actualizar los pesos de las capas ocultas

$$W_{\Omega A}^+ = W_{\Omega A} + \eta \cdot \delta_A \cdot in_{\Omega}$$

$$W_{\Omega B}^+ = W_{\Omega B} + \eta \cdot \delta_B \cdot in_{\Omega}$$

$$W_{\Omega C}^+ = W_{\Omega C} + \eta \cdot \delta_C \cdot in_{\Omega}$$

$$W_{\lambda A}^+ = W_{\lambda A} + \eta \cdot \delta_A \cdot in_{\lambda}$$

$$W_{\lambda B}^+ = W_{\lambda B} + \eta \cdot \delta_B \cdot in_{\lambda}$$

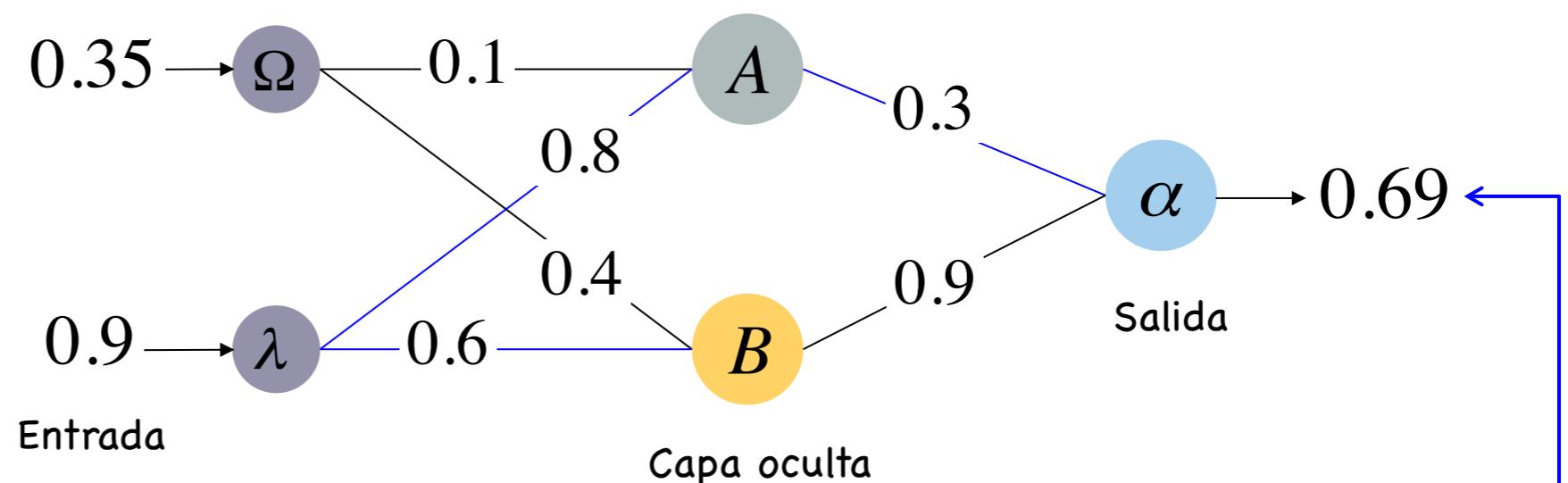
$$W_{\lambda C}^+ = W_{\lambda C} + \eta \cdot \delta_C \cdot in_{\lambda}$$

## ■ Supervisados



### ■ Backpropagation algorithm

En el ejemplo, supongamos que la salida deseada o *goal* es 0.5, asumiendo que la función de activación es sigmoidal



0

### Paso Forward:

Estimemos las salidas de cada neurona

$$\frac{1}{1 + e^{-\Sigma}}$$

Debemos aplicar la función sigmoidal

$$A = (0.35 \times 0.1) + (0.9 \times 0.8) = 0.755 \rightarrow 0.68$$

$$B = (0.9 \times 0.6) + (0.35 \times 0.4) = 0.680 \rightarrow 0.664$$

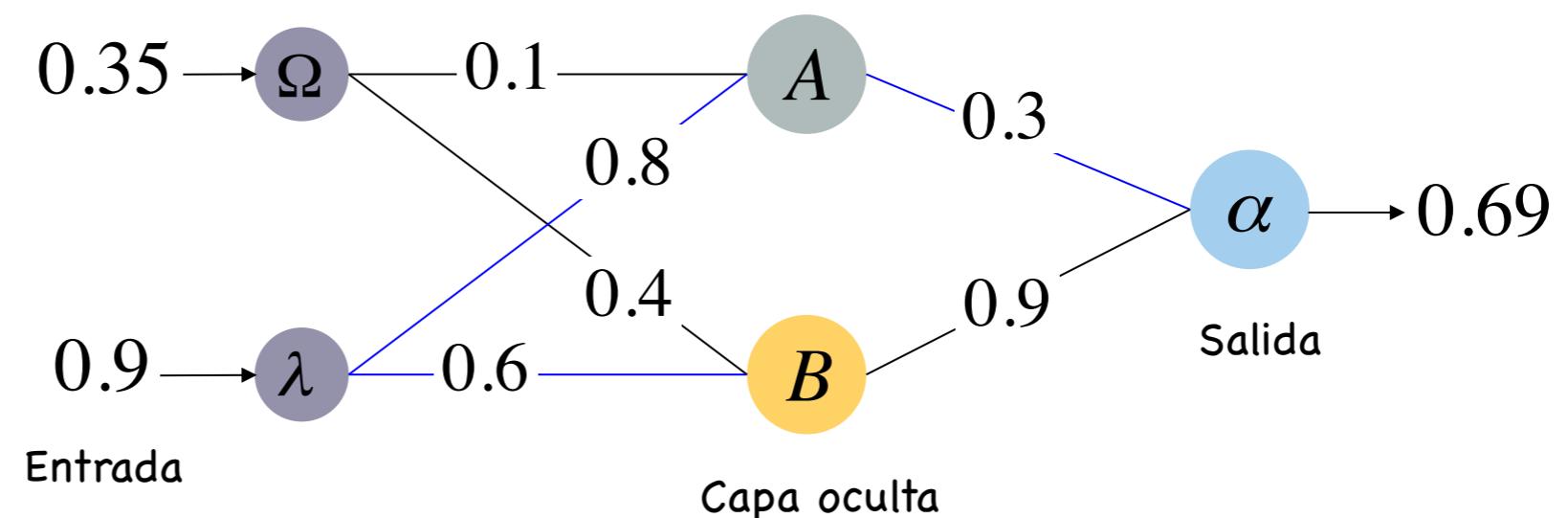
$$\alpha = (0.3 \times 0.68) + (0.9 \times 0.664) = 0.8013 \rightarrow 0.69$$

## ■ Supervisados



- Backpropagation algorithm

En el ejemplo, supongamos que la salida deseada o *goal* es 0.5, asumiendo que la función de activación es sigmoidal



- 1

Calcular el error de la neurona de salida

$$\delta_\alpha = out_\alpha \cdot (1 - out_\alpha) \cdot (goal_\alpha - out_\alpha)$$

$$\delta_\alpha = 0.69 \times (1 - 0.69) \times (0.5 - 0.69) = -0.0406$$

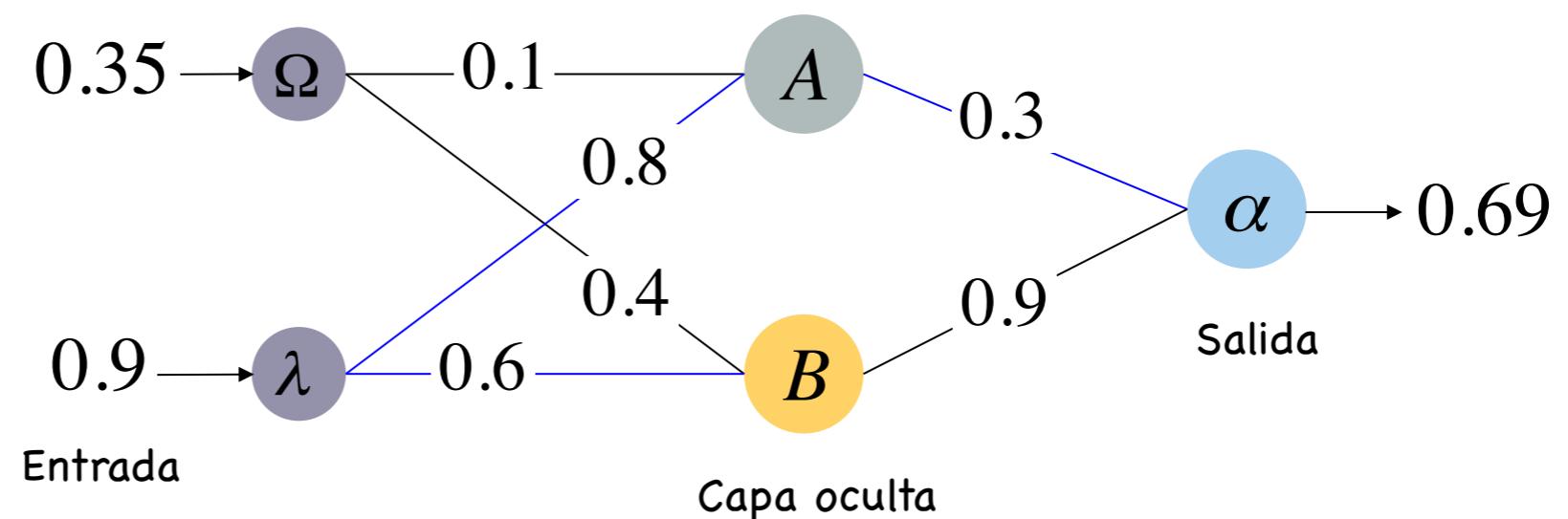
**fuente:** <http://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf>

## ■ Supervisados



### ■ Backpropagation algorithm

En el ejemplo, supongamos que la salida deseada o *goal* es 0.5, asumiendo que la función de activación es sigmoidal



2

Actualizar los pesos de la capa de salida

$$W_{A\alpha}^+ = W_{A\alpha} + \eta \cdot \delta_\alpha \cdot out_A = 0.3 + 1 \times (-0.0406 \times 0.680) = 0.2724$$

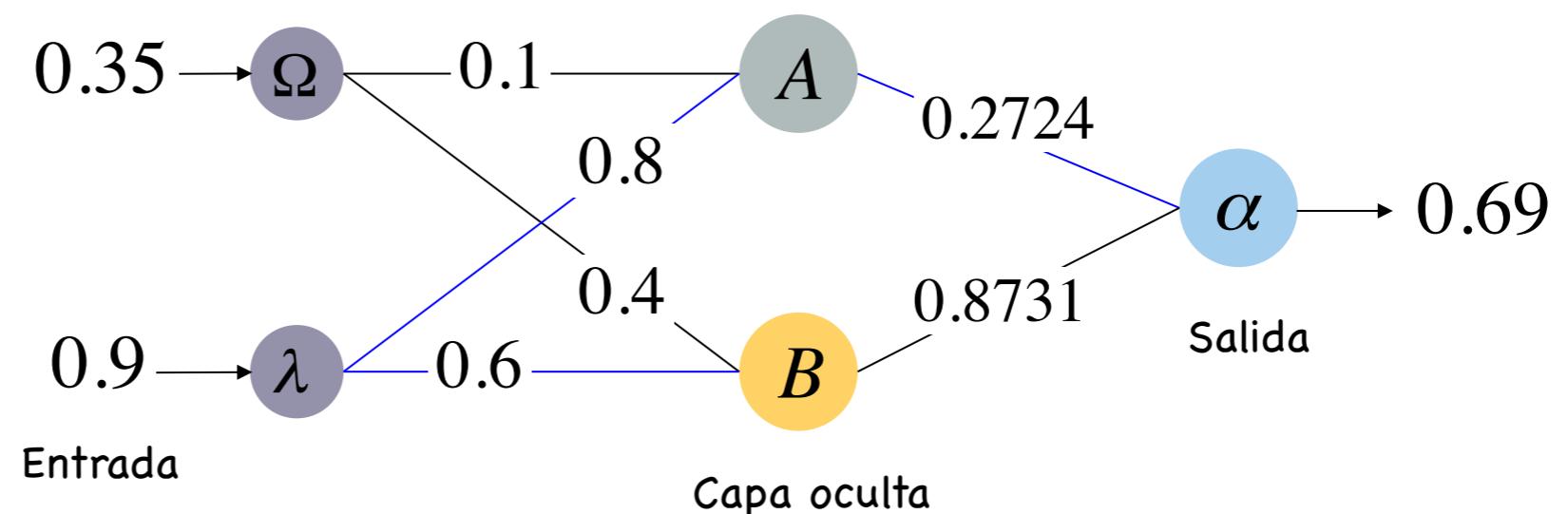
$$W_{B\alpha}^+ = W_{B\alpha} + \eta \cdot \delta_\alpha \cdot out_B = 0.9 + 1 \times (-0.0406 \times 0.664) = 0.8731$$

## ■ Supervisados



### ■ Backpropagation algorithm

En el ejemplo, supongamos que la salida deseada o *goal* es 0.5, asumiendo que la función de activación es sigmoidal



3

Backpropagate los errores de la capa oculta

$$\delta_A = out \cdot (1 - out) \cdot (\delta_\alpha \cdot W_{A\alpha}) = 0.69 \cdot (1 - 0.69) \cdot (-0.0406 \times 0.2724) = -2.36 \times 10^{-3}$$

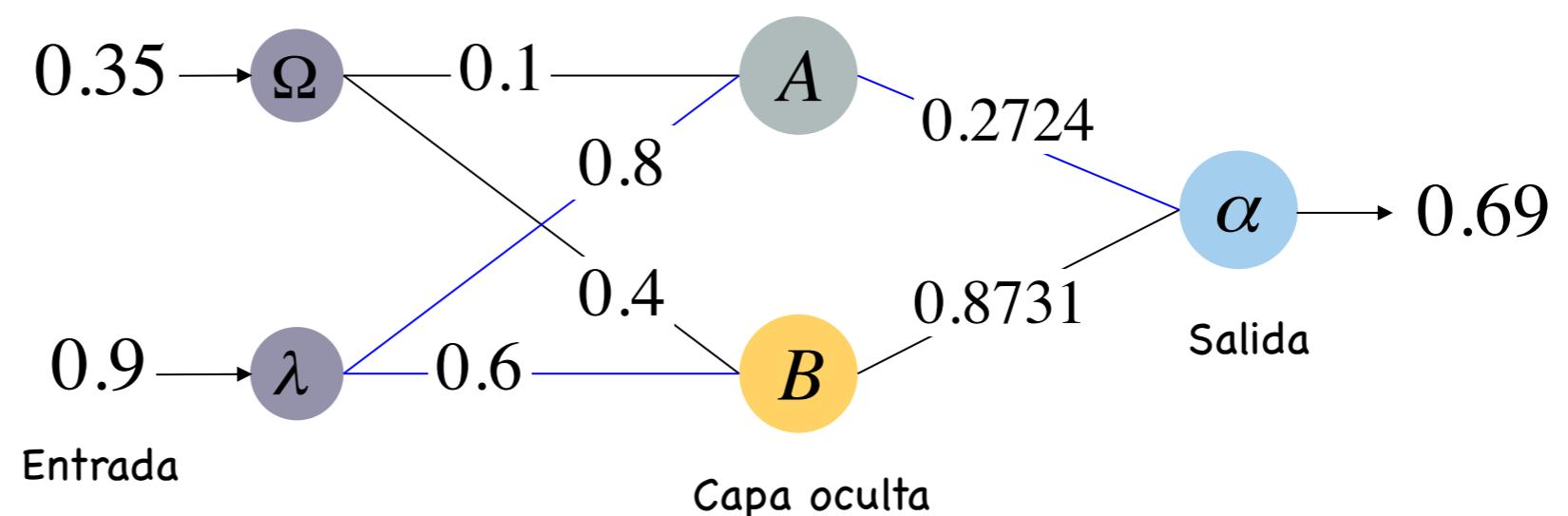
$$\delta_B = out \cdot (1 - out) \cdot (\delta_\alpha \cdot W_{B\alpha}) = 0.69 \cdot (1 - 0.69) \cdot (-0.0406 \times 0.8731) = -7.58 \times 10^{-3}$$

## ■ Supervisados



### ■ Backpropagation algorithm

En el ejemplo, supongamos que la salida deseada o *goal* es 0.5, asumiendo que la función de activación es sigmoidal



4

Actualizar los pesos de las capas ocultas

$$W_{\Omega A}^+ = W_{\Omega A} + \eta \cdot \delta_A \cdot in_{\Omega} = 0.1 + 1 \times (-2.36 \times 10^{-3} \cdot 0.35) = 0.09917$$

$$W_{\Omega B}^+ = W_{\Omega B} + \eta \cdot \delta_B \cdot in_{\Omega} = 0.4 + 1 \times (-7.58 \times 10^{-3} \cdot 0.35) = 0.3973$$

$$W_{\lambda A}^+ = W_{\lambda A} + \eta \cdot \delta_A \cdot in_{\lambda} = 0.8 + 1 \cdot (-2.36 \times 10^{-3} \times 0.9) = 0.7978$$

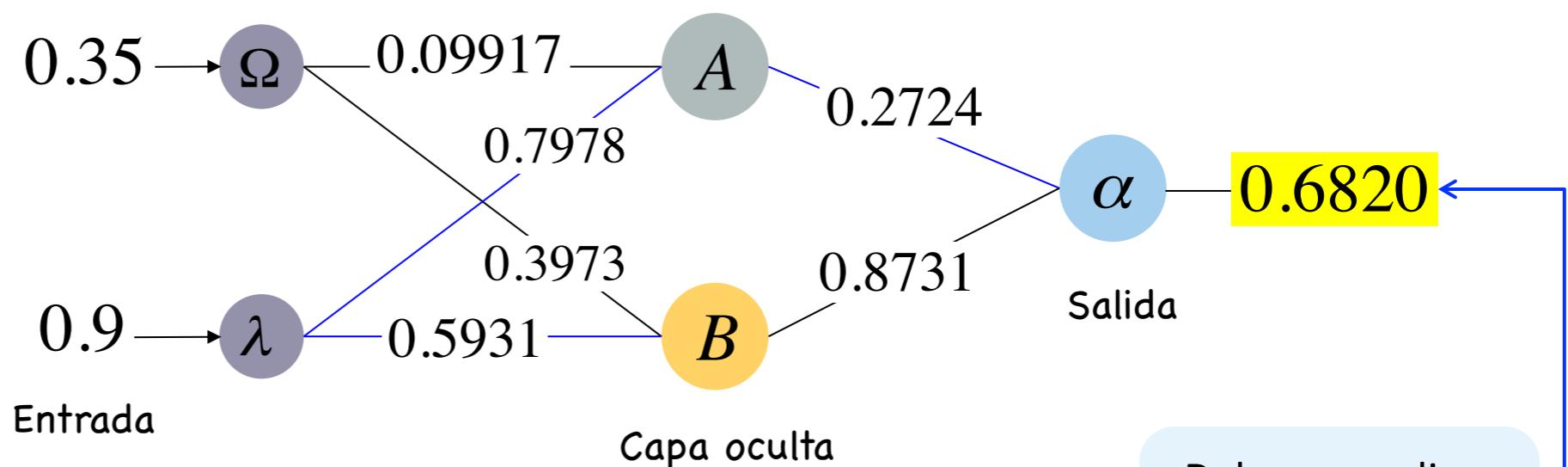
$$W_{\lambda B}^+ = W_{\lambda B} + \eta \cdot \delta_B \cdot in_{\lambda} = 0.6 + 1 \cdot (-7.58 \times 10^{-3} \times 0.9) = 0.5931$$

## ■ Supervisados



### ■ Backpropagation algorithm

En el ejemplo, supongamos que la salida deseada o *goal* es 0.5, asumiendo que la función de activación es sigmoidal



0

### Paso Forward:

Estimemos las salidas de cada neurona

$$\frac{1}{1+e^{-\Sigma}}$$

$$A = (0.35 \times 0.9917) + (0.9 \times 0.7978) = 0.7527 \rightarrow 0.6798$$

$$B = (0.9 \times 0.5931) + (0.35 \times 0.3973) = 0.6728 \rightarrow 0.6621$$

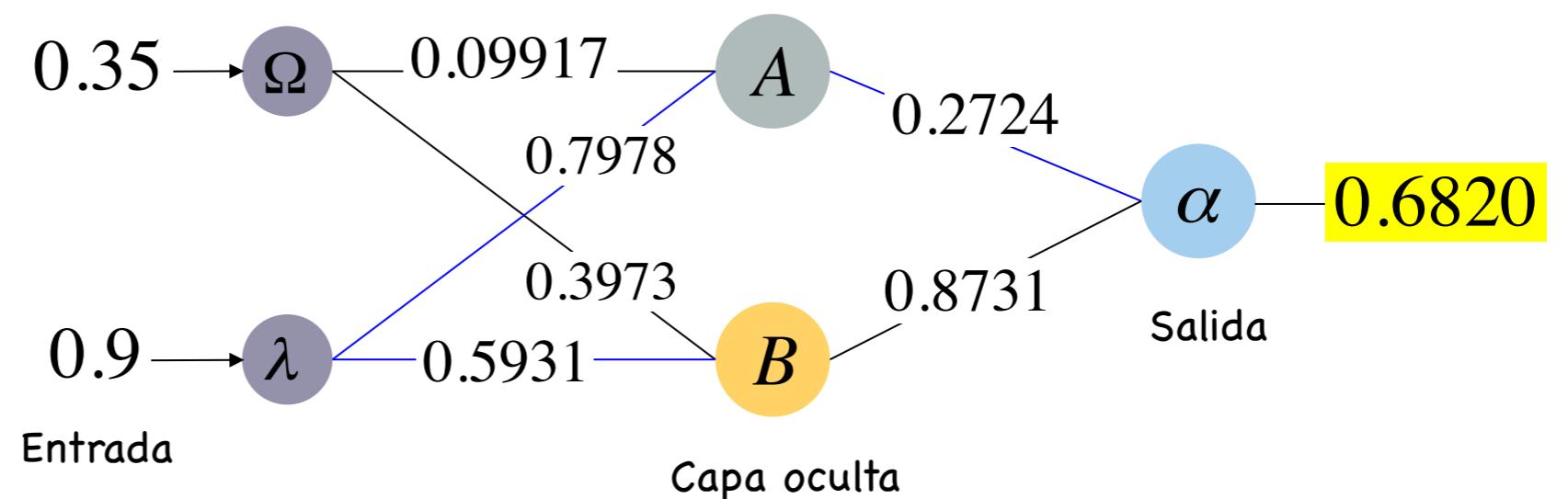
$$\alpha = (0.3 \times 0.68) + (0.9 \times 0.664) = 0.7633 \rightarrow 0.6820$$

## ■ Supervisados



## ■ Backpropagation algorithm

En el ejemplo, supongamos que la salida deseada o *goal* es 0.5, asumiendo que la función de activación es sigmoidal



R

Objetivo: 0.5

Error

Primera iteración:  $\alpha_1 = 0.6900$   $err_1 = 0.690 - 0.5 = 0.190$

Segunda iteración:  $\alpha_2 = 0.6820$   $err_1 = 0.682 - 0.5 = 0.182$

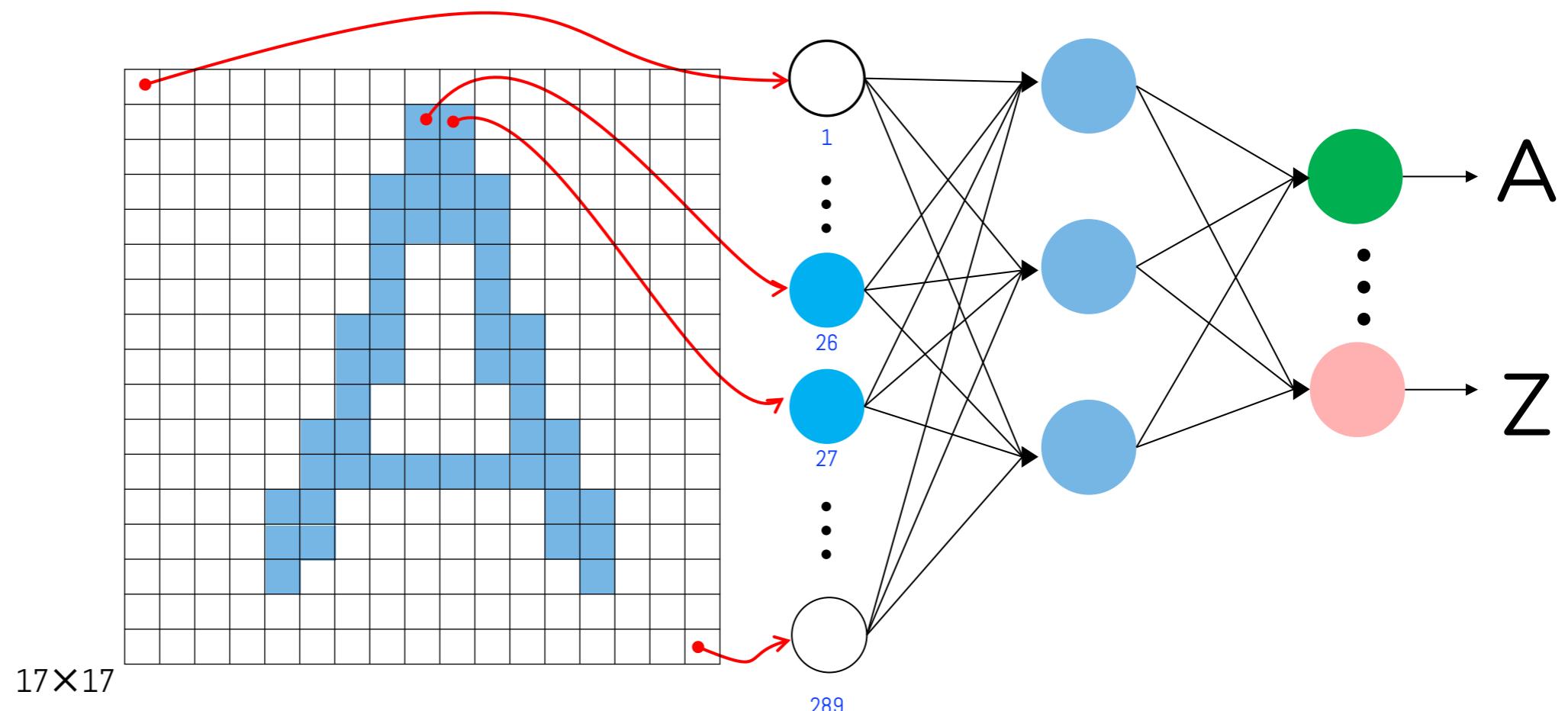
Como vemos, en cada iteración el error se reduce

## ■ Supervisados



### ■ Limitaciones de las Redes Neuronales en imágenes

Al aplicar redes neuronales sobre una imagen digital, enfrentamos problemas como rotación, desplazamiento, escala o cualquier tipo de distorsión que hace inviable cualquier tipo de aprendizaje

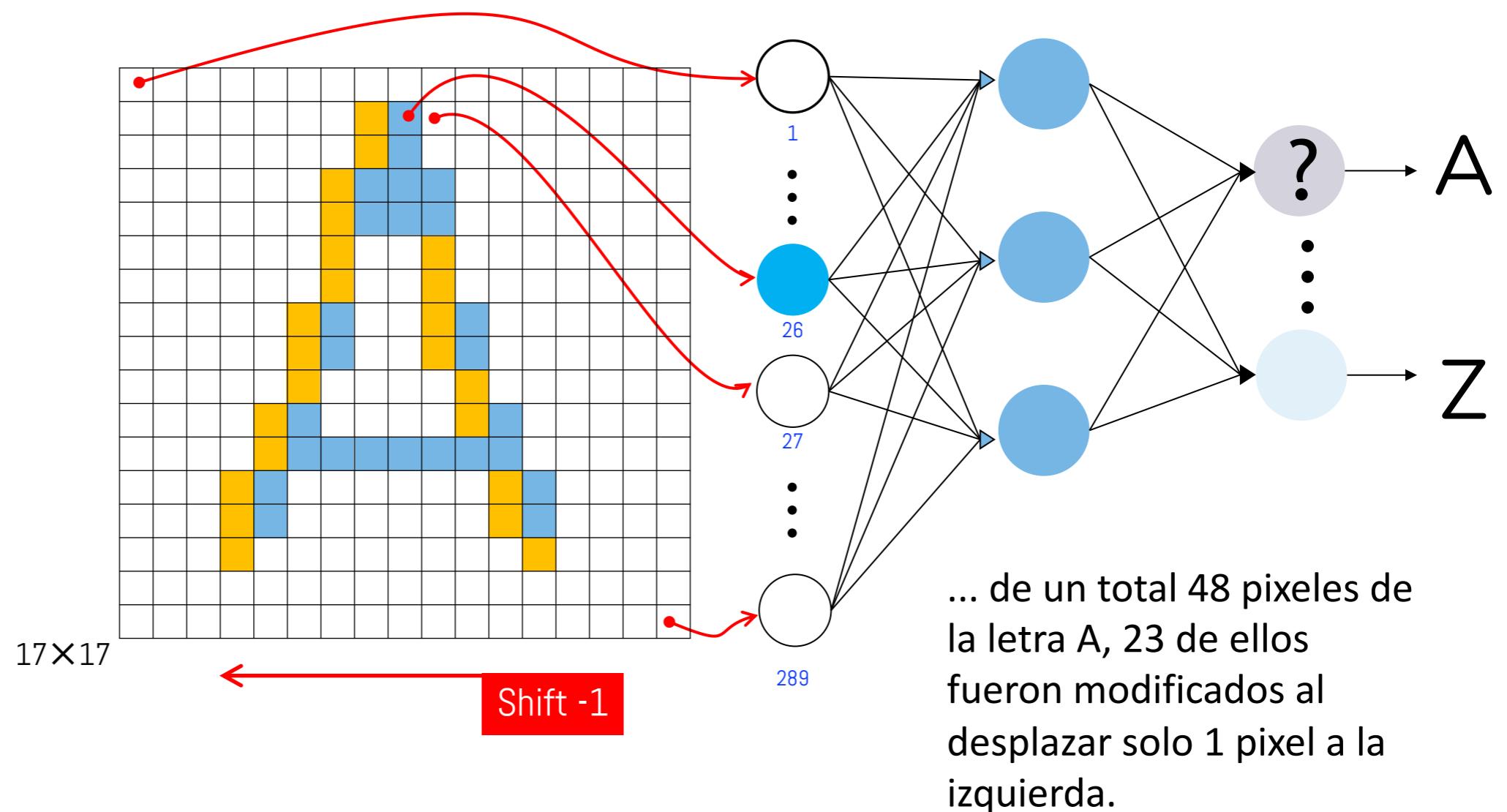


## ■ Supervisados



### ■ Limitaciones de las Redes Neuronales en imágenes

Al aplicar redes neuronales sobre una imagen digital, enfrentamos problemas como rotación, desplazamiento, escala o cualquier tipo de distorsión que hace inviable cualquier tipo de aprendizaje



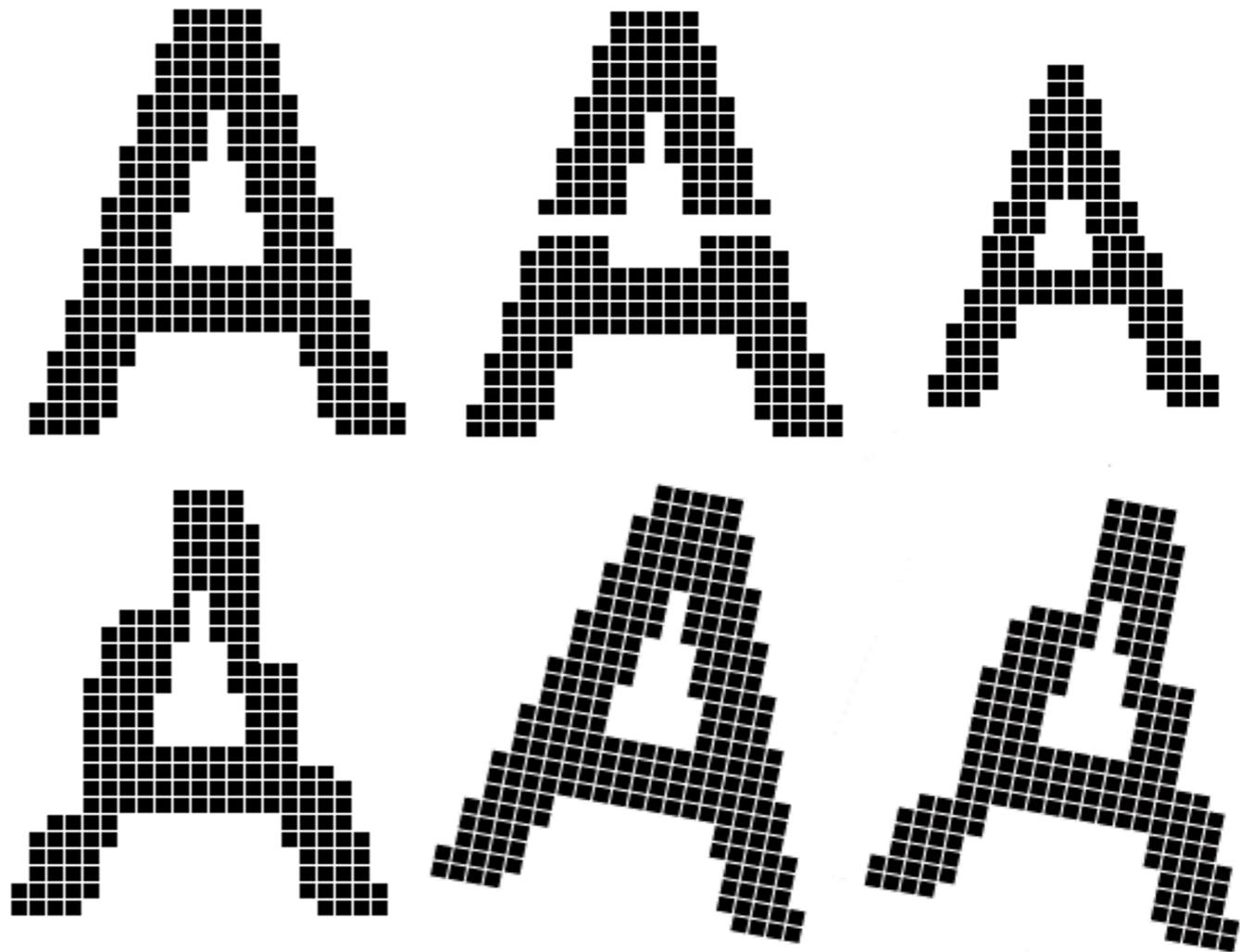
## ■ Supervisados



### ■ Limitaciones de las Redes Neuronales en imágenes

Al aplicar redes neuronales sobre una imagen digital, enfrentamos problemas como rotación, desplazamiento, escala o cualquier tipo de distorsión que hace inviable cualquier tipo de aprendizaje

Cómo podemos observar la topología de los datos de entrada se ignora por completo



*fuente:* Abin – Rozgard, Convolutional neural networks

People with no idea  
about AI, telling me my  
AI will destroy the world

Me wondering why my  
neural network is  
classifying a cat as a dog..



## ■ Supervisados



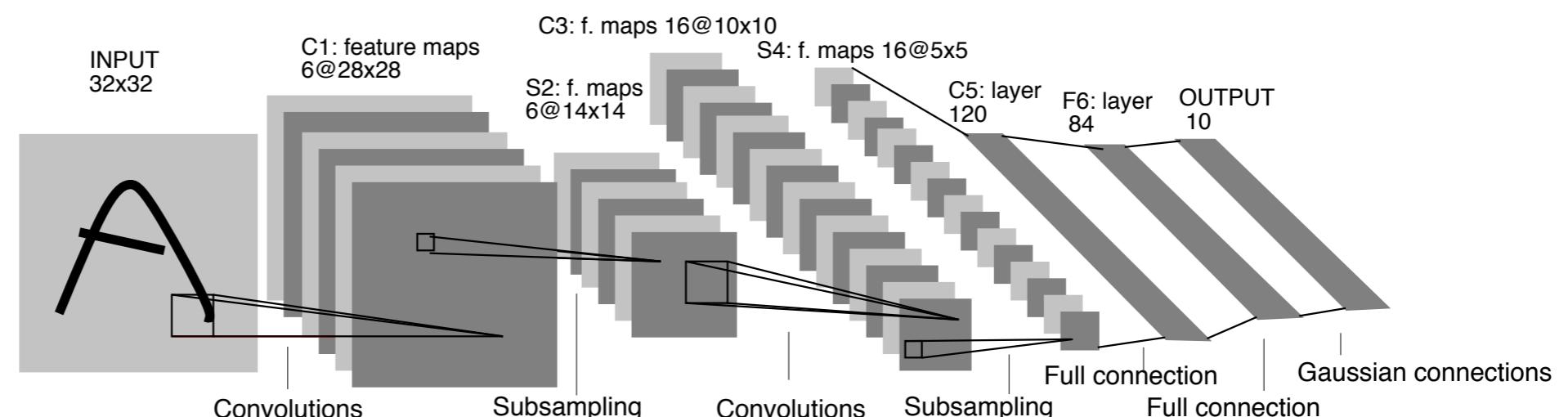
### ■ Convolutional Neural Networks

Los precursores de este tipo de red son Yann LeCun y Yoshua Bengio quienes introducen el concepto de red neuronal convolucional (CNN) en 1995.

Su principal motivación fue **emular células nerviosas localmente sensibles y selectivas a la orientación en la corteza visual.**



Yann LeCun



Las redes neuronales convolucionales son un tipo especial de redes neuronales multicapa.

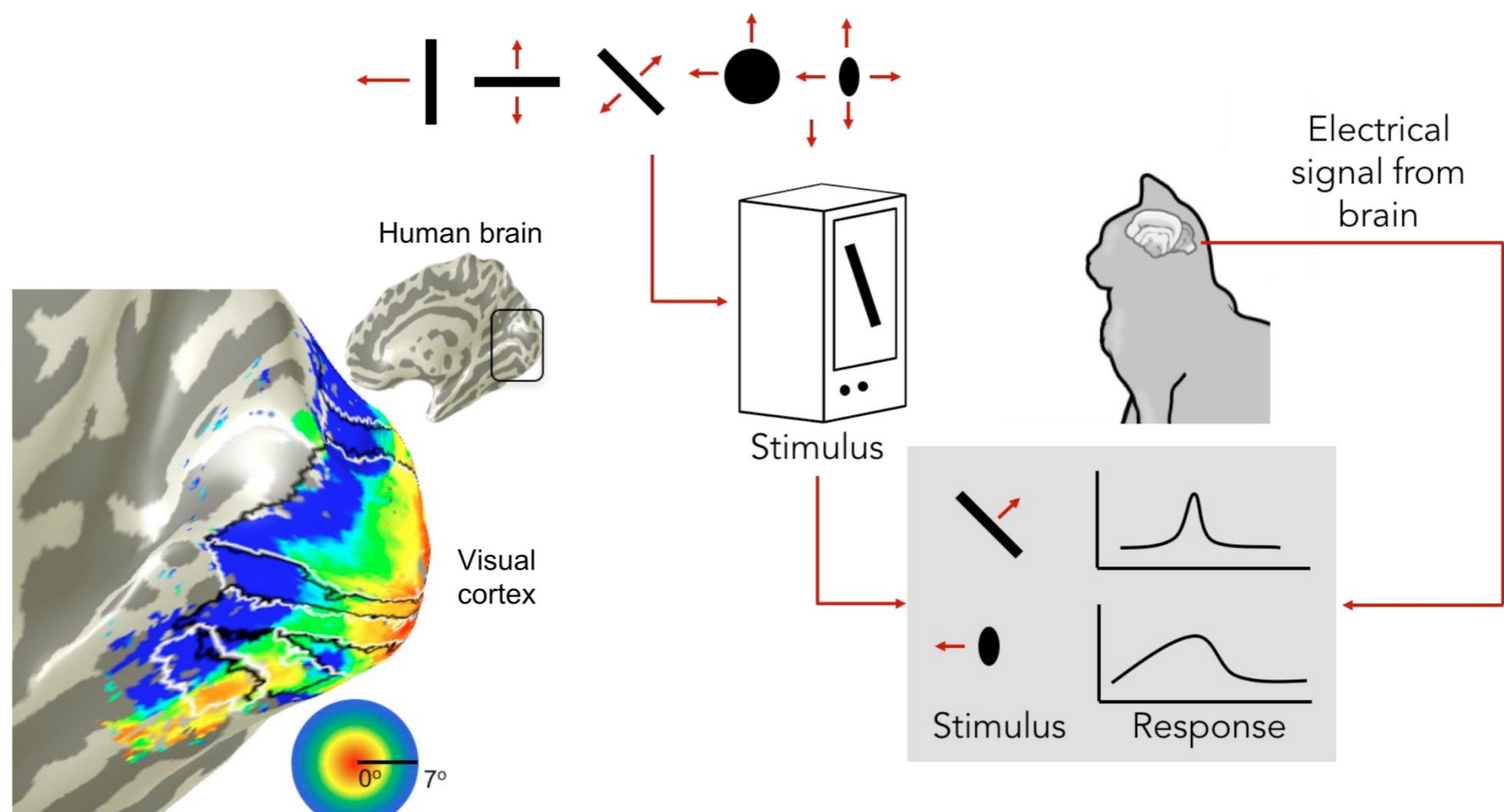
*fuente*: Yann LeCun, Kevin Murphy

## ■ Supervisados



### ■ Convolutional Neural Networks

Su principal motivación fue **emular células nerviosas localmente sensibles y selectivas a la orientación en la corteza visual**. Para ello diseñaron una estructura de red que extrae implícitamente características relevantes.



Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

**fuente:** Fei-Fei Li & Justin Johnson & Serena Yeung Cs231\_2017

## ■ Supervisados

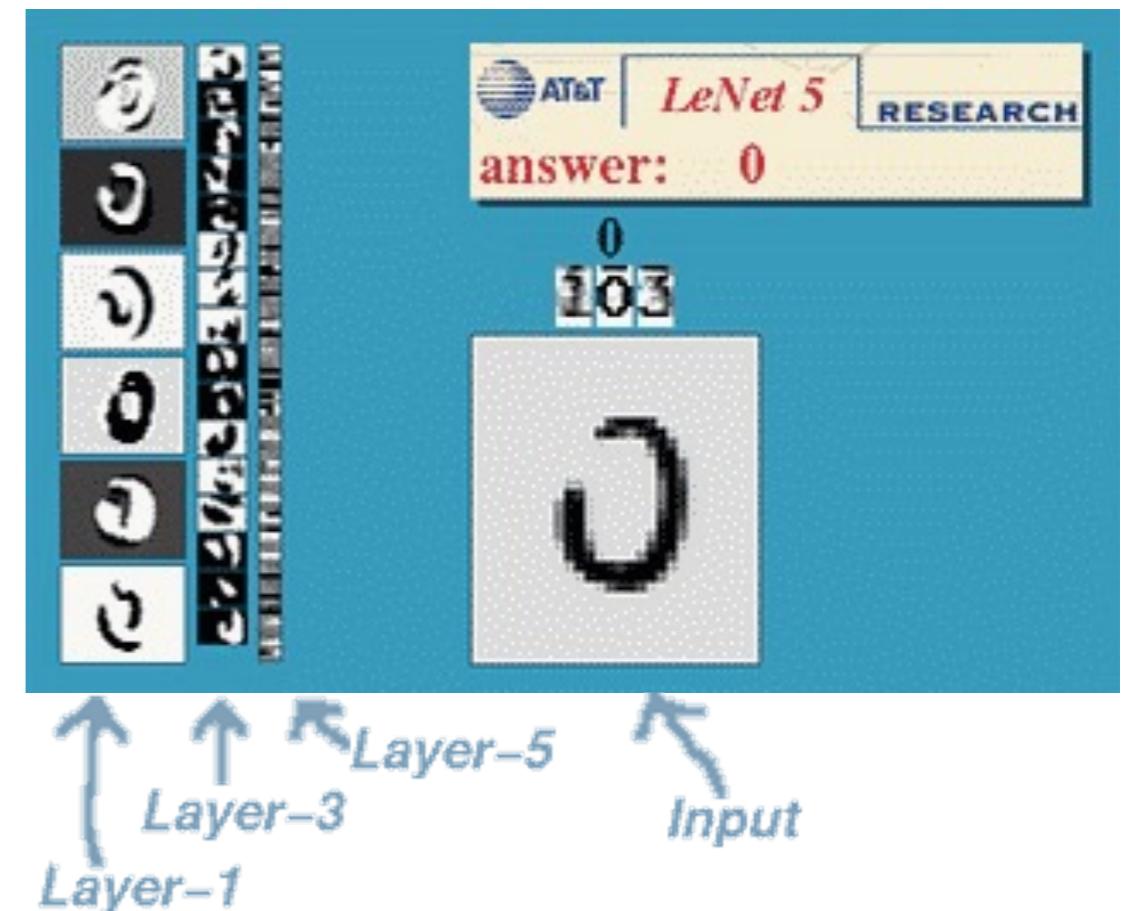


### ■ Convolutional Neural Networks

En forma simplificada las CNNs es una red que puede extraer propiedades topológicas de una imagen. Como casi todas las demás redes neuronales, están implementadas con una versión del algoritmo de retropropagación.

Permiten reconocer patrones visuales directamente a partir de imágenes de píxeles con un preprocesamiento mínimo.

Además reconocer patrones con extrema variabilidad (como caracteres escritos a mano)



*fuente* : Yann LeCun, Kevin Murphy

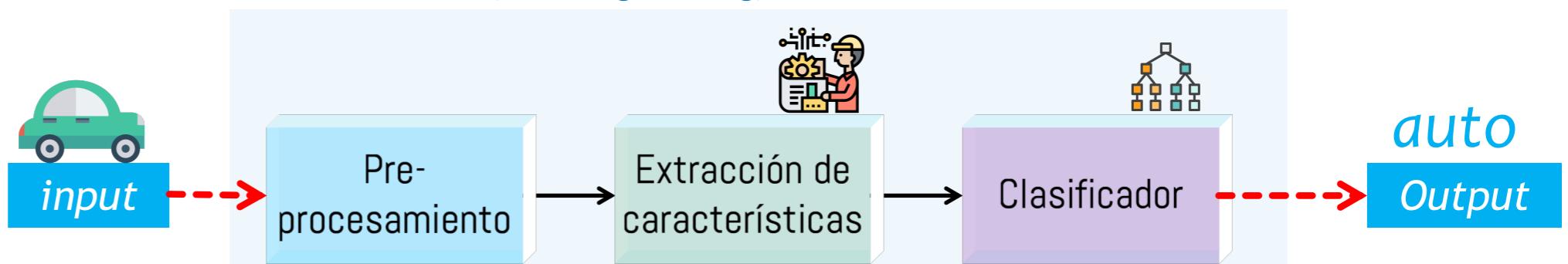
## ■ Supervisados



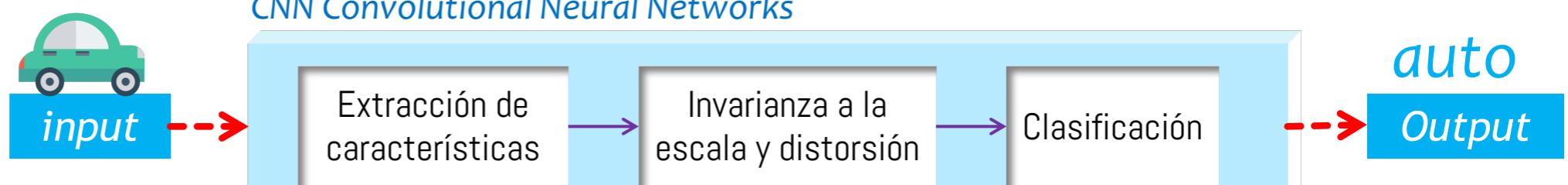
### ■ Convolutional Neural Networks

A diferencia de los métodos clásicos de extracción de características, las redes de tipo CNN se encargan de extraer los patrones de interés en forma interna a la red.

*Proceso manual (hard engineering)*



*CNN Convolutional Neural Networks*



Este proceso tiene grandes ventajas ya que permite extraer patrones que no han diseñados en forma explícita. De esta forma, la propia red es capaz de aprender por si misma patrones.

## ■ Supervisados

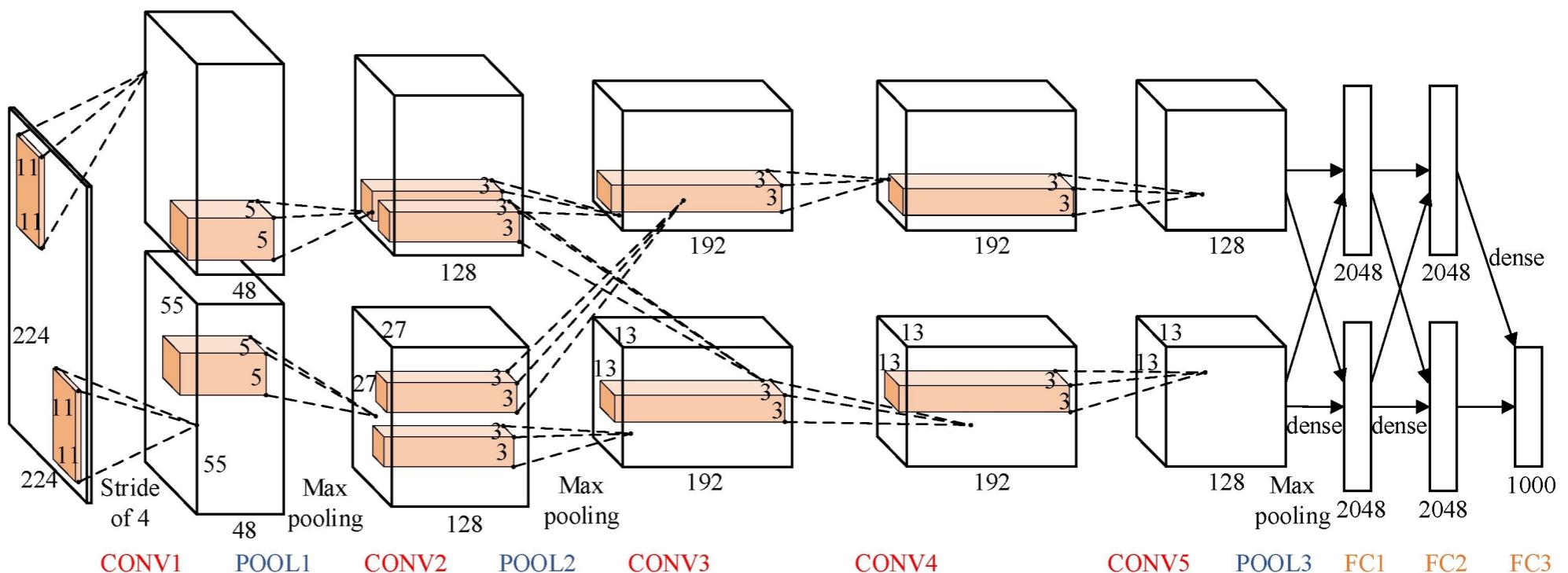


### ■ Convolutional Neural Networks

No fue hasta el año 2012 cuando Alex Krizhevsky en colaboración con Ilya Sutskever y Geoffrey Hinton ganaron la competencia imageNet Large Scale Visual Recognition Challenge en septiembre de 2012 con la red CNN denominada Alexnet.



Alex Krizhevsky



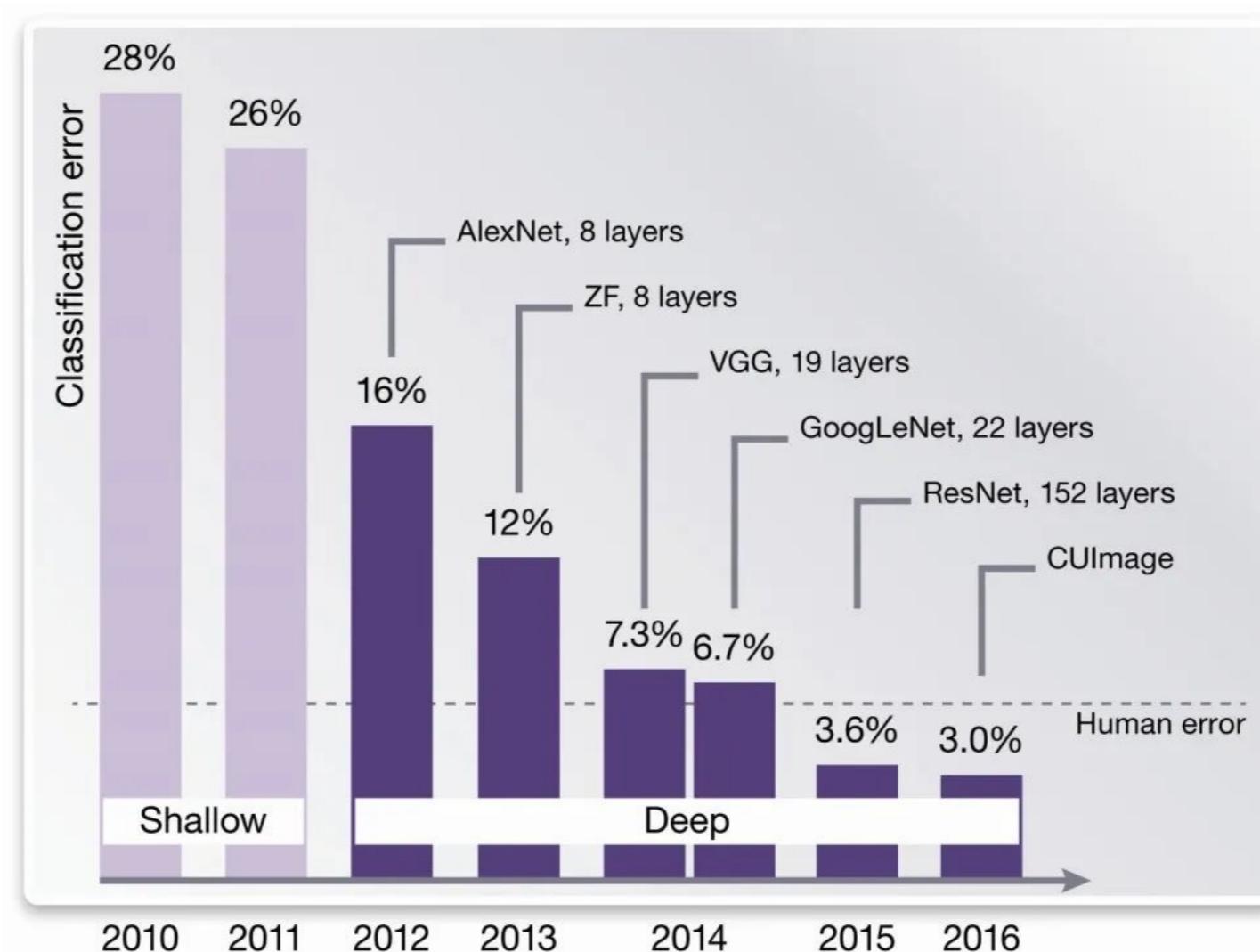
*fuente:* Alexnet (2012) <https://www.mdpi.com/2079-9292/8/3/295>

## ■ Supervisados



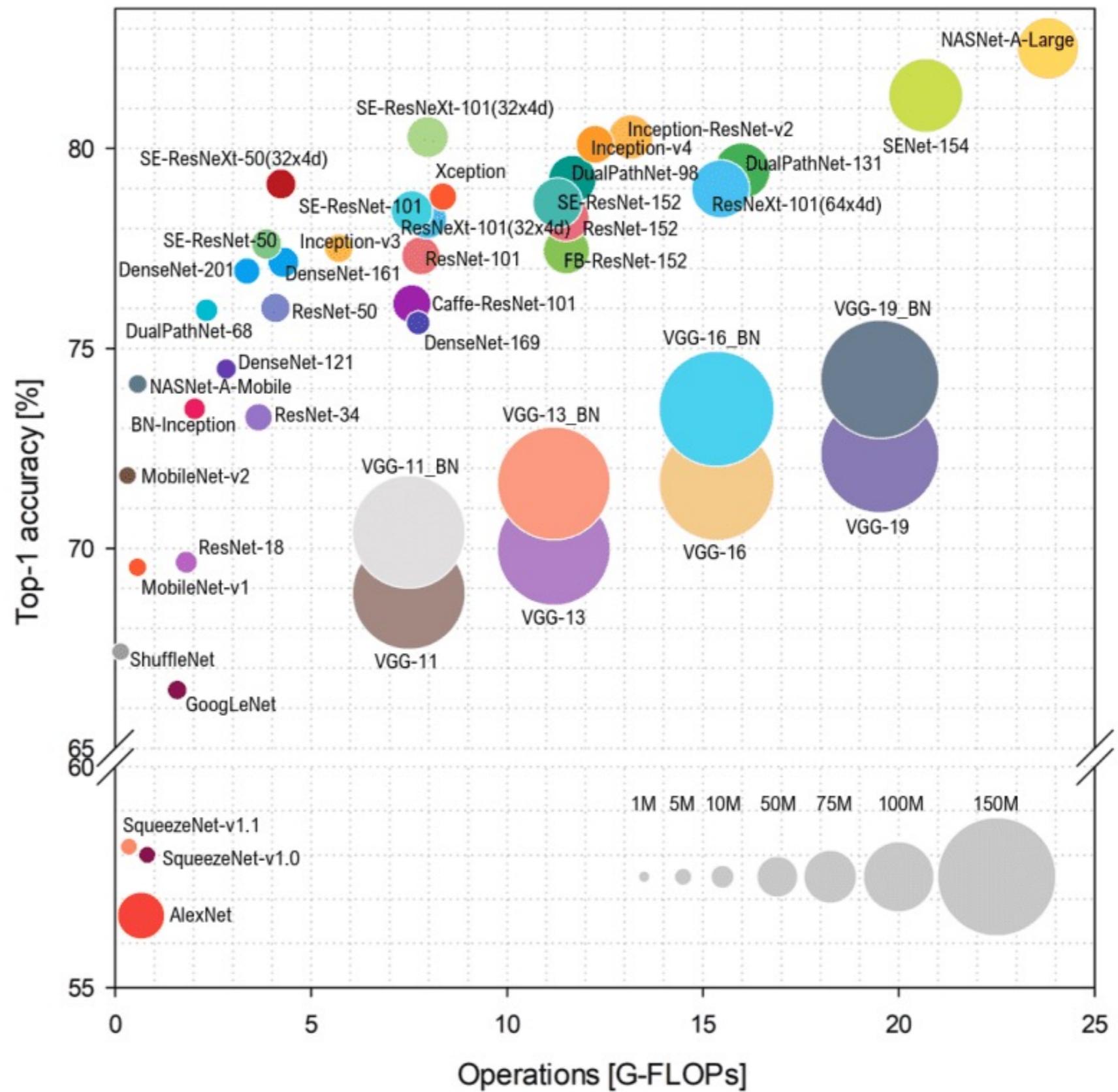
### ■ Convolutional Neural Networks

Desde la década de 2010, las CNN profundas se conocen principalmente como aprendizaje profundo, y prospera en el área de la visión por computador hasta la actualidad.



*fuente:* Gordon Cooper, <https://semiengineering.com/software-framework-requirements-for-embedded-vision/>

## ■ Supervisados

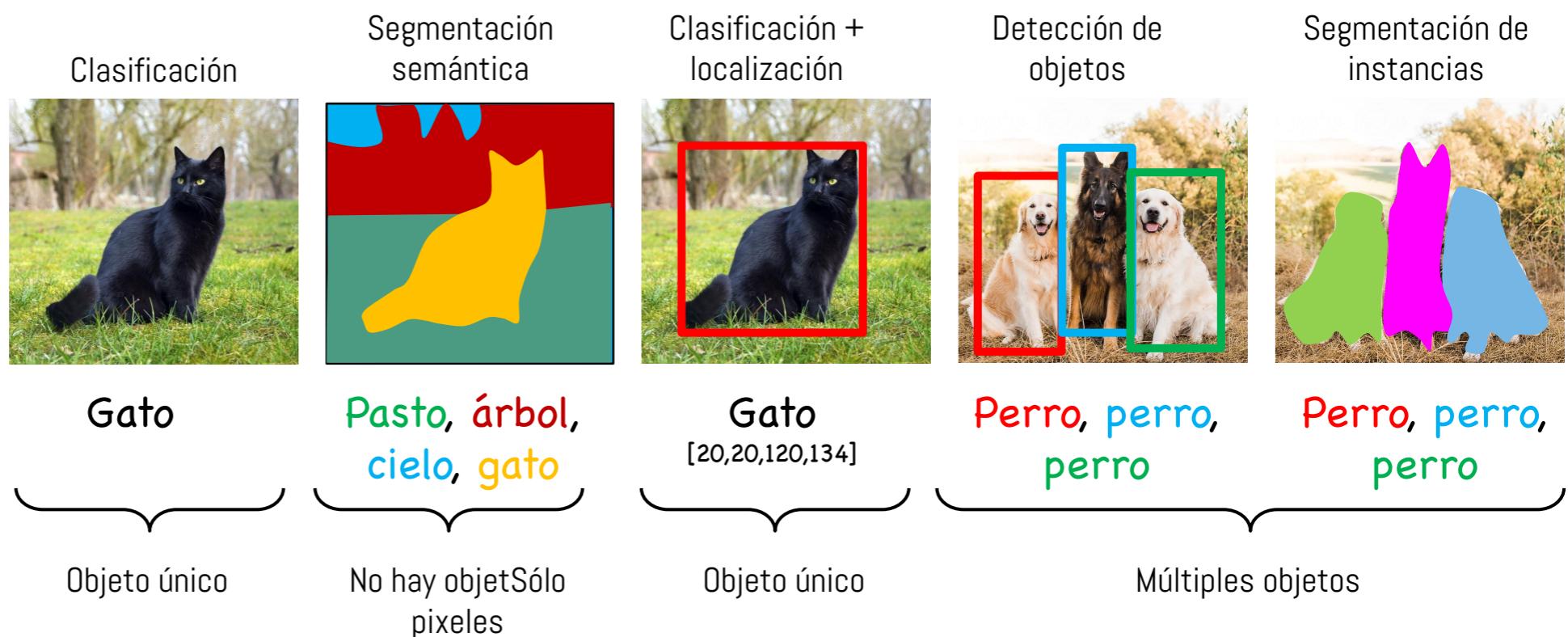


## ■ Supervisados



### ■ Convolutional Neural Networks

Las redes de tipo CNN han logrado resolver complejos problemas en el área de la visión por computador ya que permiten no solo realizar una tarea de clasificación, sino muchas otras tareas como la segmentación, localización, detección, segmentación de instancias entre otras.



### ■ Supervisados

kNN

Sin errores



*A white teddy bear sitting in the grass*

Redes  
Neuronales



*A man riding a wave on top of a surfboard*

Algo relacionado



*A woman is holding a cat in her hand*

Algunos errores



*A man in a baseball uniform throwing a ball*



*A cat sitting on a suitcase on the floor*



*A woman standing on a beach holding a surfboard*

*fuente* : Fei-Fei Li & Justin Johnson & Serena Yeung Cs231\_2017

Image transfer style

## ■ Supervisados

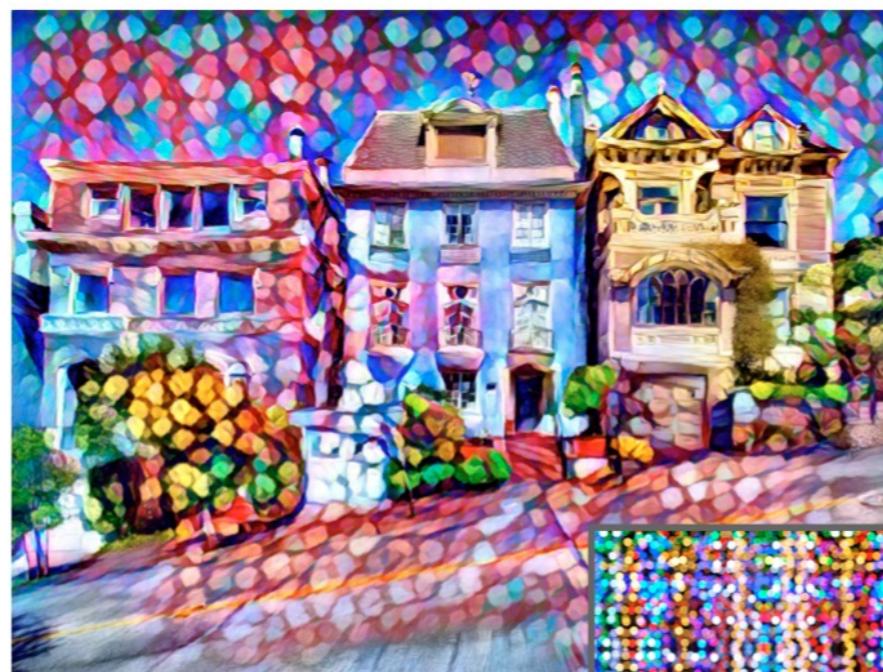
kNN



Mahalanobis



LDA



Árboles de  
decisión



Bayesiano



Redes  
Neuronales



[Original image](#) is CC0 public domain

[Starry Night](#) and [Tree Roots](#) by Van Gogh are in the public domain

[Bokeh image](#) is in the public domain

Stylized images copyright Justin Johnson, 2017;  
reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016  
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

fuente : Fei-Fei Li & Justin Johnson & Serena Yeung Cs231\_2017

## ■ Supervisados

kNN

Mahalanobis

LDA

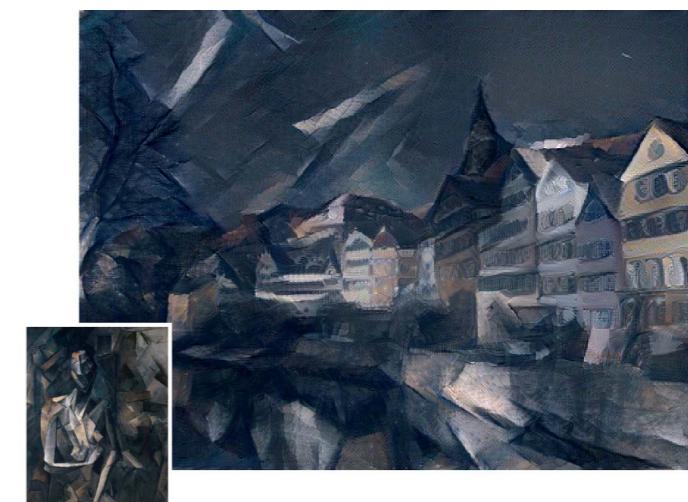
Árboles de  
decisión

Bayesiano

Redes  
Neuronales

### ■ Convolutional Neural Networks

Dentro del mundo del arte, una de las aplicaciones más interesantes tiene relación con la transferencia de estilo de obras artísticas sobre imágenes reales.



*fuente :* [http://bethgelab.org/research/machine\\_learning/style\\_transfer/](http://bethgelab.org/research/machine_learning/style_transfer/)

Deep fake videos



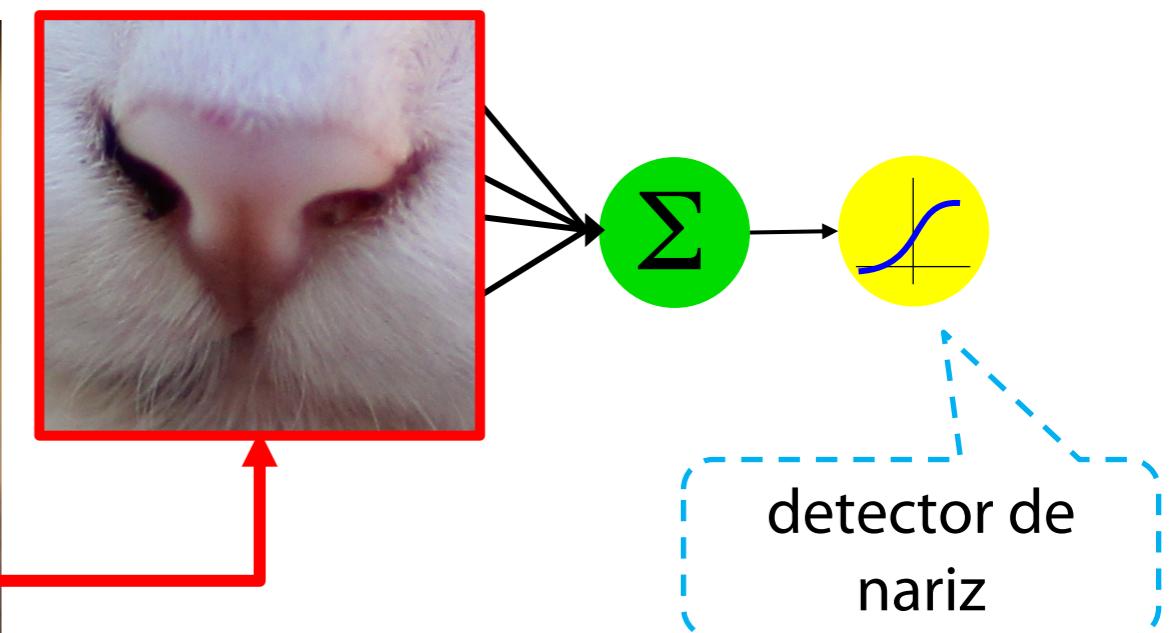
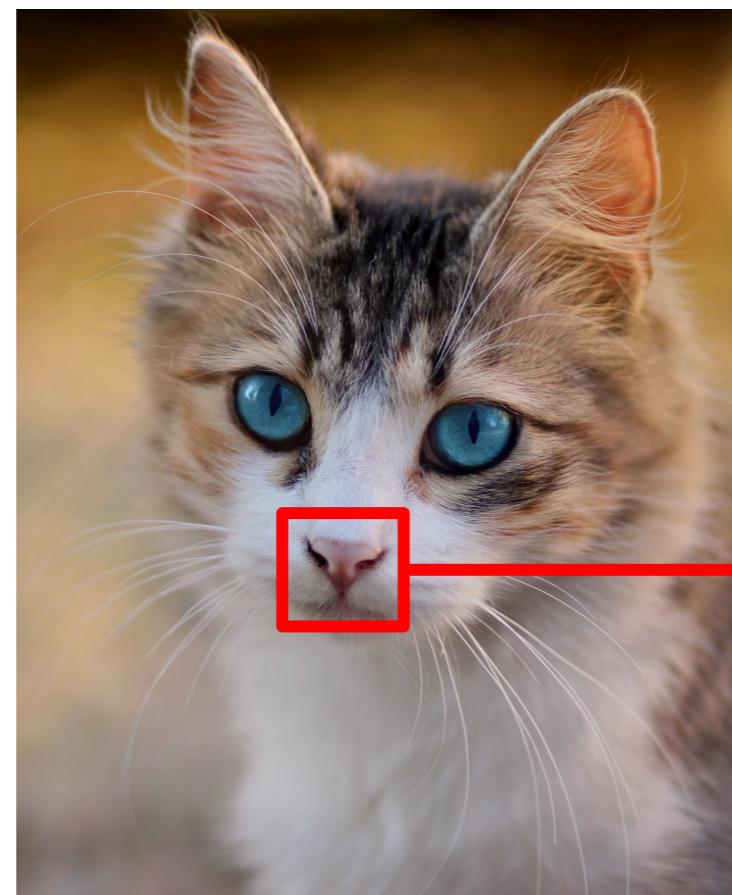
**fuentE:** <https://www.youtube.com/watch?v=51uHNgnnLWI&t=106s>

### ■ Supervisados



#### ■ Convolutional Neural Networks

Algunos patrones son más pequeños que el resto de la imagen, ¿Es posible representar una región pequeña con un menor número de parámetros?



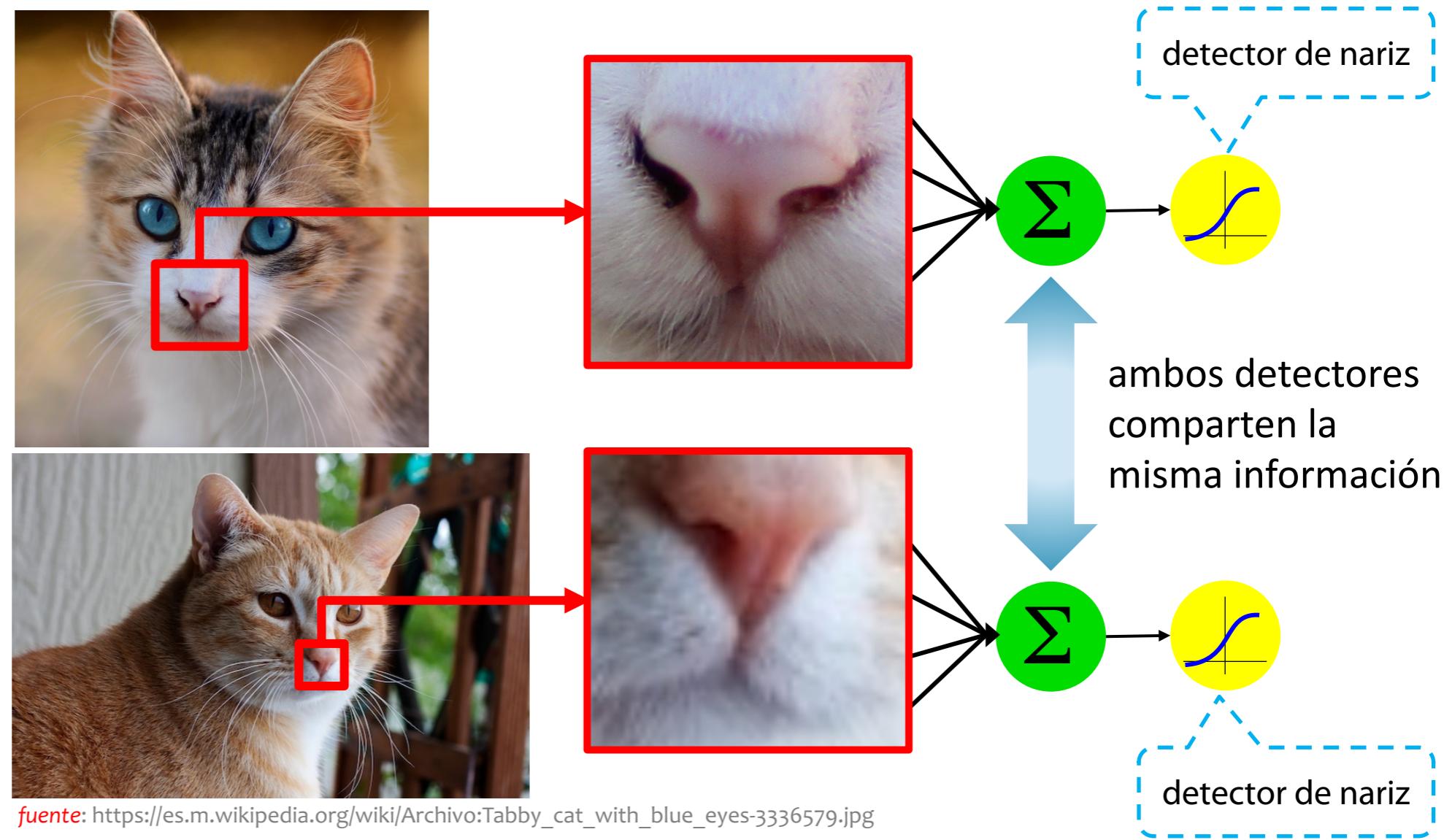
*fuente:* [https://es.m.wikipedia.org/wiki/Archivo:Tabby\\_cat\\_with\\_blue\\_eyes-3336579.jpg](https://es.m.wikipedia.org/wiki/Archivo:Tabby_cat_with_blue_eyes-3336579.jpg)

### ■ Supervisados



#### ■ Convolutional Neural Networks

El mismo patrón aparece en diferentes lugares: ¿se pueden comprimir? ¿Qué ventaja tendría emplear muchos de estos detectores "pequeños" y cada detector pueda "moverse" dentro de la imagen?

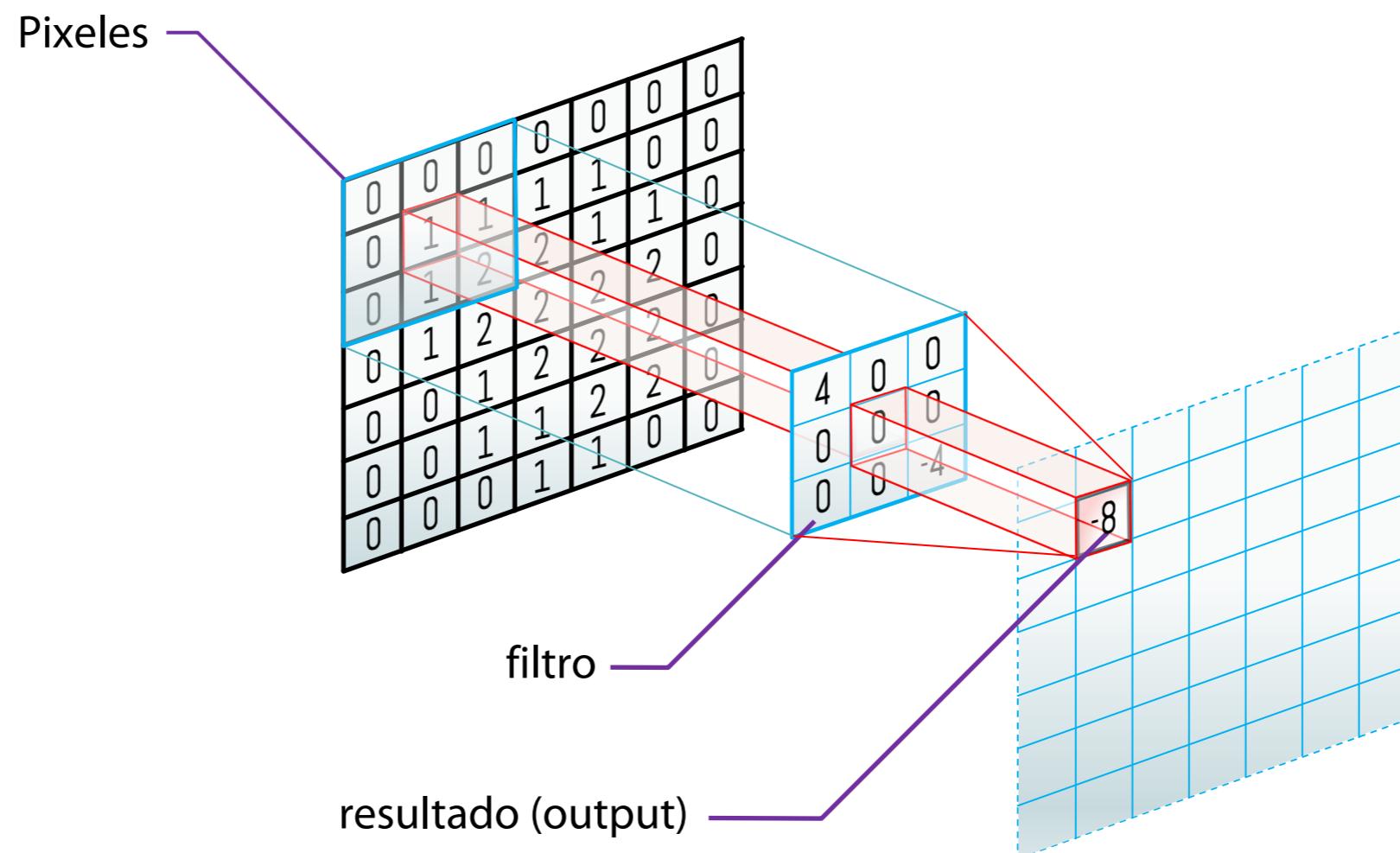


### ■ Supervisados



#### ■ Convolutional Neural Networks

Una CNN es una red neuronal con **algunas capas convolucionales** (y otras capas adicionales). Una capa convolucional tiene un número de filtros que hace operación convolucional.

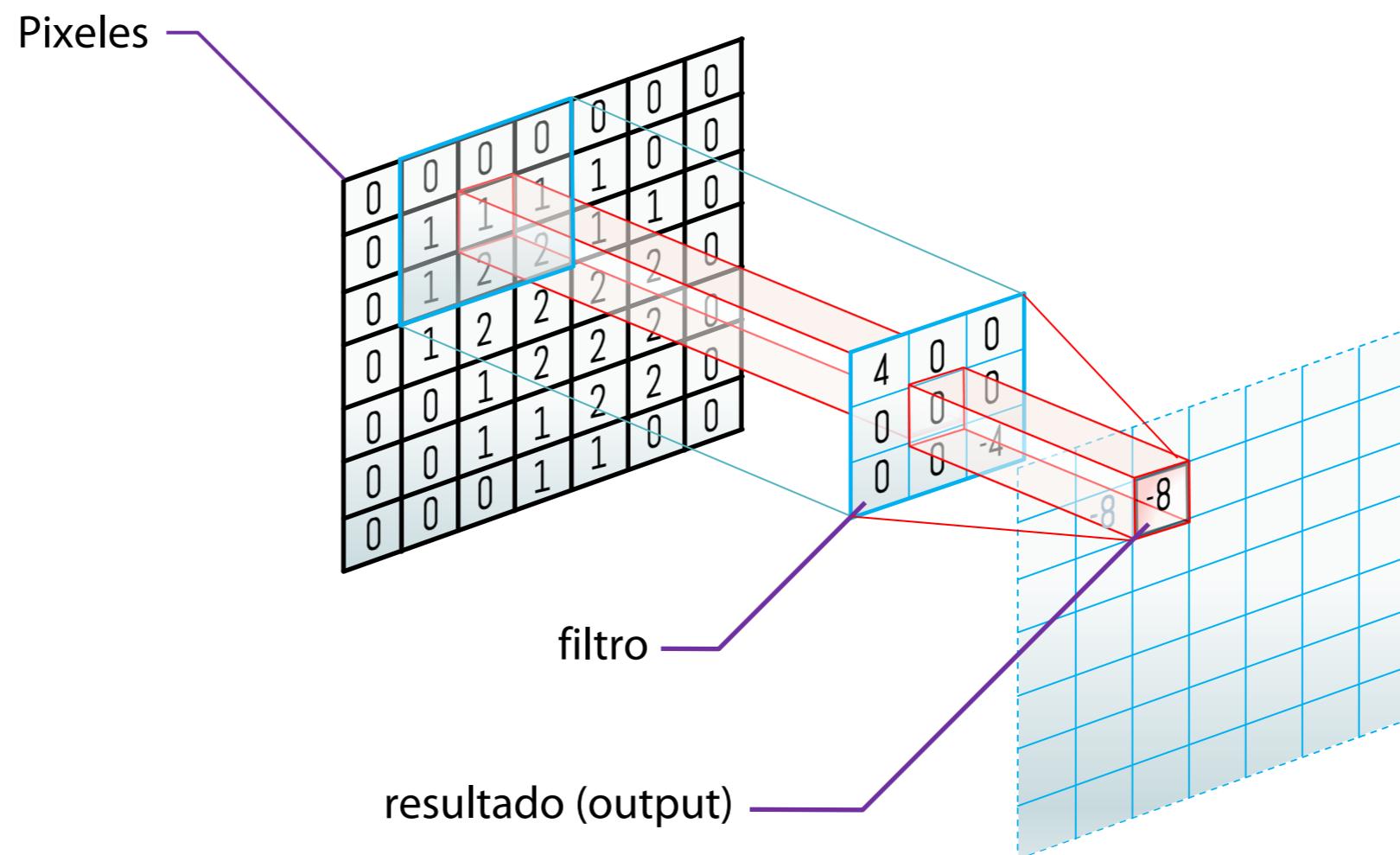


### ■ Supervisados



#### ■ Convolutional Neural Networks

Una CNN es una red neuronal con **algunas capas convolucionales** (y otras capas adicionales). Una capa convolucional tiene un número de filtros que hace operación convolucional.

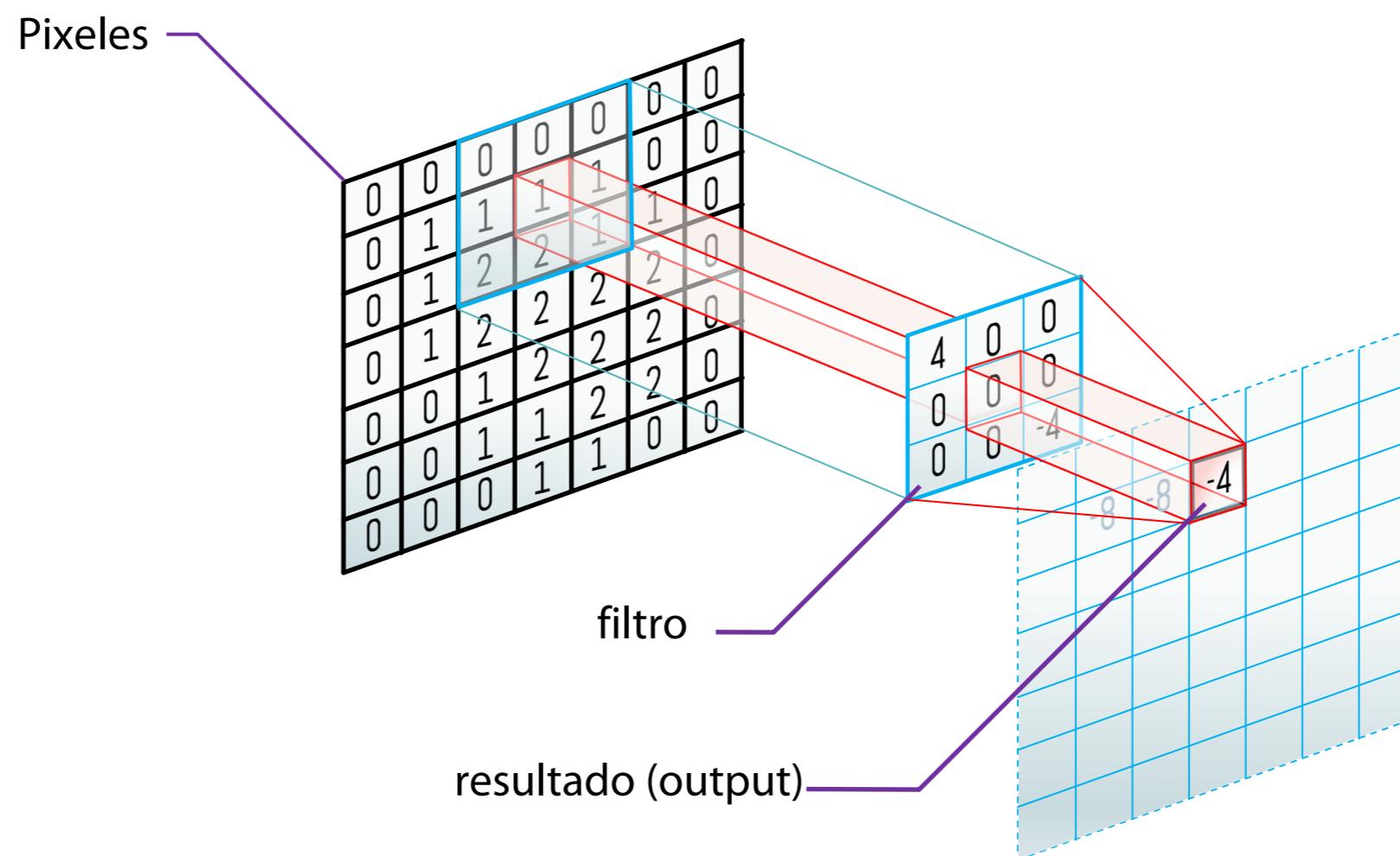


### ■ Supervisados



#### ■ Convolutional Neural Networks

Una CNN es una red neuronal con **algunas capas convolucionales** (y otras capas adicionales). Una capa convolucional tiene un número de filtros que hace operación convolucional.



demo: [https://setosa.io/ev/image-kernels/?utm\\_source=pocket\\_mylist](https://setosa.io/ev/image-kernels/?utm_source=pocket_mylist)

### ■ Supervisados



#### ■ Convolutional Neural Networks

Distintos filtros son aplicados sobre la imagen. Como resultado, obtenemos un patrón distinto según distintas características presente en la imagen.

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

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

Filter 1

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

Filter 2

6 x 6

Cada filtro está compuesto por un kernel pequeño (3 x 3, 5 x 5).

⋮

demo: [https://setosa.io/ev/image-kernels/?utm\\_source=pocket\\_mylist](https://setosa.io/ev/image-kernels/?utm_source=pocket_mylist)

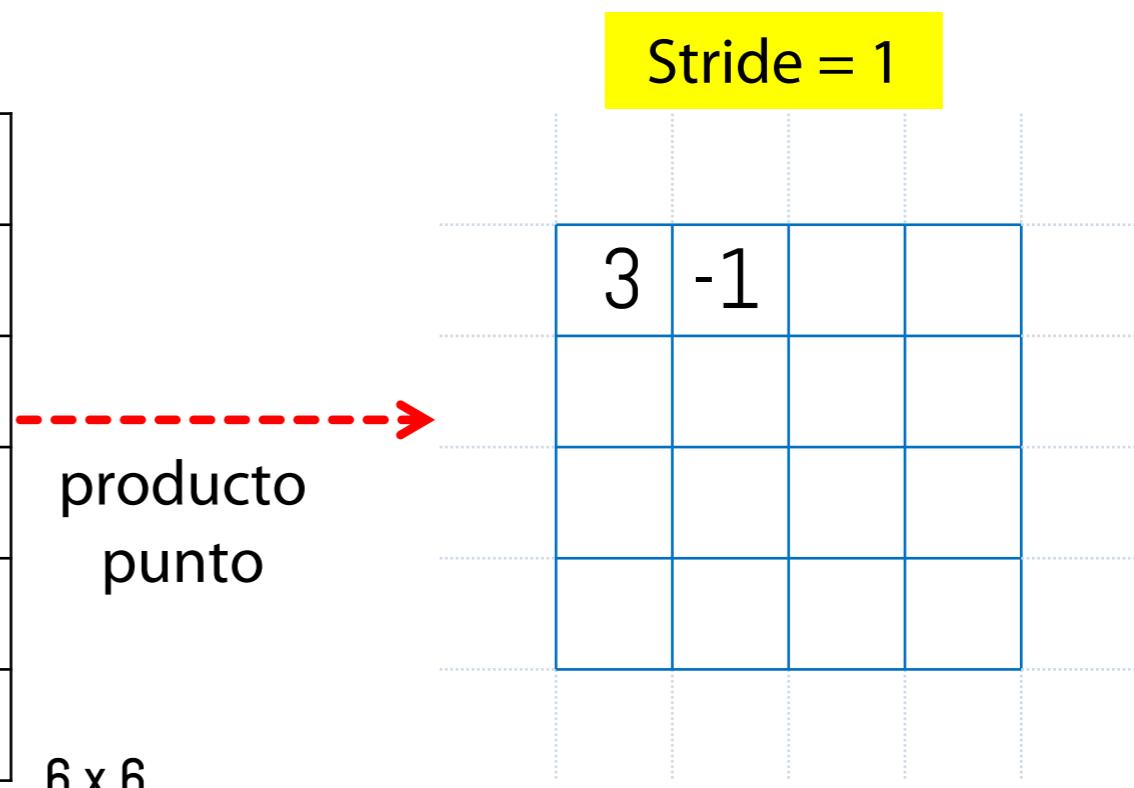
## ■ Supervisados



### ■ Convolución (CNN)

Distintos filtros son aplicados sobre la imagen. Al aplicar un filtro sobre la imagen, debemos definir el parámetro stride, y el tipo de filtro (kernel) que utilizar.

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



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

Filter 1

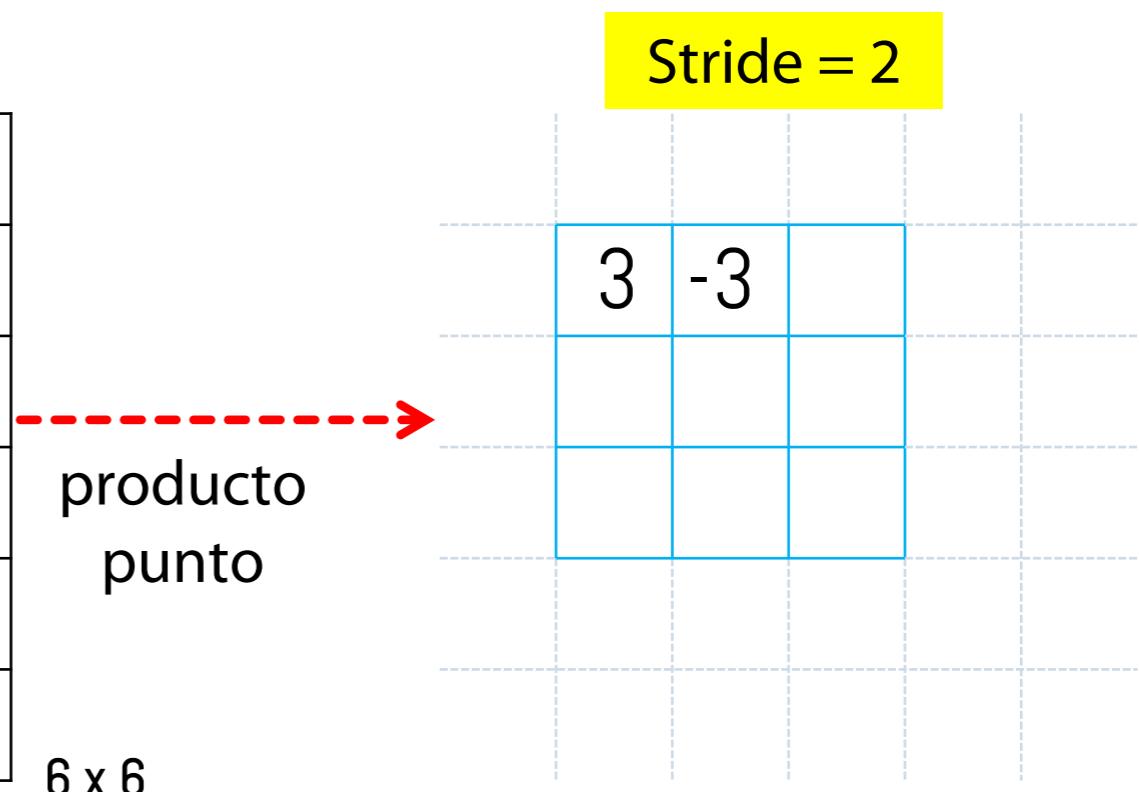
## ■ Supervisados



### ■ Convolución (CNN)

A medida que aumentamos el número de stride, el tamaño de la imagen de salida se reduce. Con un valor de stride=2 la imagen se reduce en un 50%.

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



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

Filter 1

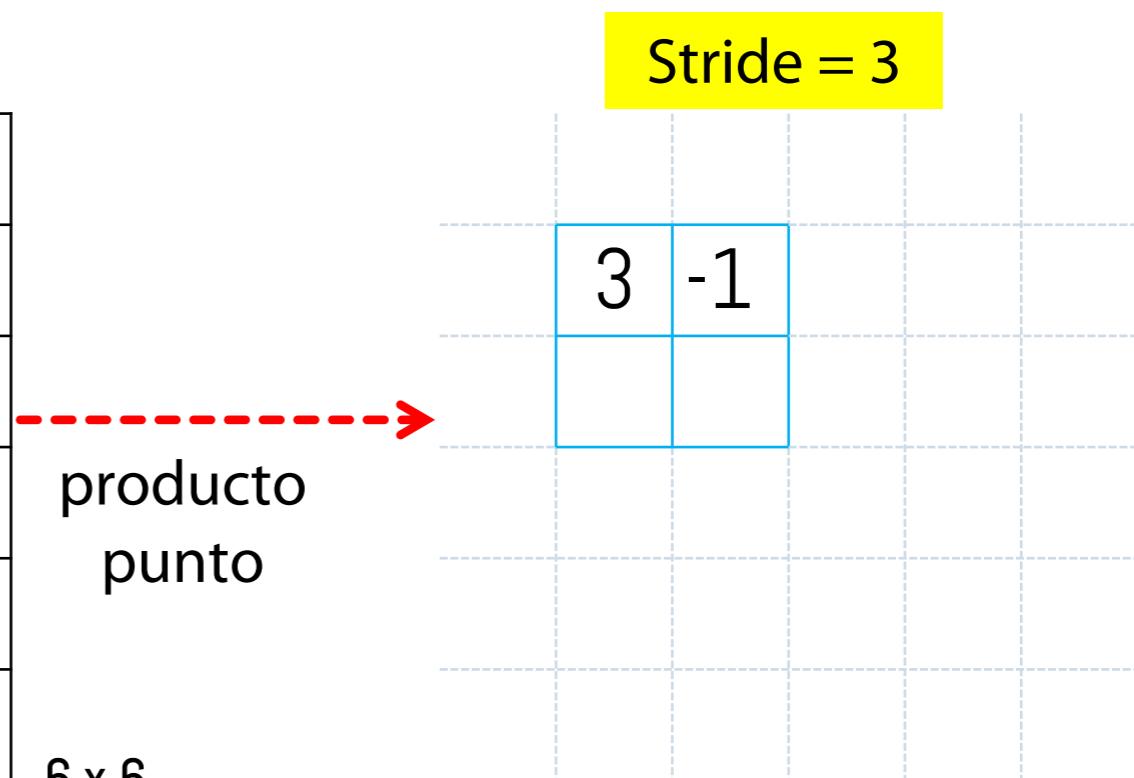
## ■ Supervisados



### ■ Convolución (CNN)

A medida que aumentamos el número de stride, el tamaño de la imagen de salida se reduce. Con un valor de stride=3 la imagen se reduce en un 66%. Sin embargo, en la práctica, no es empleado.

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



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

Filter 1

## ■ Supervisados



### ■ Convolución (CNN)

Como podemos observar, al aplicar el filtro de convolución, estamos determinando distintas propiedades sobre la misma imagen.

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

Filter 1  
producto  
punto

6 x 6

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

Filter 1

Stride = 1

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

## ■ Supervisados



### ■ Convolución (CNN)

Como podemos observar, al aplicar el filtro de convolución, estamos determinando distintas propiedades sobre la misma imagen.

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

Filter 2  
producto  
punto

6 x 6

Stride = 1

-1	-1	-1	-1
-1	-1	-1	1
-1	-1	-2	1
-1	0	-4	3

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

Filter 2

### ■ Supervisados



#### ■ Convolución (CNN)

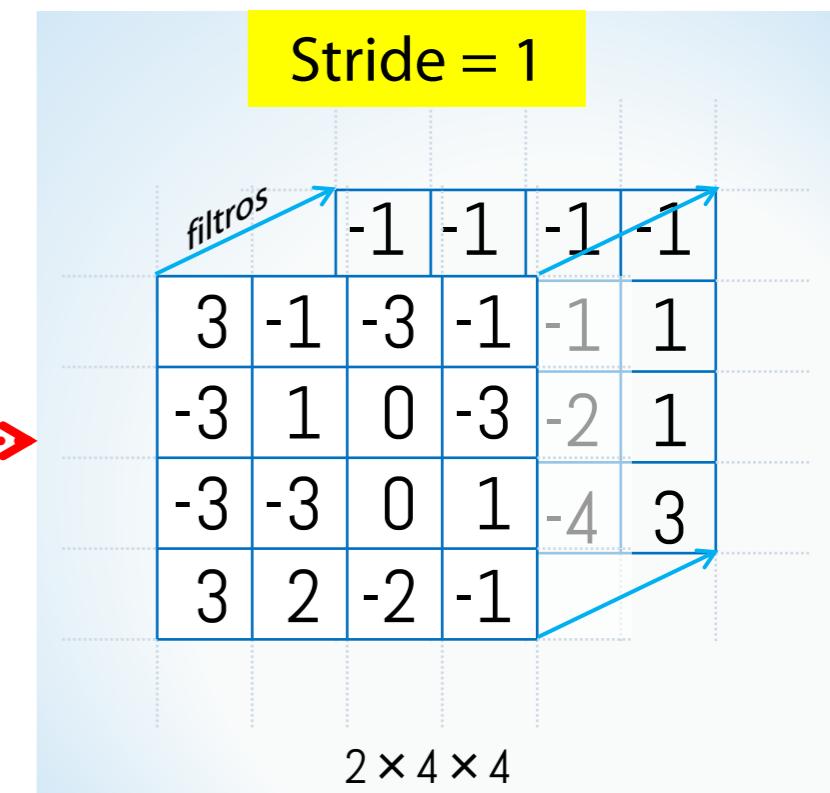
Como podemos observar, al aplicar el filtro de convolución, estamos determinando distintas propiedades sobre la misma imagen.

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

Canal R

producto punto

6 x 6



Filter 1			Filter 2		
1	-1	-1	-1	1	-1
-1	1	-1	-1	1	-1
-1	-1	1	-1	1	-1

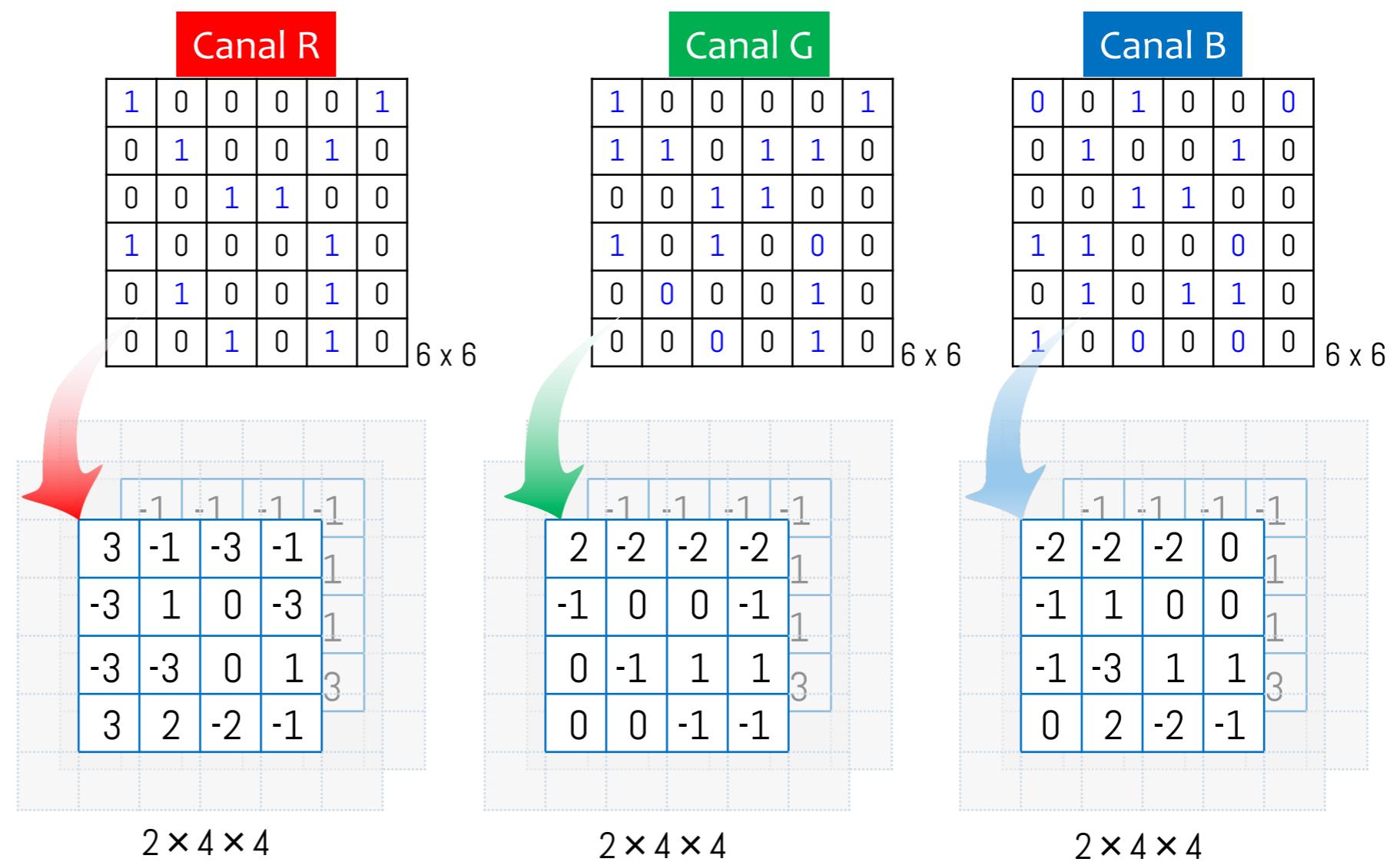
Al aplicar distintos filtros sobre un canal, extraemos distintas propiedades de los objetos

### ■ Supervisados



#### ■ Convolución (CNN)

Aplicamos los mismos filtros sobre cada uno de los canales de la imagen RGB. En cada uno de los canales, obtenemos matrices de convolución.

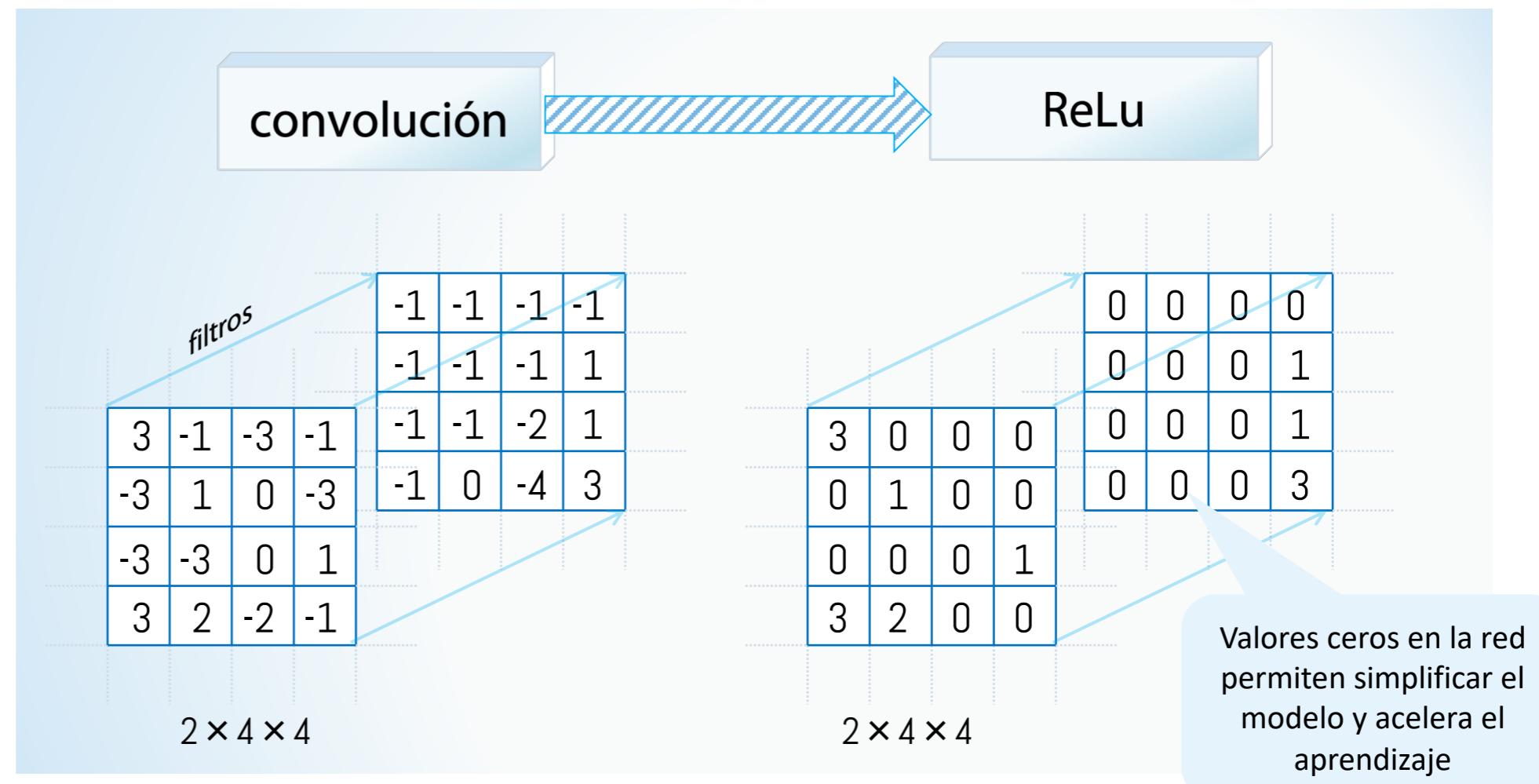


### ■ Supervisados



#### ■ Rectified Linear Unit (ReLU)

Este proceso se aplica luego de realizar una convolución. Permite fijar a cero todos aquellos valores que sean negativos como resultado de la convolución.



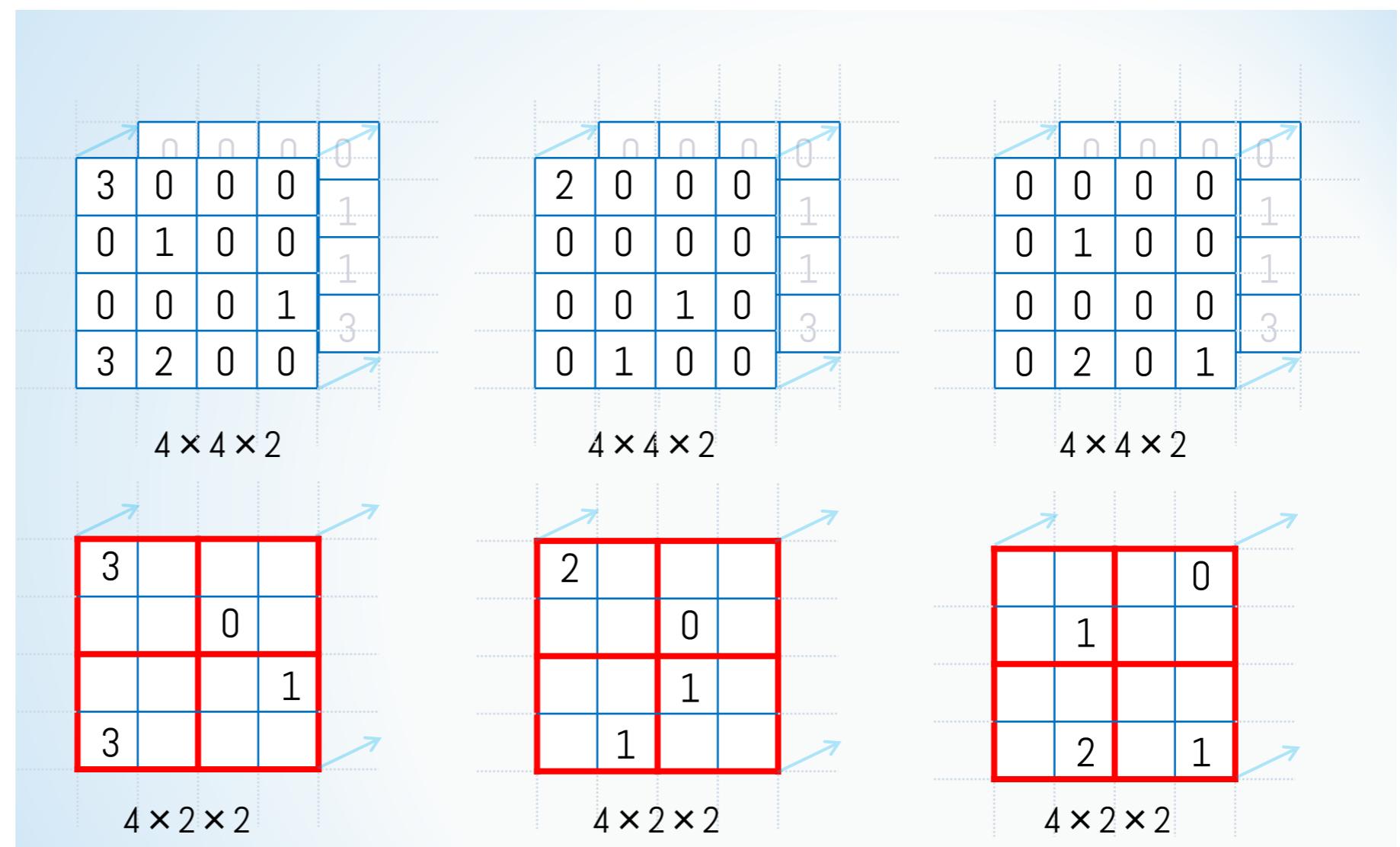
más info en : <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>

### ■ Supervisados



#### ■ Max Pooling (CNN)

Una vez aplicado los filtros, empleamos la estrategia de calcular el máximo en una determinada ventana ( $2 \times 2$ ). Lo anterior reduce la escala de la imagen, **disminuyendo el número de parámetros**.



### ■ Supervisados

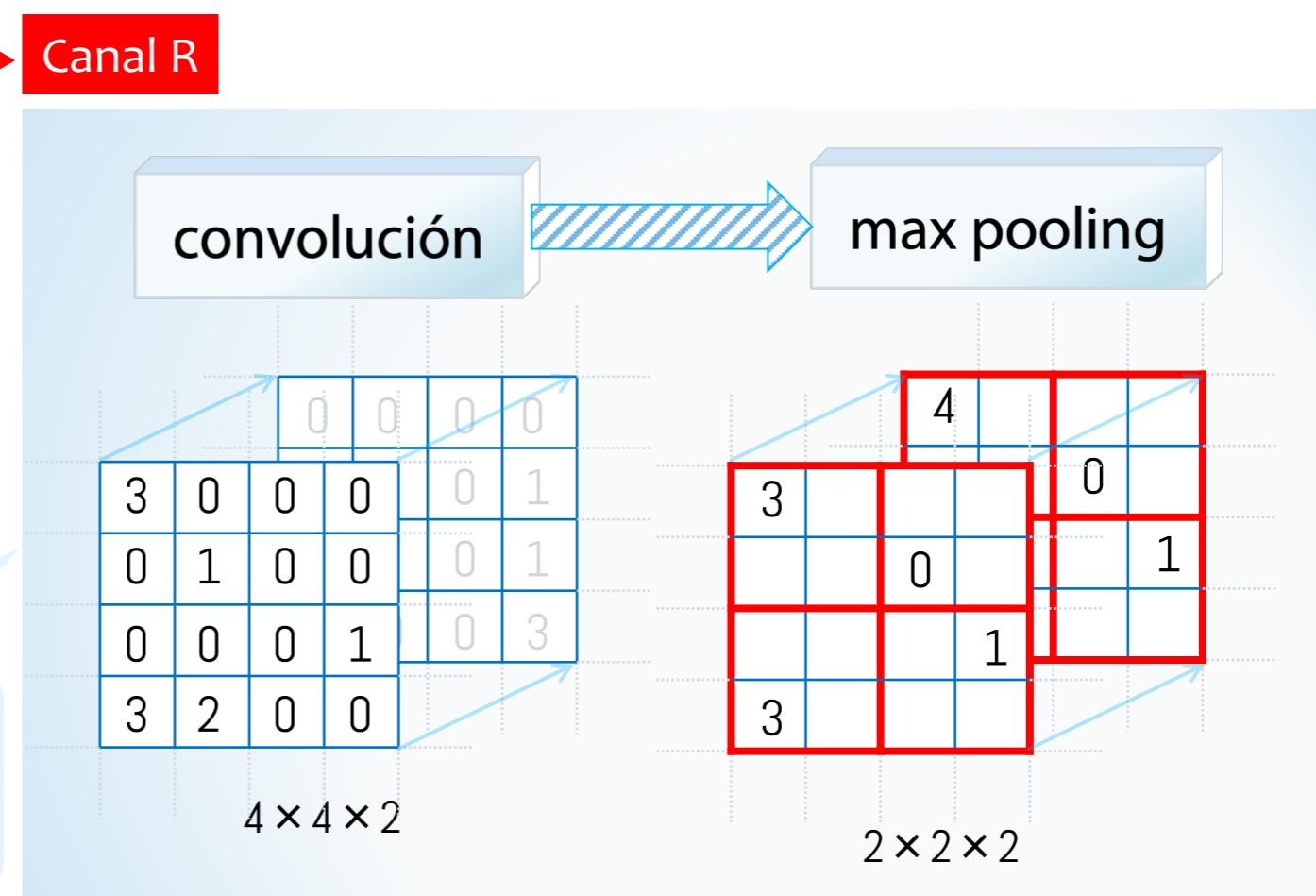


#### ■ Max Pooling (CNN)

Este proceso debe ser aplicado para cada uno de los canales. El proceso permite reducir el número de parámetros de la red y preservar las estructuras más relevantes de la imagen (**reduciendo la complejidad de la red**)



Esto genera como resultado una imagen más pequeña

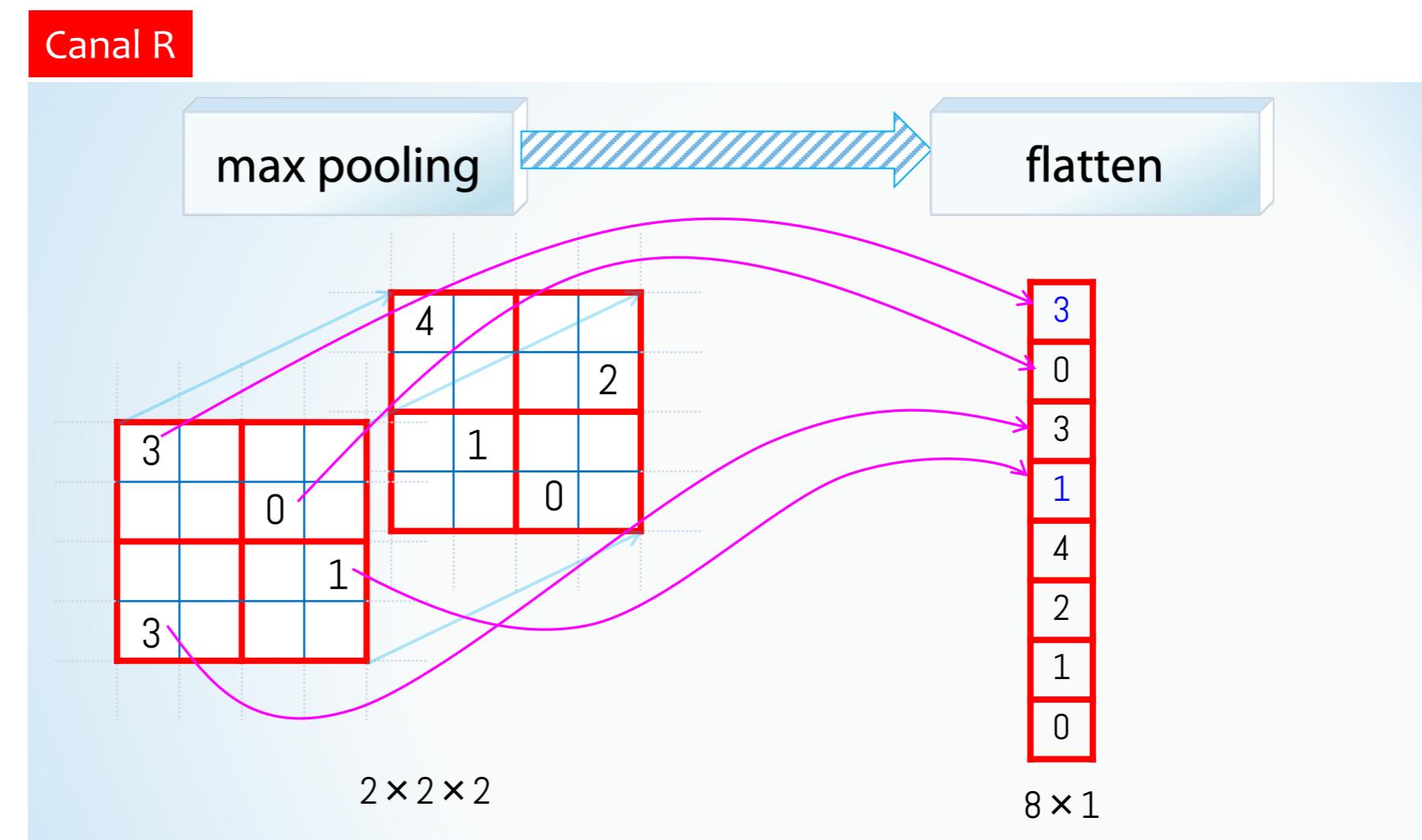


### ■ Supervisados



#### ■ Flateded (CNN)

Este proceso permite que todos los valores correspondientes a la capa anterior de max pooling sean reordenadas en un solo vector. De esta forma podemos emplear el vector como input a una red neuronal densa.

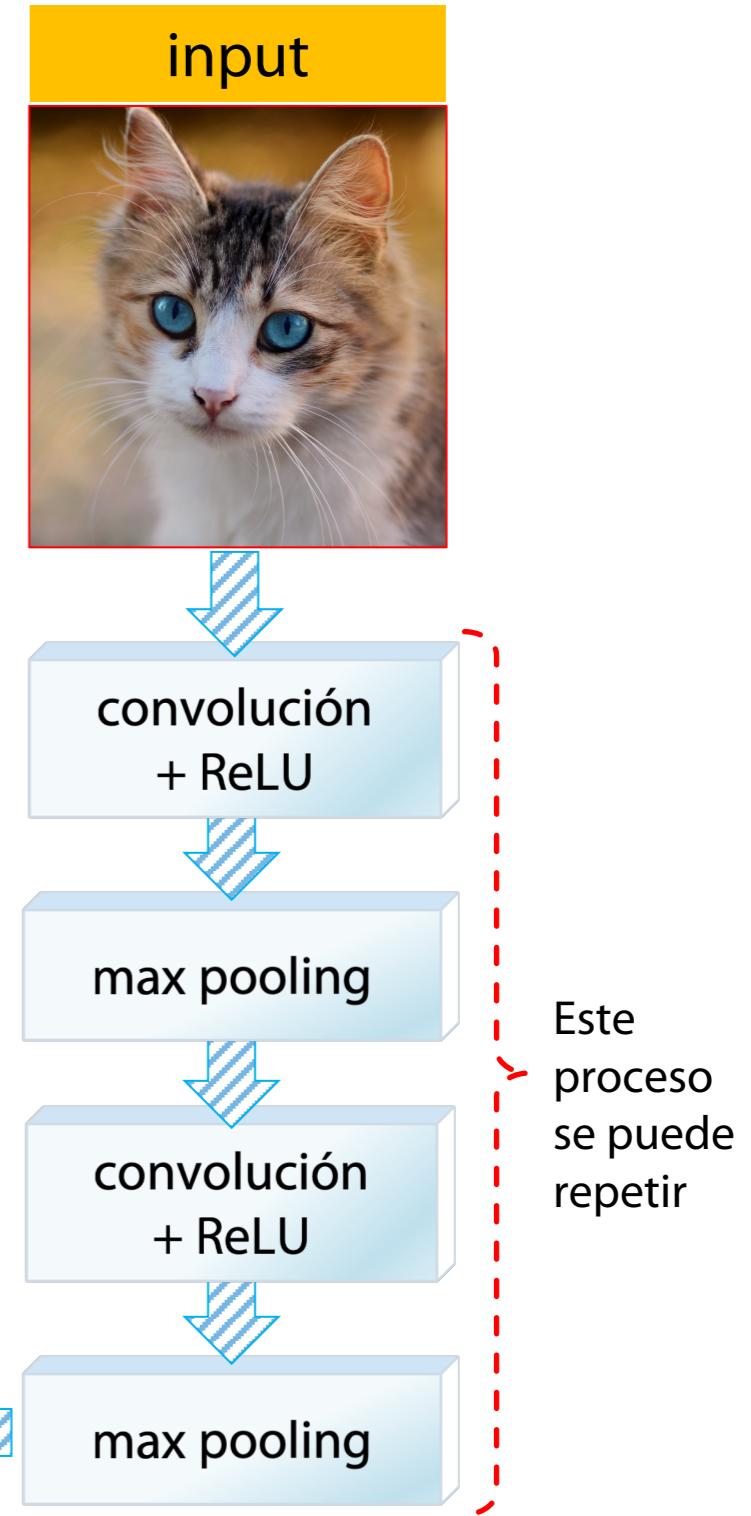
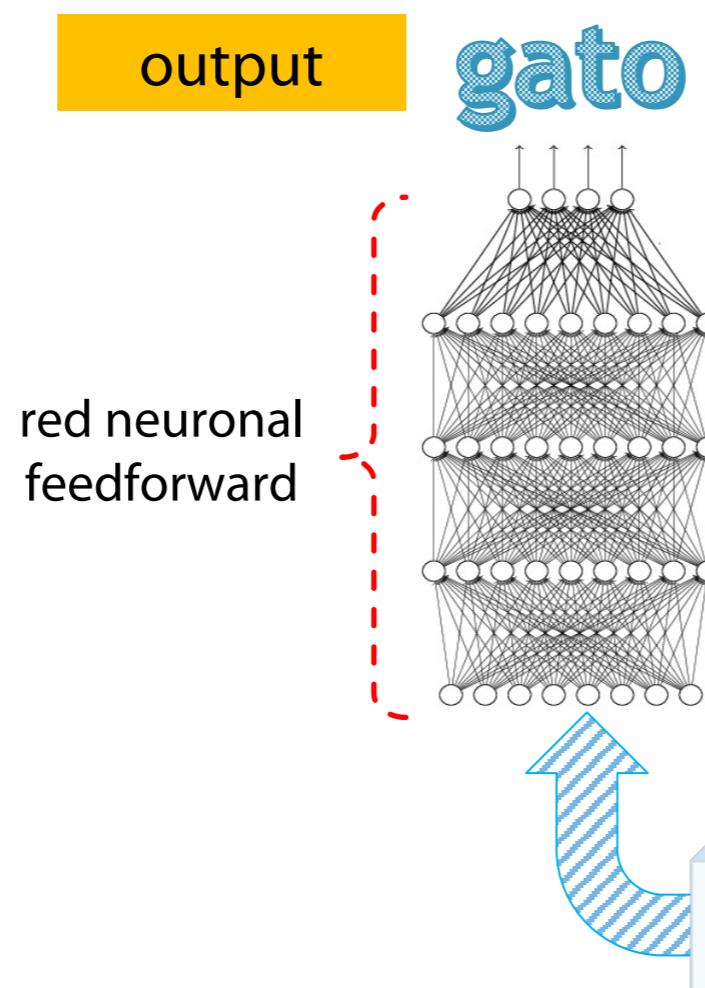


### ■ Supervisados



#### ■ Max Pooling (CNN)

El proceso anterior se repite varias veces, lo cual va generando una reducción de la imagen en sub imágenes.

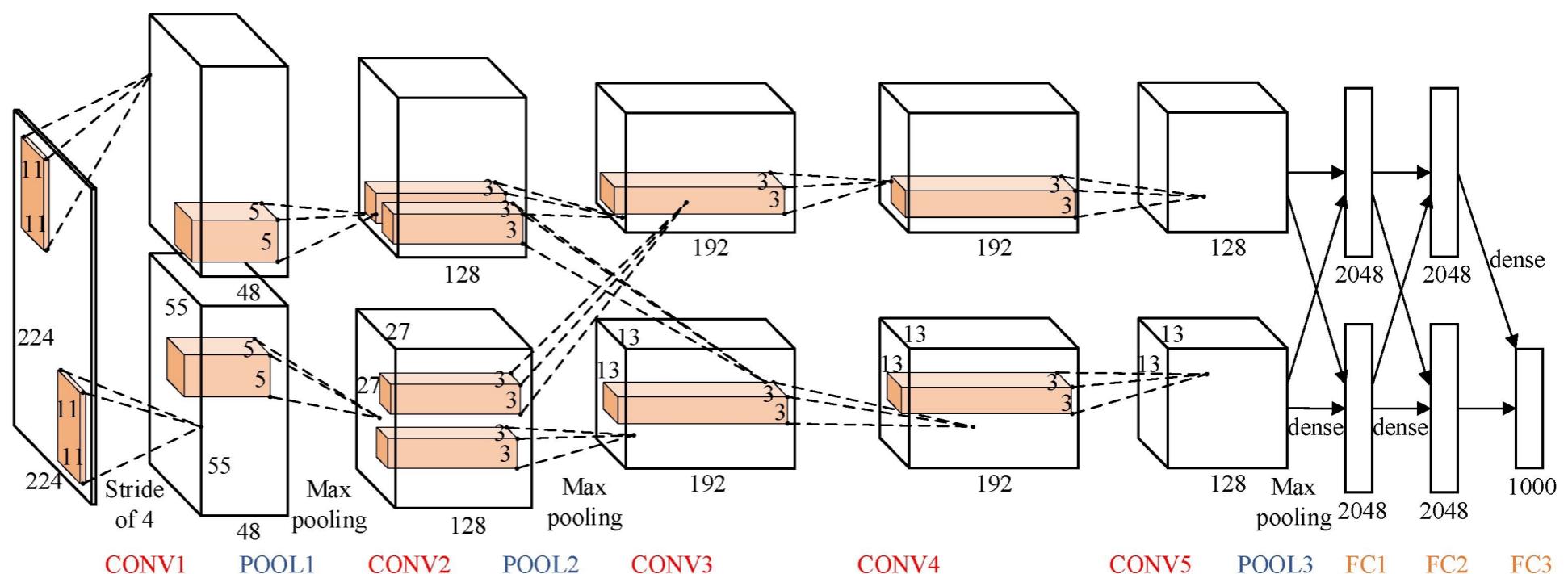


## ■ Supervisados



### ■ Arquitecturas más conocidas

Alexnet está compuesta por ocho capas; las primeras cinco son capas convolucionales, algunas de ellas seguidas por capas de agrupación máxima, y las últimas tres eran capas completamente conectadas. Usó la función de activación de ReLU no saturante, mostró un rendimiento de entrenamiento mejorado sobre tanh y sigmoide.



fuente: Alexnet (2012) <https://www.mdpi.com/2079-9292/8/3/295>

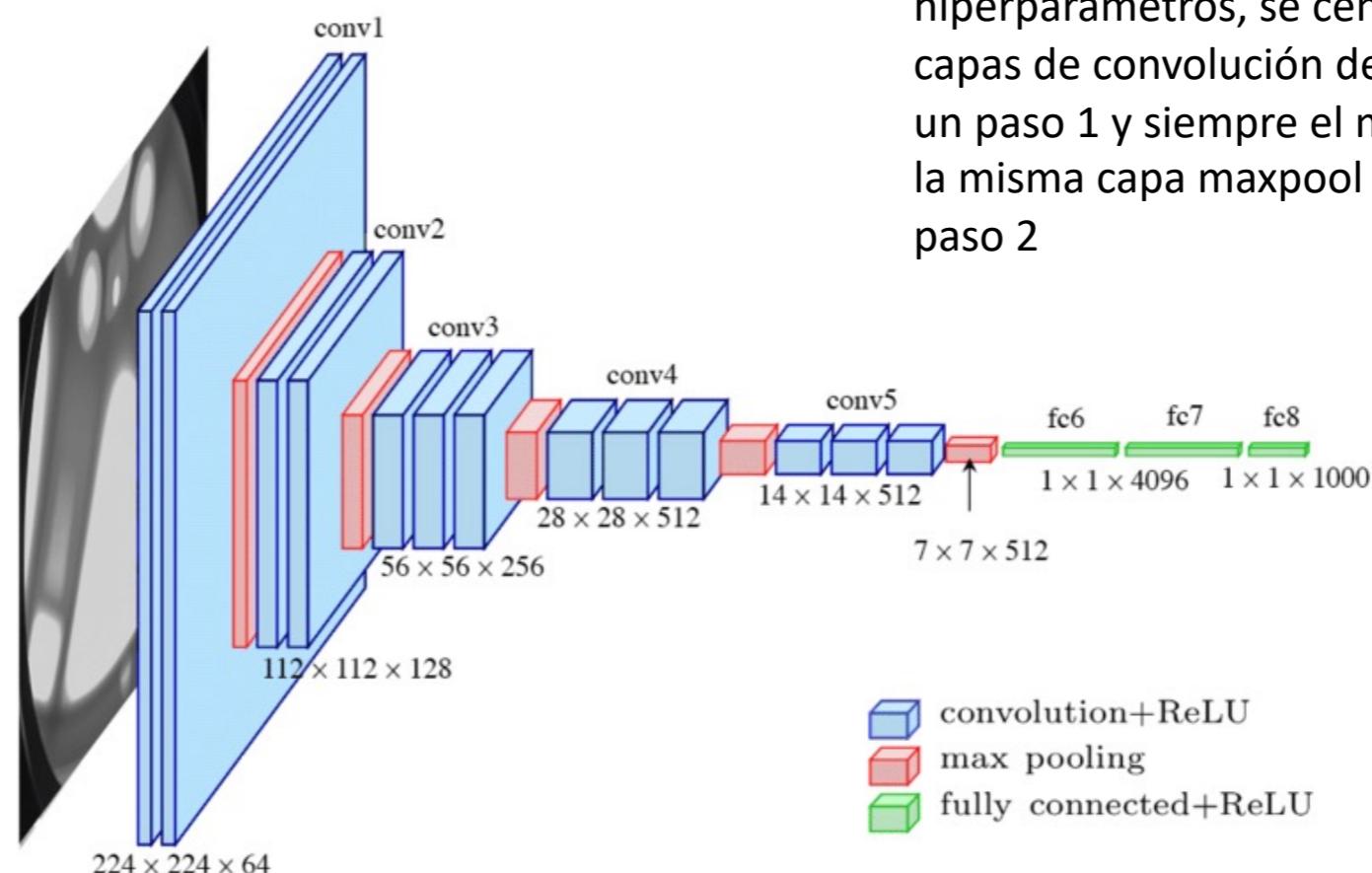
## ■ Supervisados



### ■ Arquitecturas más conocidas

VGG16 es una arquitectura de red neuronal de convolución (CNN) que se utilizó para ganar la competencia ILSVR (Imagenet) en 2014. Se considera una de las arquitecturas más empleadas en la actualidad por su versatilidad en muchos problemas.

En lugar de tener una gran cantidad de hiperparámetros, se centraron en tener capas de convolución de filtro 3x3 con un paso 1 y siempre el mismo relleno y la misma capa maxpool del filtro 2x2 del paso 2



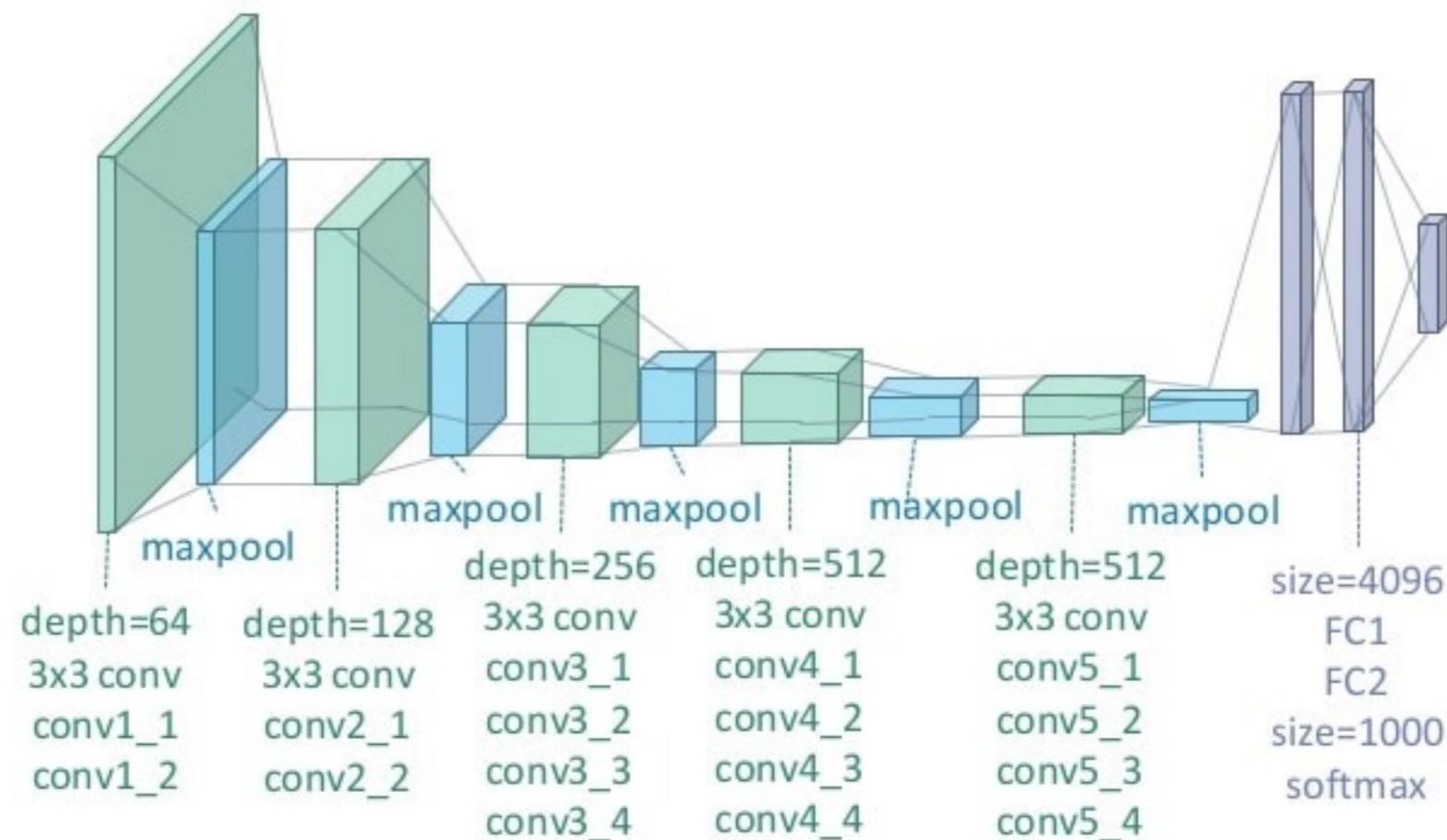
*fuente:* <https://ieeexplore.ieee.org/document/8258115>

## ■ Supervisados



### ■ Arquitecturas más conocidas

VGG-19 es una red neuronal convolucional entrenada por el Grupo de Geometría Visual, del Departamento de Ciencias de la Ingeniería, Universidad de Oxford. El número 19 representa el número de capas con pesos entrenables, 16 capas convolucionales y 3 capas totalmente conectadas



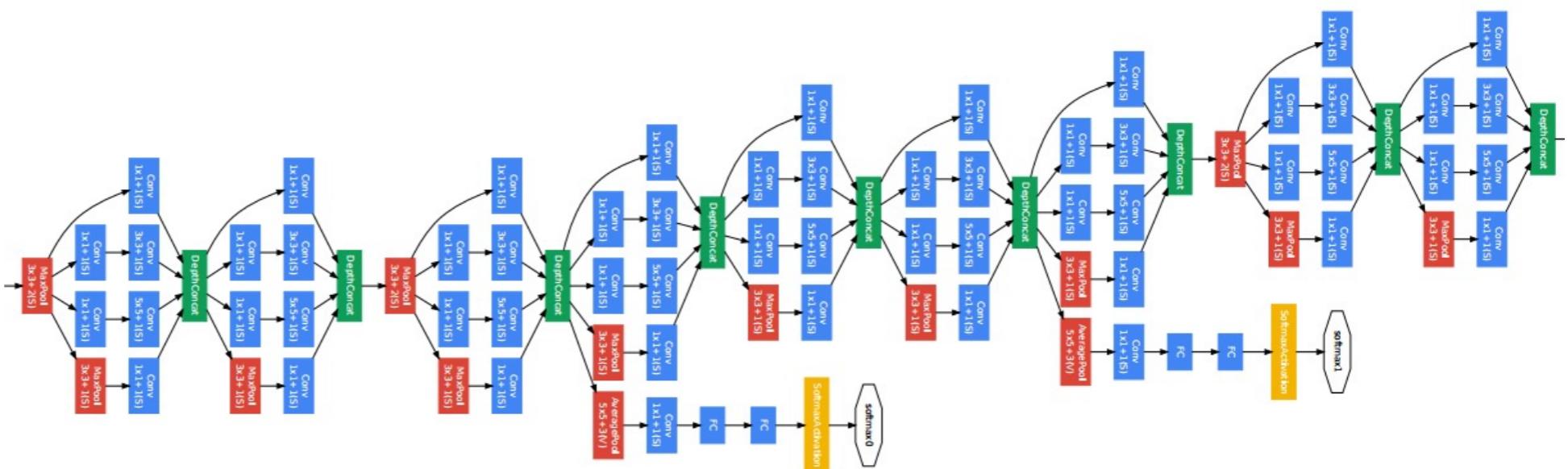
fuente: <http://dx.doi.org/10.3390/rs10071119>

### ■ Supervisados



#### ■ Arquitecturas más conocidas

GoogLeNet Google Net (o Inception V1) fue propuesto por Google (con la colaboración de varias universidades) en 2014 en el artículo de investigación titulado “Profundizando con las convoluciones”.



Esta arquitectura fue la ganadora en el desafío de clasificación de imágenes ILSVRC 2014. Esta arquitectura utiliza técnicas como convoluciones  $1 \times 1$  en el medio de la arquitectura y agrupación promedio global.

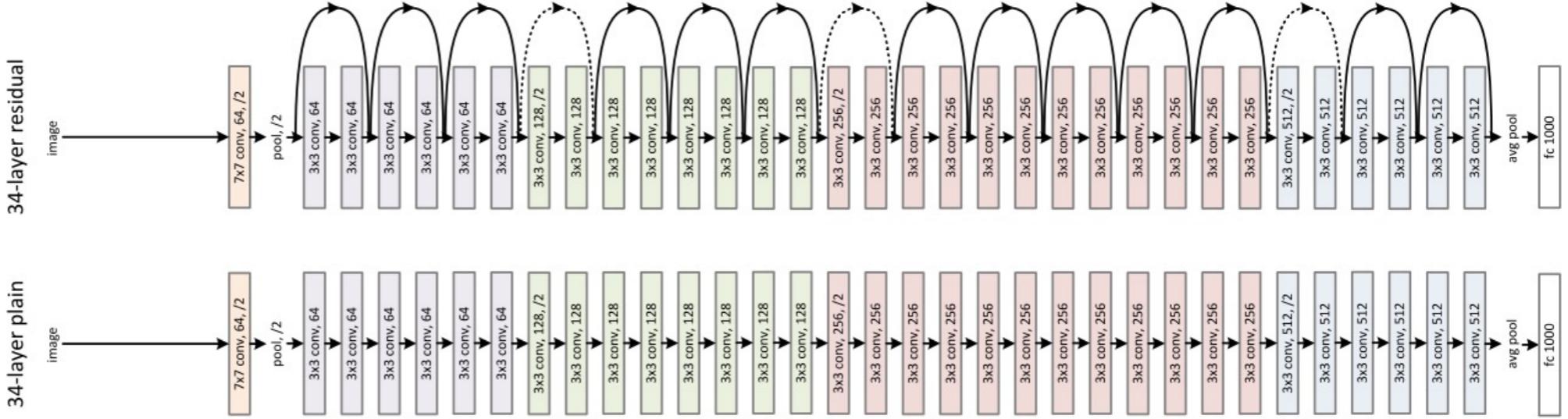
*fuente:* <https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/>

## ■ Supervisados

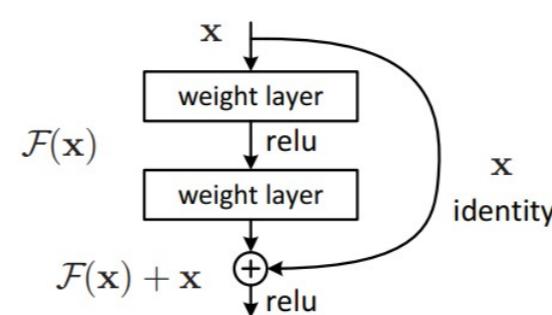


### ■ Arquitecturas más conocidas

ResNet fue propuesto en 2015 por investigadores de Microsoft Research quienes introdujeron una nueva arquitectura llamada Red residual. Esta técnica permite el salto de conexiones, de esta forma se omite el entrenamiento de algunas capas y se conecta directamente a la salida.



### Arquitectura residual



Esta red logró superar el rendimiento humano con un error aproximado de 3.6% el año 2016

*fuente:* <https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035>