

AI

SOMIA

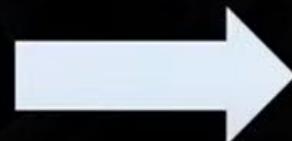
LEONARDO DANIEL AVIÑA NERI

# Programación regular

Entradas



Reglas y lógica



5

Resultado

# Aprendizaje automático

Entradas



Reglas y lógica

?

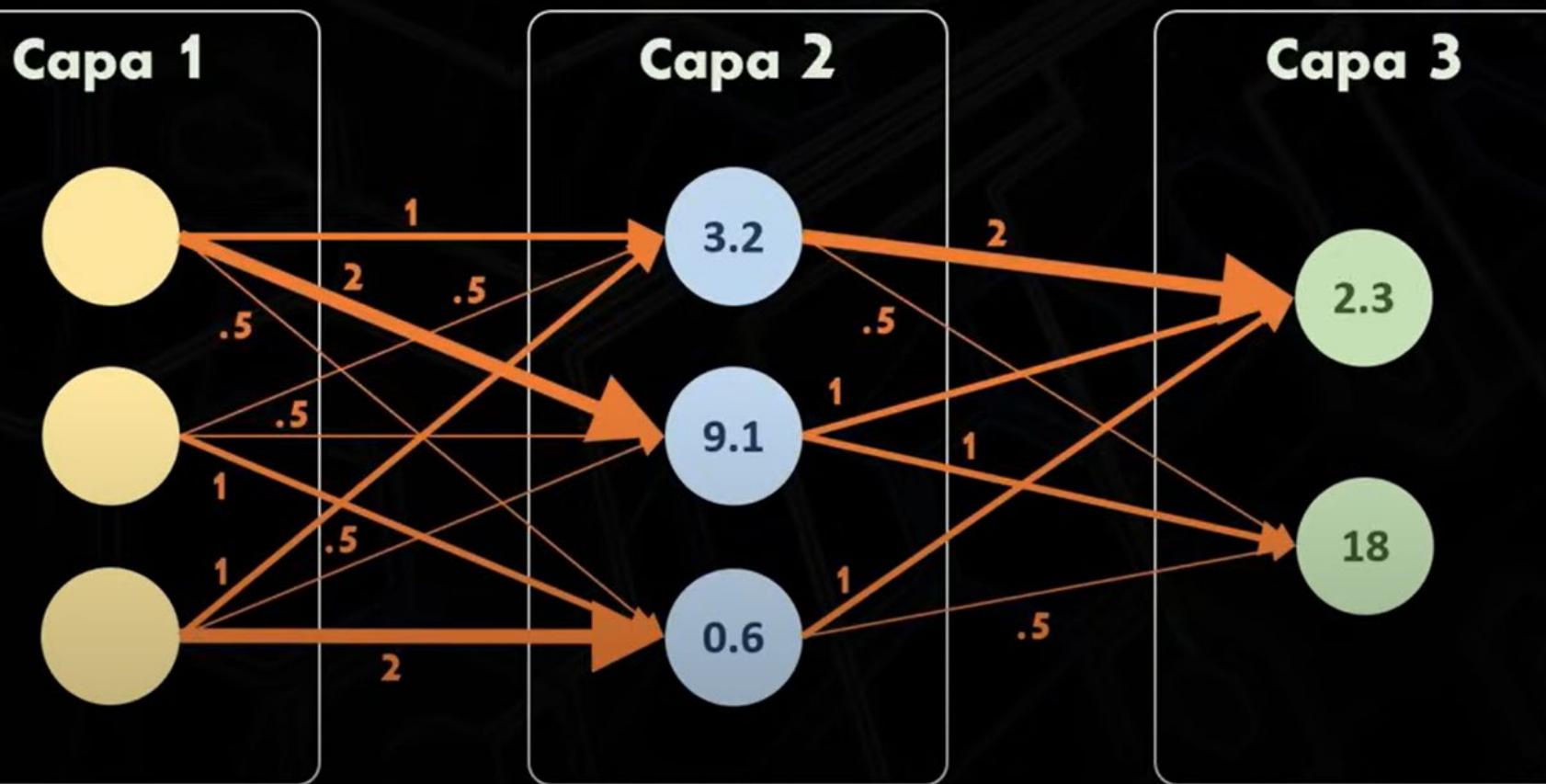


5

Resultado

# Aprendizaje automático

## Sesgos de las neuronas



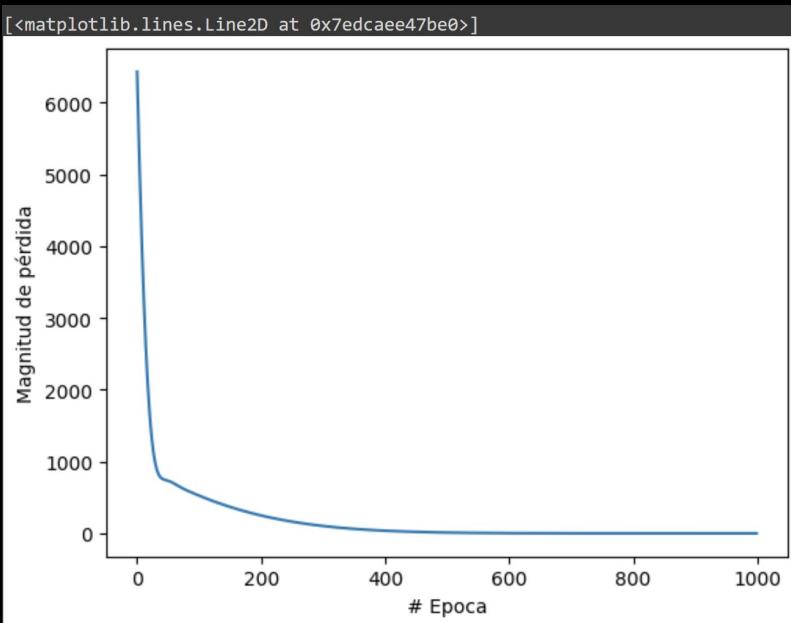
- Peso (weight): valor numérico que representa la importancia de la conexión entre las neuronas (número en las flechas)
- Sesgo: la tendencia de estas tecnologías a producir resultados que no son precisos.

# Proceso general

Salida=entrada\*peso + sesgo



$$15 * 1.5 = 22.5 + 4 = \underline{\underline{26.5}}$$



```
print("variables internas del modelo:")
print(capa.get_weights())
```

variables internas del modelo:

```
[array([[1.7983454]], dtype=float32), array([31.906666], dtype=float32)]
```

```
[14] print("haciendo una predicción")
resultado = modelo.predict(np.array([[100.0]])) # que convierta 100 celcius a fah
print("el resultado es: "+ str(resultado) +" fahrenheit" )
```

→ haciendo una predicción  
1/1 ━━━━━━ 0s 59ms/step  
el resultado es: [[211.74121]] fahrenheit

+ Código

+ Texto

# Aprendizaje automático

## Proceso general



Peso: 1.798

Sesgo: 31.9

Entrada=100

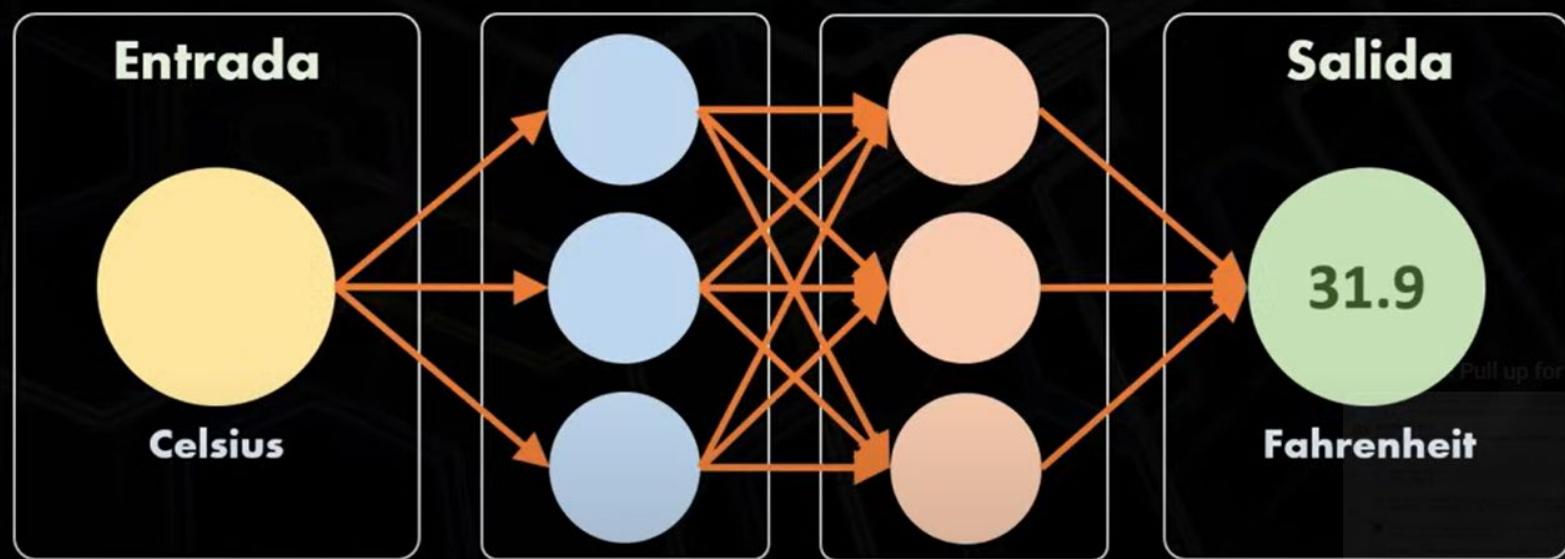
Salida=entrada\*peso + sesgo

$$100 * 1.798 = 179.8 + 31.9 = \underline{211.74}$$

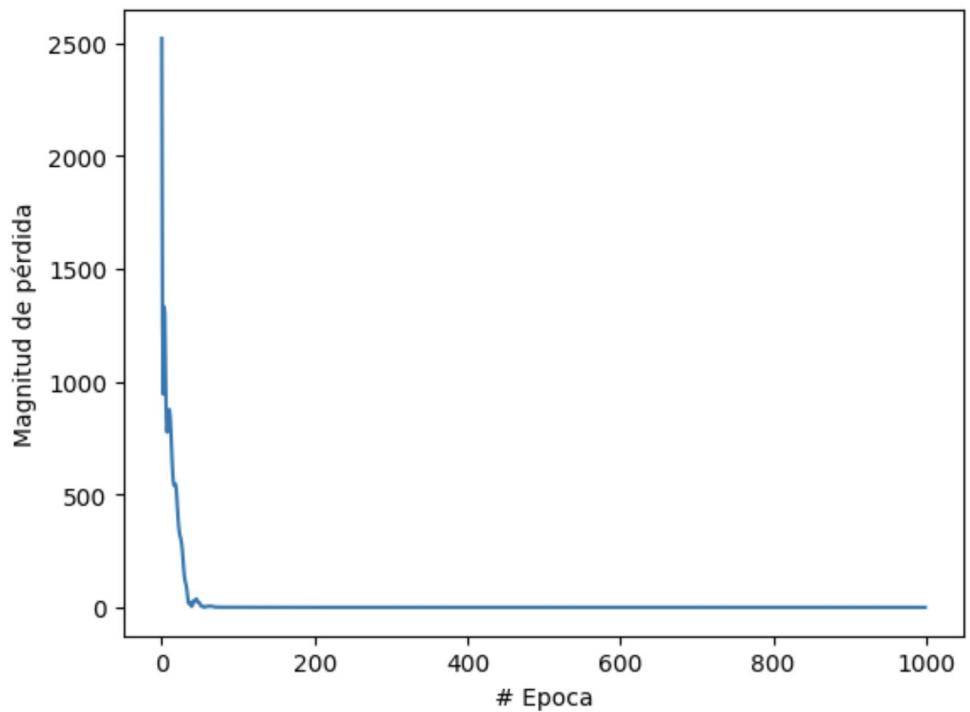
$$F = \frac{9}{5}({}^{\circ}C) + 32$$

# Aprendizaje automático

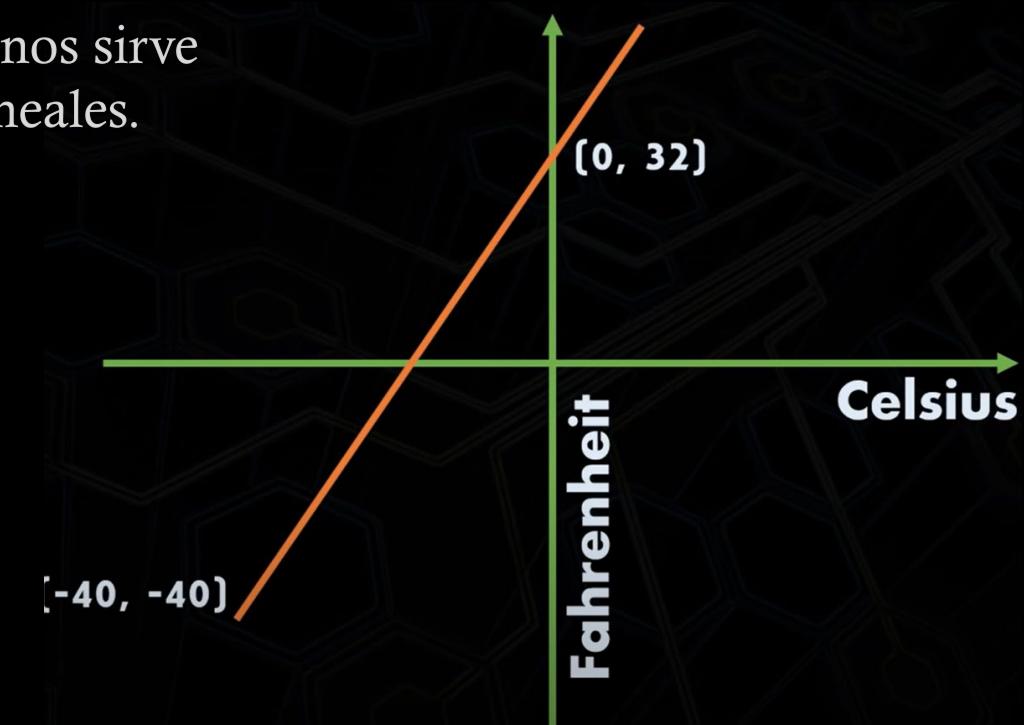
## Proceso general



Agregamos 2 capas intermedias con 3 neuronas en cada una



Este modelo solo nos sirve para problemas lineales.



```
[21] print("haciendo una predicción")
      resultado = modelo.predict(np.array([[100.0]])) # que convierta 100 celcius a fah
      print("el resultado es: " + str(resultado) +" fahrenheit")

      ➜ haciendo una predicción
      1/1 ━━━━━━━━ 0s 75ms/step
      el resultado es: [[211.74744]] fahrenheit

      ➜ print("variables internas del modelo:")
      print(oculta1.get_weights())
      print(oculta2.get_weights())
      print(salida.get_weights())

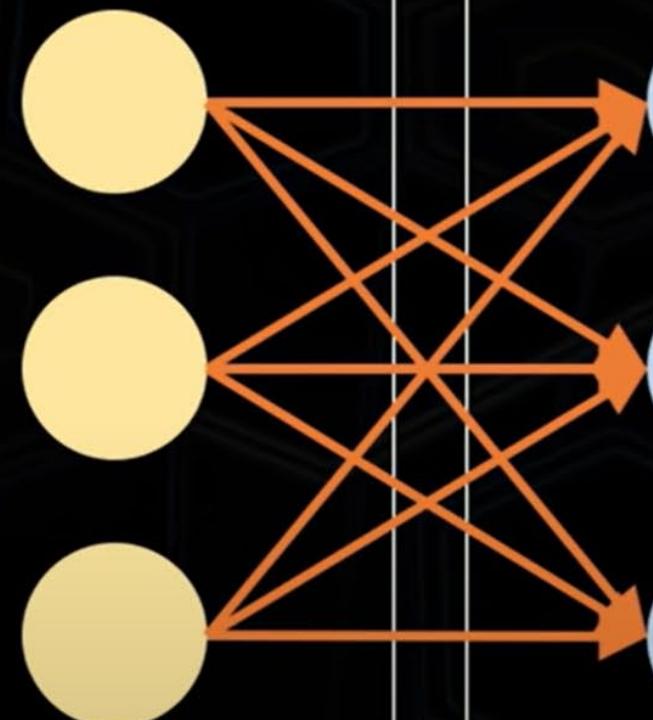
      ➜ variables internas del modelo:
      [array([[ 0.08692808,  0.6550078 , -0.39447182]], dtype=float32), array([-3.165305 ,  3.8718276, -3.8613145], dtype=float32)]
      [array([[ 0.88379365, -0.83200955,  0.83337915],
             [-1.2045394 ,  0.22273266,  0.23843646],
             [ 1.0567417 ,  0.23551258, -0.00825922]], dtype=float32), array([-3.6882803,  2.823134 , -2.4711335], dtype=float32)]
      [array([[-1.6409501 ],
             [ 0.46628627],
             [-0.1970213 ]], dtype=float32), array([3.620075], dtype=float32)]
```

# CLASIFICADOR DE IMÁGENES

VERSIÓN 1

# Regresión

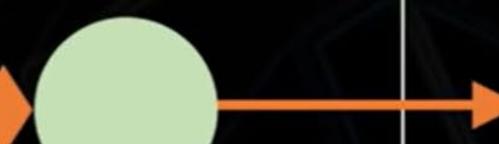
**Capa 1**



**Capa 2**



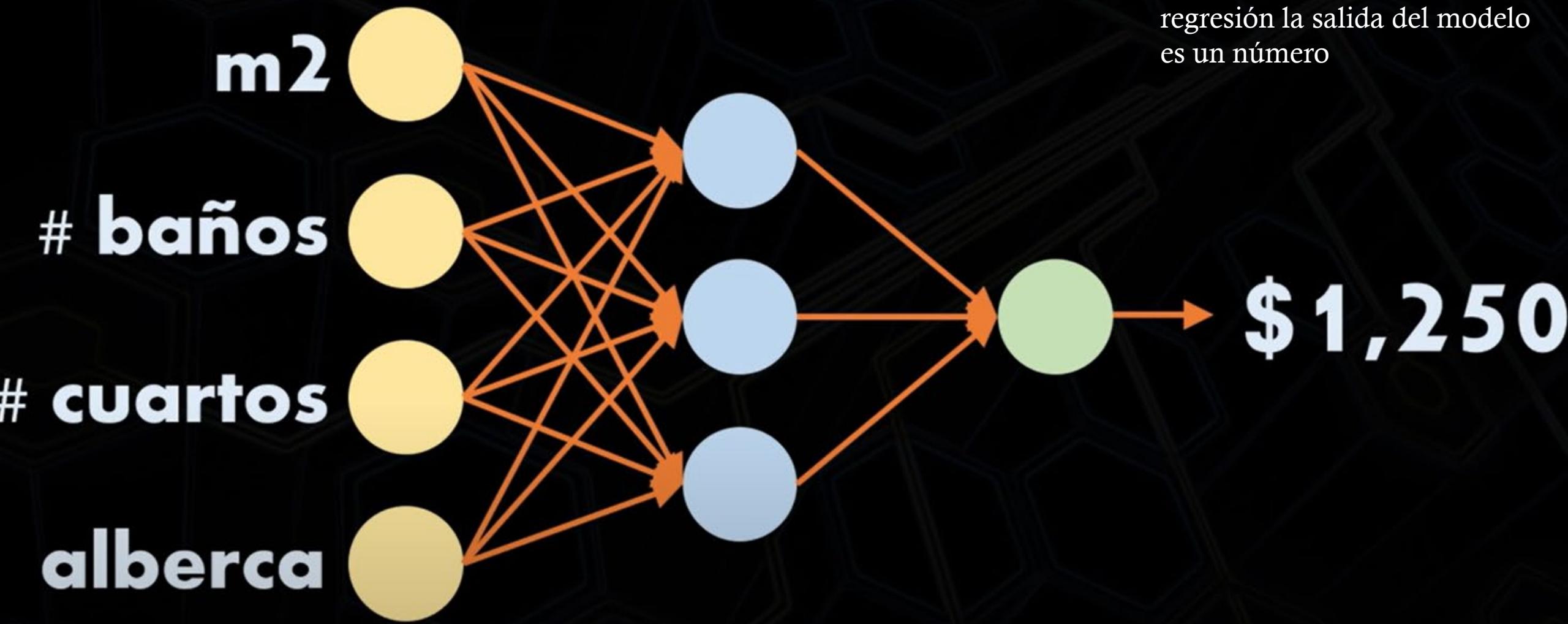
**Capa 3**



En los problemas de regresión la salida del modelo es un número

**Número**

# Valor de una casa





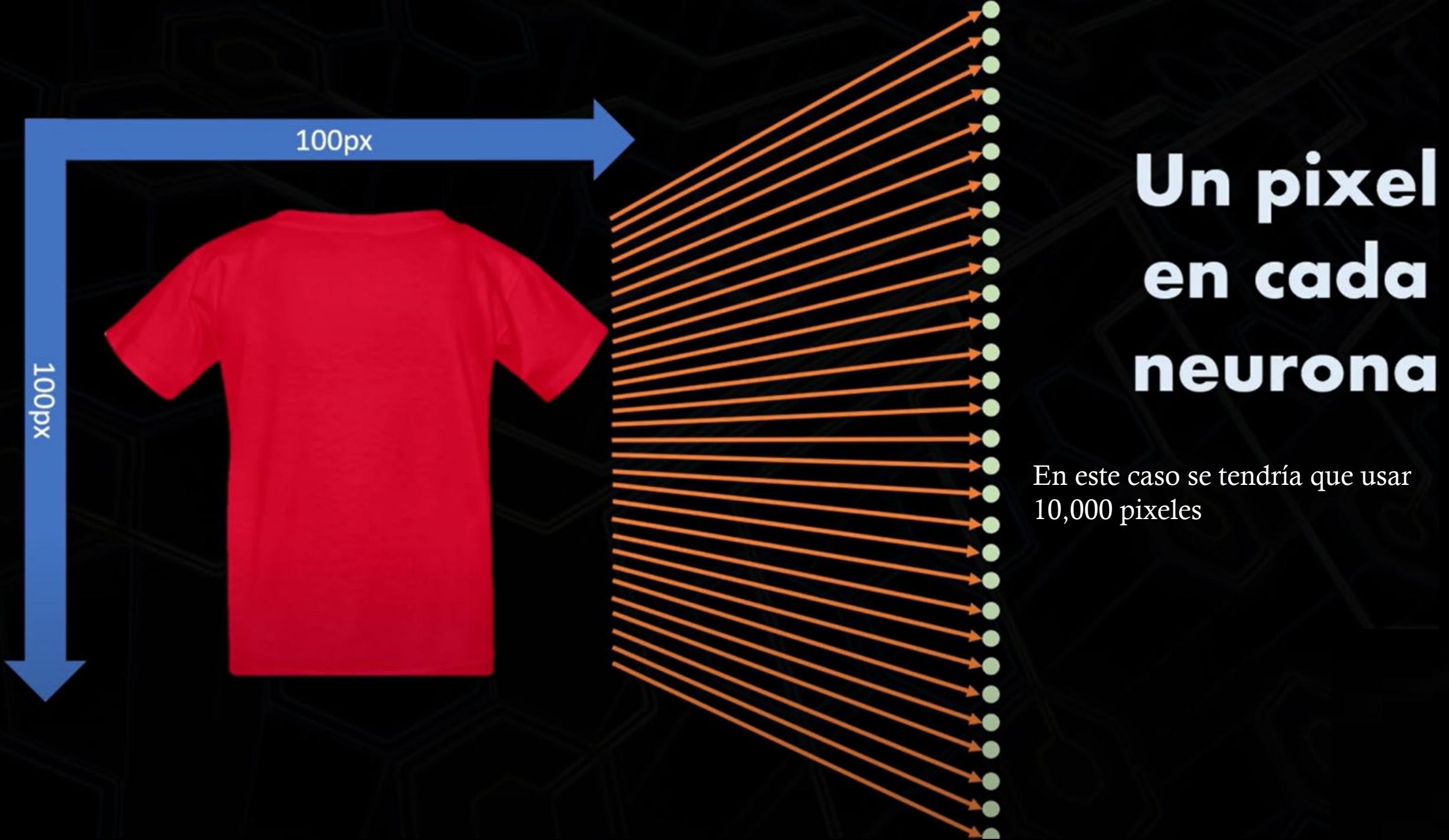
- Camiseta ✓**
- Pantalón**
- Suéter**
- Vestido**
- Saco**
- Sandalia**
- Camisa**
- Tenis**
- Bolsa**
- Botín**

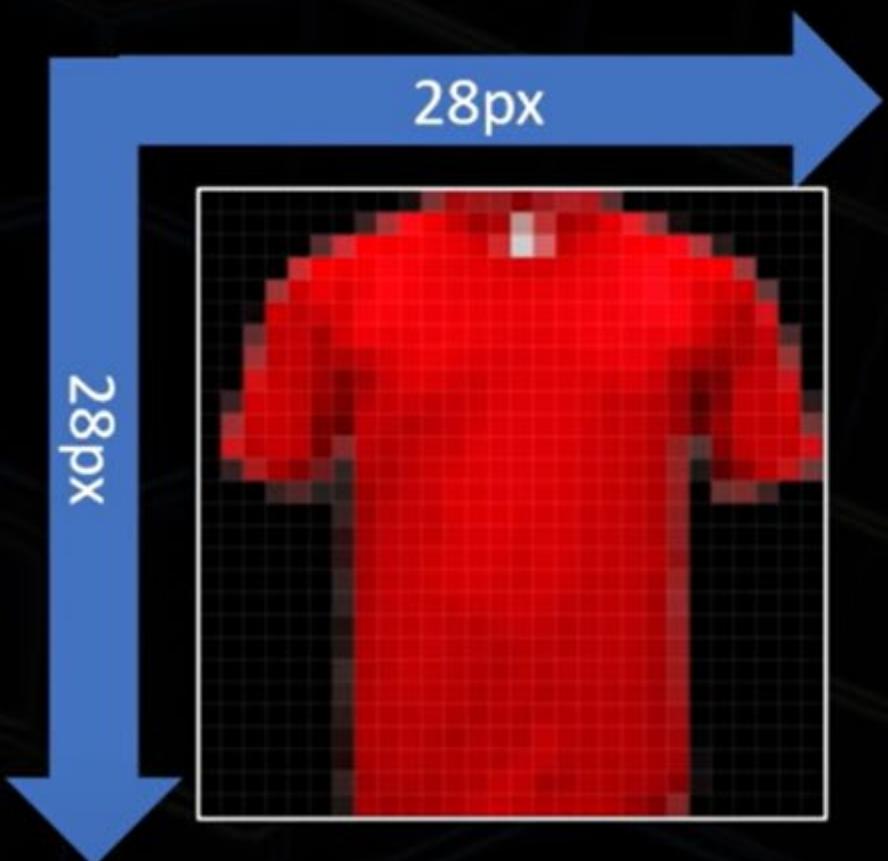
En nuestro clasificador recibirá de entrada la imagen y nos dirá a que clasificación pertenece

1. Tomamos la imagen
2. La convertimos a blanco y negro
3. Cada pixel se le asigna un valor numérico, en el intervalo: 0=negro y 255 blanco.



**100 × 100  
10,000  
pixeles**





**28 x 28**  
**784**  
**pixeles**

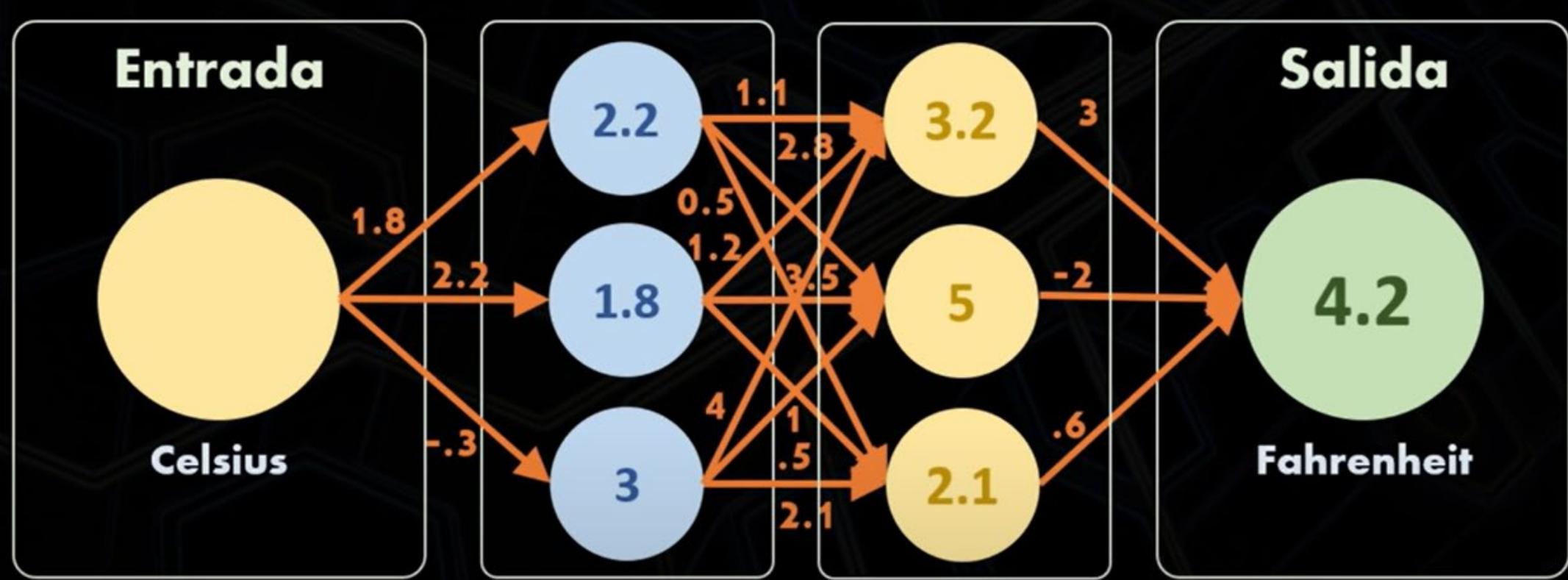
Haremos un “truco”, reducir el tamaño de las imágenes a 28 x 28px para reducir el número de neuronas

# Capas ocultas

Si queremos dar a la red más espacio para que realice transformaciones más complejas necesitamos agregar una o más capas ocultas, estas capas siguen las mismas reglas de hacer multiplicaciones y sumas.

Al agregarlas a la red, esta contará con más opciones y caminos donde puede hacer más operaciones lineales para unir el resultado.

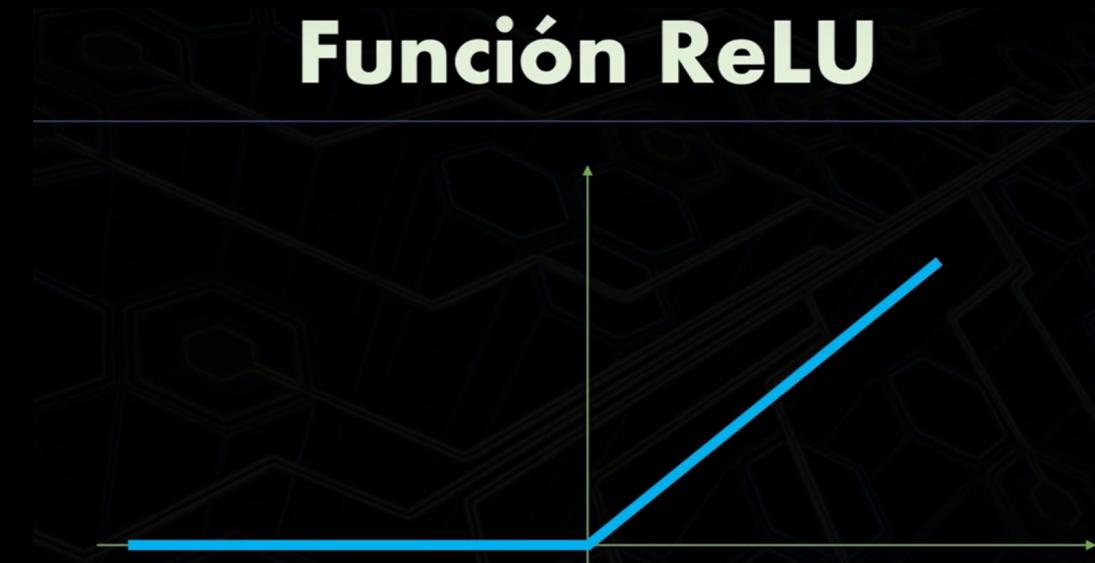
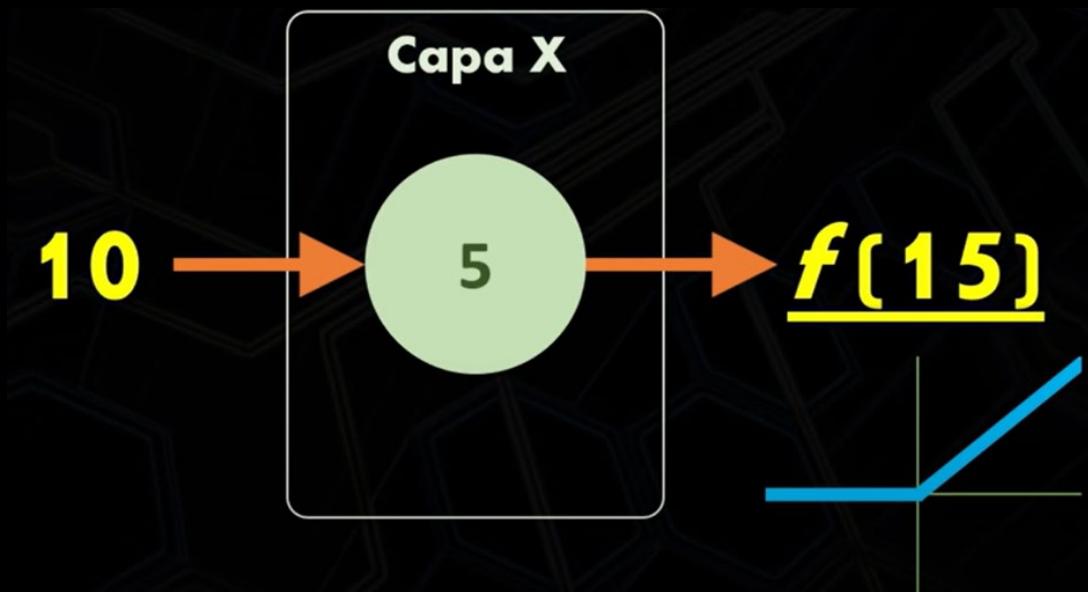
Su inconveniente es que solo solucionan problemas lineales.



# Funciones de activación

Una vez que una neurona asuma su sesgo y va a dar su resultado la pasaremos por una función llamada comúnmente función de activación.

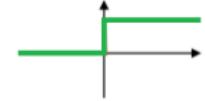
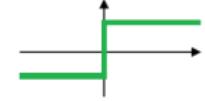
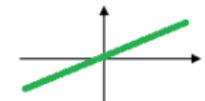
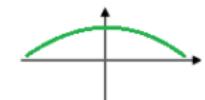
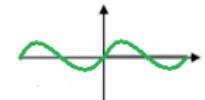
Digamos que tenemos esta neurona la neurona recibe como entrada el número 10, el sesgo es de 5 entonces al sumarlo la salida de la neurona sería 15, pero que antes de enviar ese 15 como salida hacemos ese 15 por una función. Por ejemplo, la función ReLU



Si el número es mayor a cero  
mantenlo igual.  
Si es menor a cero (negativo) que  
se haga cero.

# Normalizar los datos

Es un proceso que consiste en transformar los datos para que se ajusten a una escala estándar, normalmente entre 0 y 1 o -1 y 1. Se hace para mejorar los resultados y que sea más rápido

TIPO DE FUNCIÓN DE ACTIVACIÓN	ECUACIÓN	EJEMPLO O MODELO	GRÁFICO
Función "Step" o Heaviside	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	PERCEPTRÓN	
Función Signo	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	PERCEPTRÓN	
Función Lineal	$\phi(z) = z$	ADALINE	
Función lineal definida a trozos	$= \begin{cases} 1 & z \geq \frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} < z < \frac{1}{2} \\ 0 & z \leq -\frac{1}{2} \end{cases}$	MÁQUINAS DE VECTOR SOPORTE	
Función Gaussiana	$\phi(z) = Ae^{-Bz^2}$	REDES NEURONALES RBF	
Función Sigmoidal o Logística (Curva "S")	$\phi(z) = \frac{1}{1 + e^{-z}}$	REDES NEURONALES MULTICAPA	
Tangente hiperbólica	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	REDES NEURONALES MULTICAPA	
Función rectificadora o función ReLU Unidad Lineal Rectificada	$\phi(z) = \max(0, z)$	REDES NEURONALES MULTICAPA	
Función Sinusoidal	$\phi(z) = A \sen(\omega z + \varphi)$	CLASIFICACIÓN DE PATRONES	
Función rectificadora suavizada (softplus)	$\phi(z) = \ln(1 + e^z)$	REDES NEURONALES MULTICAPA	

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot



Lo único malo de nuestro modelo es que solo sirve si las imágenes que queramos probar sean muy similares a las de entrenamiento en cuestión de tamaño, orientación y posición



**David H.  
Hubel**



**Torsten  
Wiesel**



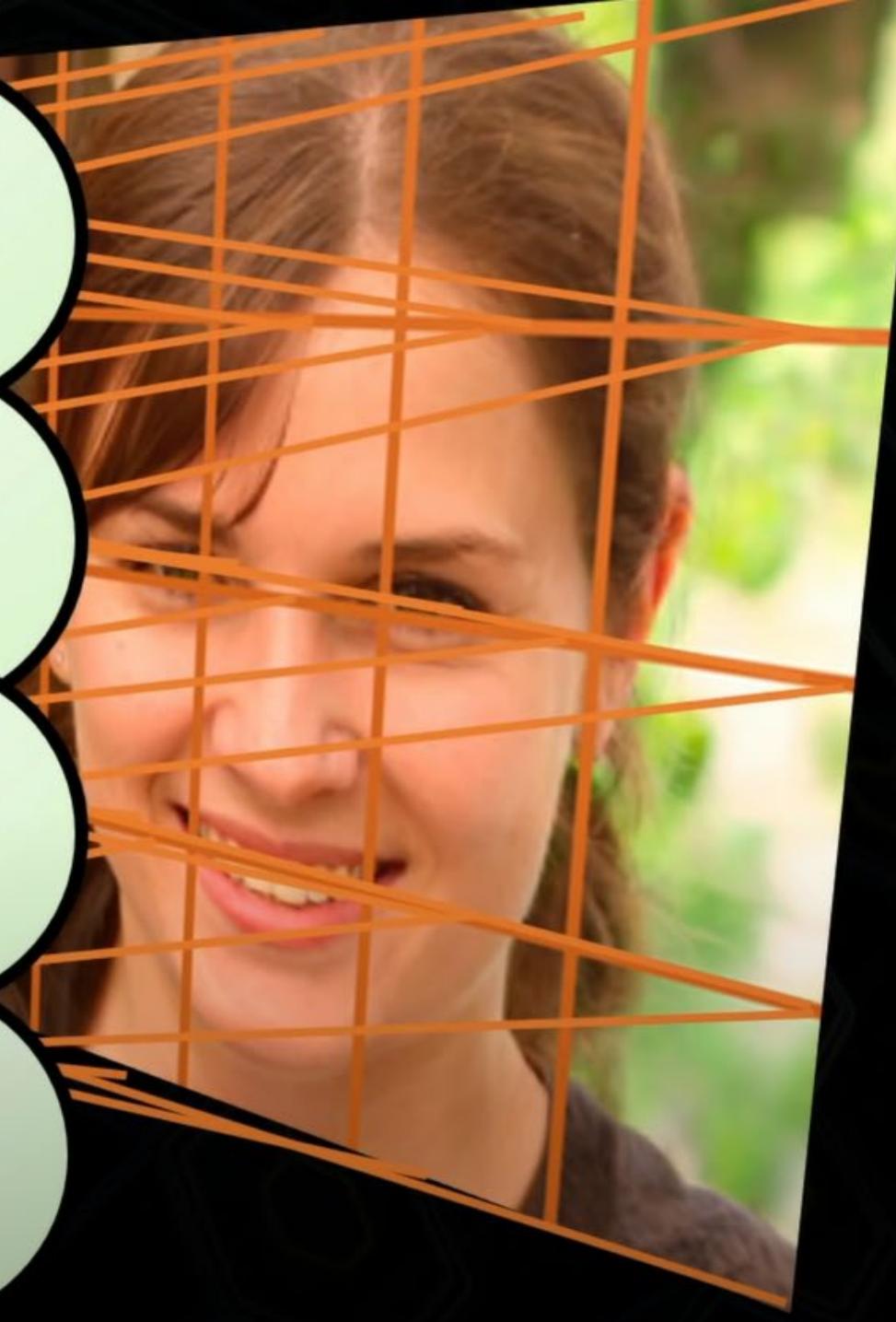
**Gato**



Simple



Compleja





**Neuronas  
Simples**



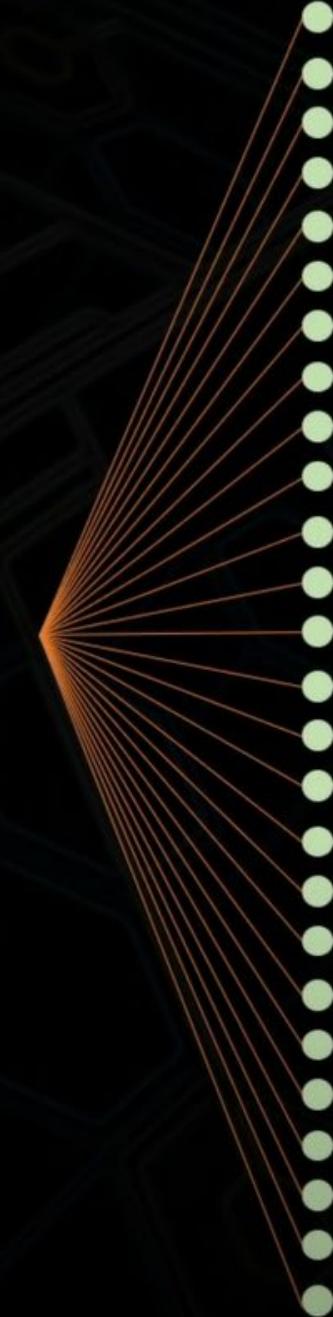
**Neuronas  
Complejas**

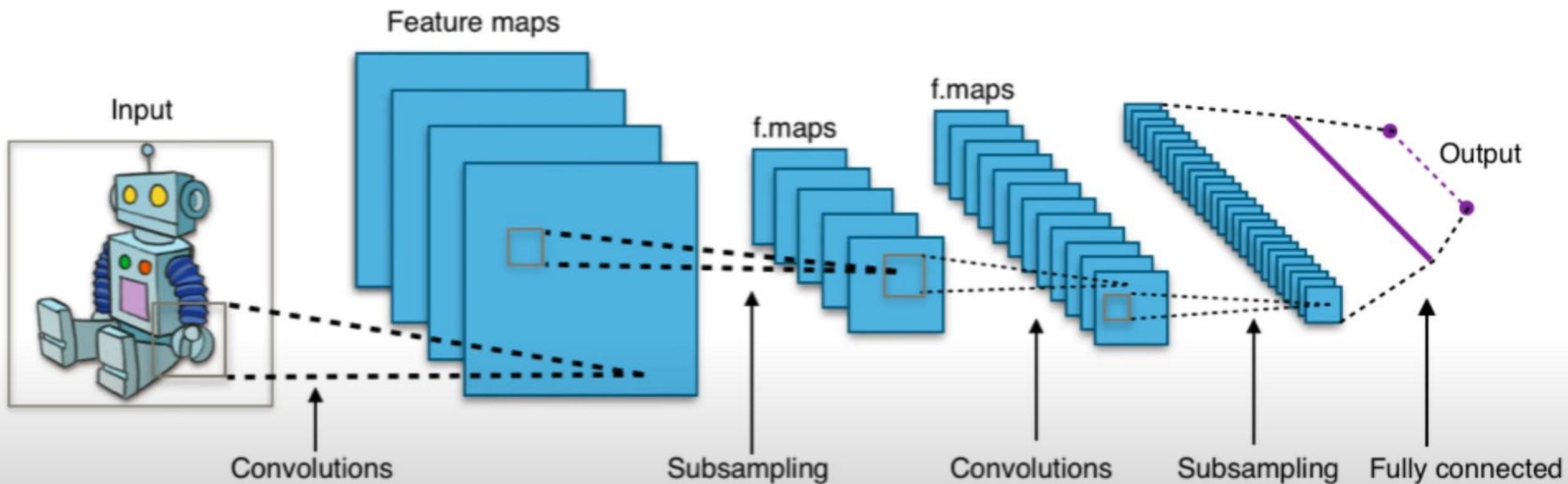


**Neuronas  
Simples**

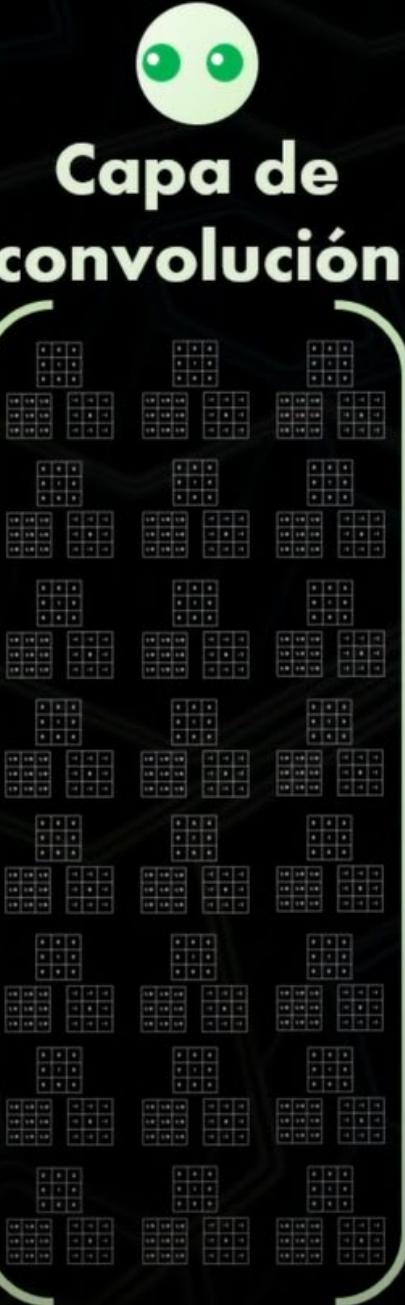
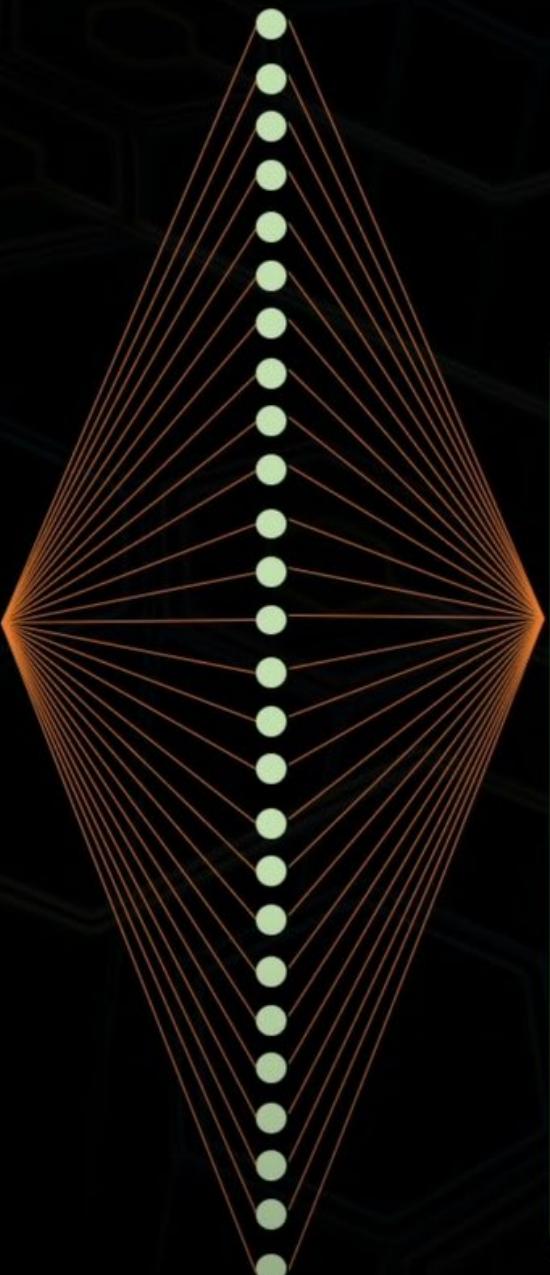


**Neuronas  
Complejas**





# Capa de entrada



Capa de  
convolución

32  
núcleos  
  
96  
convoluciones