Fundamentos de las Redes Neuronales

Redes Convolucionales (CNNs)

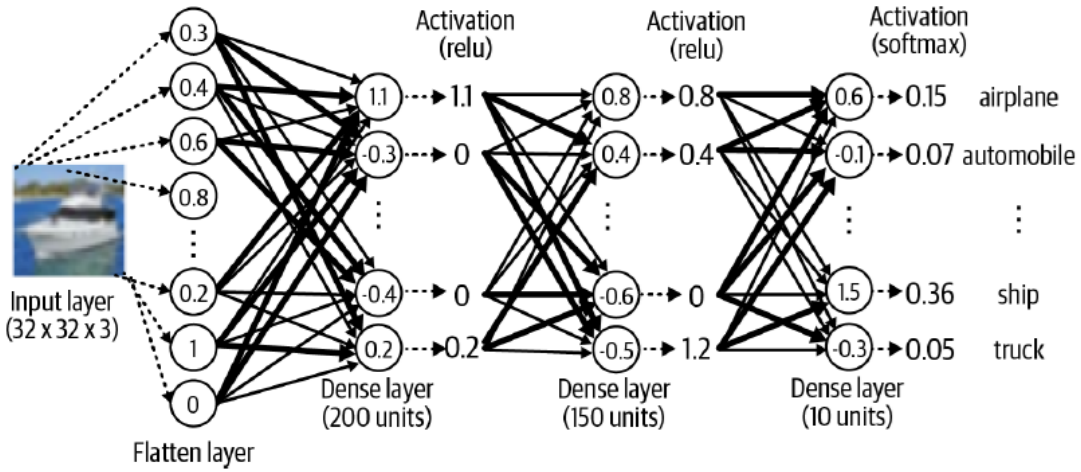
Clase 8

Dra. Wendy Aguilar

Modelos Generativos Profundos

UN ENFOQUE DESDE LA CREATIVIDAD COMPUTACIONAL

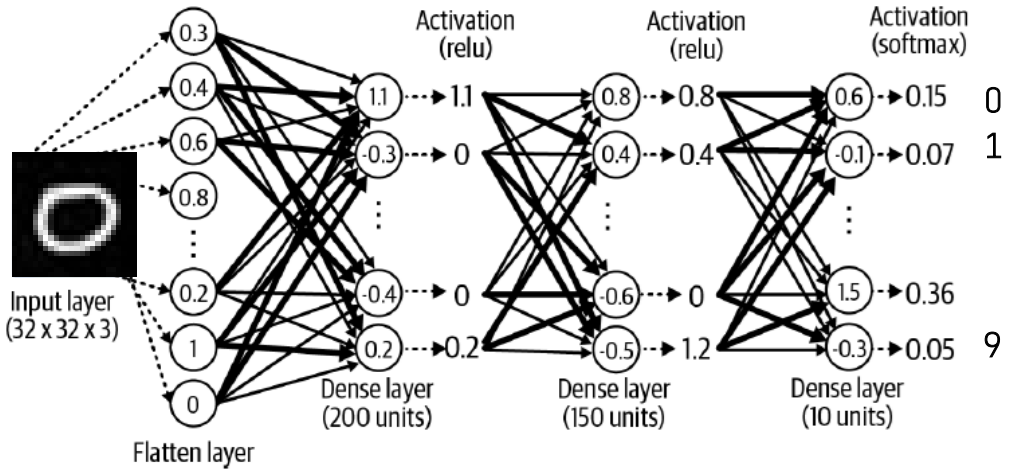
Perceptrón Multicapa usando Keras




```
model.evaluate(x_test, y_test)

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4879 - loss: 1.4613
[1.47357976436615, 0.4787999987602234]
```

Es alrededor del 50%.

Perceptrón Multicapa usando Keras



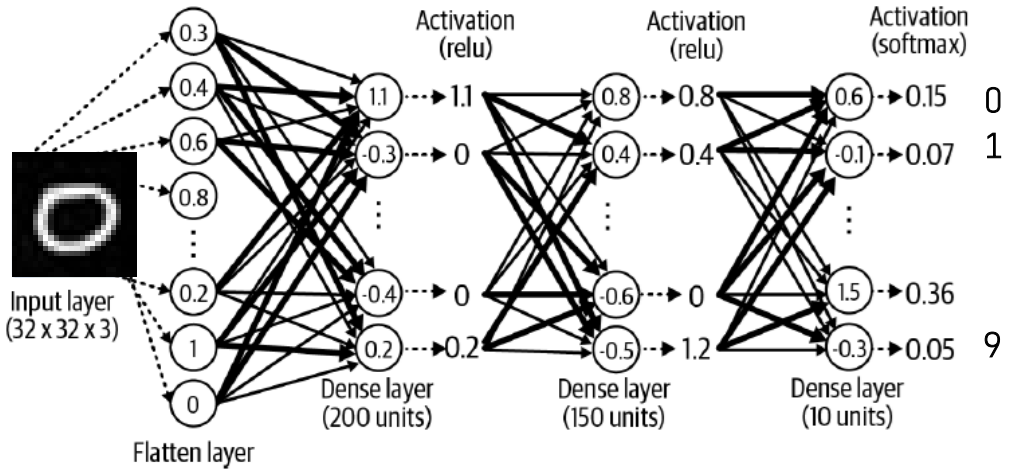
¿Qué accuracy obtuvieron con MNIST?

mlp_MNIST.ipynb

Google colab

¿Qué tuvieron que modificar de la red usada con CIFAR10?

Perceptrón Multicapa usando Keras




```
[13] model.evaluate(x_test, y_test)

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9720 - loss: 0.1116
[0.09902395308017731, 0.9753999710083008]
```

Es alrededor del 98%

60,000 imágenes de 32 ×32 pixeles

¿Por qué el desempeño es diferente?

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

```
model.evaluate(x_test, y_test)
```

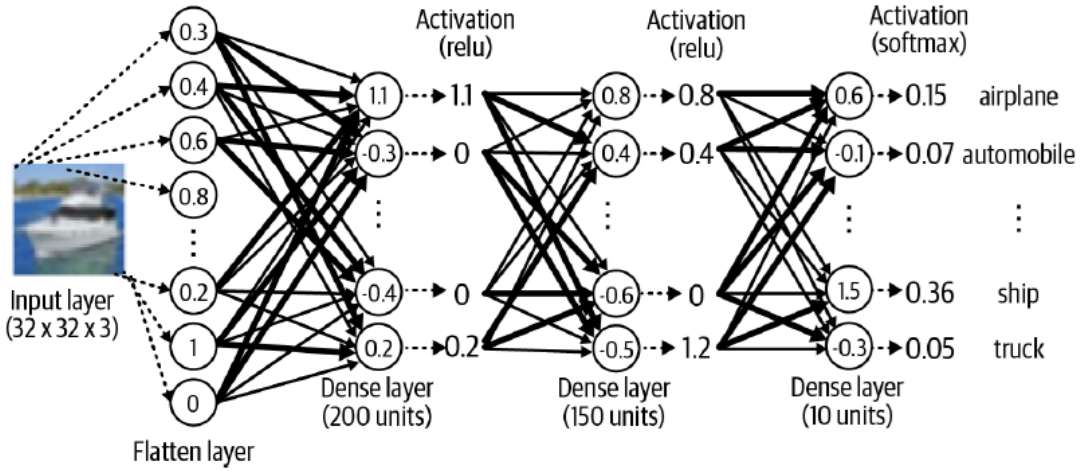
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4879 - loss: 1.4613
[1.47357976436615, 0.4787999987602234]
```

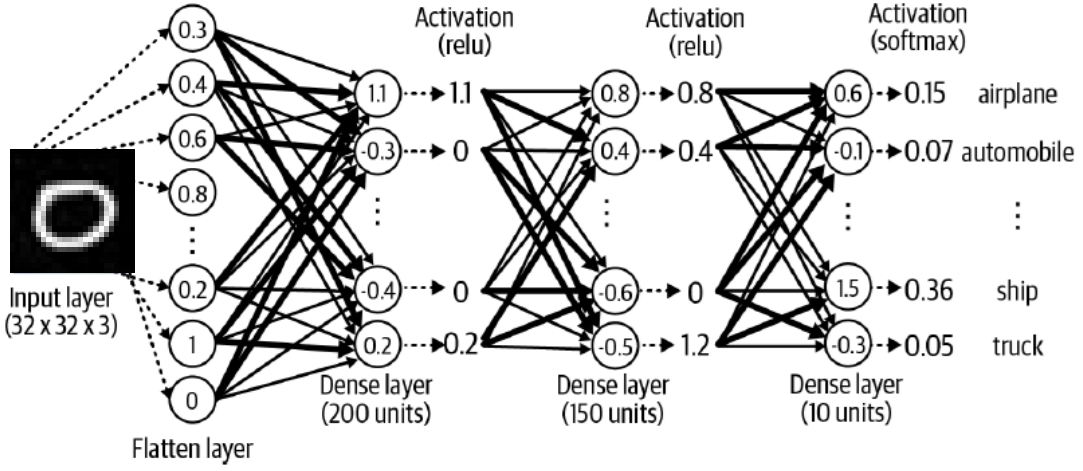
Es alrededor del 50%.

```
[13] model.evaluate(x_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9720 - loss: 0.1116
[0.09902395308017731, 0.9753999710083008]
```

Es alrededor del 98%





60,000 imágenes de 32 ×32 pixeles

¿Será por la complejidad que agrega el que sean a color y el tamaño de las imágenes?

mlp_CIFAR10_gris.ipynb

Google colab

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

60,000 imágenes de 28 ×28 pixeles

¿Será por la complejidad que agrega el que sean a color y el tamaño de las imágenes?

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

mlp_CIFAR10_gris.ipynb

Google colab

```
model.evaluate(x_test, y_test)

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4163 - loss: 1.6376
[1.645226001739502, 0.41269999742507935]
```

Es alrededor del 40%.

Empeoró, antes era alrededor del 50%

```
[13] model.evaluate(x_test, y_test)

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9720 - loss: 0.1116
[0.09902395308017731, 0.9753999710083008]
```

Es alrededor del 98%

60,000 imágenes de 32 ×32 pixeles

¿Por qué el desempeño es diferente?

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

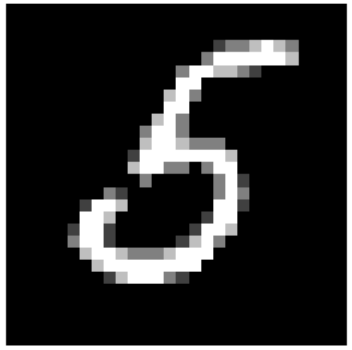
60,000 imágenes de 32 ×32 pixeles

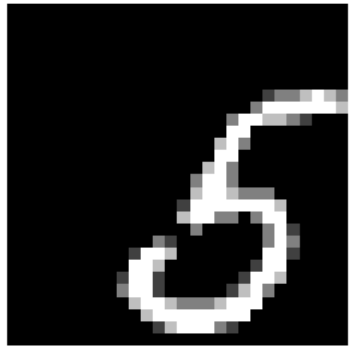
¿Qué pasa si trasladamos 4 pixeles abajo y a la derecha las imágenes del conjunto de entrenamiento de MNIST?

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

mlp_MINIST_traslaciones.ipynb

Google colab





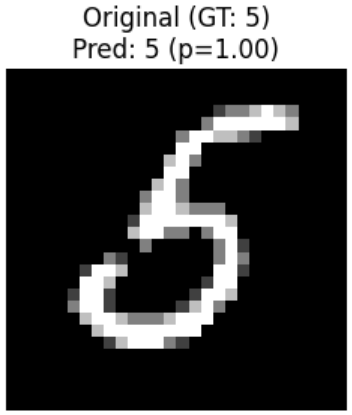
60,000 imágenes de 32 ×32 pixeles

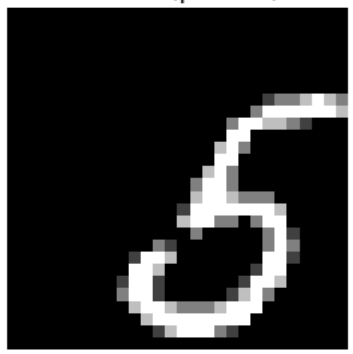
¿Qué pasa si trasladamos 4 pixeles abajo y a la derecha las imágenes del conjunto de entrenamiento de MNIST?

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

mlp_MINIST_traslaciones.ipynb

Google colab





¡Ya no clasificó bien!

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.0802 - loss: 13.9055
[14.555965423583984, 0.07609999924898148]
```

60,000 imágenes de 32 ×32 pixeles

¿Y si entrenamos MNIST con aumento de datos?

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

60,000 imágenes de 32 ×32 pixeles

¿Y si entrenamos MNIST con aumento de datos?

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

Necesitaríamos darle como ejemplo los dígitos **prácticamente bajo todas las diferentes condiciones posibles**: traslaciones, rotaciones, cambios de iluminación, escala, etc.

Además de ser costoso y no escalar bien a datasets grandes:

Tendería a memorizar y no generalizaría bien.

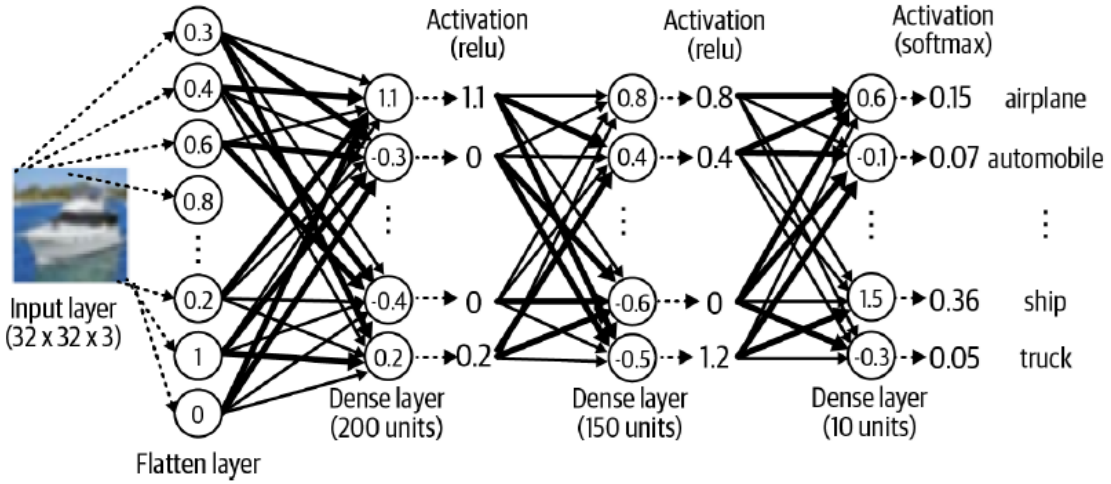
No resolvería el problema de fondo

Los MLP Ignoran la estructura espacial inherente de los datos visuales

No resolvería el problema de fondo

Los MLP Ignoran la estructura espacial inherente de los datos visuales

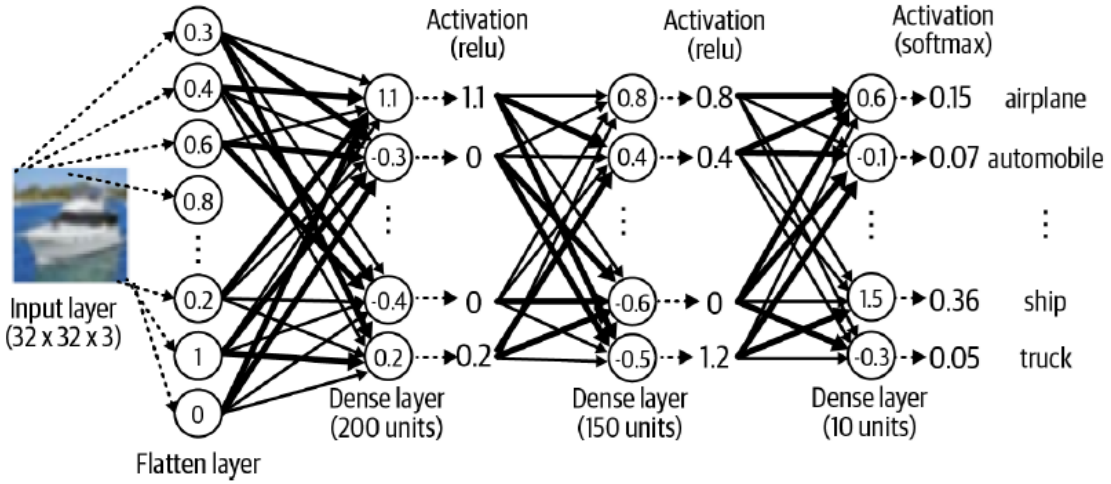
Se aplana la imagen en un vector, con lo cual **se pierde toda la estructura espacial de los pixeles**.



No resolvería el problema de fondo

Ignoran la estructura espacial inherente de los datos visuales

Se aplana la imagen en un vector, con lo cual **se pierde toda la estructura espacial de los pixeles**.

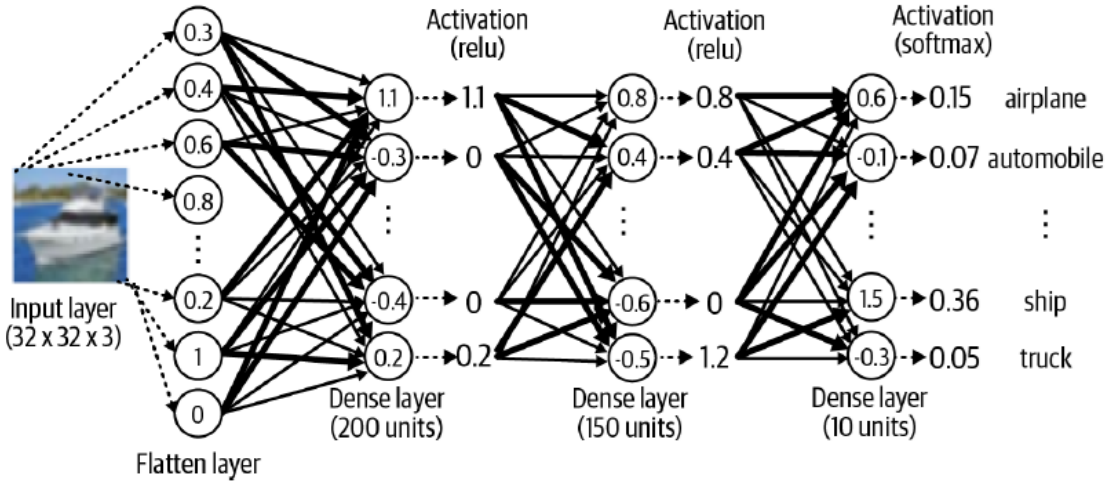


Las capas densas **reciben como entrada todos los píxeles de la imagen, sin importar su posición relativa**.

No resolvería el problema de fondo

Ignoran la estructura espacial inherente de los datos visuales

Se aplana la imagen en un vector, con lo cual **se pierde toda la estructura espacial de los pixeles**.



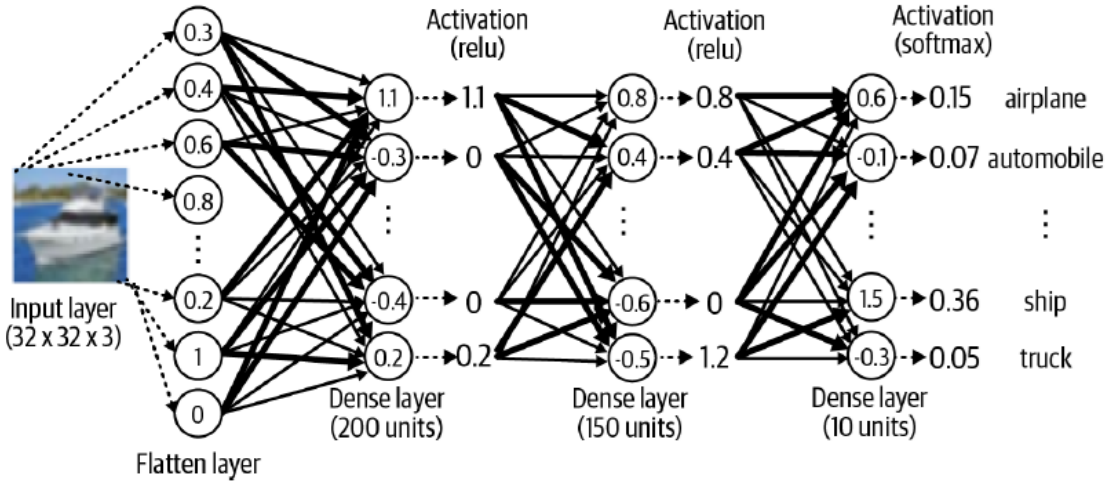
Las capas densas **reciben como entrada todos los píxeles de la imagen, sin importar su posición relativa**.

El modelo **trata cada entrada como independiente**, desconociendo las relaciones de vecindad que conforman bordes, texturas y formas.

No resolvería el problema de fondo

Ignoran la estructura espacial inherente de los datos visuales

Se aplana la imagen en un vector, con lo cual **se pierde toda la estructura espacial de los pixeles**.



Las capas densas **reciben como entrada todos los píxeles de la imagen, sin importar su posición relativa**.

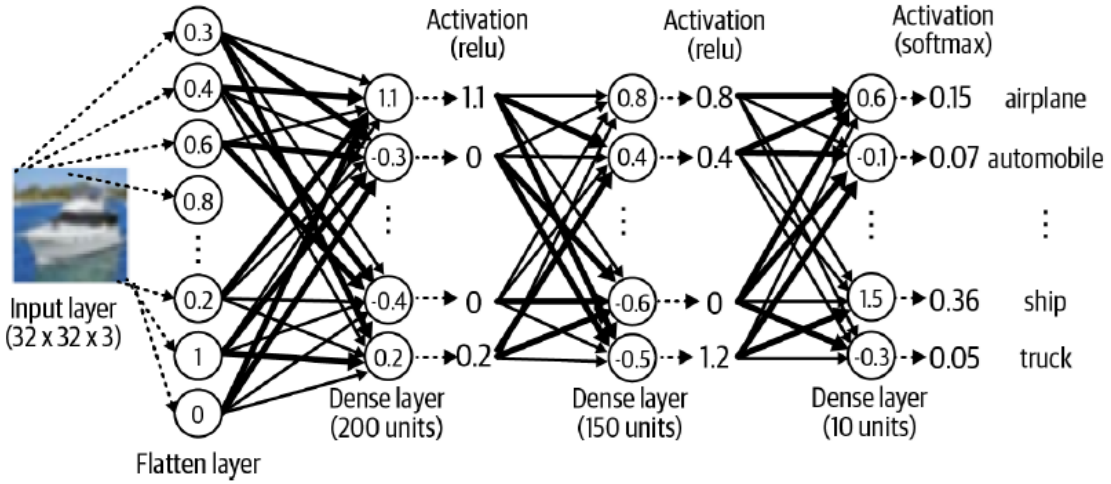
El modelo **trata cada entrada como independiente**, desconociendo las relaciones de vecindad que conforman bordes, texturas y formas.

Esto provoca que un patrón visual —por ejemplo, el trazo curvo característico de un "3"— sea interpretado como algo completamente distinto si se desplaza apenas unos píxeles dentro del campo de visión.

No resolvería el problema de fondo

Ignoran la estructura espacial inherente de los datos visuales

Se aplana la imagen en un vector, con lo cual **se pierde toda la estructura espacial de los pixeles**.



Las capas densas **reciben como entrada todos los píxeles de la imagen, sin importar su posición relativa**.

El modelo **trata cada entrada como independiente**, desconociendo las relaciones de vecindad que conforman bordes, texturas y formas.

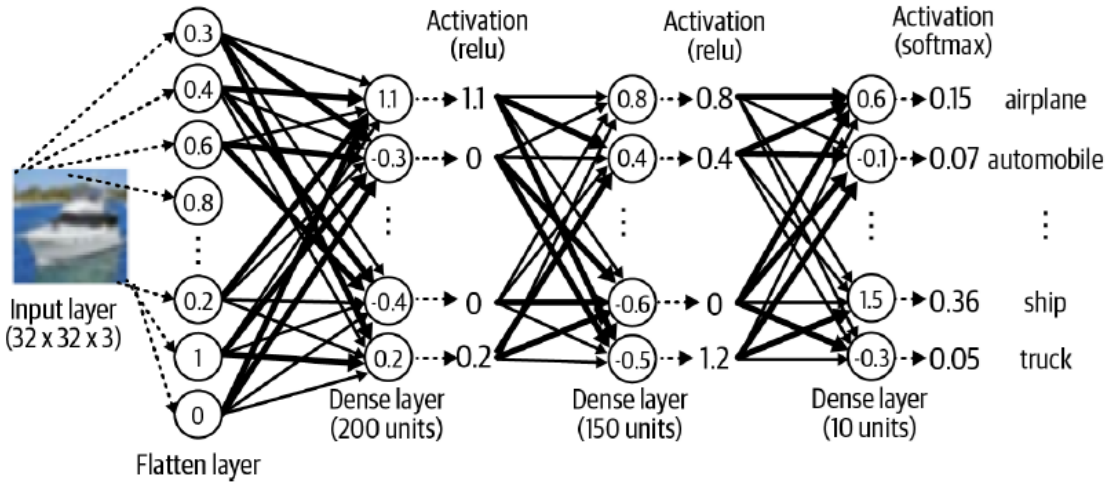
Esto provoca que un patrón visual —por ejemplo, el trazo curvo característico de un "3"— sea interpretado como algo completamente distinto si se desplaza apenas unos píxeles dentro del campo de visión.

Un MLP debe "reaprender" el mismo concepto en todas las posiciones posibles de la imagen.

No resolvería el problema de fondo

Ignoran la estructura espacial inherente de los datos visuales

Se aplana la imagen en un vector, con lo cual **se pierde toda la estructura espacial de los pixeles**.



Las capas densas **reciben como entrada todos los píxeles de la imagen, sin importar su posición relativa**.

El modelo **trata cada entrada como independiente**, desconociendo las relaciones de vecindad que conforman bordes, texturas y formas.

Esto provoca que un patrón visual —por ejemplo, el trazo curvo característico de un "3"— sea interpretado como algo completamente distinto si se desplaza apenas unos píxeles dentro del campo de visión.

Un MLP debe "reaprender" el mismo concepto en todas las posiciones posibles de la imagen.

Esto conduce a un **crecimiento explosivo en la necesidad de datos**, y a una **incapacidad de generalizar** de forma robusta frente a translaciones, rotaciones u otras variaciones espaciales.

Además ...

- **Complejidad visual:** gatos, perros, autos… tienen texturas y formas mucho más variadas que los dígitos.
- **Variabilidad intra-clase:** un “gato” puede ser pequeño, grande, acostado, corriendo, visto de frente o de perfil.
- **Colores y fondos:** los objetos aparecen con distintos colores y en contextos distintos, a diferencia de los dígitos en blanco y negro sobre fondo negro.

Necesitamos otro tipo de capas

- Respeten la estructura local:

  Aprovechar que los píxeles vecinos forman bordes, texturas y patrones.

- Robustas a transformaciones:

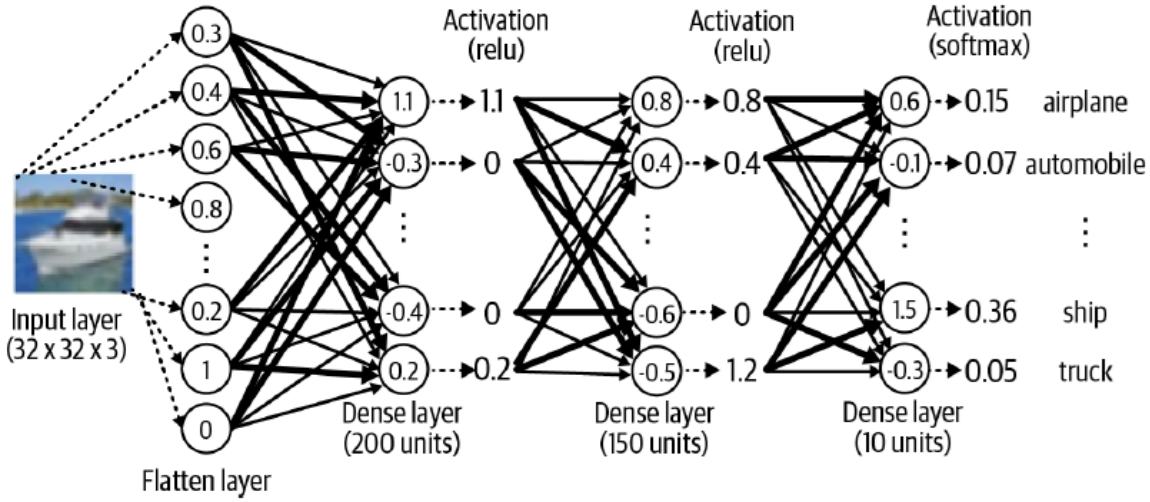
  Reconocer un objeto aunque esté desplazado, rotado, escalado, reflejado o deformado ligeramente dentro de la imagen.

- Eficiencia en escalamiento:

  Permitir que la red crezca con el tamaño de la imagen sin un aumento exponencial de parámetros

- Mejor generalización:

  Aprender patrones reutilizables en distintas posiciones y contextos, en lugar de memorizarlos.



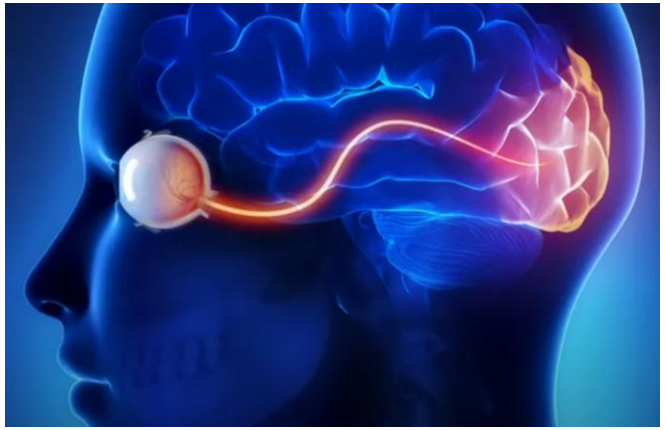
Redes Convolucionales

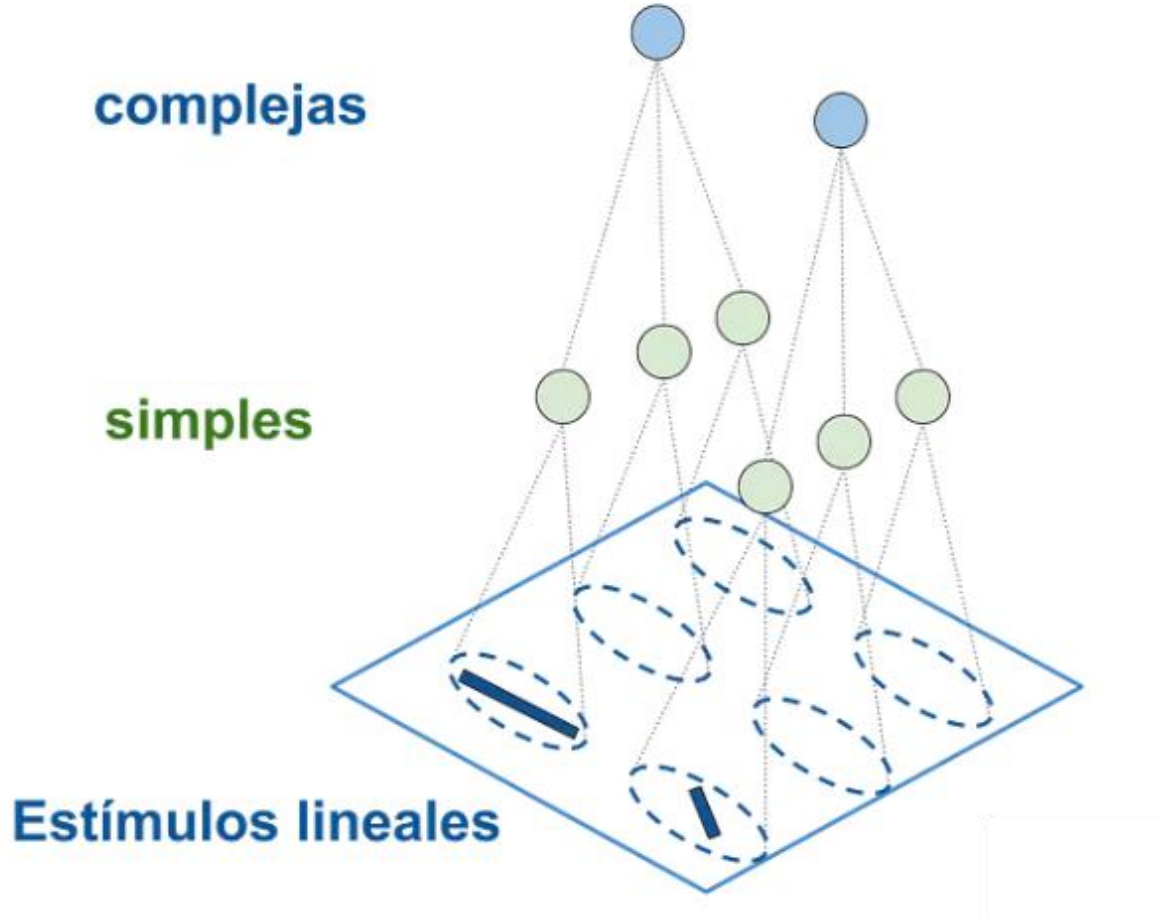
Un tipo de redes neuronales que se inspiran en la corteza visual.

1962

David Huber and Torsten Weisel

Descubrieron que la corteza visual de los gatos (donde se procesa lo que ven los ojos) tiene dos tipos de neuronas diferentes:





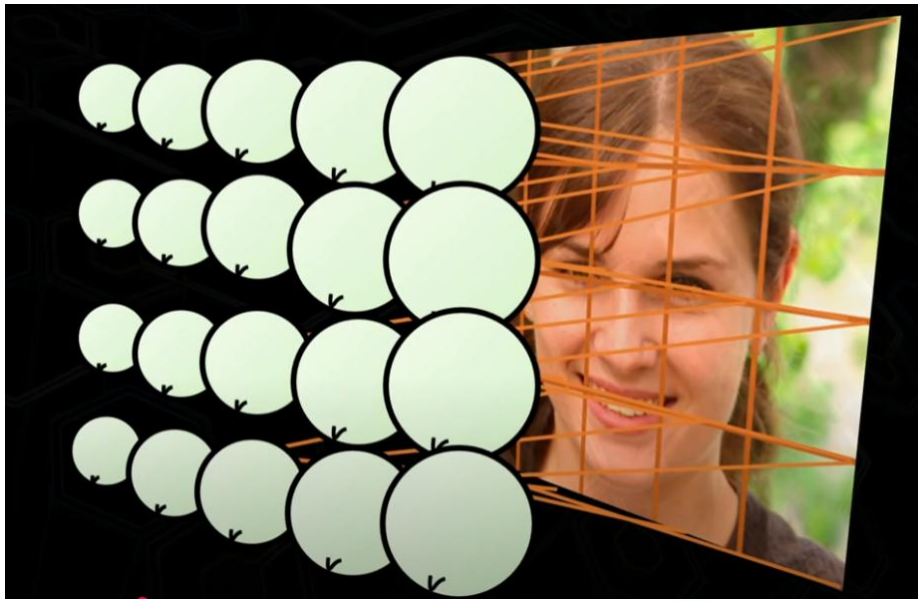
Redes Convolucionales

Un tipo de redes neuronales que se inspiran en la corteza visual.

Neuronas simples

Sólo ven una pequeña parte del campo visual

Tenemos muchas, cada una cubriendo un pedacito, hasta cubrir todo el campo visual.



Se activan y disparan cuando ven líneas con cierta orientación, en la pequeña parte que perciben.

Se activa.

No se activa.

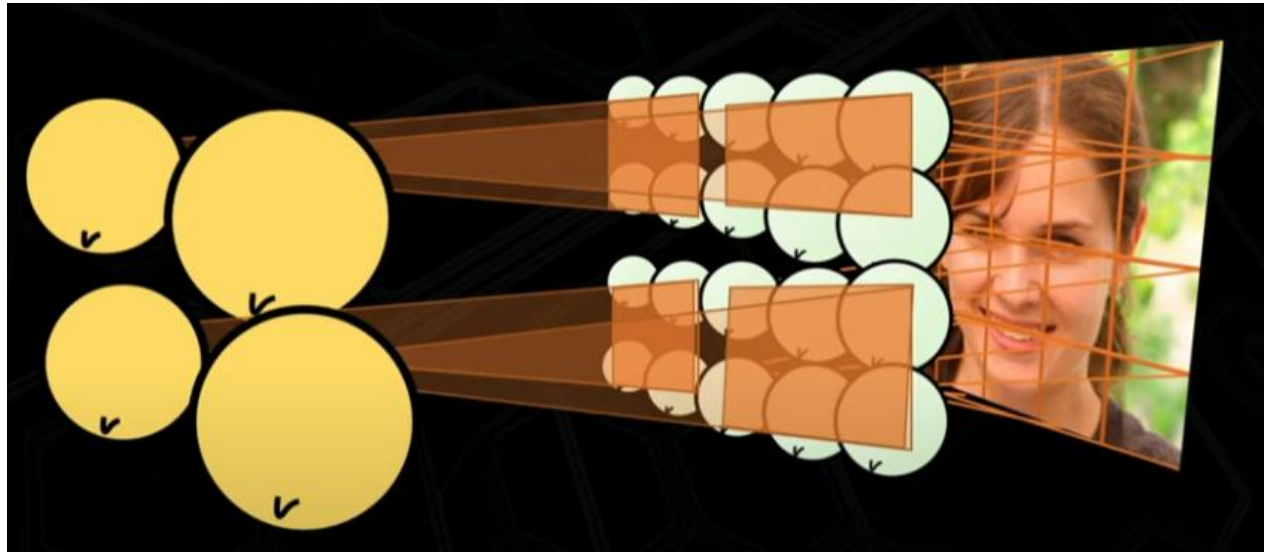
Redes Convolucionales

Un tipo de redes neuronales que se inspiran en la corteza visual.

Neuronas complejas

Agrupan lo que ven varias neuronas simples.

Entonces, su campo receptivo es más grande.



Se activan y disparan también con las líneas, pero a ellas no les importa su dirección ni posición, pero dan preferencia a las neuronas que fueron fuertemente activadas

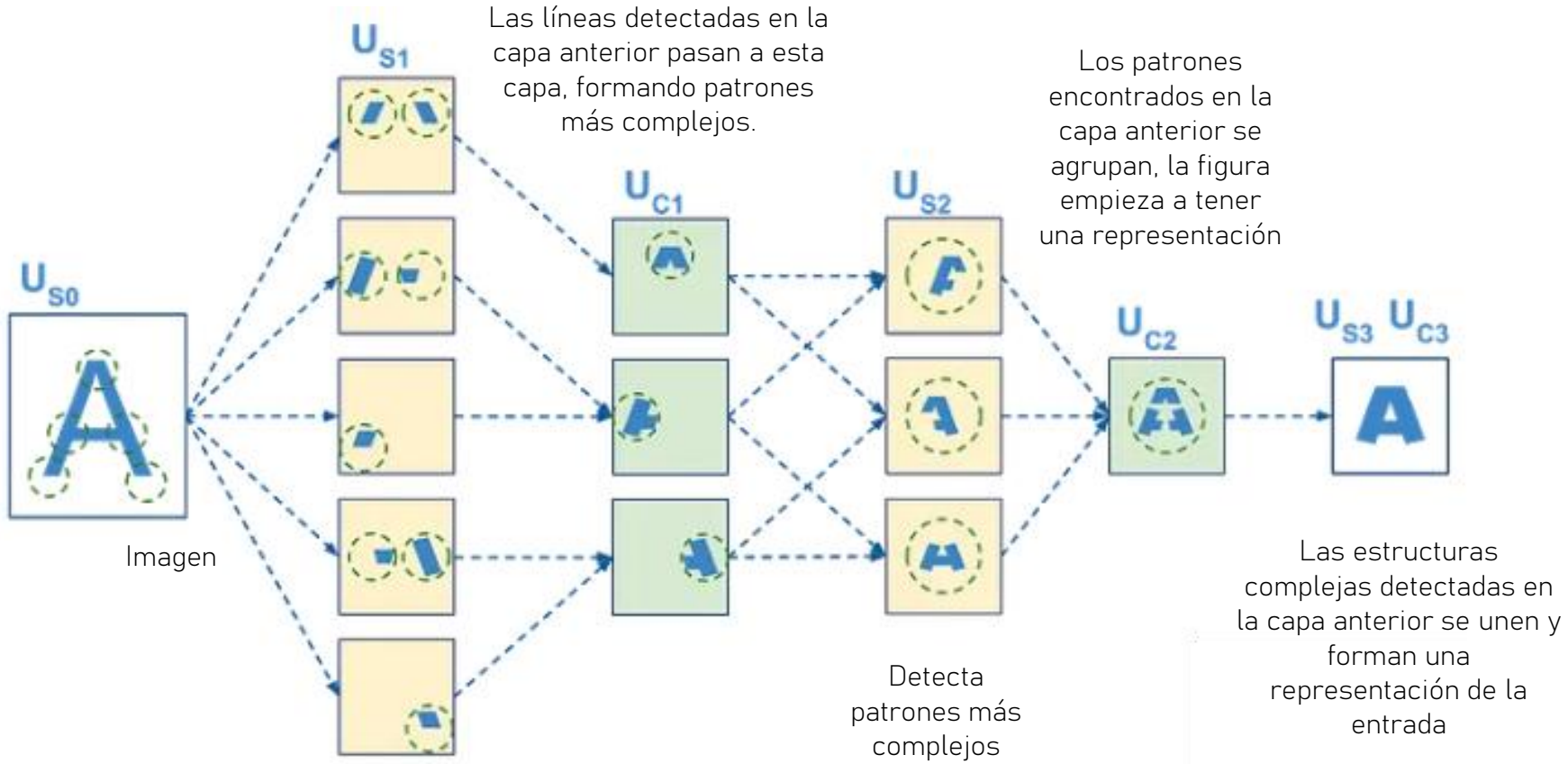
Se activa.

Se activa.

Se activa.

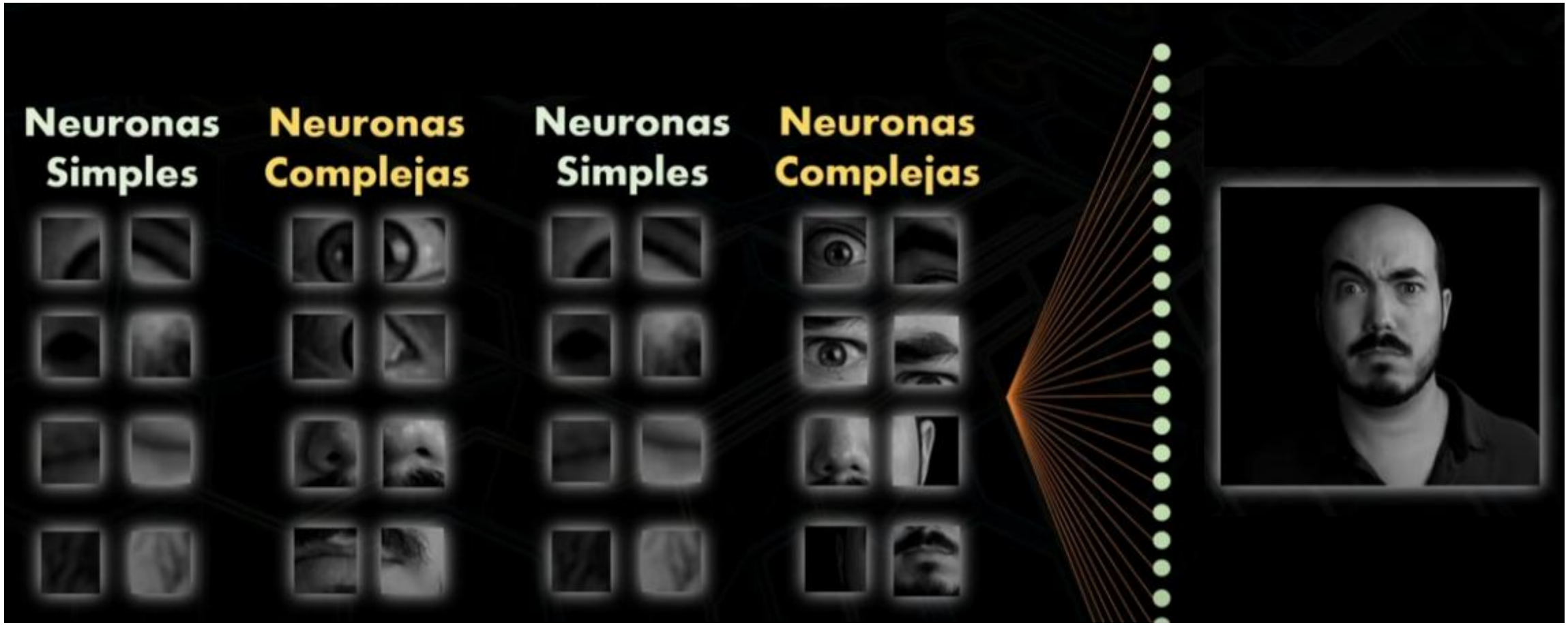
Redes Convolucionales

Un tipo de redes neuronales que se inspiran en la corteza visual.



Redes Convolucionales

Un tipo de redes neuronales que se inspiran en la corteza visual.



Redes Convolucionales

Un tipo de redes neuronales que se inspiran en la corteza visual.

1980

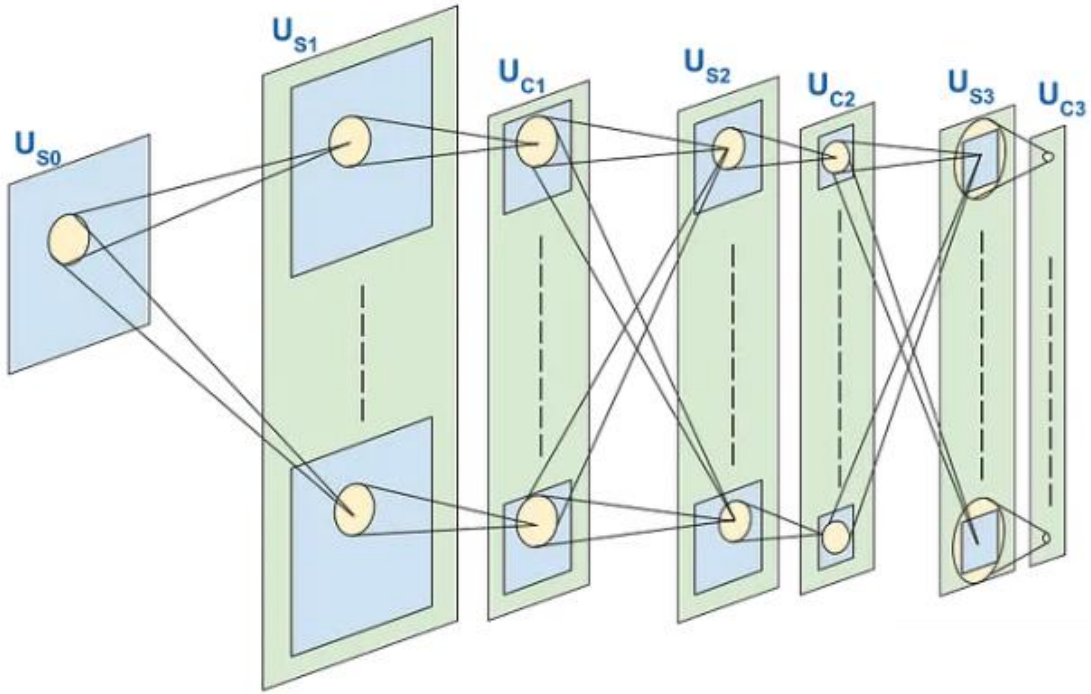
Fukushima: Neocognitron

- Basado en el trabajo de Hubel y Wiesel propuso:
  - La primera arquitectura artificial con capas jerárquicas de "receptive fields" → antecesor conceptual de las CNN.

Propuesta para el reconocimiento de números y caracteres escritos a mano.

Con la novedad de que podía identificarlos sin importar su ubicación en la imagen de entrada.

No usaba backpropagation, porque aún no se popularizaba.



Redes Convolucionales

Un tipo de redes neuronales que se inspiran en la corteza visual.

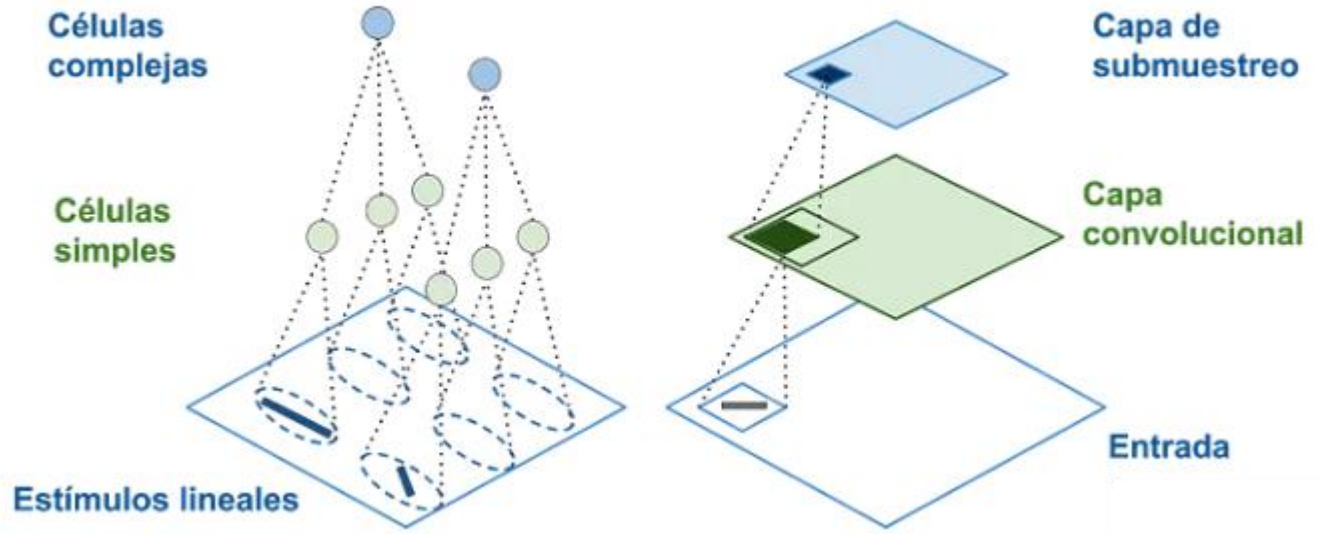
1998

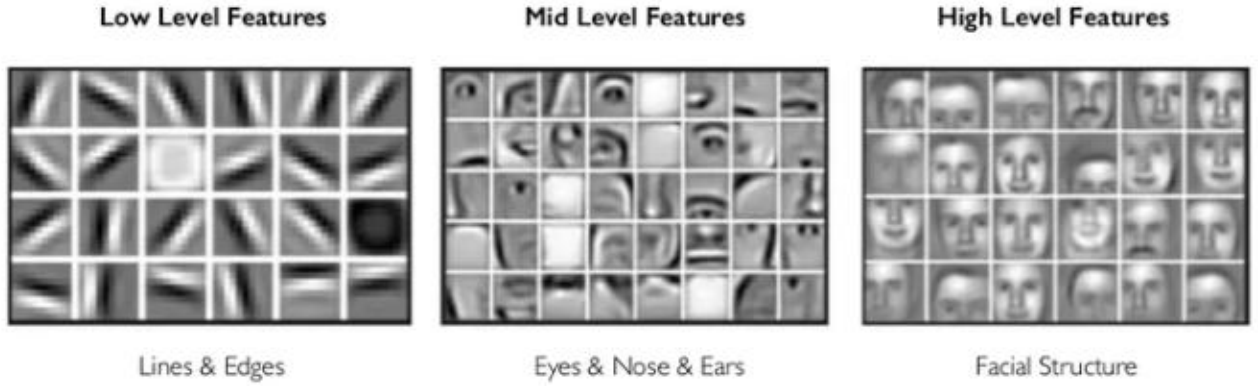
Lecun y su equipo

Propusieron un modelo de red neuronal basado en el operador matemático de la **convolución**.

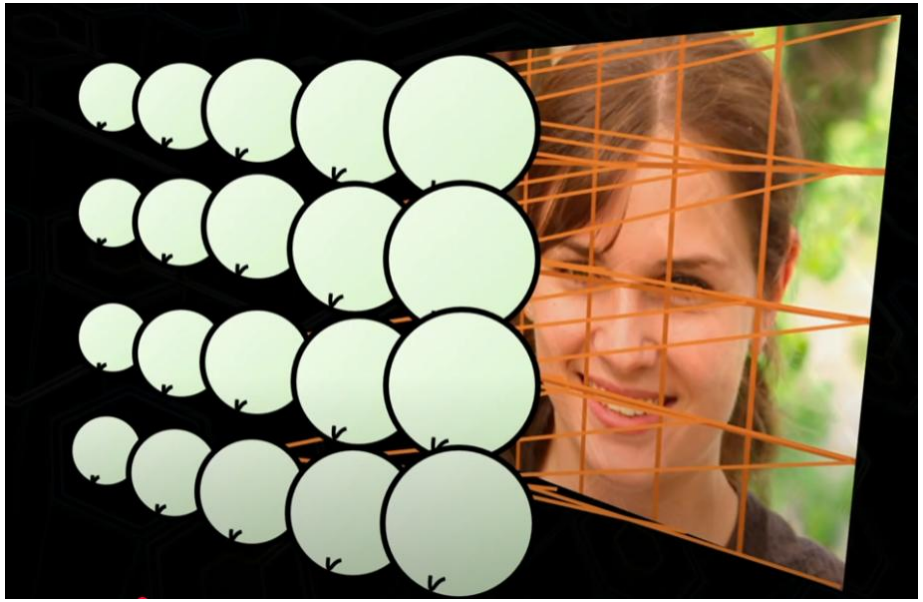
Usa filtros para detección de bordes, suavizado, eliminación de ruido, etc.

Primera CNN entrenada con **retropropagación**.





Neuronas simples



¿Cómo podemos detectar líneas en imágenes?

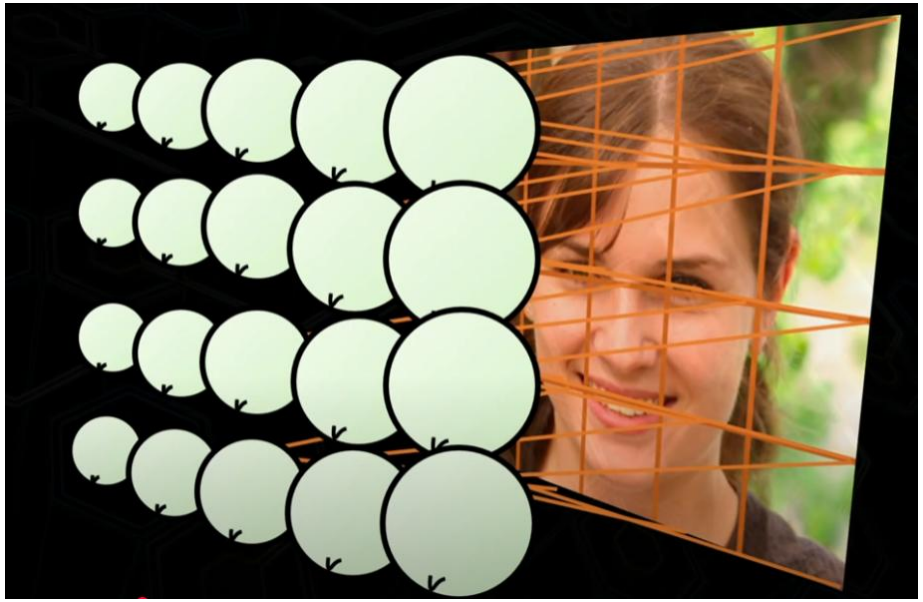
0

255

175

Si nos fijamos en un solo pixel no podemos determinar si ahí hay una línea.

Neuronas simples



¿Cómo podemos detectar líneas en imágenes?

0

255

176	182	184	166	91
180	182	183	88	62
181	191	175	66	62
181	188	157	97	75
182	184	110	106	90

Necesitamos fijarnos en una vecindad.

Comparar sus valores.

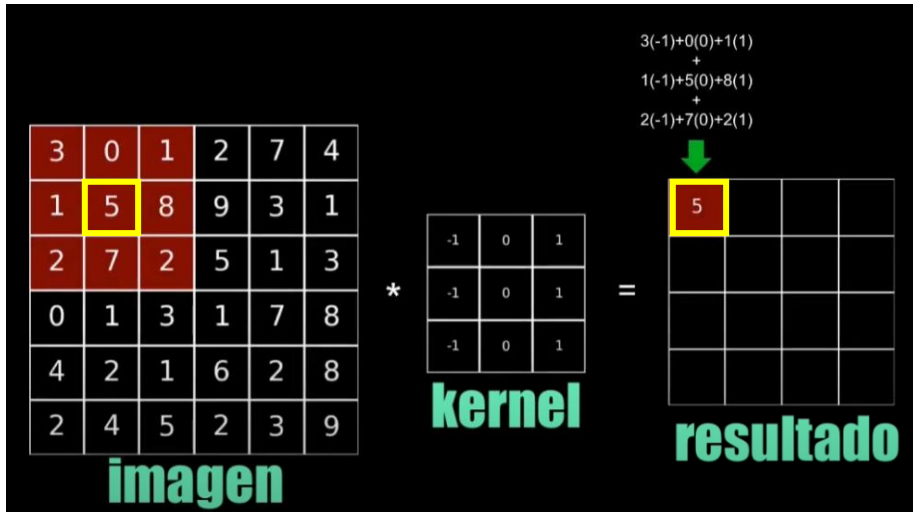
Saber si hay un cambio fuerte en los valores.

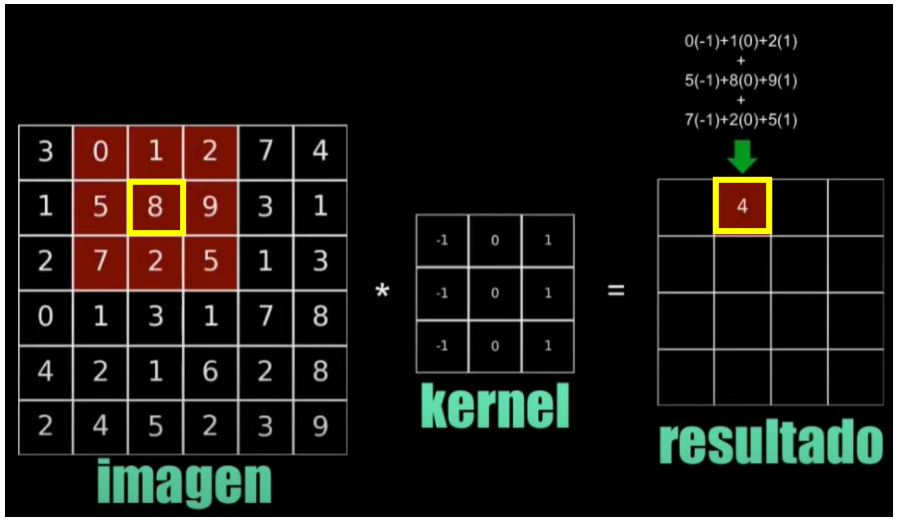
Operación de convolución

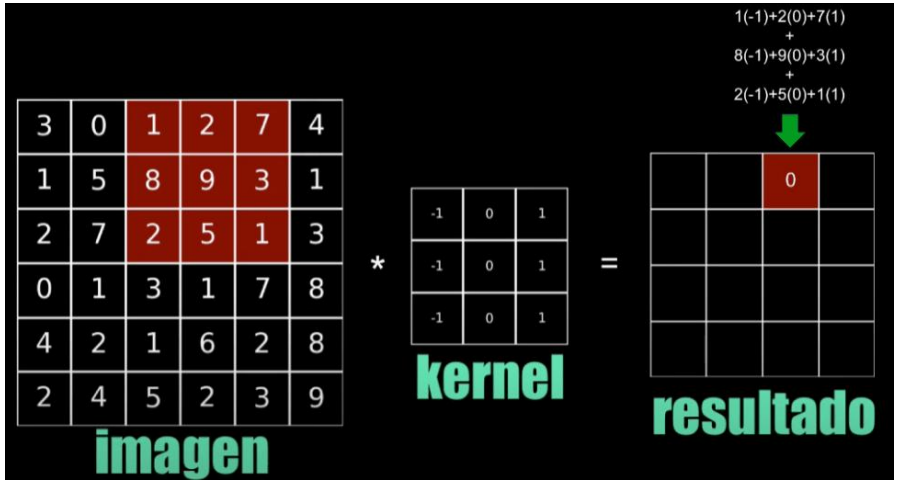
Con imágenes en escala de grises

$$S(i,j) = \sum_{m=0}^{k-1}\sum_{n=0}^{k-1} I(i+m, j+n) \cdot K(m,n)$$

Nota que no podemos iniciar desde el primer pixel.







… para todos los pixeles de la imagen.

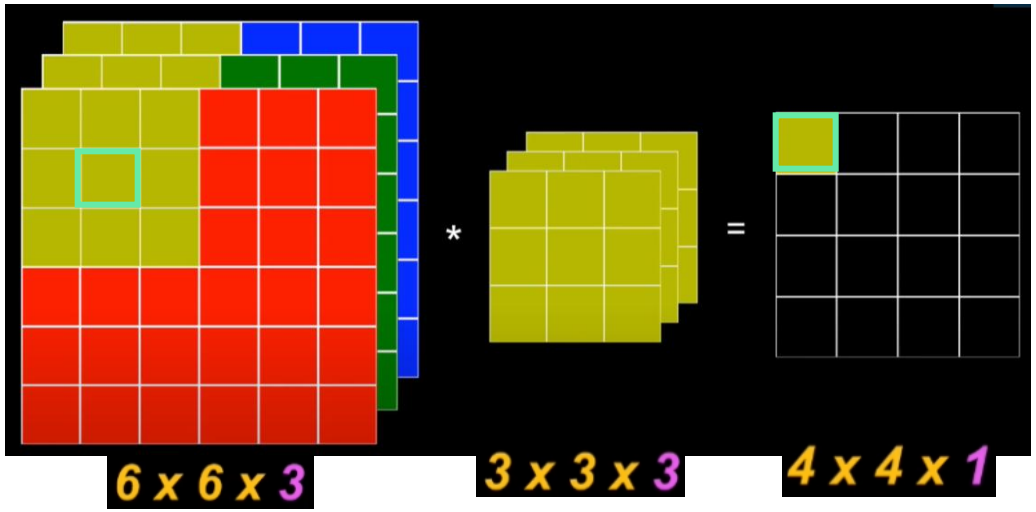
Nota que la imagen resultante es un renglón y una columna más pequeña que la original.

Operación de convolución

Con imágenes a color

$$S(i,j) = \sum_{c=1}^{3} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I_c(i+m, j+n) \cdot K_c(m,n)$$

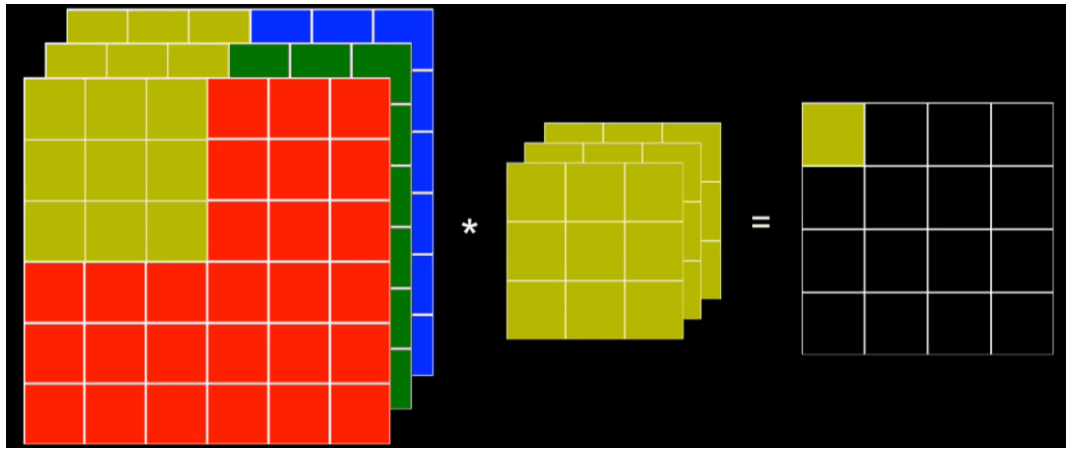
$\sum$



El filtro tiene ahora 3 planos de profundidad.

La imagen de salida tiene un solo plano de profundidad.

• • •



... para todos los pixeles de la imagen.

Operación de convolución

- Distintos valores de sus pesos enfatizan, atenúan o transforman de manera selectiva patrones visuales, generando resultados tan diversos como la detección de bordes, el suavizado, la realce de texturas o la supresión de ruido.

Original

0	0	0
0	1	0
0	0	0

Identidad

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Blur

0	-1	0
-1	5	-1
0	-1	0

Sharpen

Se conocen del orden de decenas de kernels, principalmente para:

- reducción de ruido,
- detección de bordes,
- realce de detalles,
- extracción de texturas y orientaciones.

-1	-1	-1
0	1	0
1	1	1

Gradiente horiz.

-1	0	1
-1	0	1
-1	0	1

Gradiente vert.

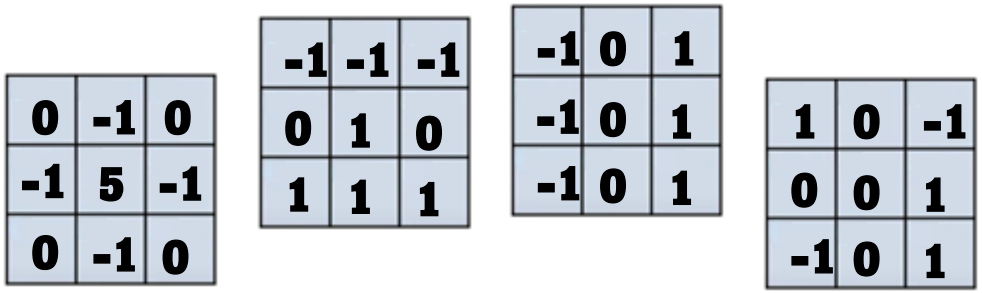
1	0	-1
0	0	1
-1	0	1

Laplaciano diag.

De filtros hechos a mano a filtros aprendidos

Antes
(Visión por computadora clásica)

- Filtros diseñados a mano por expertos.
- Se usaban para detectar bordes, esquinas, texturas.
- El flujo era:
  - Aplicar filtros fijos.
  - Extraer *features* (HOG, SIFT, SURF).
  - Entrenar un clasificador tradicional (SVM, k-NN).
- Limitación: rígidos, poco adaptables a nuevas tareas.



Ahora
(Deep Learning con CNNs)

- Cada kernel es un pequeño bloque de pesos entrenables.
- Al entrenar la red, esos pesos cambian para detectar patrones útiles (bordes, texturas, formas…).
- En capas más profundas, los kernels representan patrones cada vez más complejos.

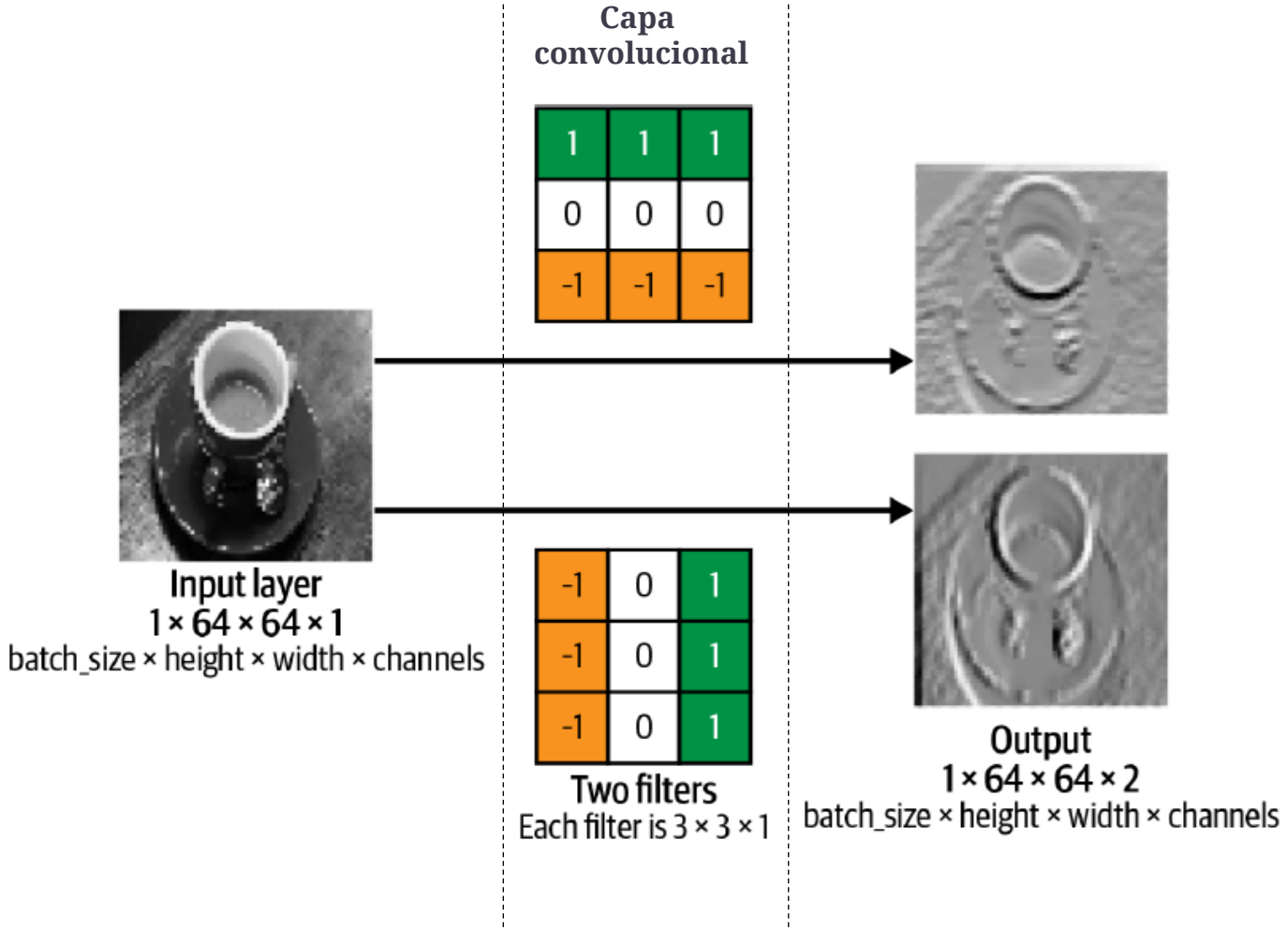
Ventaja: adaptabilidad, mejor generalización y escalabilidad.

Capas convolucionales

Una capa convolucional es simplemente una **colección de filtros**, donde los **valores almacenados en los filtros son los pesos** que la red neuronal aprende durante el entrenamiento.

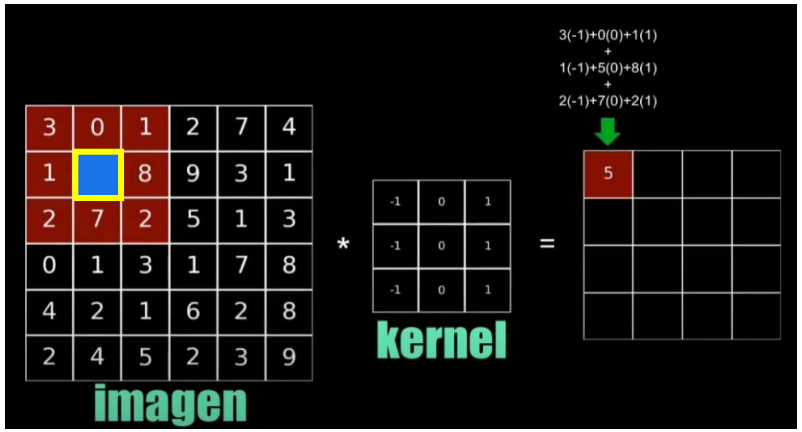
Inicialmente, estos son **aleatorios**, pero gradualmente los filtros ajustan sus pesos para comenzar a **identificar características interesantes**, como bordes o combinaciones específicas de colores.

```
from tensorflow.keras import layers

input_layer = layers.Input(shape=(64,64,1))
conv_layer_1 = layers.Conv2D(
    filters = 2
    , kernel_size = (3,3)
    , strides = 1
    , padding = "same"
    )(input_layer)
```



Capas convolucionales

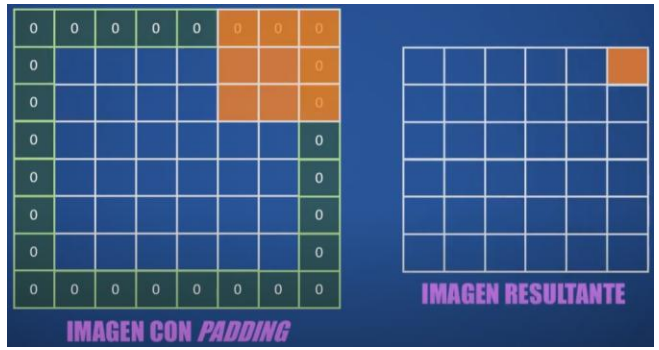
La imagen resultante de la convolución es un renglón y una columna más pequeña que la original.



Si queremos que la imagen resultante tenga el mismo tamaño que la original, podemos usar:

Padding

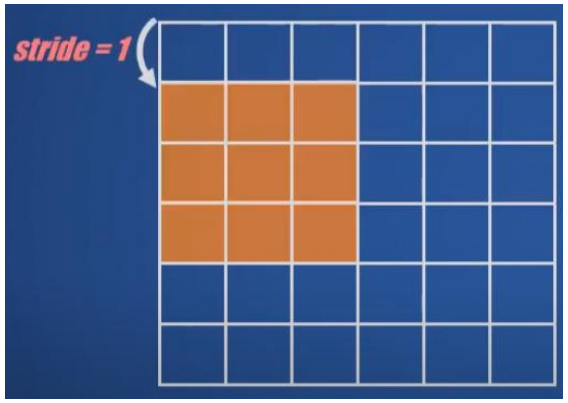
- Agrega filas y columnas alrededor de la imagen original.
- Generalmente con ceros.
- Antes de aplicar la convolución.
- Esto permite:
  1. Conservar el tamaño original de la imagen tras la convolución.
  1. Asegurar que los bordes también sean procesados por el kernel.



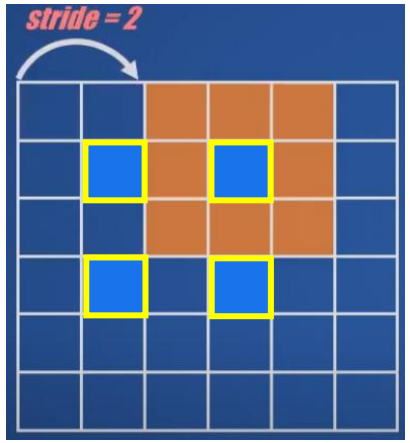
Capas convolucionales

Stride (paso)

- Indica **cuántos pixeles se avanza al mover el kernel** en cada dirección (horizontal y vertical).
- **Valor por defecto:** stride=1 -> el kernel se mueve pixel a pixel.
- Si stride=2 -> el kernel "salta" de dos en dos pixeles -> la salida es más pequeña.



- *Strides* más grandes reducen el tamaño de la imagen de salida.



Capas convolucionales

Padding + Stride
1 2

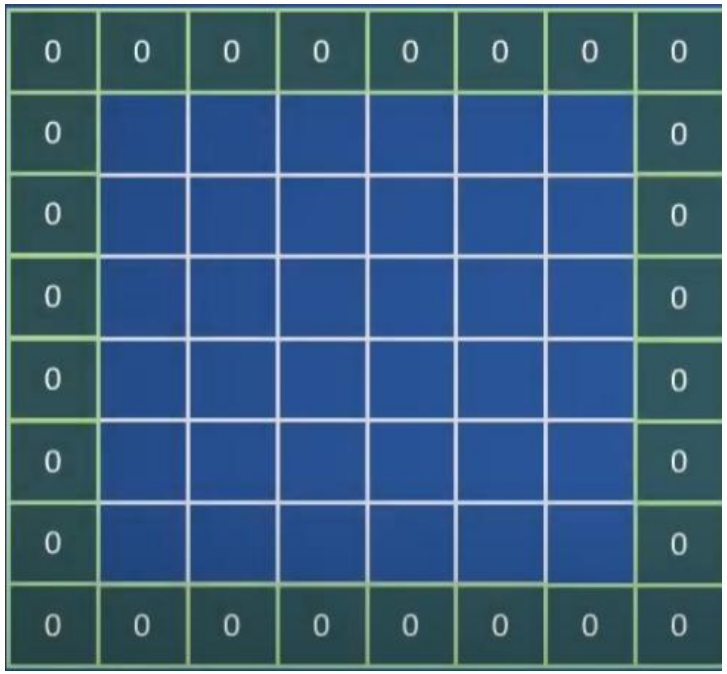


Imagen original

¿De qué tamaño queda la imagen convolucionada?

Capas convolucionales

Padding + Stride
1 2

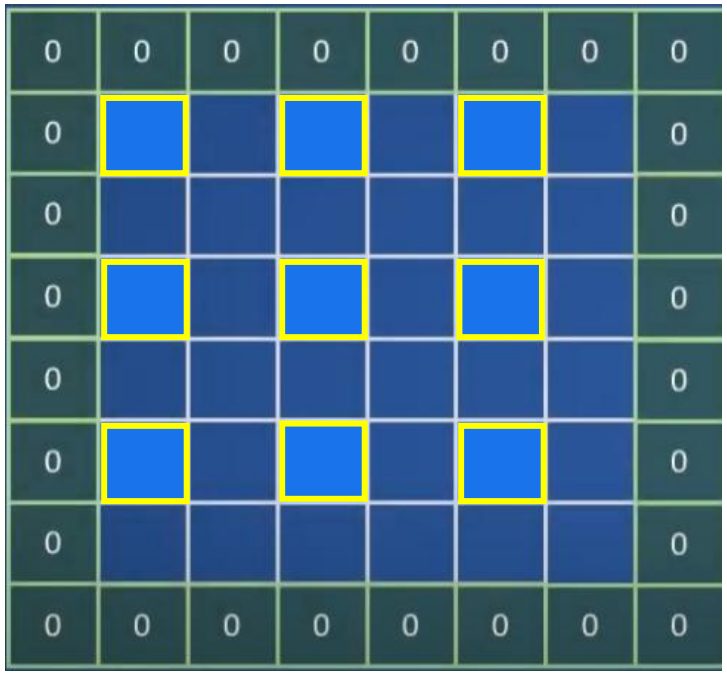


Imagen original
8x8

Kernel
3x3

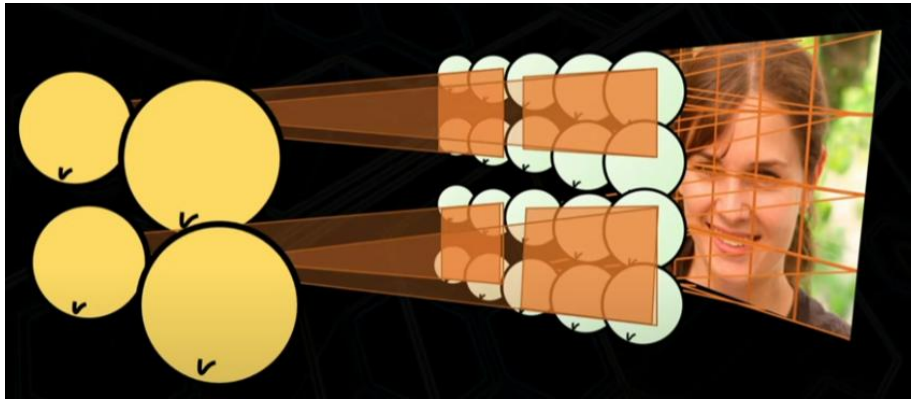
¿De qué tamaño queda la imagen convolucionada?

Imagen resultante
3x3

Capas convolucionales

Neuronas complejas

Agrupan lo que ven varias neuronas simples.



Se puede implementar con:

Padding + Stride

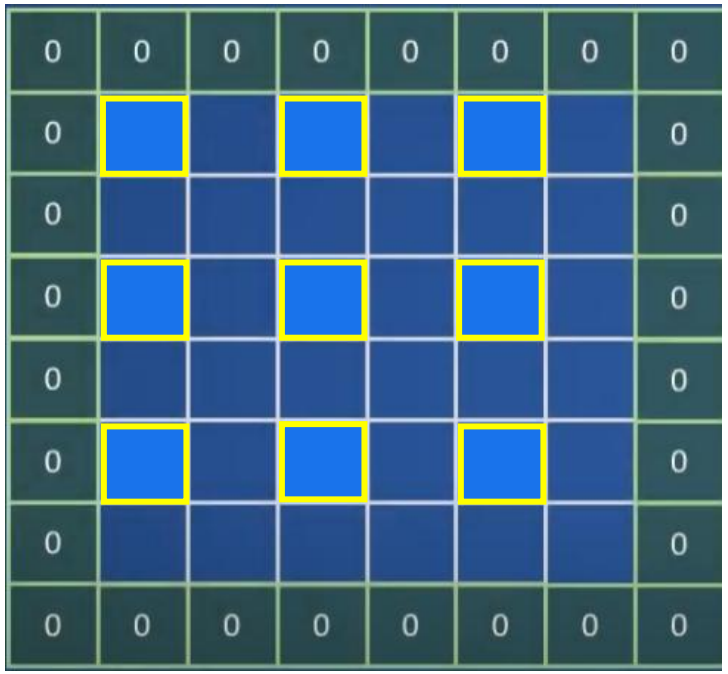


Imagen original

8x8

Kernel

3x3

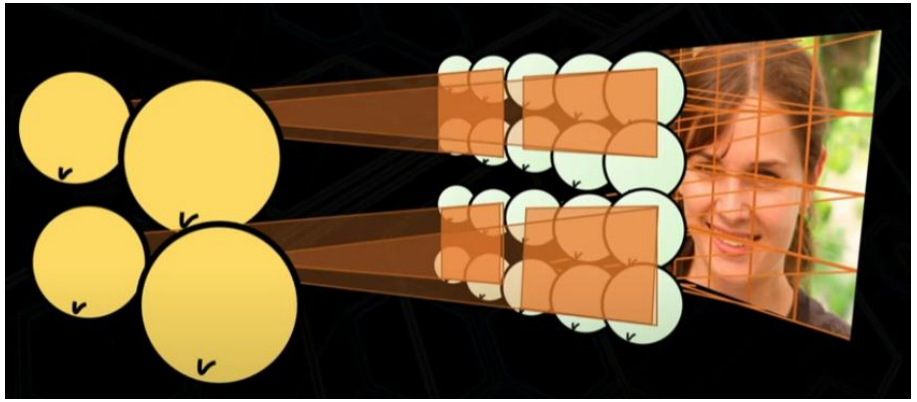
Imagen resultante

3x3

Capas convolucionales

Neuronas complejas

Agrupan lo que ven varias neuronas simples.



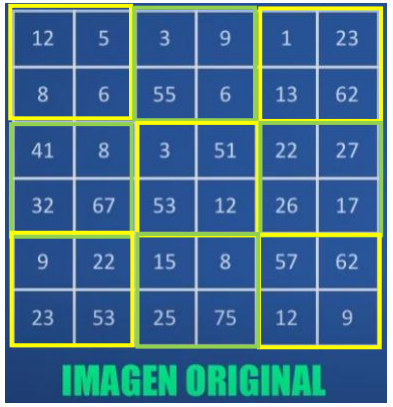
Max Pooling

Reduce la cantidad de datos y preserva la información.

- Se divide la imagen en regiones del mismo tamaño.
- Para cada región se extrae el valor máximo.

Max pooling = 2

9 regiones



12	55	62
67	53	27
53	75	62

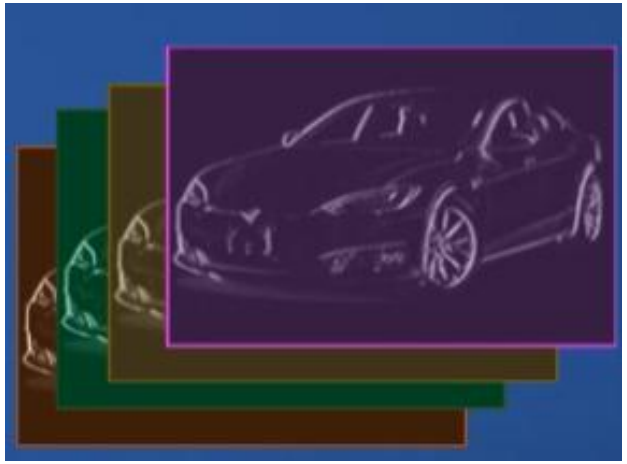
Imagen resultante

* Average pooling: En lugar de tomar el **valor máximo**, toma el **promedio** de la ventana.

Capas convolucionales

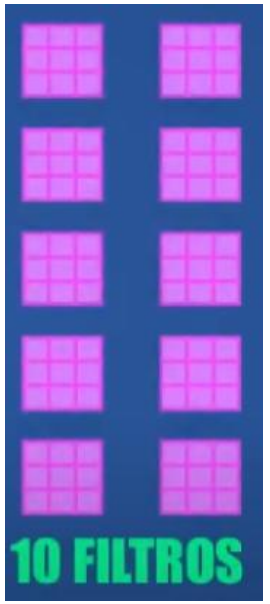
Stacking

Apilar múltiples imágenes resultantes de convolucionar la imagen con diferentes kernels.

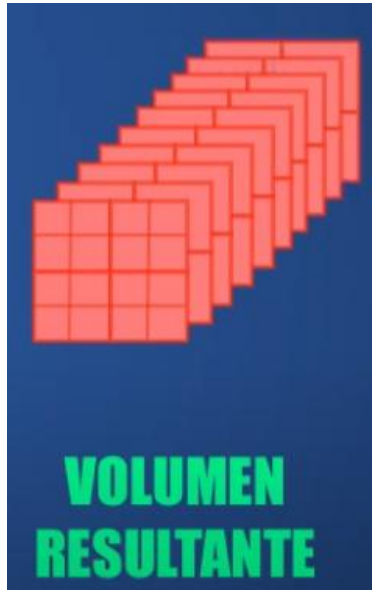


Padding=0
Stride = 1

6x6

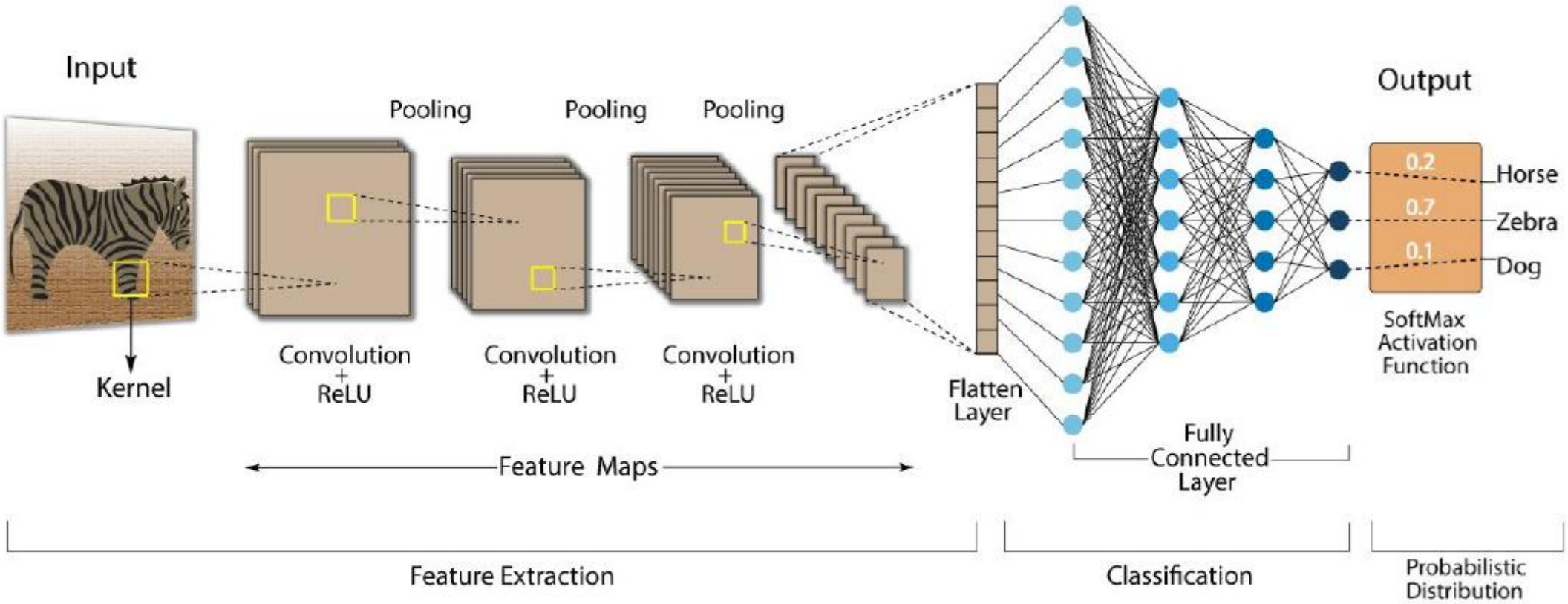


3x3



4x4x10

Poniendo todo junto

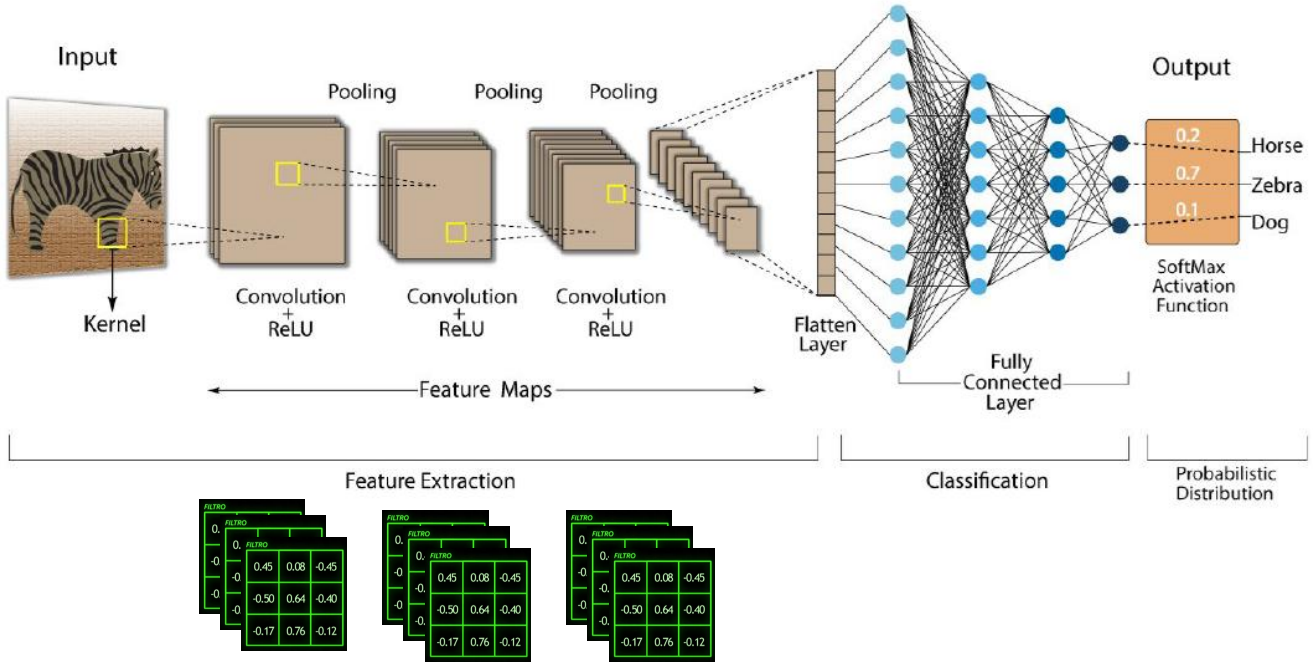


¿Qué aprenden las CNNs?

En una **CNN**, cada **filtro** funciona como un conjunto de pesos compartidos, aplicado a todos los píxeles de la entrada.

Los parámetros entrenables son:

- **Pesos de los kernels** de las capas convolucionales.
- **Sesgos** asociados a cada filtro.
- (Opcional) **Pesos y biases de las capas densas al final de la red**.



Aprende **qué patrones buscar** (bordes, texturas, formas, etc.) ajustando los valores de estos filtros.

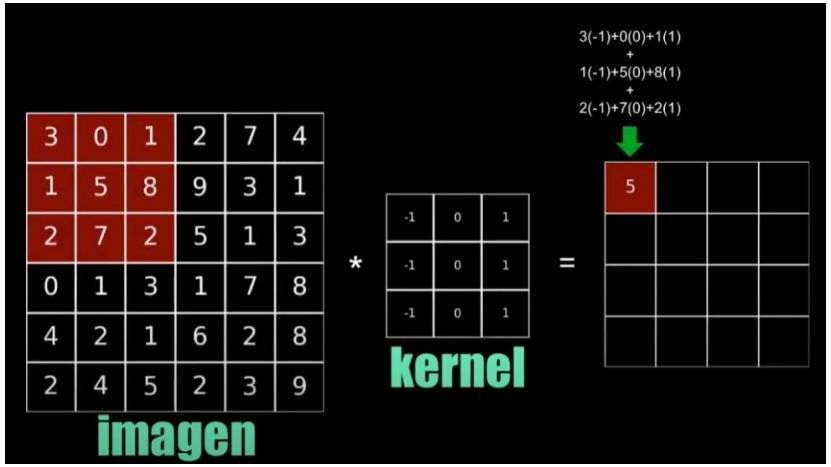
Función de activación

Para que pueda aprender relaciones no lineales entre las características de la imagen, se aplica una función de activación después de cada convolución.

- Una convolución sin activación es una operación lineal (multiplicación + suma + sesgo).

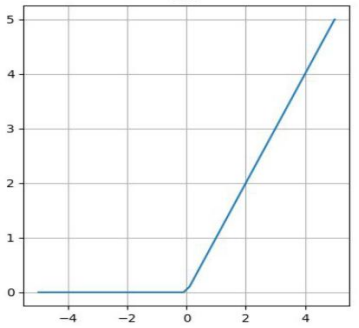
$$\sum_{i,j} W_{i,j} \cdot X_{i,j} + b$$

*cada filtro tiene un sesgo único



- Si encadenamos varias convoluciones sin activaciones, todo el bloque sigue siendo equivalente a una sola convolución grande → la red pierde capacidad expresiva.

$$\mathrm{ReLU}(x) = \max(0, x)$$



Simplicidad computacional

- ReLU solo requiere una comparación → **muy barata** de calcular.
- Esto acelera el entrenamiento en GPU comparado con activaciones más pesadas como **sigmoid** o **tanh**.

Mejora el desvanecimiento del gradiente

- En sigmoid/tanh, los gradientes se vuelven muy pequeños en las regiones saturadas.
- En ReLU, para $x > 0$, la derivada es 1, lo que permite un flujo de gradientes más estable en redes profundas.

Induce dispersidad (sparsity)

- Como los valores negativos se cortan a 0, muchas neuronas quedan inactivas.
- Esto hace a la red más eficiente y ayuda a representar datos de manera dispersa, lo que mejora la generalización.

Funciona muy bien en la práctica

- funciona en casi todos los casos

¿Cómo aprenden los pesos?

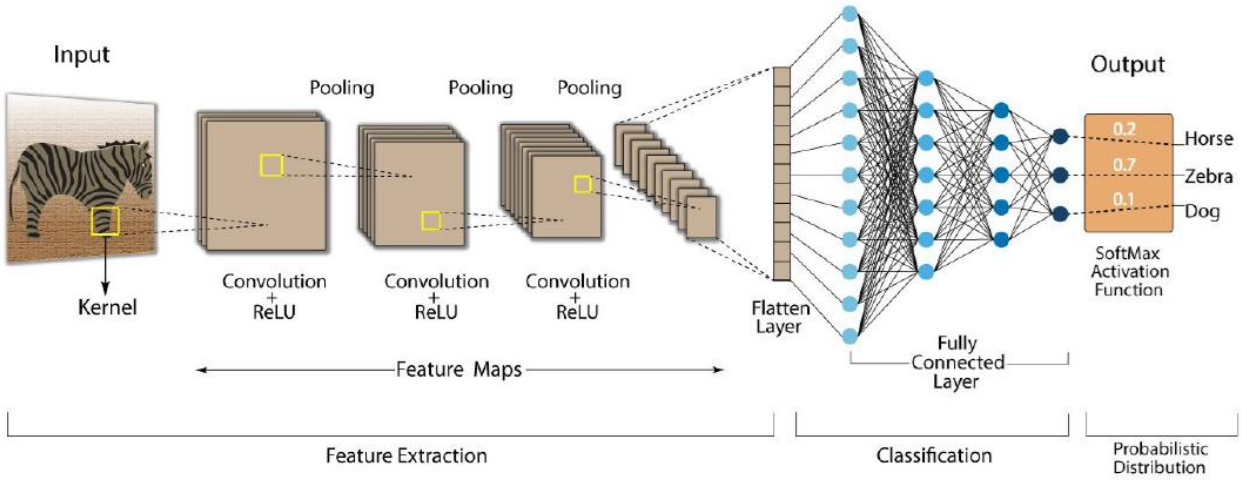
De manera similar a cómo se lo hacen los perceptrones multicapa.

1. Propagación hacia adelante (forward pass)

- La imagen entra.
- Cada capa convolucional aplica sus filtros.
- Pasa por activaciones, pooling, y al final capas densas.
- Se genera una predicción $\hat{y}$
- Cálculo del error (función de pérdida)
    - Compara la predicción $\hat{y}$ con la etiqueta real y.

2. Propagación hacia atrás

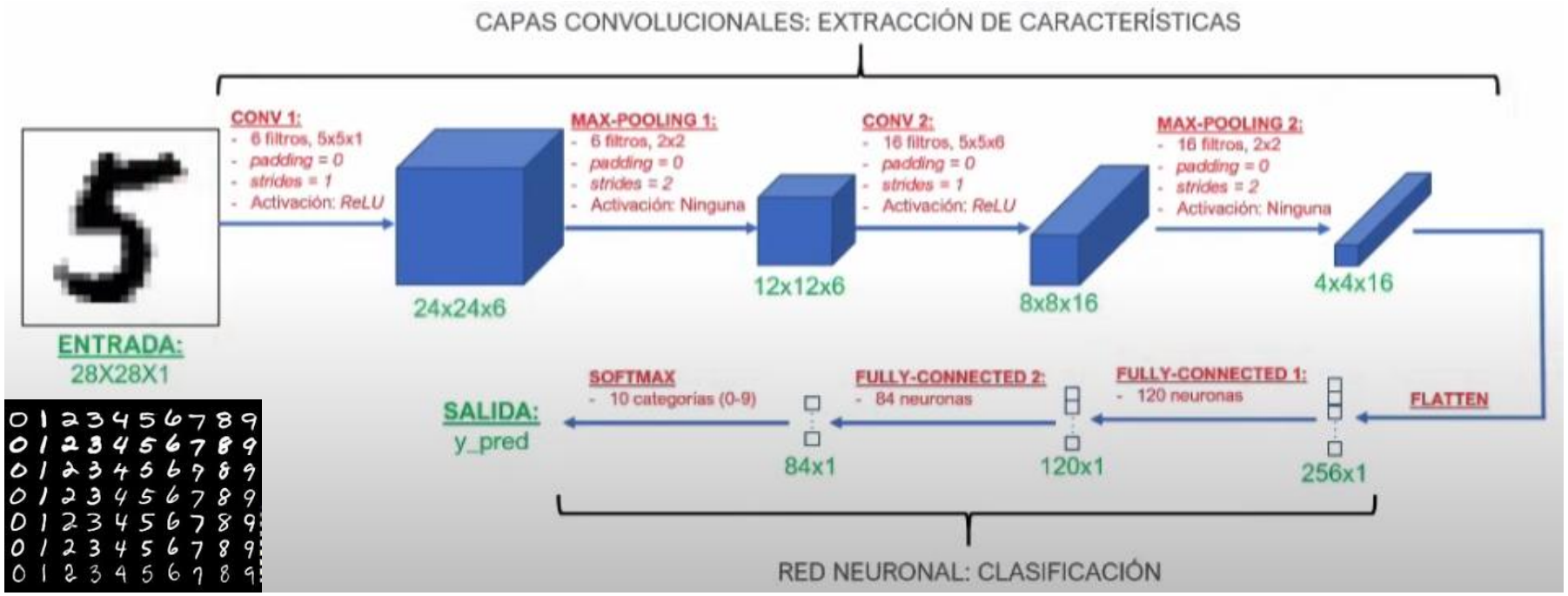
- Se calculan gradientes de la pérdida respecto a cada peso (filtros, biases).
- Se actualizan los pesos y sesgos (optimizado: SGD, Adam, etc.).



LeNet-5

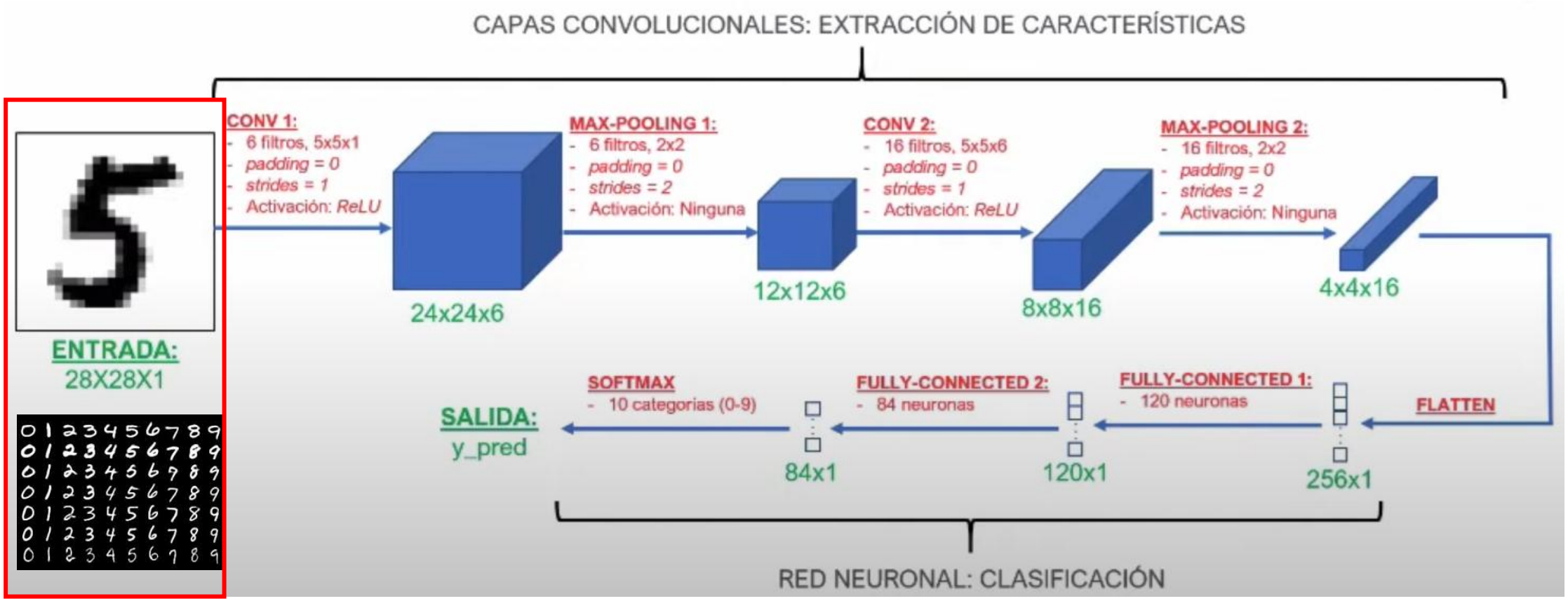
Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324

- Fue la primera CNN entrenada exitosamente en tareas reales (lectura automática de cheques bancarios).
- Mostró que las convoluciones y el submuestreo (pooling) podían aprender características jerárquicas de imágenes.
- Sentó las bases de arquitecturas modernas.



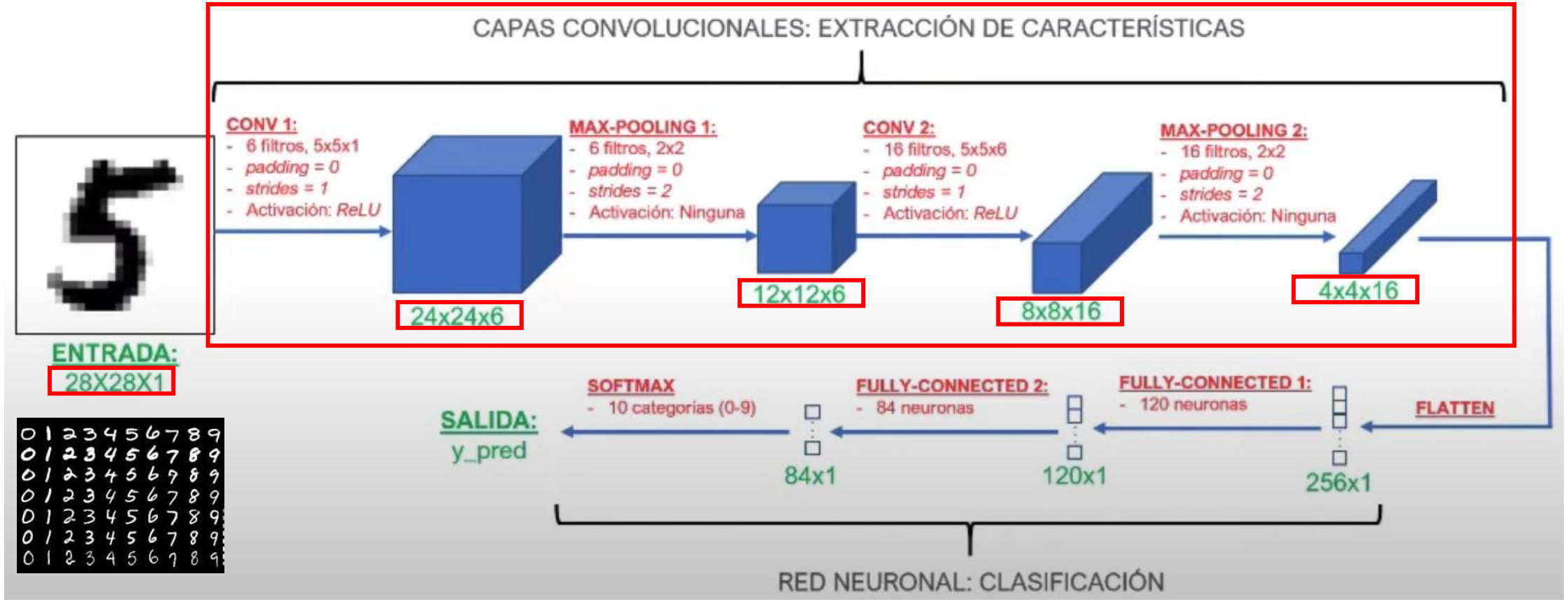
LeNet-5

Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324



LeNet-5

Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324

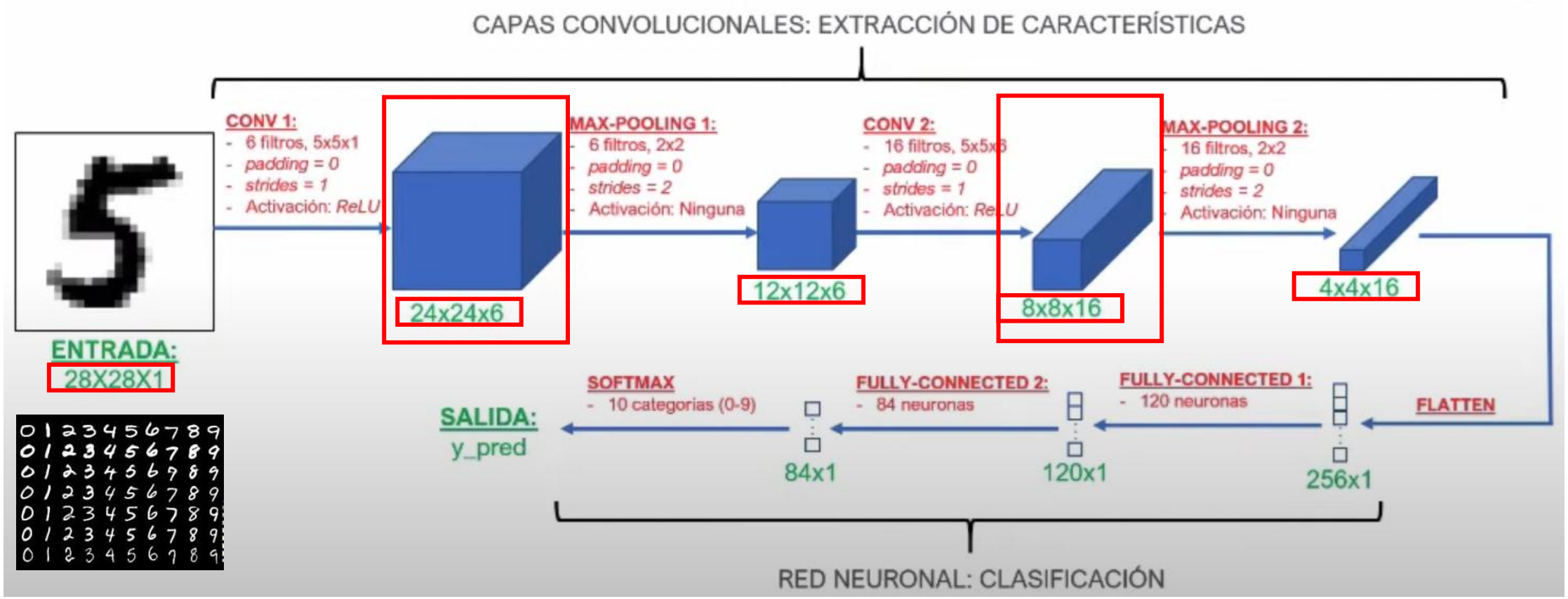


* Nota:

- Aquí estamos usando una función de activación ReLU, pero originalmente se usaba la función sigmoide o tanh.

LeNet-5

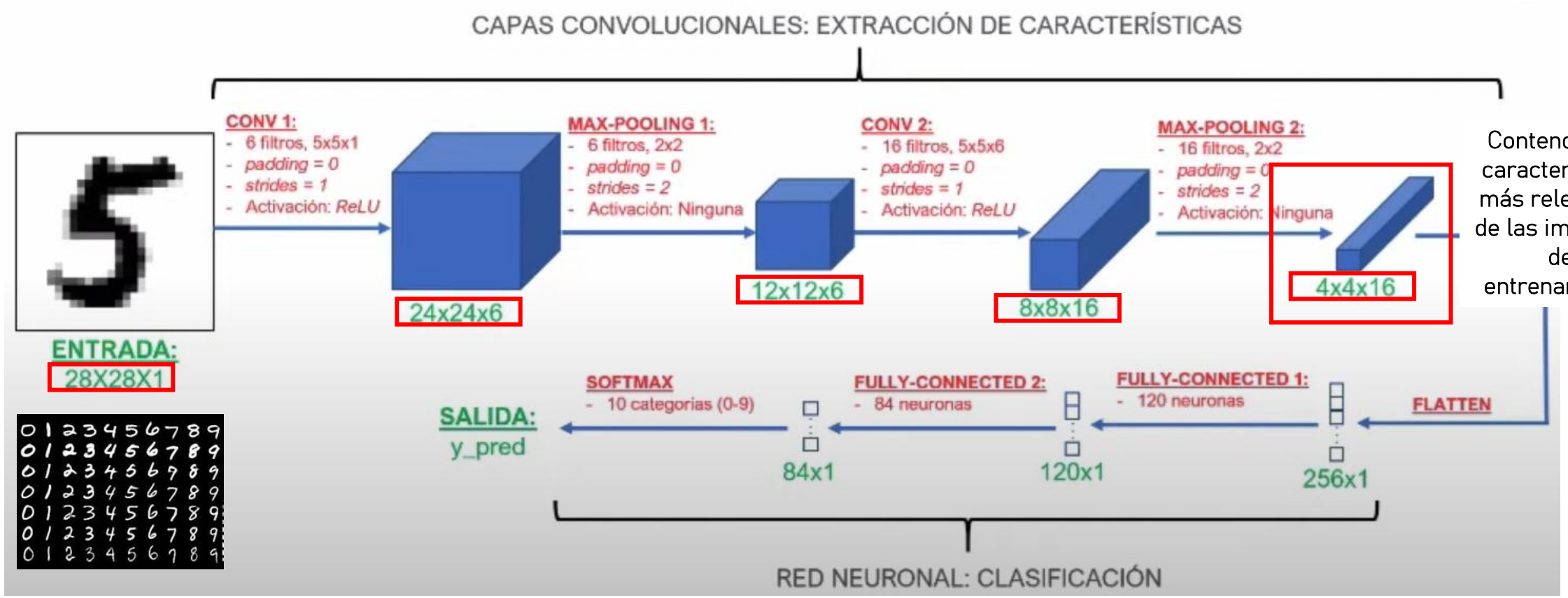
Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324



- El **objetivo del entrenamiento** de esta red convolucional es que los filtros de CONV1 y CONV2 sean capaces de aprender a extraer las características relevantes de las 60,000 imágenes de entrada.

LeNet-5

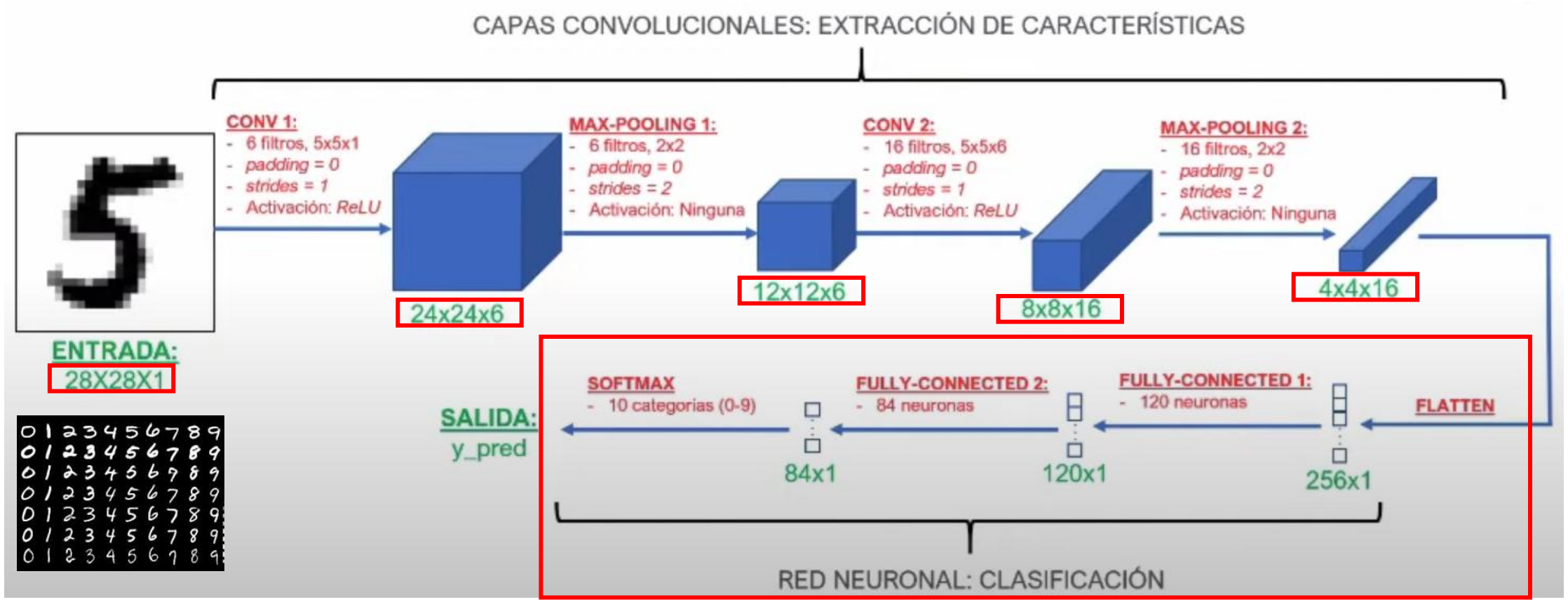
Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324



Contendrá las características más relevantes de las imágenes de entrenamiento

LeNet-5

Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324



LeNet-5

LeNet5_MNIST.ipynb

Usen GPU

¿Qué accuracy se obtiene con el conjunto test?

LeNet-5

LeNet5_MNIST.ipynb

¿Qué accuracy se obtiene con el conjunto test?

```
Epoch 20/20
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9986 - loss: 0.0052
```

Versión MLP

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 200)	157,000
dense_1 (Dense)	(None, 150)	30,150
dense_2 (Dense)	(None, 10)	1,510

Total params: 188,660 (736.95 KB)
Trainable params: 188,660 (736.95 KB)
Non-trainable params: 0 (0.00 B)

Versión CNN

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 24, 24, 6)	156
max_pooling2d (MaxPooling2D)	(None, 12, 12, 6)	0
conv2d_1 (Conv2D)	(None, 8, 8, 16)	2,416
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 16)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 120)	30,840
dense_1 (Dense)	(None, 84)	10,164
dense_2 (Dense)	(None, 10)	850

(size(kernel)+1sesgo)(#kernels)

Total params: 44,426 (173.54 KB)
Trainable params: 44,426 (173.54 KB)
Non-trainable params: 0 (0.00 B)

LeNet-5

LeNet5_MNIST.ipynb

¿Qué pasará cuando probamos con las imágenes trasladadas?

LeNet-5

LeNet5_MNIST.ipynb


```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.1392 - loss: 8.9618
```

¿Qué pasará cuando probamos con las imágenes trasladadas?

LeNet5_MNIST_mejor_configuracion.ipynb

LeNet-5

1. Aumentarle borde a las imágenes

```
input_layer = layers.Input(shape=(28, 28,1))
x = layers.ZeroPadding2D(padding=2)(input_layer)  # dar “aire” al dígito: pad a 32×32
```

2. Aumentar número de capas convolucionales

```
x = layers.Conv2D(filters=16, kernel_size=(5,5), padding="same", strides=(1,1), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(filters=32, kernel_size=(5,5), padding="same", strides=(1,1), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
x = layers.Conv2D(64, kernel_size=(5, 5), padding="same", strides=(1,1), activation="relu")(x)
x = layers.MaxPooling2D(pool_size=(2, 2))(x)
```

3. Aumento de datos

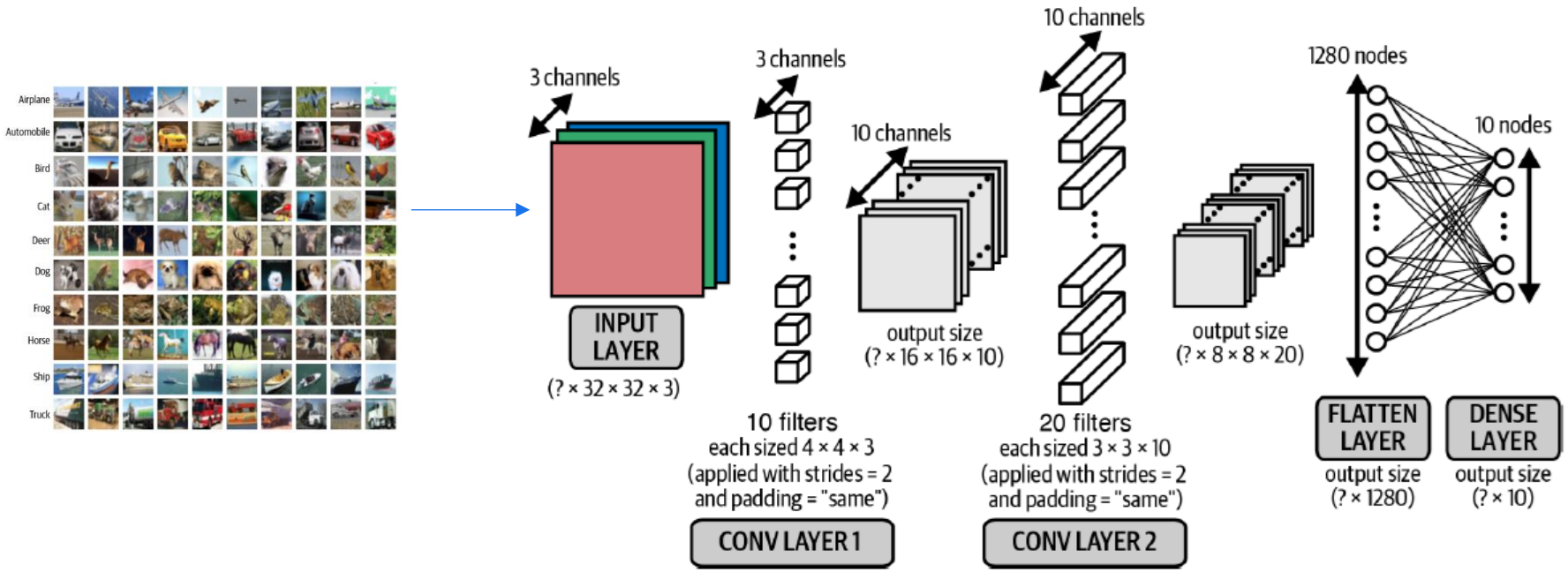
```
data_aug = Sequential([
    layers.RandomTranslation(0.08, 0.08, fill_mode="constant", fill_value=0.0), # traslación ±4 px
    layers.RandomRotation(0.1, fill_mode="constant", fill_value=0.0),           # rotación ~±10°
    layers.RandomZoom(0.1, fill_mode="constant", fill_value=0.0),               # zoom ±10%
    layers.RandomShear(0.1, fill_mode="constant", fill_value=0.0)               # “inclina” la imagen hasta un 10%
], name="aug")

input_layer = layers.Input(shape=(28, 28,1))
x = layers.ZeroPadding2D(padding=2)(input_layer)  # dar “aire” al dígito: pad a 32×32
x = data_aug(x)
```

4. Aumentar número de épocas

```
opt = optimizers.SGD(learning_rate=0.1)
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
```

Ejercicio

De manera similar a MNIST, implementa esta red convolucional para CIFAR10.



Predicción de la clase

* Nota: sin las mejoras de la diapositiva anterior.