



CNN - Deep Learning Python





Redes Neuronales Convolucionales

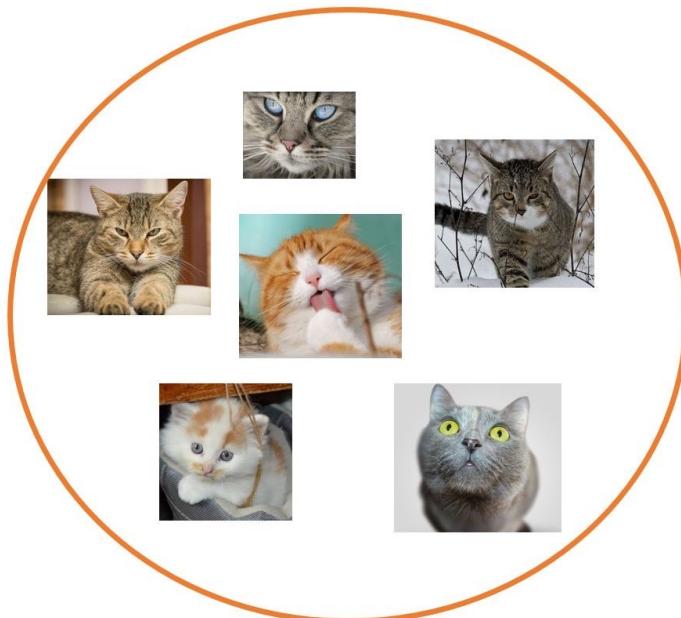
(Convolutional Neural Networks)

(CNN)

Aprendizaje Supervisado

El conjunto de datos tiene etiquetas

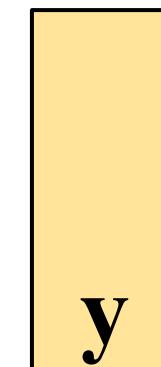
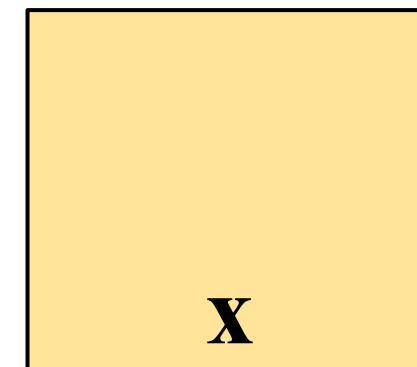
(categoría en la variable a predecir - creadas por humanos)



Aprendizaje Supervisado

MNIST Dataset (dígitos escritos a mano)

Cada imagen tiene 28×28 pixeles

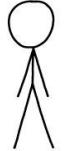


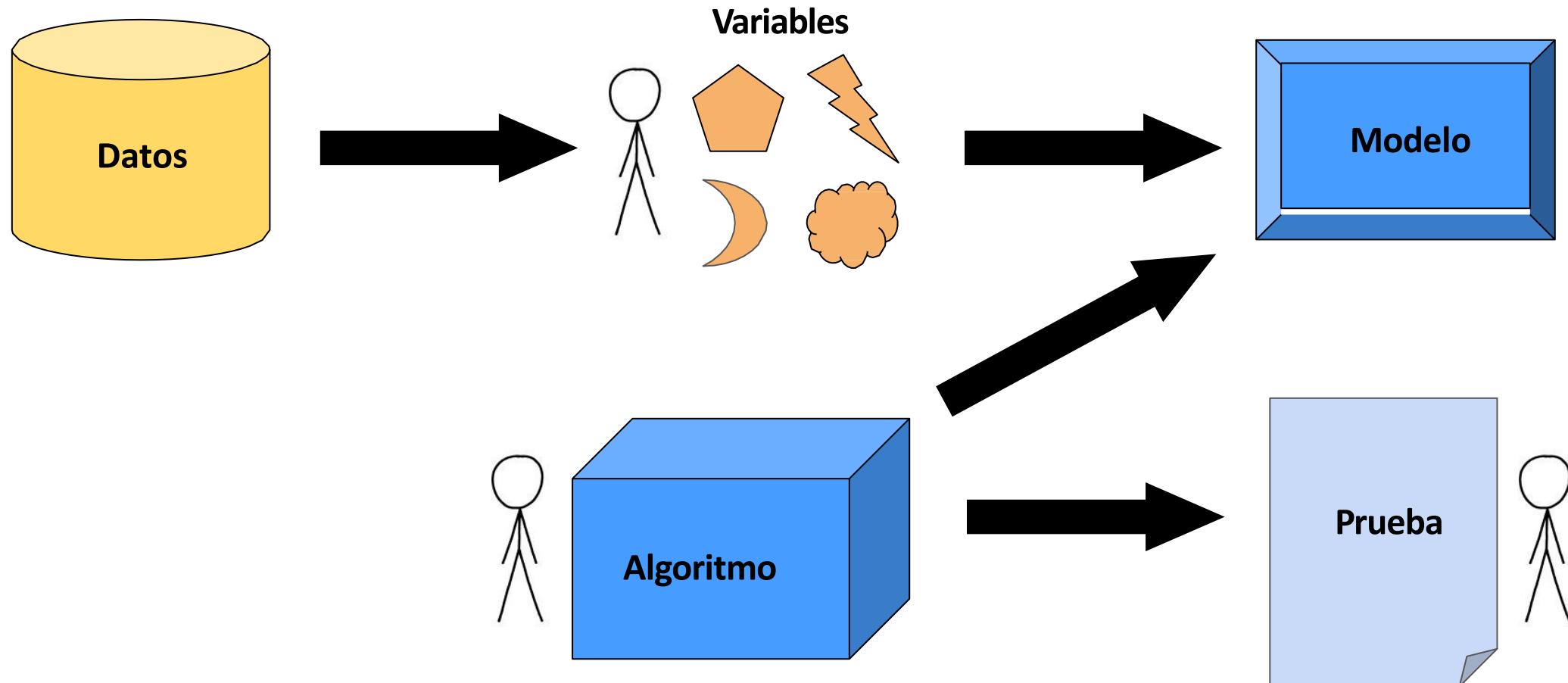
60,000 x 784

60,000 x 10

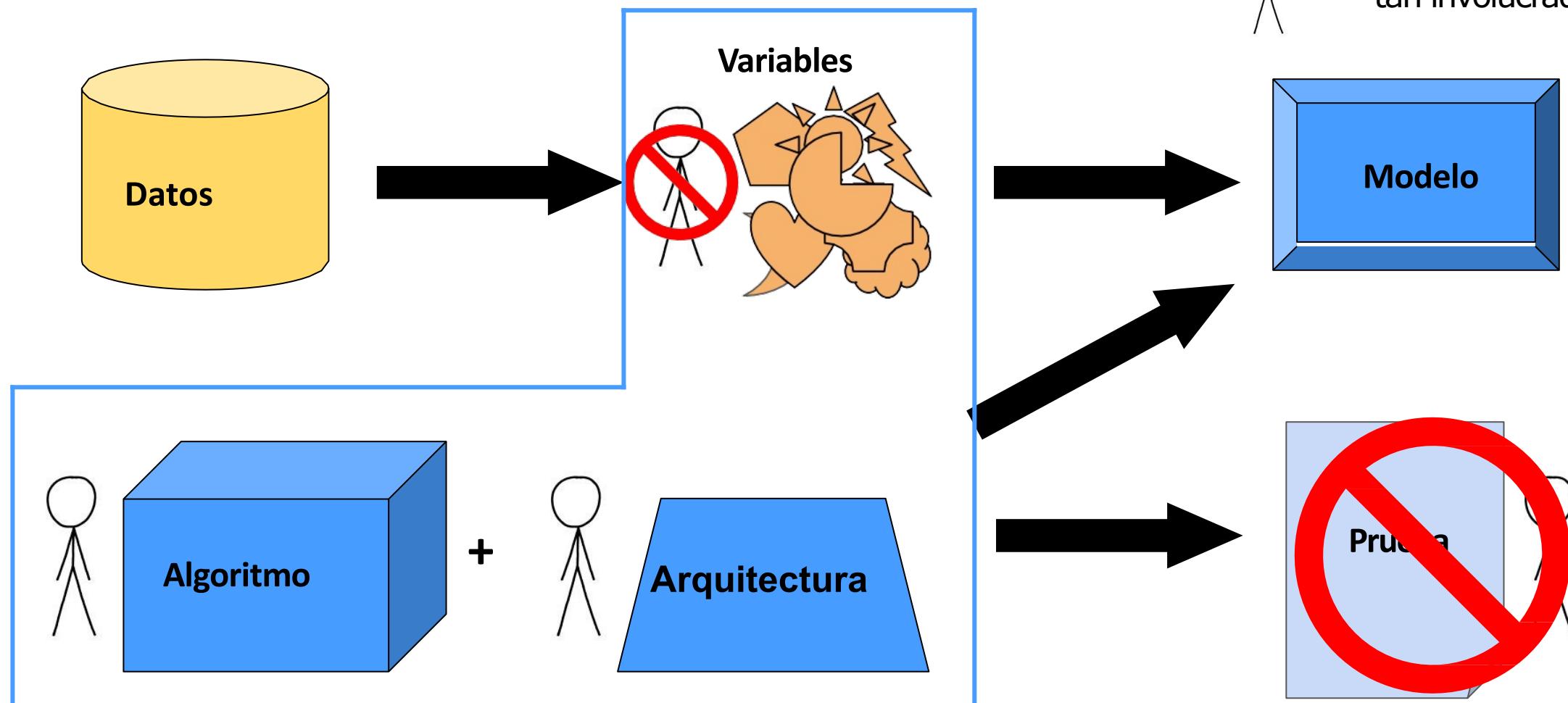
Codificado
con 0 y 1

Machine Learning - Tradicional

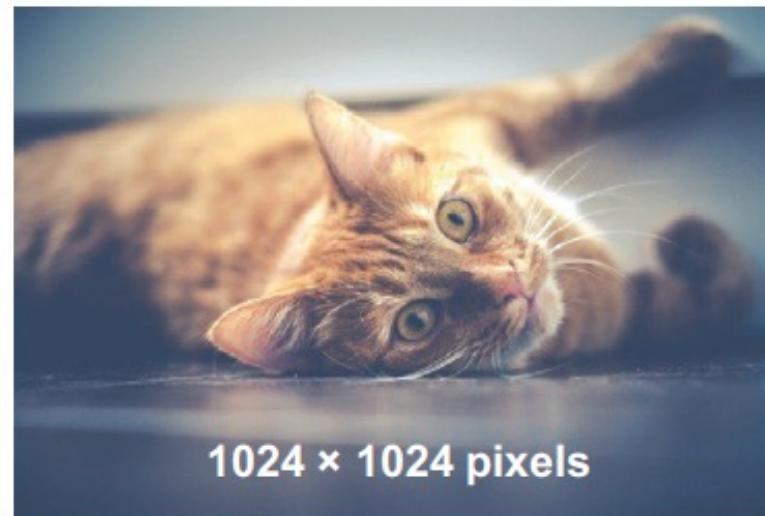
 = Diseño humano involucrado.



Deep Learning



Deep Learning → primer paso es reducir la resolución



1024-node input layer

1 billion weights to learn

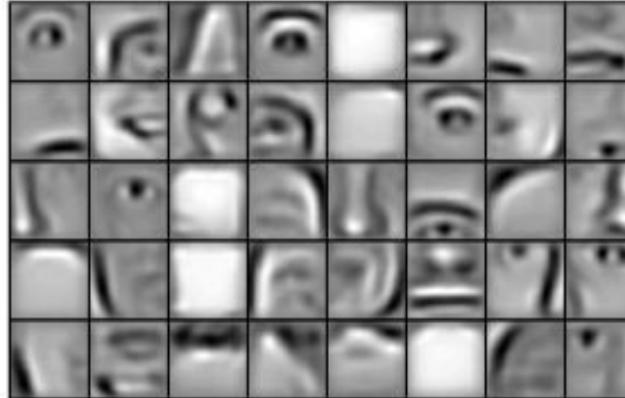


1024-node input layer

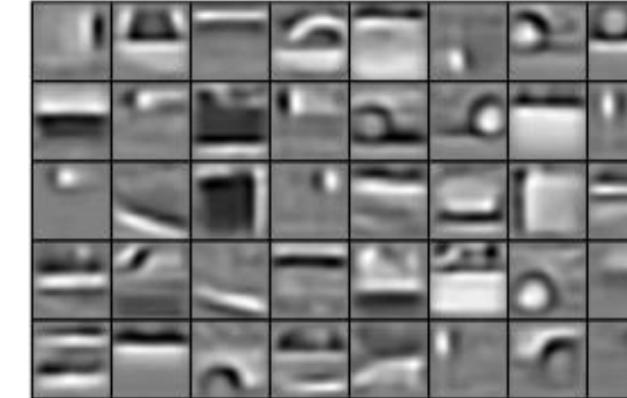
154 million weights to learn

Deep Learning – Generación de Variables

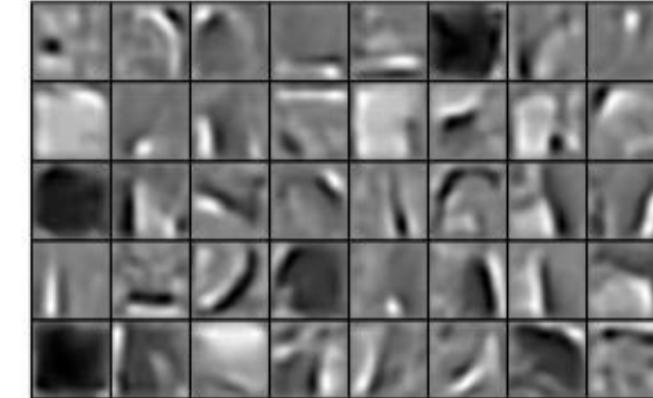
faces



cars



elephants



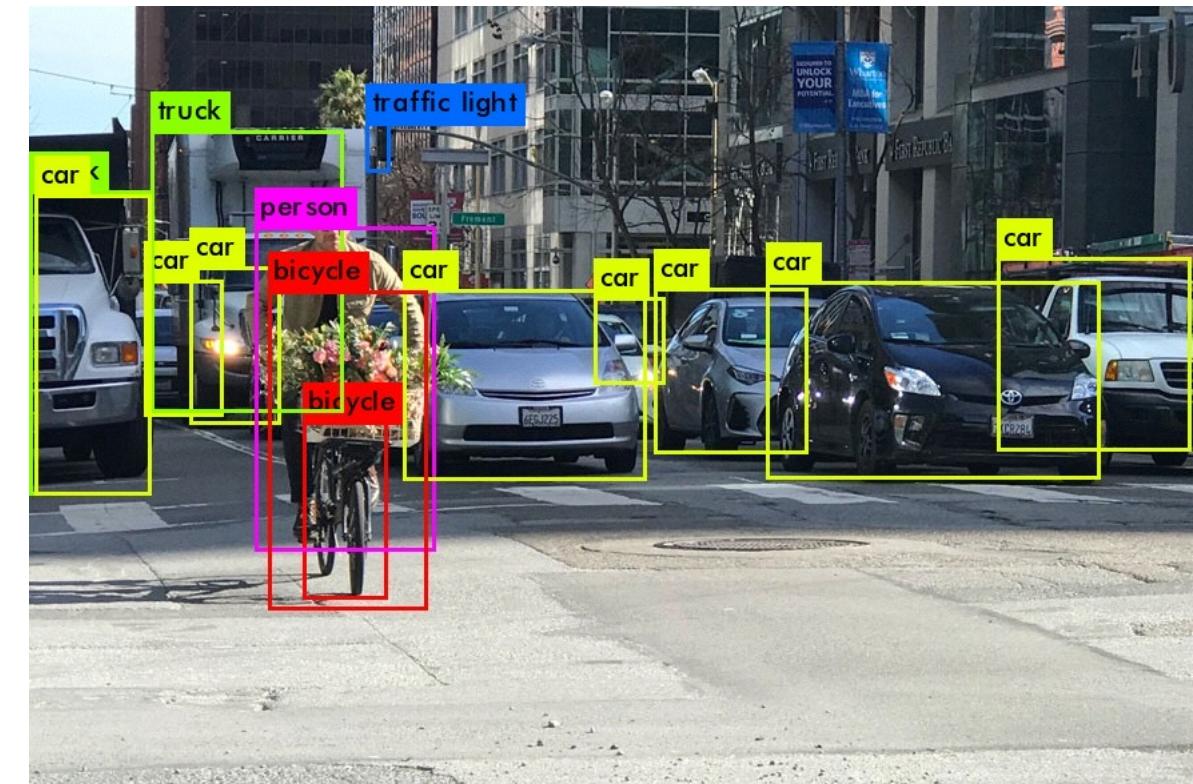
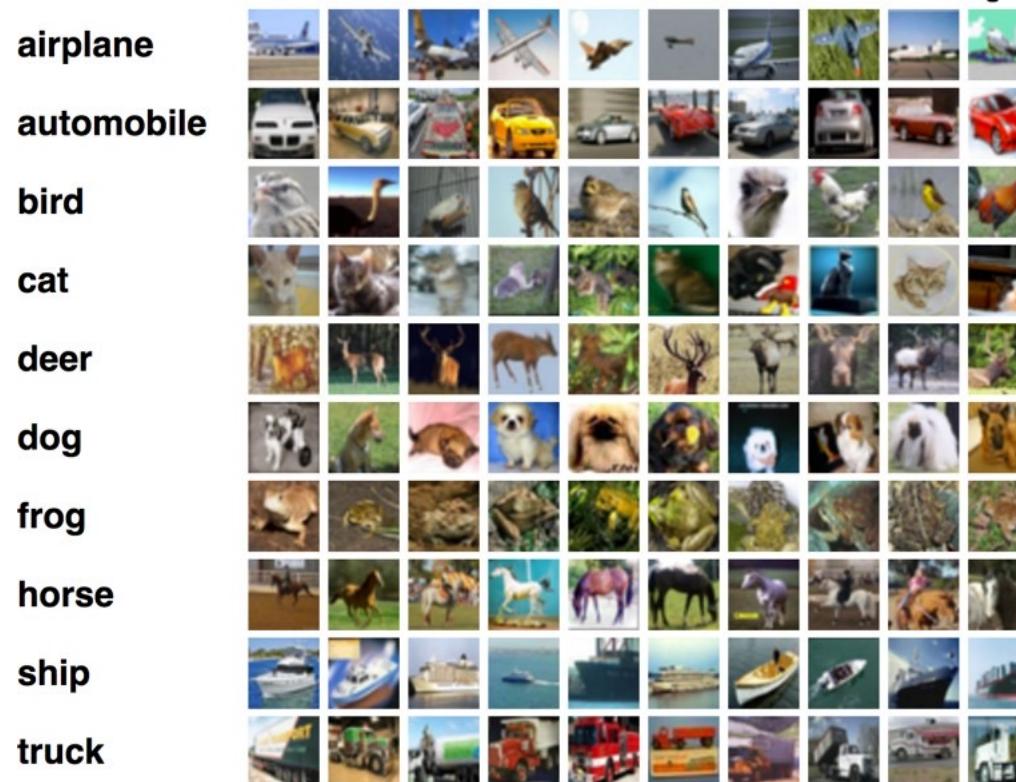


¿Qué son las CNN?

- Las CNN son un tipo de red neuronal artificial; están inspiradas en el concepto de que la corteza visual humana procesa imágenes y permite que nuestro cerebro reconozca objetos en el mundo e interactúe con ellos, lo que nos permite hacer una serie de cosas, como conducir, practicar deportes, leer, ver películas, etcétera.
- Se ha descubierto que en nuestro cerebro tienen lugar cálculos que se asemejan a las **convoluciones** (veremos qué son más adelante).
- Además, nuestros cerebros poseen células simples y complejas. Las células simples recogen características básicas, como bordes y curvas, mientras que las células complejas muestran invariancia espacial, mientras que también responden a las mismas señales que las celdas simples.
- **Se usan fundamentalmente en el reconocimiento de imágenes.**

Red Neuronal Convolucional

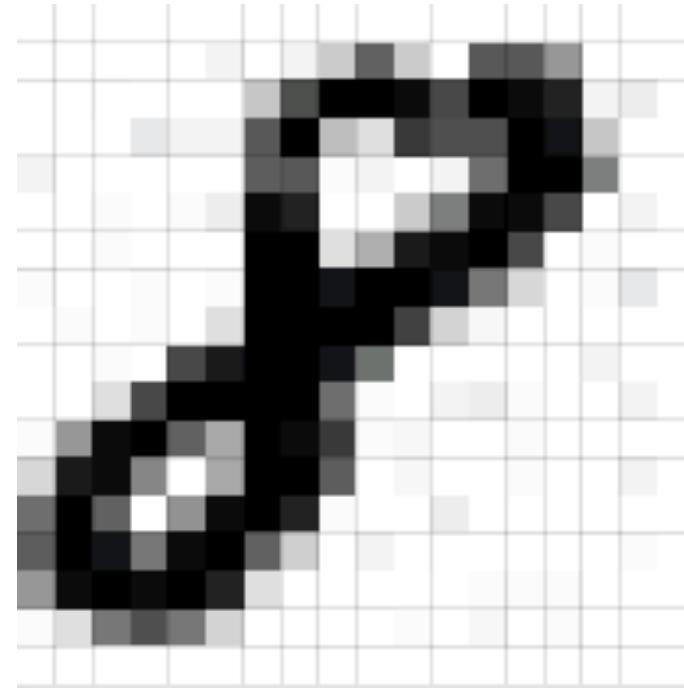
- Las Redes Neuronales Convolucionales han tenido mucho éxito en la clasificación de imágenes y la detección de objetos.



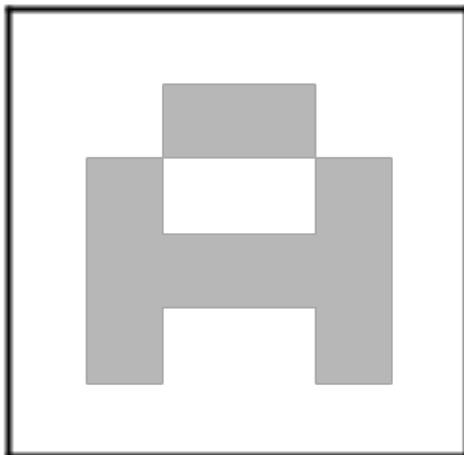
Tipos de Datos en CNN

- Las CNN funcionan excepcionalmente bien en **tareas visuales**, como la clasificación de objetos y el reconocimiento de objetos en imágenes y videos y el reconocimiento de patrones en música, clips de sonido, entre otros.
- Trabajan eficazmente en estas áreas porque son capaces de explotar la estructura de los datos para aprender sobre ellos. Esto significa que no podemos alterar las propiedades de los datos. Por ejemplo, las imágenes tienen una estructura fija y si modificáramos esto, la imagen dejaría de tener sentido.

¿Cómo se representa una imagen?



- Cada pixel será un número entre 0 y 1.
- Cada pixel será una variable en la tabla de aprendizaje y testing.

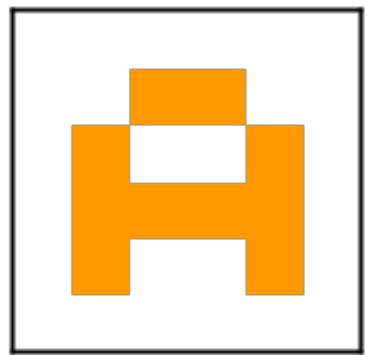


Una imagen...

		0.6	0.6		
0.6				0.6	
0.6	0.6	0.6	0.6		
0.6				0.6	

...es una matriz de píxeles.

El valor de los píxeles va de 0 a 255 pero se normaliza para la red neuronal de 0 a 1



	0.2	0.2	
0.2			0.2
0.2	0.2	0.2	0.2
0.2			0.2

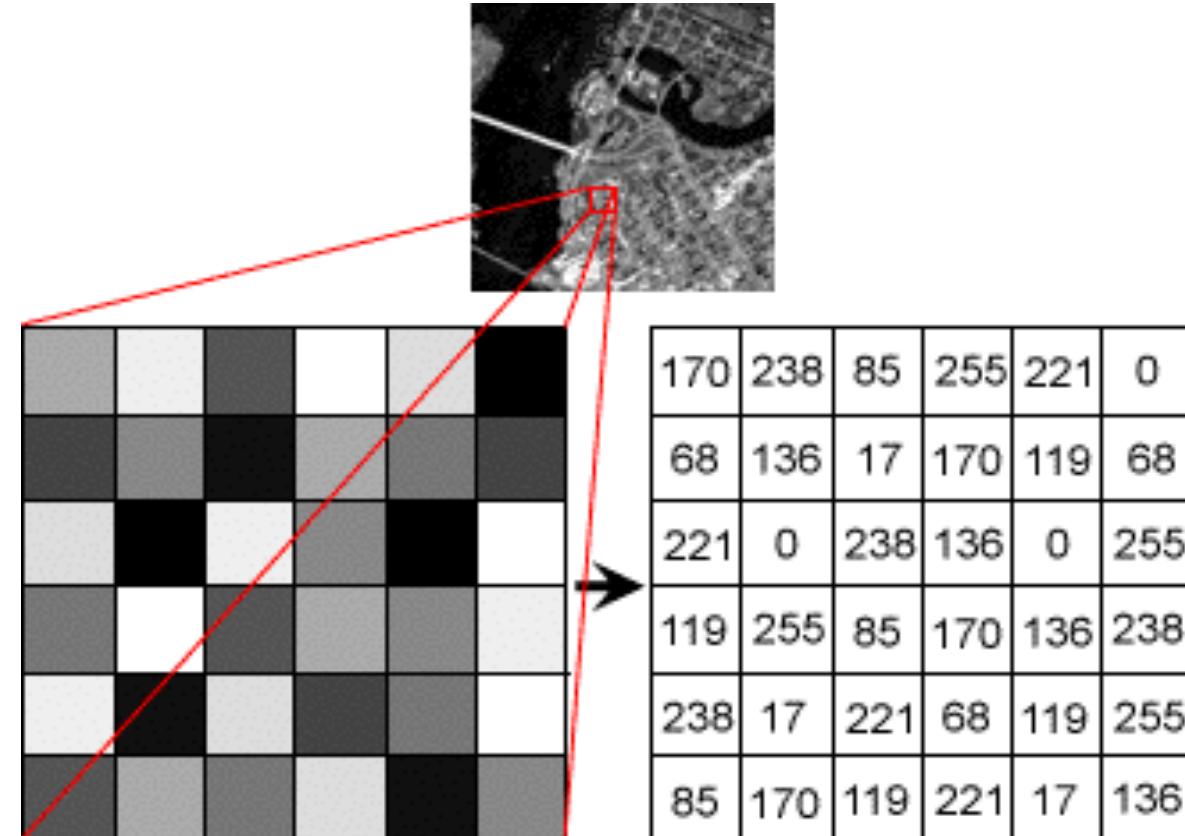
	0.4	0.4	
0.4			0.4
0.4	0.4	0.4	0.4
0.4			0.4

	0.2	0.2	
0.2			0.2
0.2	0.2	0.2	0.2
0.2			0.2

Si la imagen es a color, estará compuesta de tres canales: rojo, verde, azul.

- Antes de alimentar la red neuronal, es muy importante normalizar los valores.
- Los colores de los pixeles tienen valores que van de 0 a 255, se hace una transformación de cada pixel: “valor/255” de manera que el valor quede siempre un valor entre 0 y 1.

Las imágenes como tensores

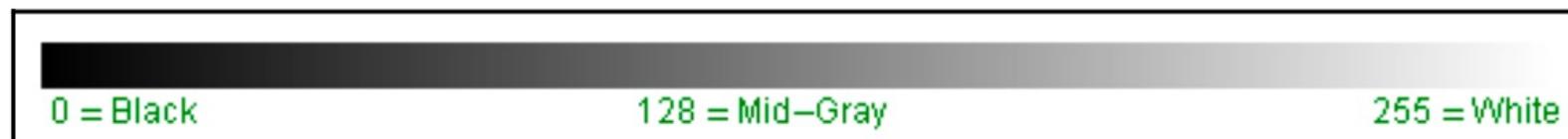




Tipos de Datos en CNN

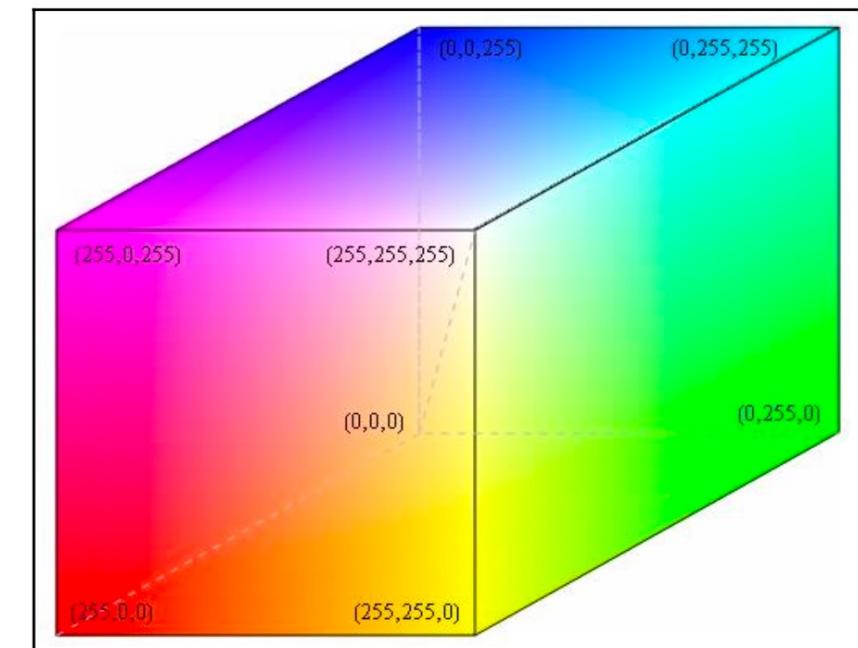
En las computadoras, las imágenes están en escala de grises (blanco y negro) o en colores (RGB) mientras que los videos (RGB-D) están formados por píxeles. Un **píxel** es la unidad más pequeña de una imagen digitalizada que se puede mostrar en una computadora y contiene valores en forma de [0, 255]. El valor de píxel representa su intensidad.

Si el valor de píxel es 0, entonces es negro, si es 128, entonces es gris, y si es 255, entonces es blanco.



Tipos de Datos en CNN

- Las imágenes en color, por otro lado, se componen de tres valores diferentes: *rojo, verde y azul (RGB Red-Green-Blue)* ya que cualquier color se puede mostrar usando una combinación de estos tres colores.
- Se puede ver como un cubo o con diferentes intensidades de color, podemos verlo como si tuviera tres canales separados: rojo, azul y verde. Luego, cada pixel requiere 3 bytes de almacenamiento.





Tipos de Datos en CNN

- Supongamos que tenemos una imagen en escala de grises con un tamaño de $512 \times 512 \times 1$ (alto \times ancho \times canal).
- Podemos almacenarla en un tensor bidimensional (matriz), $\mathbb{R}^{alto \times ancho \times 1}$ donde cada valor de i y j es un píxel con cierta intensidad. Para almacenar esta imagen en nuestro disco, necesitamos $512 \times 512 = 262,144$ bytes.
- Ahora, supongamos que tenemos una imagen en color con un tamaño de $512 \times 512 \times 3$ (alto \times ancho \times canal). Podemos almacenarlo en un tensor tridimensional, donde cada valor de i, j y k es un píxel de color con cierta intensidad. Para almacenar esta imagen en nuestro disco, necesitaríamos $512 \times 512 \times 3 = 786,432$ bytes, lo que nos dice que almacenar una imagen en color requiere mucho más espacio y, por lo tanto, más tiempo para procesar.



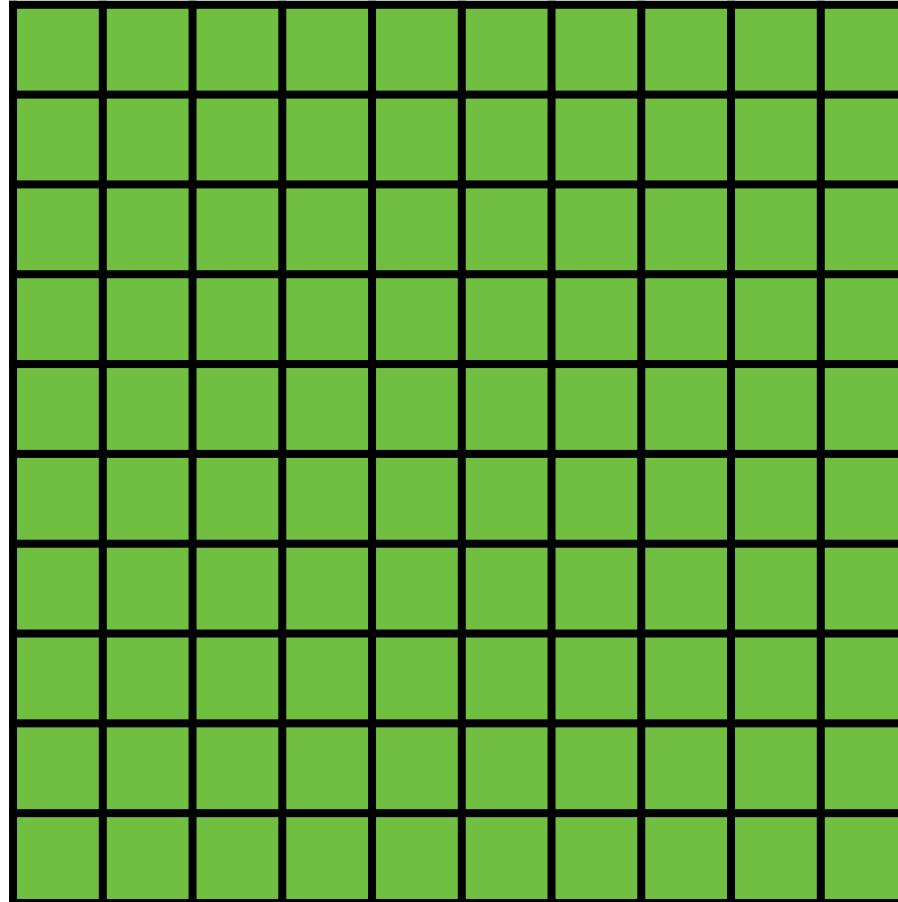
Convolución

Las capas en las CNN están conectadas a través de una operación lineal conocida como **convolución**, que es de donde proviene su nombre y lo que la convierte en una arquitectura muy poderosa para las imágenes.

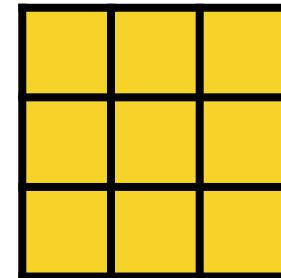
- *Una dimensión:* Se usan para datos de series de tiempo, como los asociados a los precios de las acciones o datos de audio.
- *Dos dimensiones:* Más utilizado para imágenes en escala de grises.
- *Tres dimensiones:* Es frecuentemente utilizado en tareas que requieren buscar relaciones en 3D. Es particularmente utilizado en detección de acciones o movimientos en videos.

¿Qué es una convolución?

Imagen

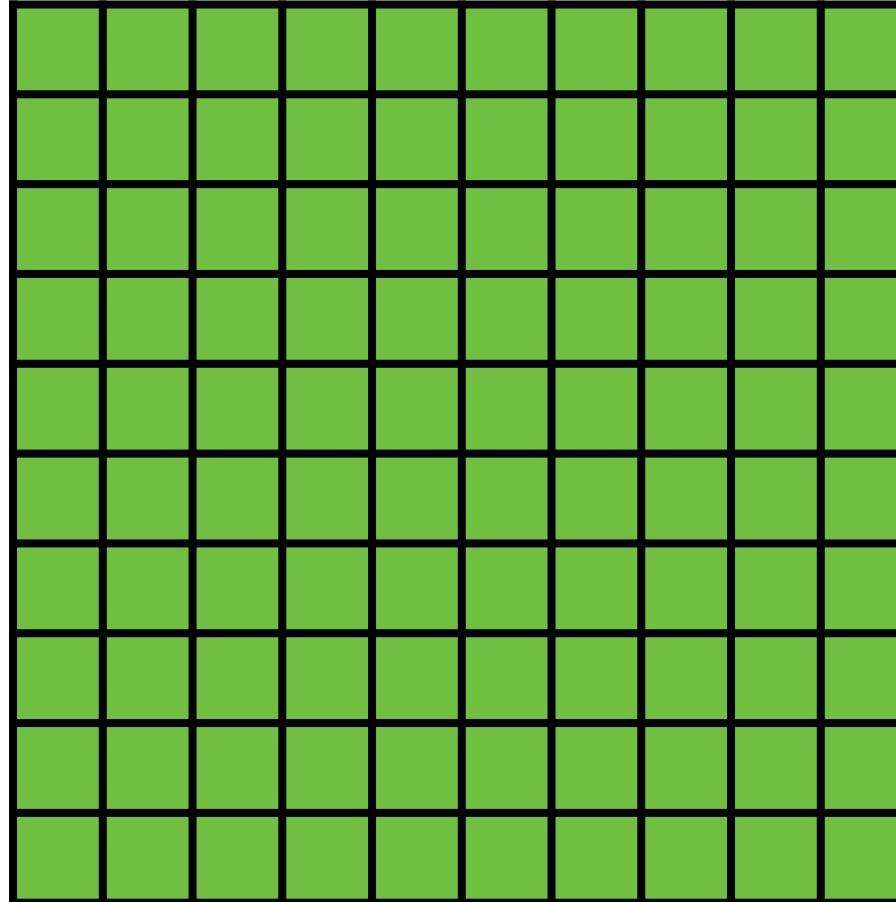


Filtro



¿Qué es una convolución?

Imagen



Filtro

1	0	1
0	1	0
1	0	1

Pesos aprendibles

¿Qué es una convolución?

- Un ejemplo en una imagen con 1 canal.
- Observe que las dimensiones de la salida son menores que las dimensiones de la imagen por 2 píxeles.

1 <small>×1</small>	1 <small>×0</small>	1 <small>×1</small>	0	0
0 <small>×0</small>	1 <small>×1</small>	1 <small>×0</small>	1	0
0 <small>×1</small>	0 <small>×0</small>	1 <small>×1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Imagen Original

4		

Variable de
Convolución

¿Qué es una convolución?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



1	0	1
0	1	0
1	0	1

Imagen Original 5x5

Variable de convolución
Matriz filtro 3x3

¿Qué es una convolución?

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

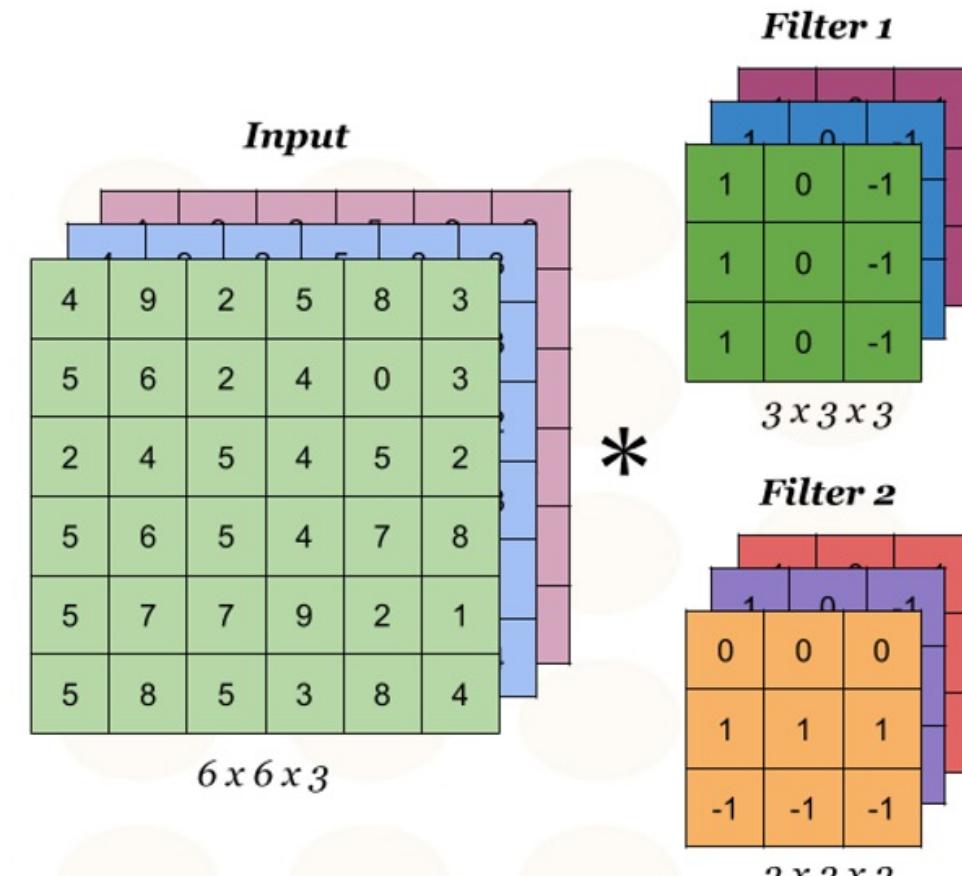
Imagen Original 5x5

4		

Variable de convolución
Matriz filtro 3x3

Convoluciones: 3 canales

La mayoría de las imágenes son RGB:



¿Qué es una convolución?

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
0	154	152	152	157	167	167	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...

Input Channel #3 (Blue)

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

Kernel Channel #1



308

+

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

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

164 + 1 = -25

Bias = 1

-25				...
				...
				...
				...
...

Output



Capas convolucionales

1. Las capas convolucionales son un tipo especial de capa para entradas de imágenes.
2. Las convoluciones reemplazan la multiplicación de matrices con la operación de convolución.

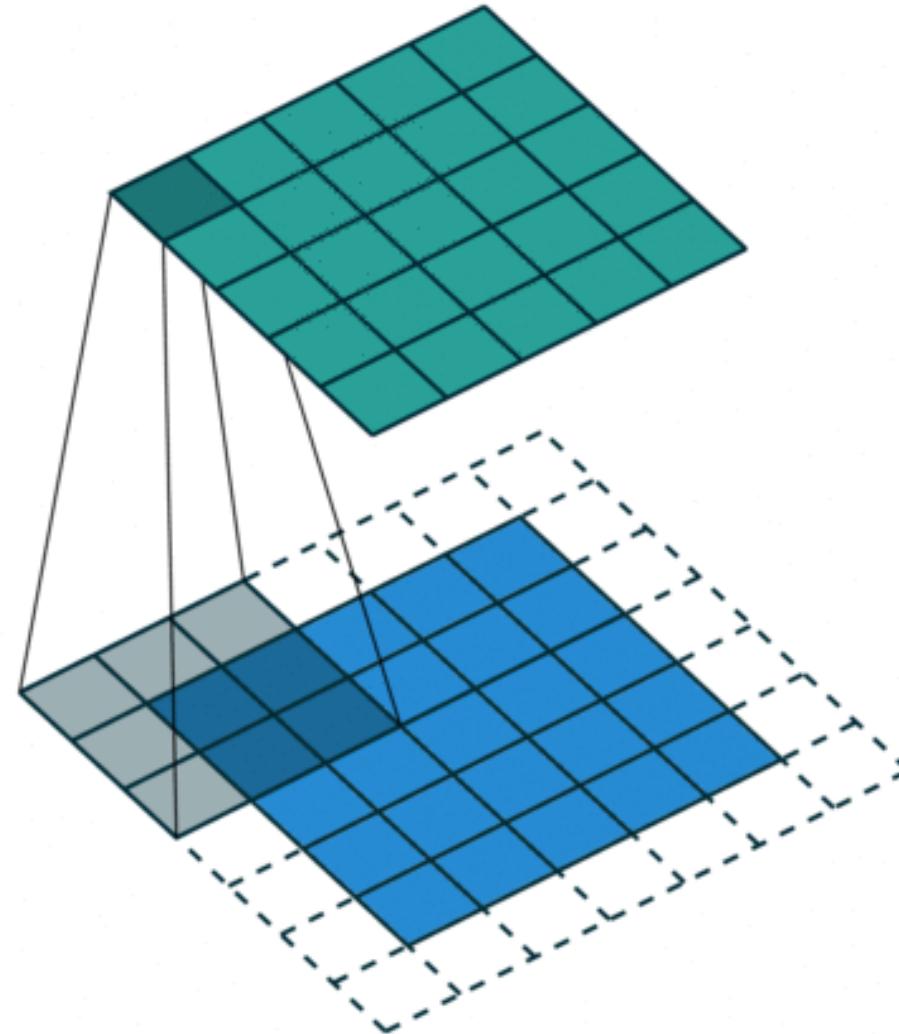
$$g(\mathbf{Wx} + \mathbf{b}) \longrightarrow g(\mathbf{f} * \mathbf{x} + \mathbf{b})$$



¿Qué es una convolución?

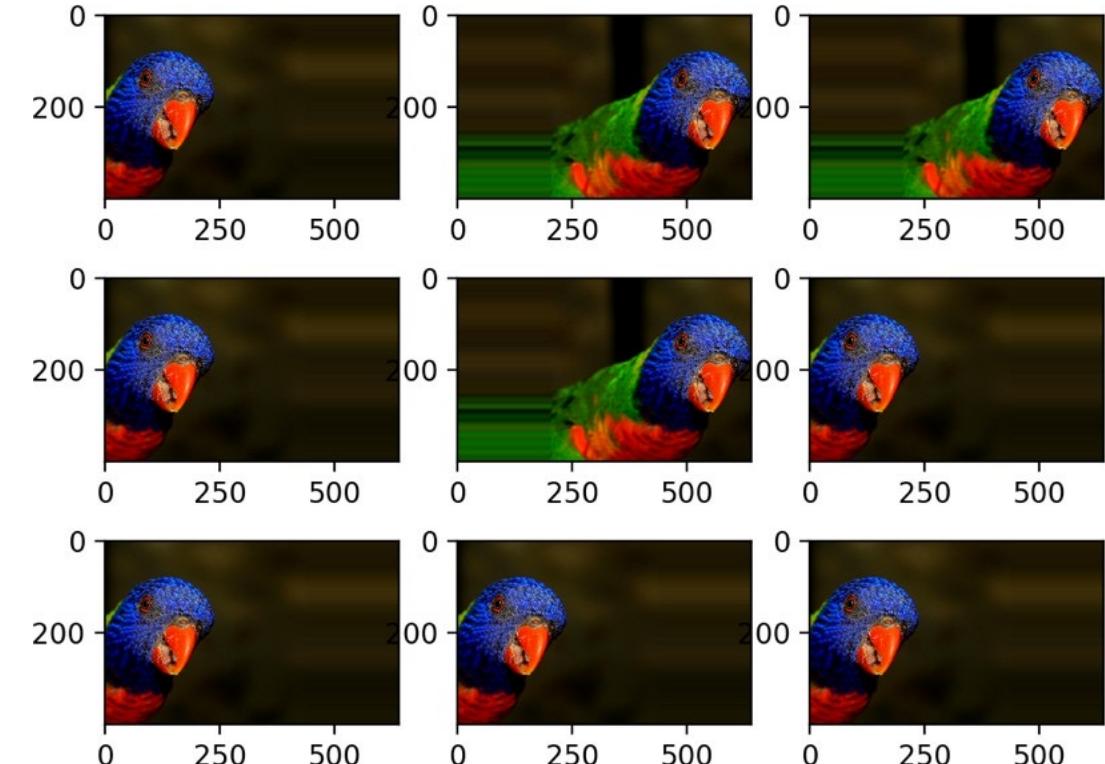
- El kernel se desliza sobre la entrada y produce un **mapa de características** con una altura de 2 y un ancho de 2. Este mapa de características nos dice el **grado** en que las funciones **f** y **g** se **superponen** cuando uno pasa sobre el otro.
- Podemos pensar en esto como escanear la entrada en busca de un patrón determinado; en otras palabras, el mapa de características busca el mismo patrón en diferentes lugares de la entrada.

¿Qué es una convolución?



¿Por qué usar convoluciones?

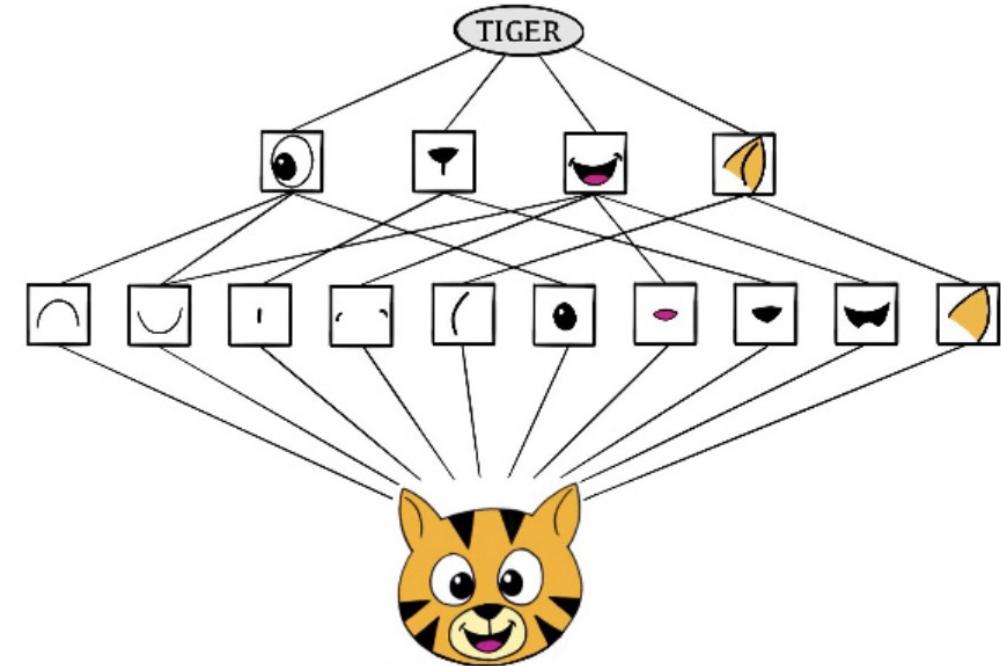
- Las entidades de las imágenes deben ser invariantes a las translaciones.
- Compartir el peso reduce el número de parámetros para aprender.



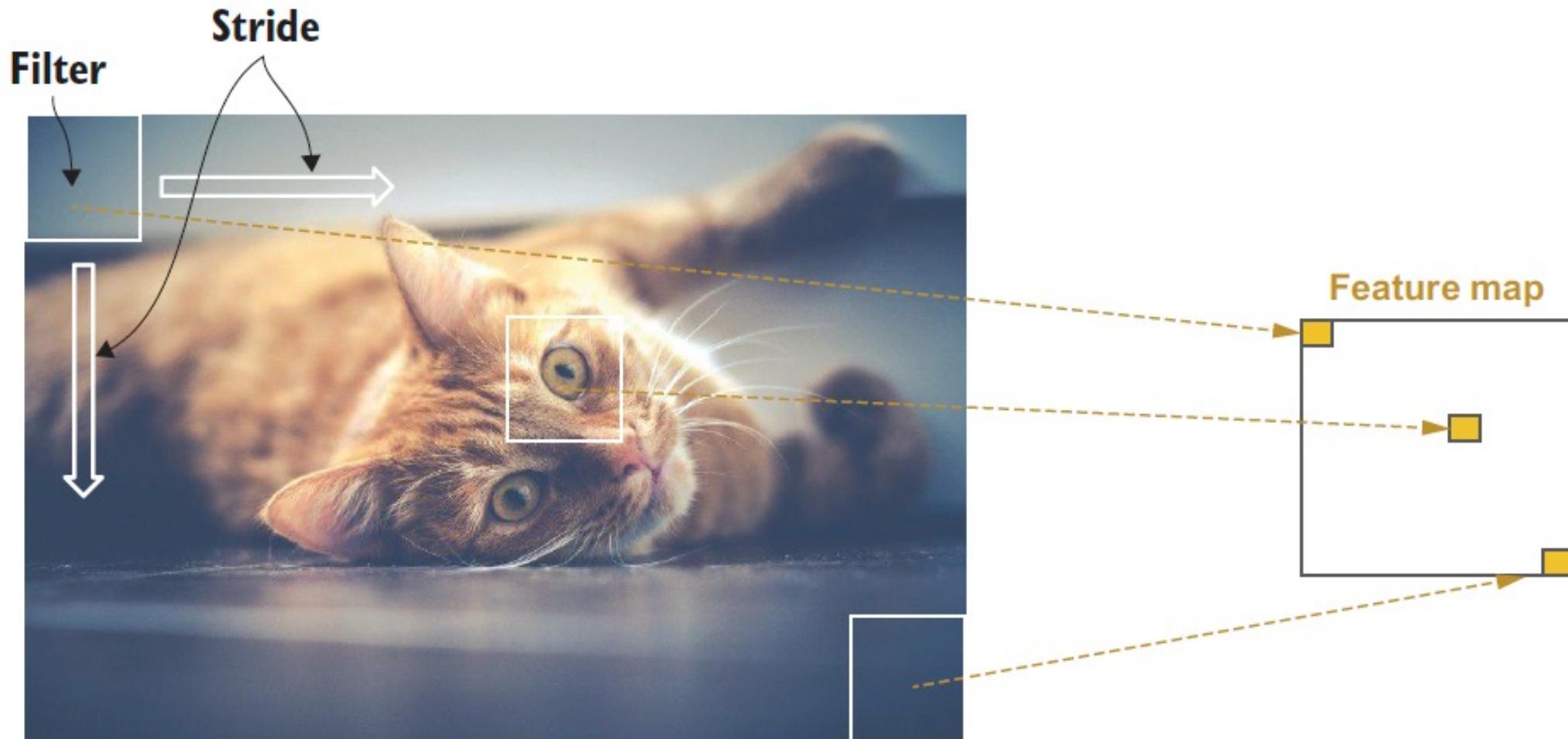
No importa dónde esté el loro en la imagen,
la imagen todavía contiene un loro.

¿Por qué usar convoluciones?

- En el esquema se muestra cómo clasifica una red neuronal convolucional en una imagen de un tigre.
- La red capta la imagen e identifica las características locales (**deep**).
- Luego combina las características locales para crear características compuestas, que en este ejemplo incluyen ojos y oídos. Estas funciones compuestas se utilizan para generar la etiqueta "tigre".



¿Por qué usar convoluciones?



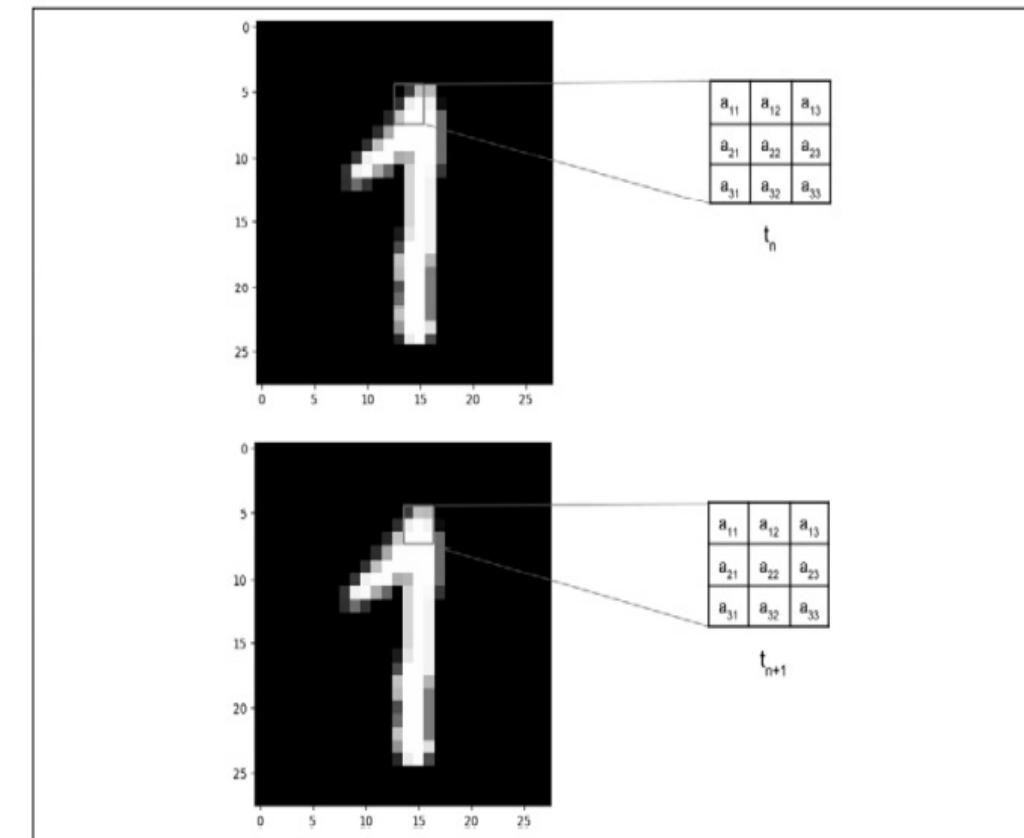


¿Qué es una convolución?

- Podemos repetir este proceso tantas veces como se quiera, usando diferentes núcleos y produciendo múltiples mapas de características.
- Luego se apilan estas salidas juntas y formamos una *matriz tridimensional de mapas de características*, que llamamos **capa**.
- Por ejemplo, digamos que tenemos una imagen con un tamaño de 52×52 y un núcleo con un tamaño de 12×12 y un paso de 2. Aplicamos esto a nuestra entrada 15 veces y apilamos las salidas. Obtenemos un tensor tridimensional con un tamaño $\mathbb{R}^{21 \times 21 \times 15}$.

¿Qué es una convolución?

El kernel se puede visualizar como un *parche* o *ventana* rectangular que se desliza a través de toda la imagen de izquierda a derecha y de arriba a abajo. Transforma la imagen de entrada en un mapa de características (variables), que es una representación de lo que el kernel ha aprendido de la imagen de entrada. Luego, el mapa de características se transforma en otro mapa de características en la capa siguiente y así sucesivamente. El número de mapas de características generados por Conv2D está controlado por el argumento de filtros (*filters*).



¿Qué es una convolución?

- Si en el modelo **MLP** (Multi-Layer Perceptron), el número de **unidades** caracteriza las capas densas (Dense Layers), el **kernel** (núcleo) caracteriza las operaciones de la **CNN**.
- En las CNN, generalmente usamos convoluciones discretas, que se escriben de la siguiente manera:

$$(f * g)(x) = \sum_t f(t)g(t - x)$$

- Supongamos que tenemos una matriz bidimensional con una altura de 3 y un ancho de 3, y un kernel bidimensional con una altura de 2 y un ancho de 2. Entonces, la convolución y su salida se verán de la siguiente manera:

$$\begin{bmatrix} I_{1,1} & I_{1,2} & I_{1,3} \\ I_{2,1} & I_{2,2} & I_{2,3} \\ I_{3,1} & I_{3,2} & I_{3,3} \end{bmatrix} * \begin{bmatrix} K_{1,1} & K_{1,2} \\ K_{2,1} & K_{2,2} \end{bmatrix} = \begin{bmatrix} O_{1,1} & O_{1,2} \\ O_{2,1} & O_{2,2} \end{bmatrix}$$

$$\begin{aligned} O_{1,1} &= I_{1,1}K_{1,1} + I_{1,2}K_{1,2} + I_{2,1}K_{2,1} + I_{2,2}K_{2,2} \\ O_{1,2} &= I_{1,2}K_{1,1} + I_{1,3}K_{1,2} + I_{2,2}K_{2,1} + I_{2,3}K_{2,2} \\ O_{2,1} &= I_{2,1}K_{1,1} + I_{2,2}K_{1,2} + I_{3,1}K_{2,1} + I_{3,2}K_{2,2} \\ O_{2,2} &= I_{2,2}K_{1,1} + I_{2,3}K_{1,2} + I_{3,2}K_{2,1} + I_{3,3}K_{2,2} \end{aligned}$$

Ejemplo

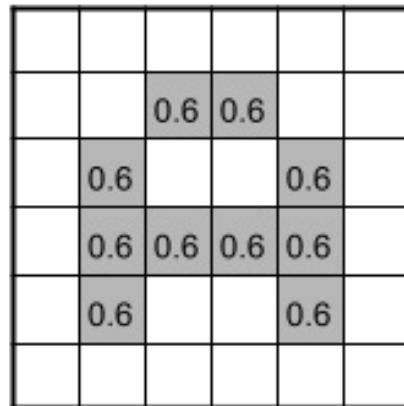
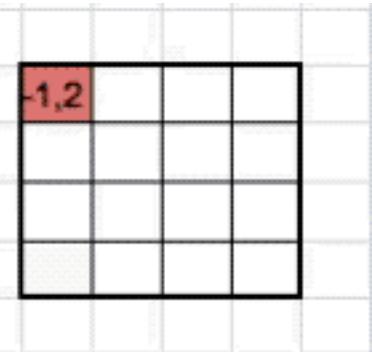
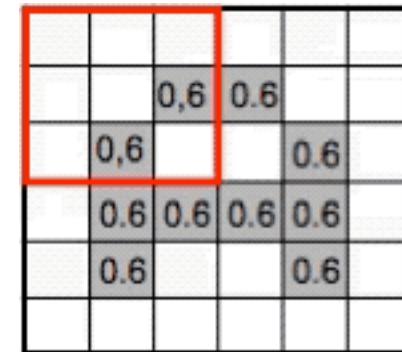


Imagen de
entrada

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

kernel



- El kernel tomará inicialmente valores aleatorios y se irán ajustando mediante backpropagation.
- Una mejora es hacer que siga una distribución normal siguiendo simetrías, pero sus valores son aleatorios.

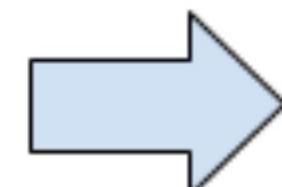
Ejemplo

IMAGEN

		0,6	0,6		
0,6				0,6	
0,6	0,6	0,6	0,6		
0,6			0,6		

KERNEL

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



CONVOLUCION
DEL KERNEL

-1,2	-0,6	0,6	1,2
-1,2	0,6	-0,6	1,2
-1,2	1,2	-1,2	1,2
-0,6	1,2	-1,2	0,6

Ejemplo:

<https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

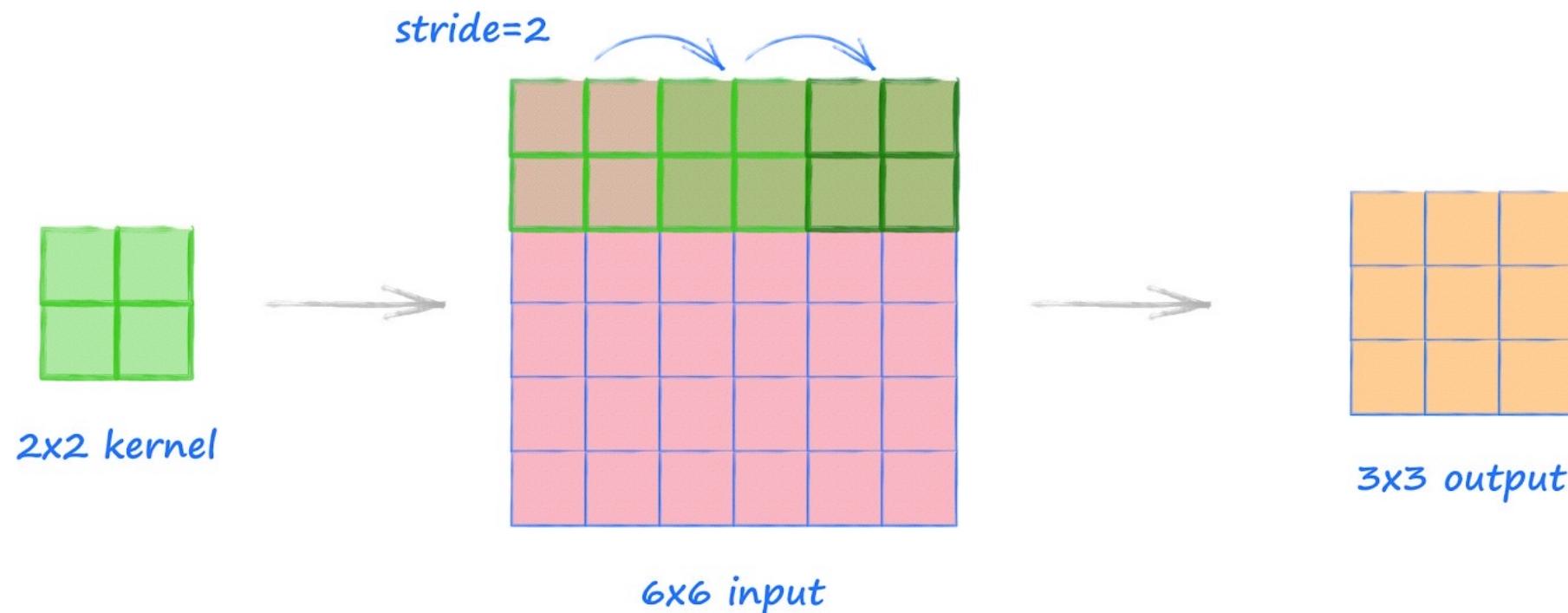
Figure 7 : Some common filters

Observación: Se aplica un conjunto de kernels (Filtros)

- En realidad, no se aplica un sólo kernel, si no que se toman muchos kernels (su conjunto se llaman filtros).
- En una primer convolución se podrían tener, por ejemplo, **32 filtros**, con lo cual realmente se obtiene 32 matrices (variables) de salida (este conjunto se conoce como “feature mapping”).
- Por ejemplo, si cada matriz (imagen) es de tamaño 30x30x1 daría un total del $30 \times 30 \times 1 \times 32 = 28,800$ neuronas para la PRIMER CAPA OCULTA de neuronas. Son muchas para una imagen cuadrada de apenas 30 píxeles.
- Imagínese cuántas más serían si tomáramos una imagen de entrada de 224x224x3 (que aún es considerado un tamaño pequeño).
- Ver un ejemplo en: <https://setosa.io/ev/image-kernels/>
- <https://ringa-tech.com/vision01/camara.html>

Zancada (Stride)

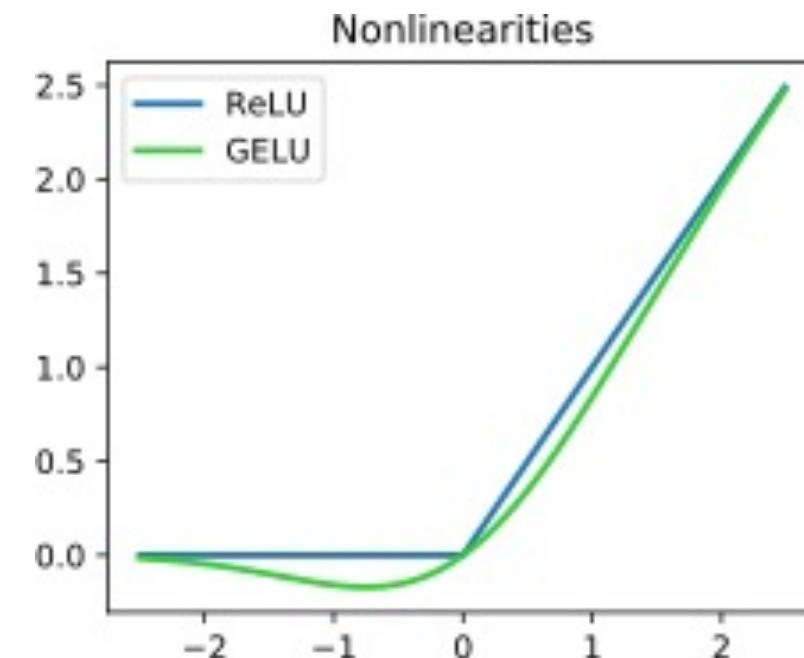
- Stride (zancada) se refiere a cuántos píxeles desplazamos el filtro (kernel) en cada paso de la convolución:



<http://makeyourownneuralnetwork.blogspot.com/2020/02/calculating-output-size-of-convolutions.html>

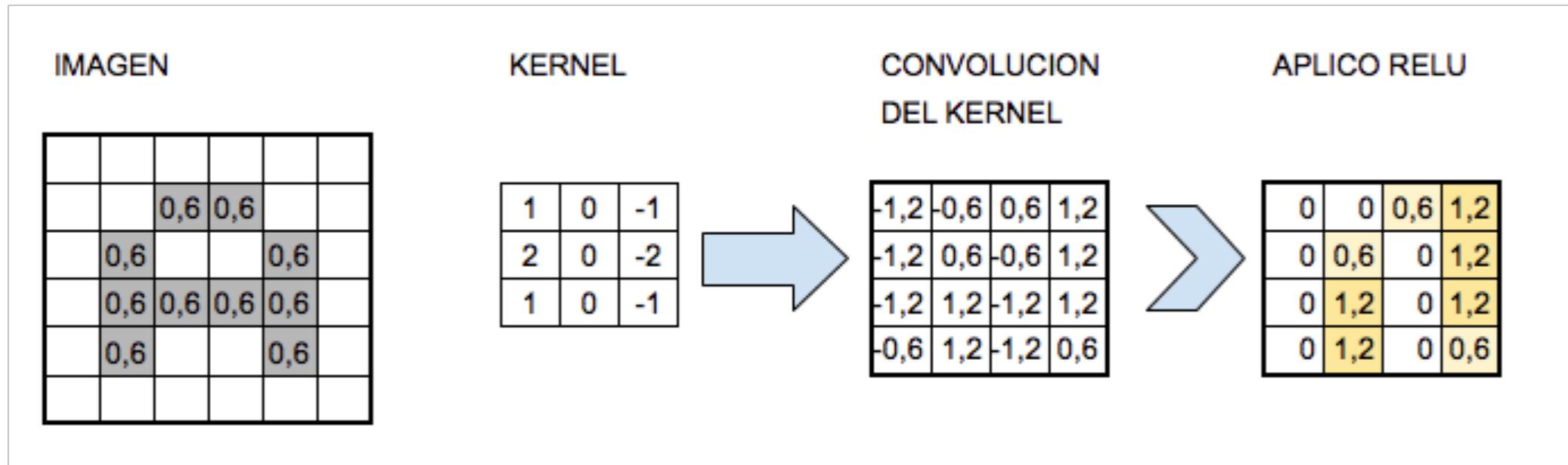
La función de Activación - ReLU

- La función de activación más utilizada para este tipo de redes neuronales es la llamada ReLU por Rectifier Linear Unit y consiste en $f(x) = \max(0, x)$.



$$f(x) = x^+ = \max(0, x)$$

La función de Activación - ReLU



FINALMENTE
OBTENGO UN MAPA
DE DETECCIÓN DE
CARACTERÍSTICAS



Ejemplo: Subsampling (Submuestreo)

- Es importante reducir la cantidad de neuronas antes de hacer una nueva convolución. ¿Por qué? Como se vio en el ejemplo anterior, con una imagen blanco y negro de 30x30x1 tenemos una primer capa de entrada de 900 neuronas y luego de la primer convolución obtenemos una capa oculta de 28,800 neuronas.
- Si se hace una nueva convolución a partir de esta capa, el número de neuronas de la próxima capa se iría por las nubes (y ello implica mayor procesamiento)!
- Para reducir el tamaño de la próxima capa de neuronas se hace un proceso de subsampling en el que se reduce el tamaño de las imágenes filtradas en el cual deberán prevalecer las características más importantes que detectó cada filtro.
- Hay diversos tipos de subsampling, los “más usado” son: Max-Pooling y Mean-Pooling.

Pooling layers (Agrupación de capas)

Es común insertar capas de agrupación entre capas (Conv layers). La agrupación reduce el tamaño espacial de la representación.

La capa de agrupación (pooling) más común:

2	3	1	9
4	7	3	5
8	2	2	2
1	3	4	5

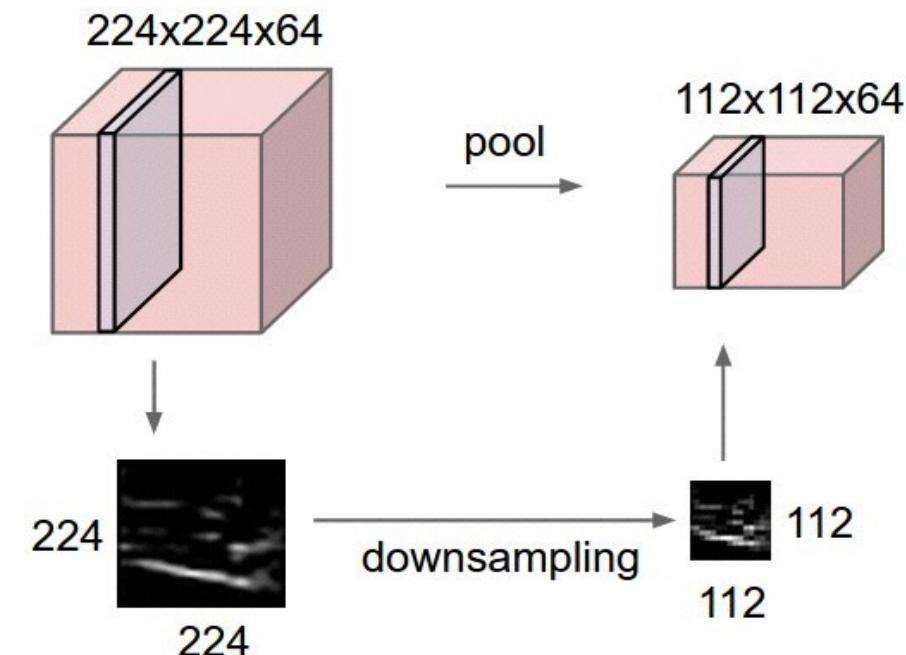


7	9
8	5

Max-Pool with a
2 by 2 filter and
stride 2.

Pooling layers (Agrupación de capas)

Las capas de agrupación no contienen parámetros entrenables.



Pooling (máximo o promedio)

Otra operación de uso frecuente en las CNN se conoce como **pooling (agrupación, submuestreo)**. Esto funciona de manera similar a la operación de convolución, excepto que reduce el tamaño del mapa de características deslizando una ventana a través del mapa de características y **promedia** todos los valores dentro de cada ventana en cada paso o genera el **valor máximo**. El pooling se diferencia de la convolución en que no tiene ningún parámetro, por lo tanto, no se puede aprender ni ajustar.

Es un tensor bidimensional en forma de $n \times n$, la operación de pooling es un tensor bidimensional en forma de $r \times r$ y s es la zancada/paso.



Pooling

A continuación, se muestra un ejemplo de agrupación máxima con un paso de 1 y un ejemplo de agrupación promedio con un paso de 2:

$$\begin{bmatrix} 2 & 1 & 1 & 3 \\ 4 & 5 & 2 & 1 \\ 1 & 4 & 5 & 3 \\ 5 & 1 & 7 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & 5 & 3 \\ 5 & 5 & 5 \\ 5 & 7 & 7 \end{bmatrix}$$

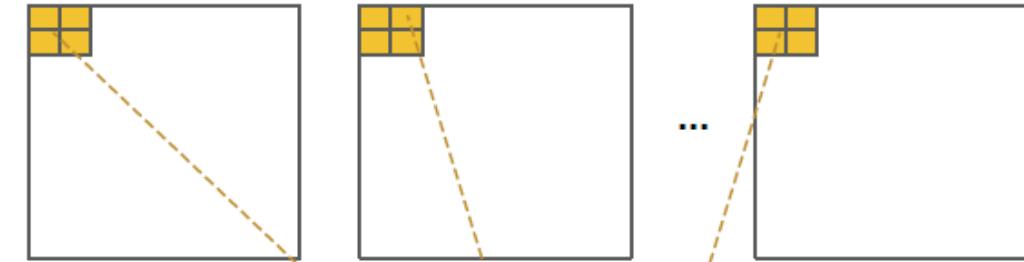
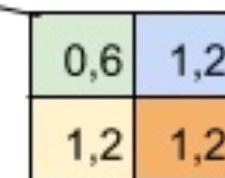
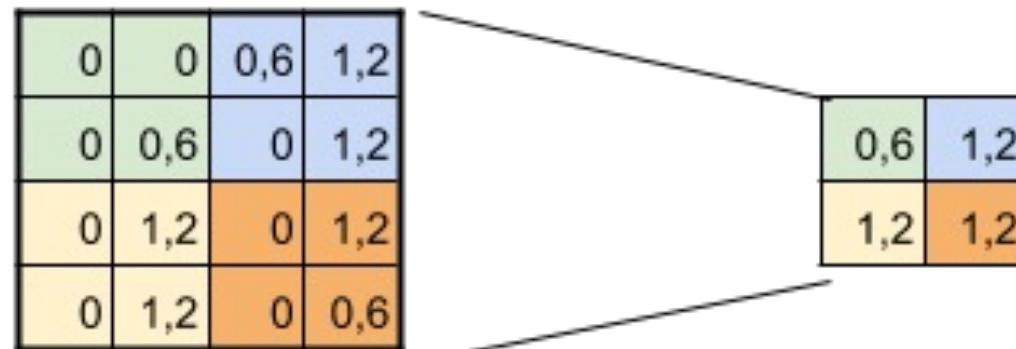
Agrupación máxima con
zancada de 1

$$\begin{bmatrix} 2 & 1 & 1 & 3 \\ 4 & 5 & 2 & 1 \\ 1 & 4 & 5 & 3 \\ 5 & 1 & 7 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 1.75 \\ 2.75 & 5.25 \end{bmatrix}$$

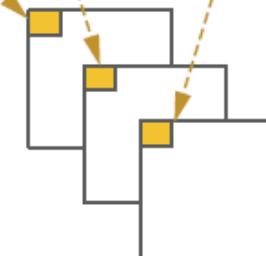
Agrupación promedio con
zancada de 2

Como regla general, se ha descubierto que la operación de **agrupación máxima** funciona **mejor**.

Max-Pooling



Feature maps



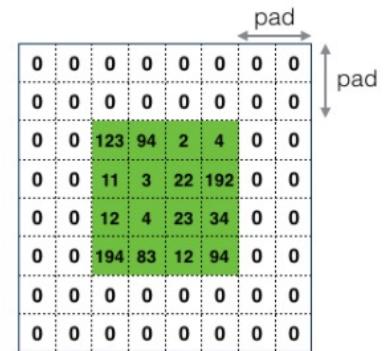
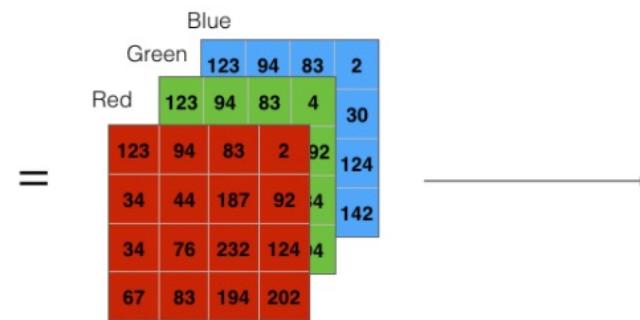
Pooled maps

SUBSAMPLING:
Aplico Max-Pooling de 2x2
y reduzco mi salida a la mitad

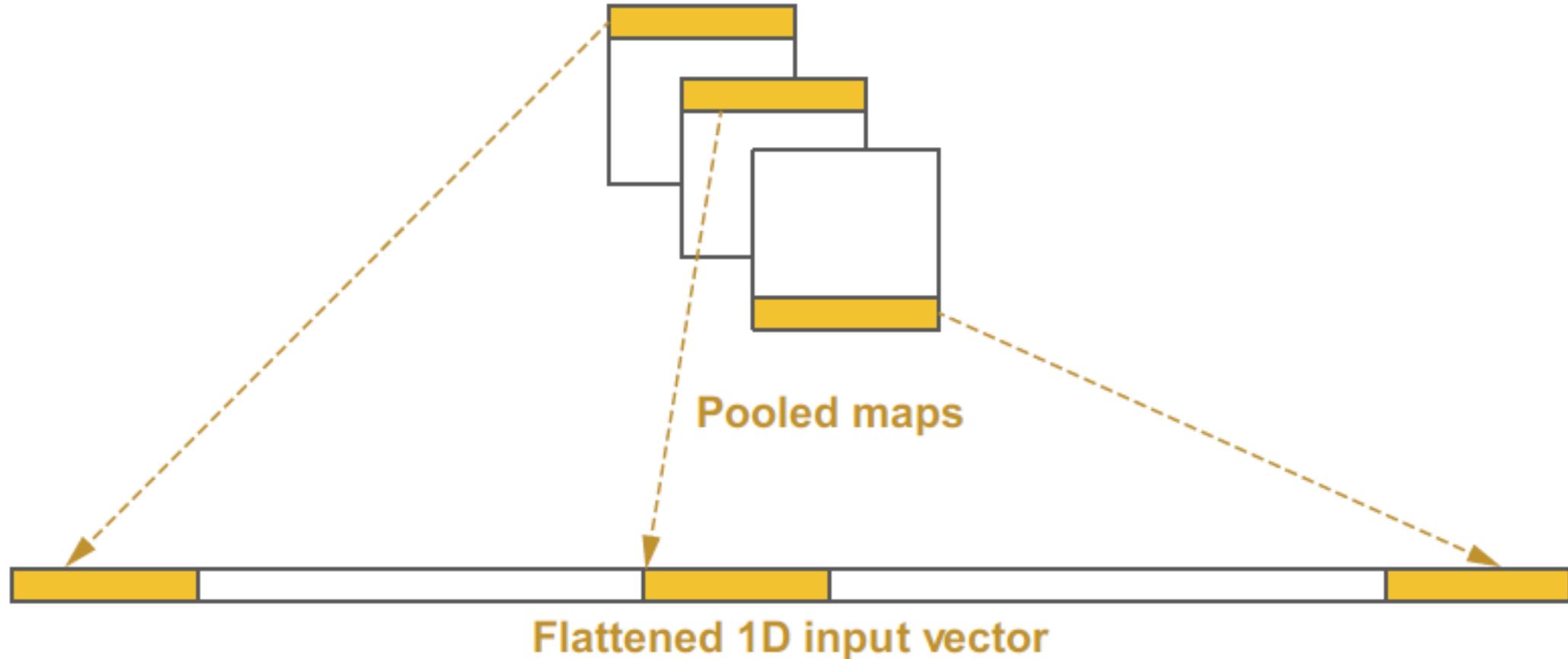
Relleno con Cero (Zero-Padding)

El relleno cero (***pad***) agrega ceros alrededor del borde de una imagen.

En el ejemplo $pad=2$.

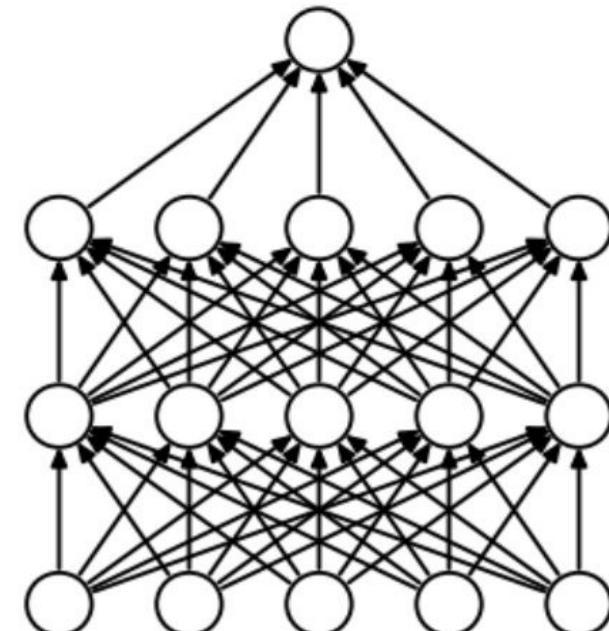


Aplanado (Flattening)

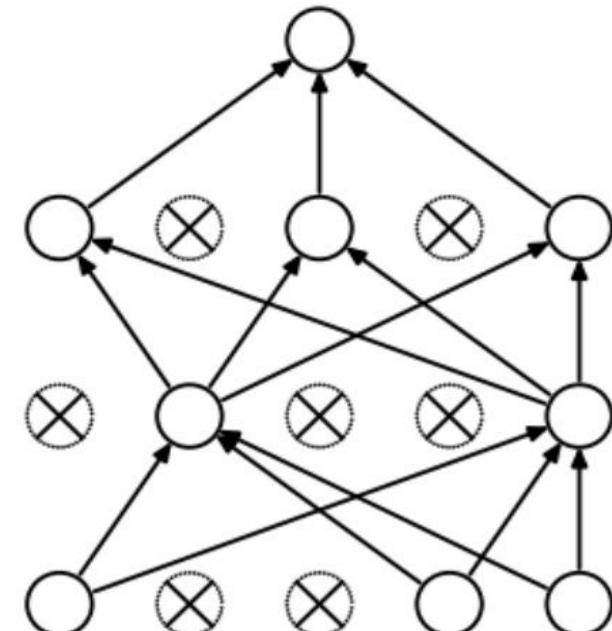


Dropout

Otra forma de regularizar las redes neuronales es eliminar aleatoriamente alguna fracción de neuronas en una capa durante el entrenamiento.



(a) Standard Neural Net



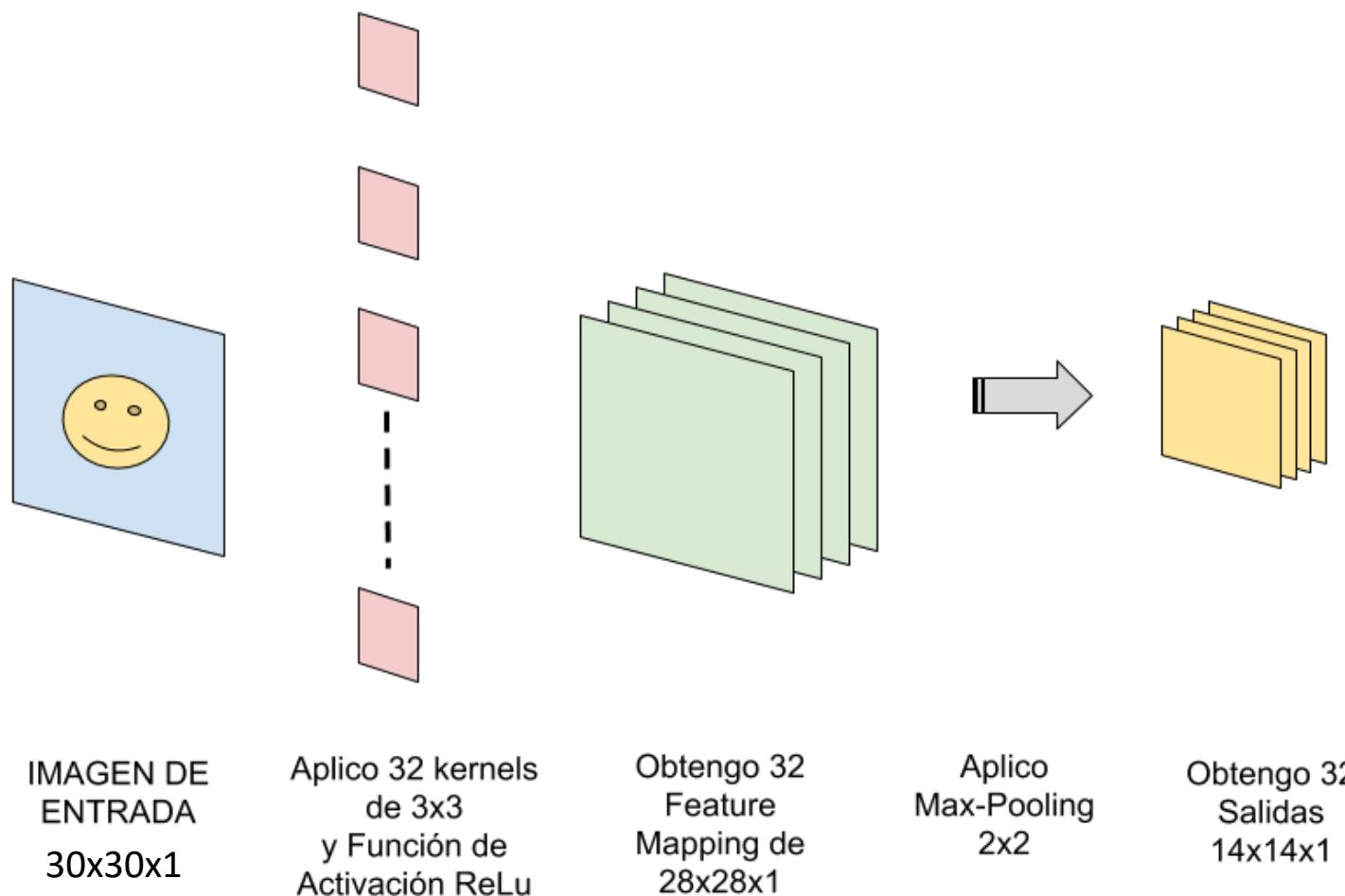
(b) After applying dropout.



Continuación del Ejemplo

- En el ejemplo anterior (de imágenes 30×30 , filmina 43), aplicamos 32 filtros de 3×3 con una zancada (stride) de 1.
- Esto quiere decir que se recorre cada imagen 32 veces obteniendo 32 características (variables-matrices) de $28 \times 28 \times 1$.
- Luego se hace Max-Pooling de tamaño 2×2 con un paso de 2.
- En este caso, usando 2×2 , la imagen resultante es reducida "a la mitad" y quedará de 14×14 pixeles. Luego de este proceso de subsampling nos quedarán 32 imágenes de 14×14 , pasando de haber tenido $28 \times 28 \times 32 = 22,088$ neuronas a $14 \times 14 \times 32 = 6272$, son bastantes menos y al menos en teoría, siguen almacenando la información más importante para detectar características deseadas.

PRIMERA CONVOLUCIÓN



Tamaño de convolución y pooling

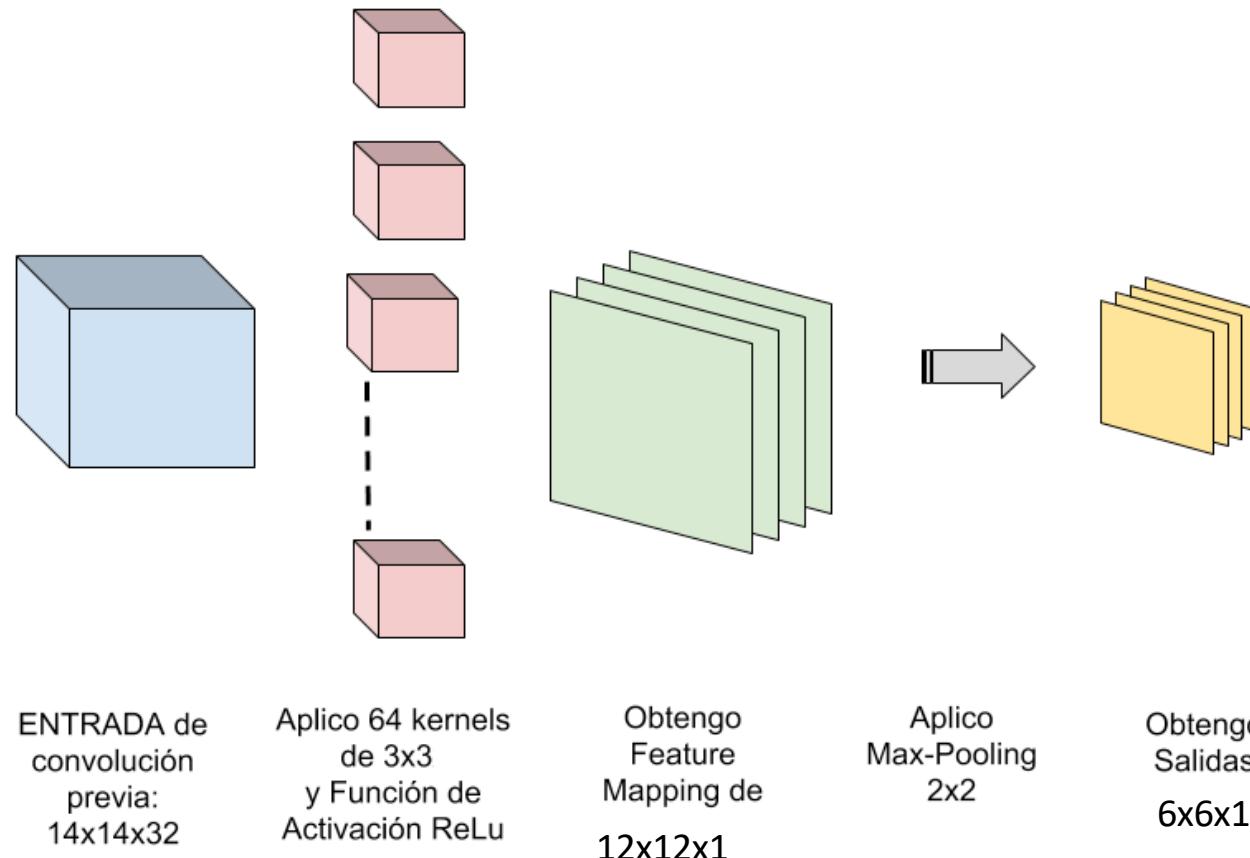
- Como ha visto, cuando aplicamos una convolución a una imagen, la salida será de un tamaño menor que el de la entrada. El tamaño de salida está determinado por el tamaño del kernel, la zancada y si tenemos o no relleno.
- En la práctica, generalmente usamos convoluciones más grandes con una zancada más grande para generar un mapa de características de un tamaño más pequeño para así reducir la restricción computacional.
- Por defecto se usan núcleos 3×3 y 5×5 ó más. Esto se debe a que son *computacionalmente más factibles*. Generalmente, tener un kernel más grande nos permitirá ver un espacio más grande en la imagen y capturar más relaciones. Tener múltiples kernels de 3×3 ha demostrado tener un rendimiento similar mientras que es menos intensivo en computación, lo que es mejor.



Segunda Convolución

- La primer convolución es capaz de detectar características primitivas como líneas o curvas. A medida que se hacen más capas con las convoluciones, los mapas de características serán capaces de reconocer formas más complejas, y el conjunto total de capas de convoluciones podrá “ver” mejor.
- Siguiendo con el ejemplo de imágenes $30 \times 30 \times 1$ (que quedaron $14 \times 14 \times 32$), se hace una **segunda convolución**, se supone que se aplican 64 **kernels** (núcleos).
- Se muestra en el siguiente gráfico.

SEGUNDA CONVOLUCIÓN (y sucesivas)



Tercera Convolución

- Entonces, la tercer convolución comenzará en tamaño 6×6 pixeles y luego de aplicar kernels (núcleos) de tamaño 3×3 quedan de tamaño 4×4 y si luego se aplica un Max-Pooling quedará variables (matrices) de 2×2 .
- En este ejemplo se inició con una imagen de $28 \times 28 \times 1$ e hicimos 3 convoluciones. Si la imagen inicial hubiese sido mayor (de $224 \times 224 \times 1$, por ejemplo) aún se hubiera podido seguir haciendo convoluciones.

1)Entrada: Imagen	2)Aplico Kernel	3)Obtengo Feature Mapping	4)Aplico Max- Pooling	5)Obtengo “Salida” de la Convolución
$6 \times 6 \times 64$	128 filtros de 3×3	$4 \times 4 \times 128$	de 2×2	$2 \times 2 \times 128$



Continuando con el ejemplo, finalmente:

- Se toma la última capa oculta a la que se le hizo sub-sampling, que se dice que es “tridimensional” por tomar la forma, en este ejemplo, de $2 \times 2 \times 128$, (alto, ancho, mapas) y se luego “aplana”, esto es que deja de ser tridimensional, y pasa a ser una capa de neuronas “tradicionales”, de las que ya conocíamos.
- Por ejemplo, podríamos aplanar (y conectar) a una nueva capa oculta de 100 neuronas feed-forward, aplicamos una Red Neuronal tipo **Perceptrón**.
- Por ejemplo, a esta nueva capa oculta “tradicional”, le aplicamos una función llamada Softmax que conecta contra la capa de salida final que tendrá la cantidad de neuronas correspondientes con las clases que estamos clasificando.
- Si clasificamos perros, gatos y pájaros, por ejemplo, serán 3 neuronas.

ARQUITECTURA DE UNA CNN

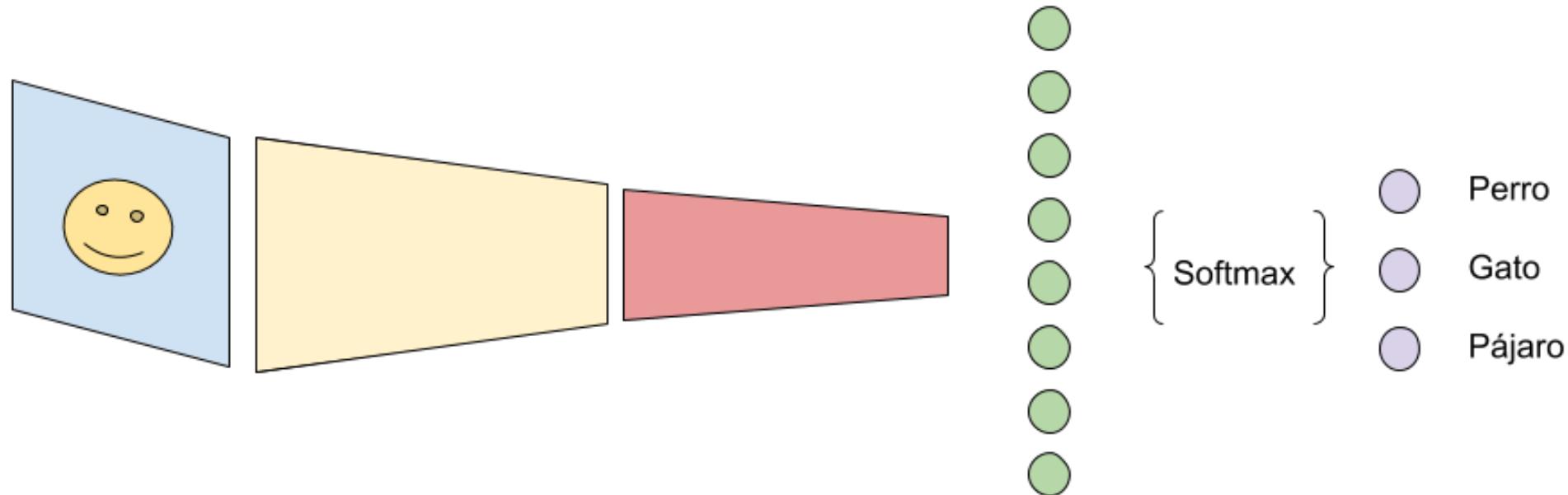


IMAGEN DE
ENTRADA

1er CONVOLUCIÓN

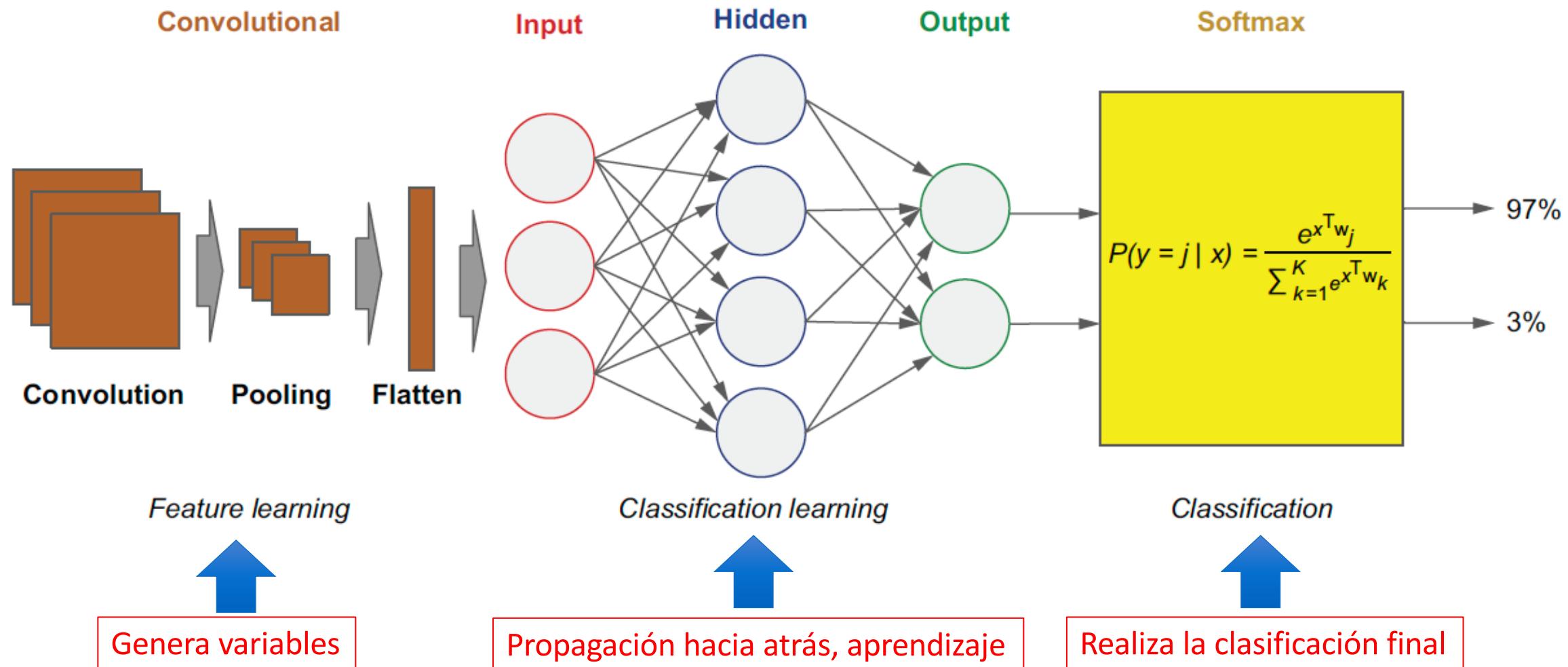
- FILTROS
- RELU
- SUBSAMPLING

2^a CONVOLUCIÓN (Y SUCESIVAS)

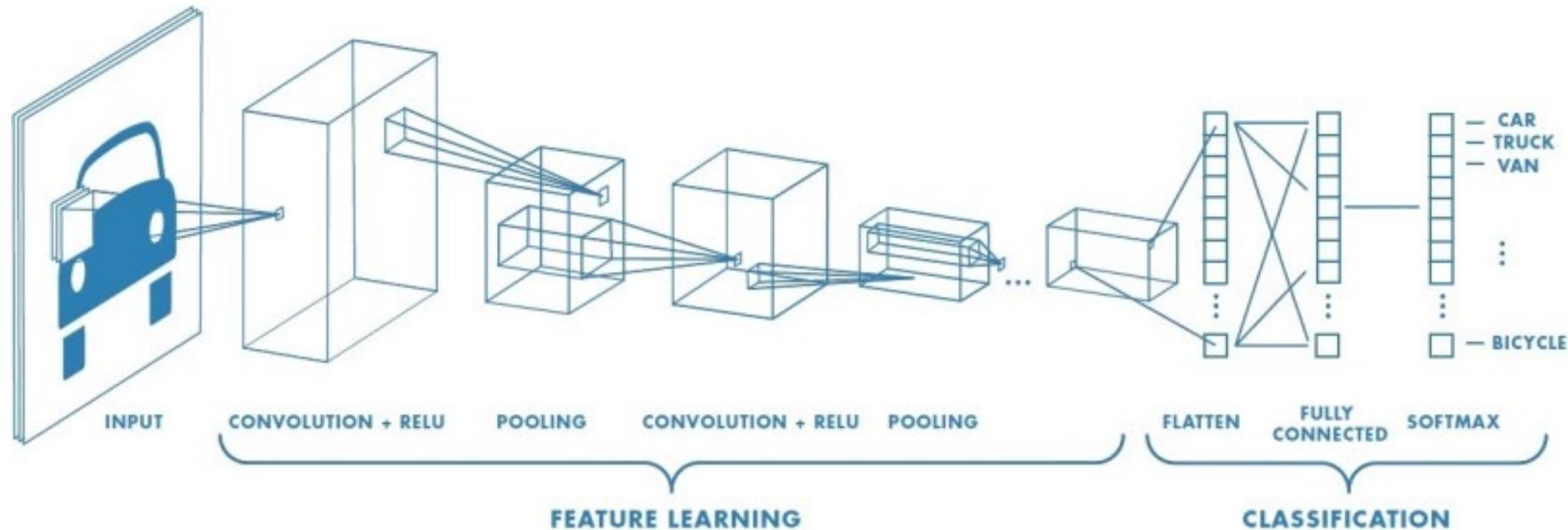
RED NEURONAL
MULTICAPA
(fully connected)

CAPA DE SALIDA:
CLASIFICACIÓN
ONE-HOT
ENCODED

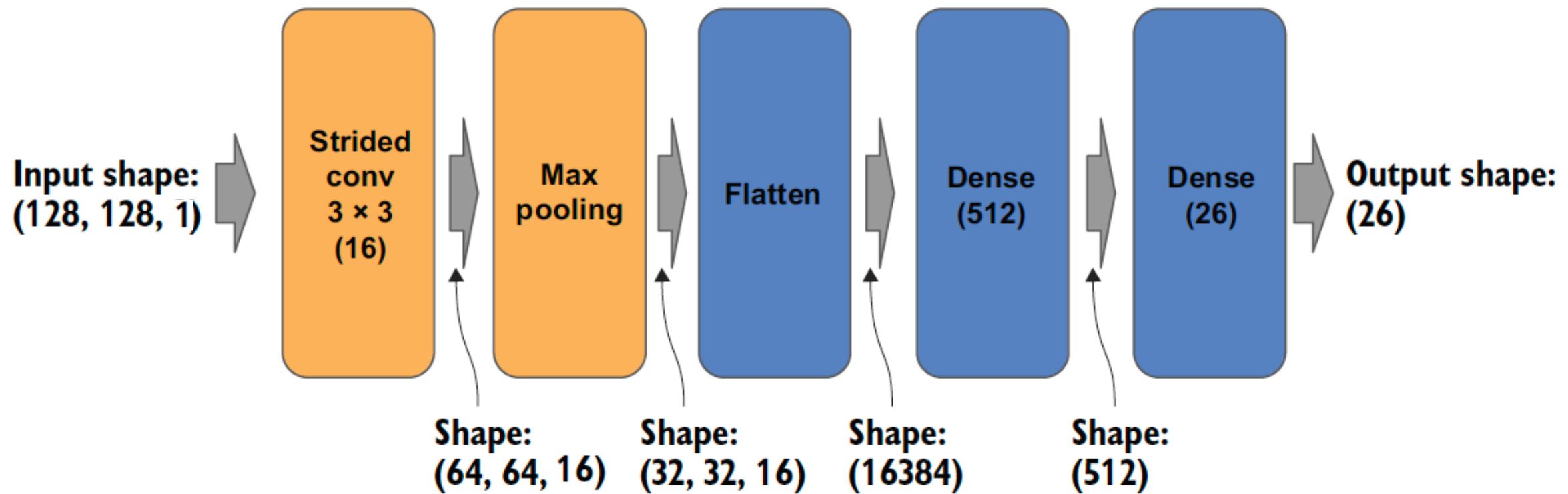
Arquitectura General de CNN



Arquitectura General de CNN



Ejemplo 1: Diseño de una CNN la clase de Python Conv2D





Ejemplo: Diseño de una CNN la clase de Python Conv2D

1. Entran imágenes de tamaño 128×128 , es decir, $128 \times 128 \times 1$.
2. Se aplican 16 convoluciones tamaño 3×3 con un “stride” (paso) de 2 y queda de tamaño $64 \times 64 \times 16$.
3. Se aplica un Max-Pooling de tamaño 2 y queda de tamaño $32 \times 32 \times 16$.
4. Luego se aplana y queda de tamaño $32 \times 32 \times 16 = 16,384$ variables (columnas).
5. Luego con Dense(512) se crea una capa oculta con 512 neuronas.
6. Luego con Dense(26) se crea una capa oculta con 26 neuronas.

Ejemplo: Diseño de una CNN la clase de Python Conv2D

Here's the code:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, ReLU, Activation
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten

model = Sequential()
model.add(Conv2D(16, kernel_size=(3, 3), strides=(2, 2), padding="same",
                input_shape=(128, 128, 1))) ←
model.add(ReLU())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2))) ←
model.add(Flatten()) ←
model.add(Dense(512))
model.add(ReLU())
model.add(Dense(26))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Image data is
inputted to a
convolutional
layer.

The 2D feature maps are
flattened into a 1D vector
before the output layer.

The size the
feature maps
is reduced by
pooling.

Let's look at the details of the layers in our model by using the `summary()` method:

The output from the convolutional layer is 16 feature maps of 2D size 64×64 .

```
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 64, 64, 16)	160
re_lu_1 (ReLU)	(None, 64, 64, 16)	0
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 16)	0
flatten_1 (Flatten)	(None, 16384)	0
dense_1 (Dense)	(None, 512)	8389120
re_lu_2 (ReLU)	(None, 512)	0
dense_2 (Dense)	(None, 26)	13338
activation_1 (Activation)	(None, 26)	0

The final dense layer has 26 nodes, one for each letter in the English alphabet.

The output from the pooling layer reduces the feature map sizes to 32×32 .

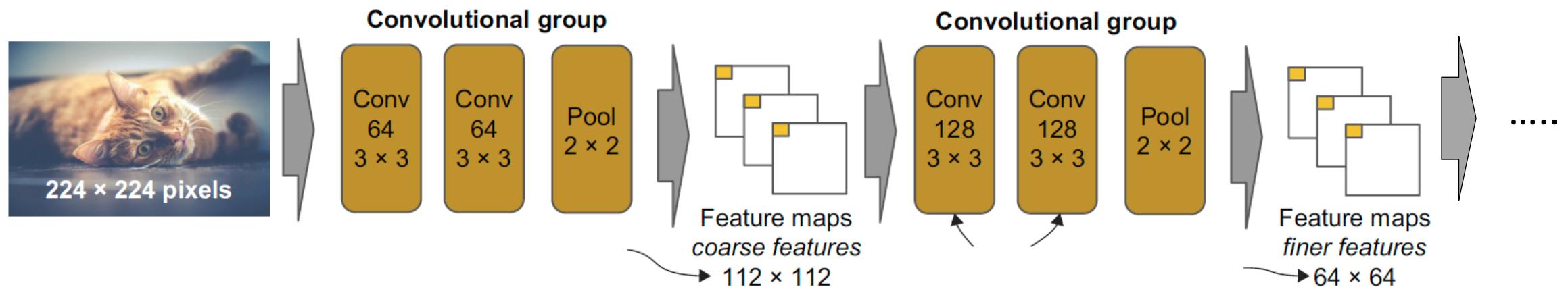
The number of parameters for the 512-node dense layer is over 8 million; every node in the flatten layer is connected to every node in the dense layer.

Filtro de 3×3 implica 9 parámetros y son 16 filtros (convoluciones) implica 144 parámetros + 16 bias entonces son en total $(9 \times 16) + 16 = 160$ parámetros.

Número_variables x Número_Neuronas + bias) = $(16384 \times 512) + 512 = 8389120$

Número_variables x Número_Neuronas + bias) = $(512 \times 26) + 26 = 13338$

Ejemplo con una imagen a color: Diseño de una CNN la clase de Python Conv2D



Apilamiento de capas

Denotamos:

- n_H = número de filas de la matriz.
- n_W = número de columnas de la matriz.
- $n_{H_{prev}}$ = número de filas de la matriz de la capa anterior.
- $n_{W_{prev}}$ = número de columnas de la matriz de la capa anterior.
- f = tamaño de la convolución o del pooling.
- pad = tamaño del relleno de ceros (padding).
- $stride$ = tamaño de la zancada.

Entonces:

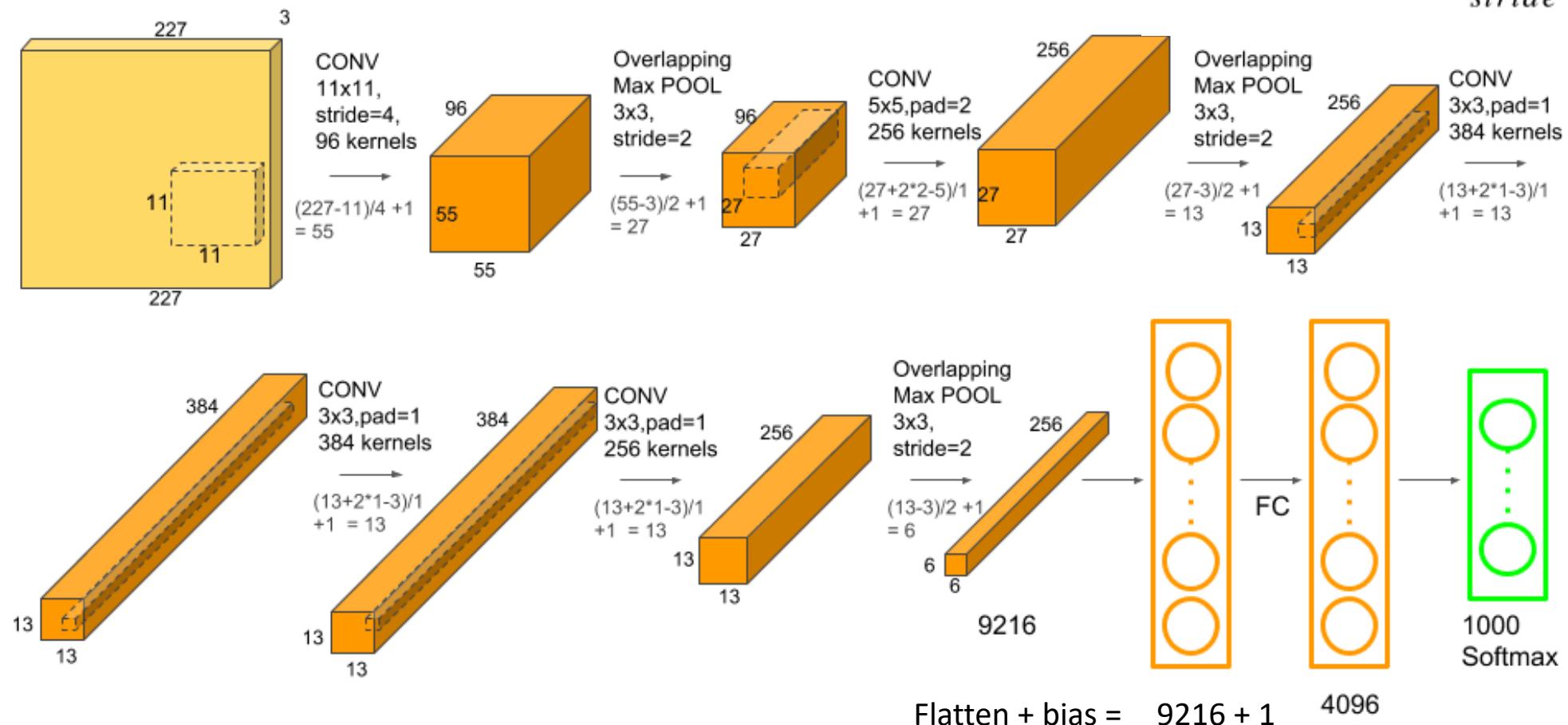
$$n_H = \lfloor \frac{n_{H_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

$$n_W = \lfloor \frac{n_{W_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

Apilamiento de capas:

$$n_H = \left\lfloor \frac{n_{H\ prev} - f + 2 \times pad}{stride} \right\rfloor + 1$$

$$n_W = \left\lfloor \frac{n_{W\ prev} - f + 2 \times pad}{stride} \right\rfloor + 1$$



Número de Parámetros y Resumen del modelo

- Esta CNN anterior requiere de 80,226 parámetros, vemos porqué.
- La capa **conv2d_1** con núcleos (kernels) de tamaño 3×3 **tiene 640 parámetros** porque cada kernel tiene $3 \times 3 = 9$ parámetros, y son 64 convoluciones y cada kernel tiene un parámetro de sesgo (bias). Así tenemos el **Total Parámetros = $(9 \times 64) + 64 = 640$** .
- La capa **conv2d_2** con núcleos (kernels) de tamaño 24×24 **tiene 36928 parámetros** porque cada kernel tiene $24 \times 24 = 576$ parámetros y son 64 convoluciones y cada kernel tiene un parámetro de sesgo (bias). Así tenemos el **Total Parámetros = $(576 \times 64) + 64 = 36928$** .

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d_1 (MaxPooling2)	(None, 13, 13, 64)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	36928
max_pooling2d_2 (MaxPooling2)	(None, 5.5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3.3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dropout_1 (Dropout)	(None, 576)	0
dense_1 (Dense)	(None, 10)	5770
activation_1 (Activation)	(None, 10)	0

Total params: 80,266
Trainable params: 80,266
Non-trainable params: 0

$577 \times 10 = 5770$

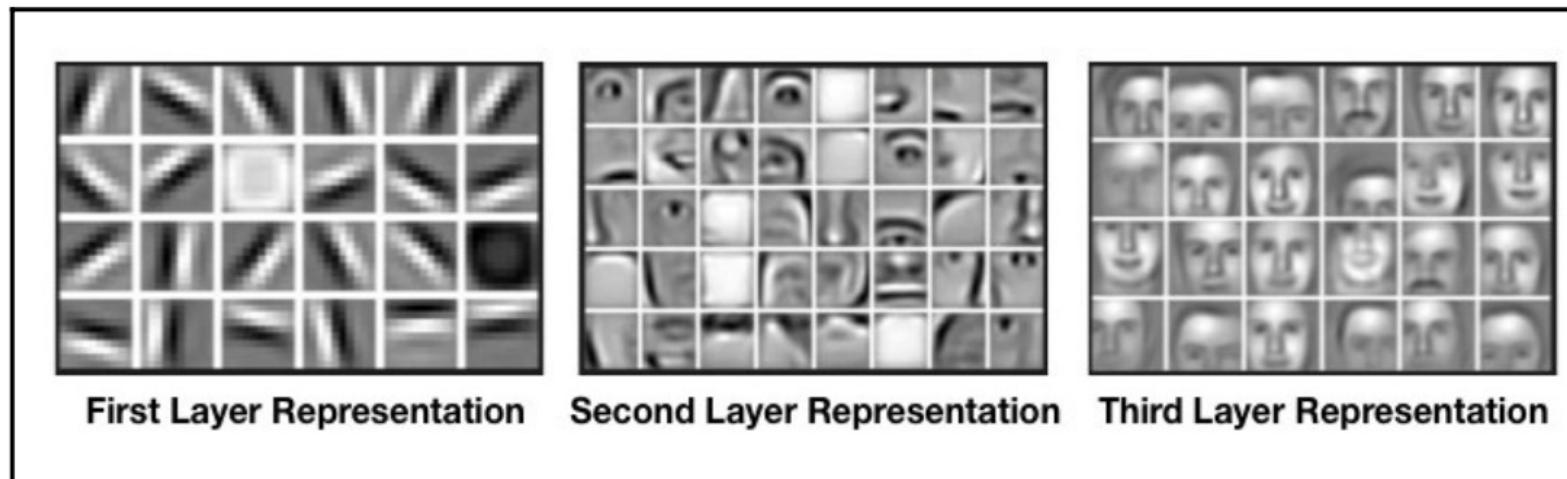


Trabajar con la arquitectura ConvNet

- Ahora que conocemos todos los diferentes componentes de una ConvNet (Red Convolucional), se puede unir todo y ver cómo construir una CNN profunda (Deep). En general, lo que aprende la arquitectura se puede demostrar con el siguiente flujo:
- *Imagen de entrada → degradados+bordes → texturas → patrones → características básicas del objeto→objeto→salida*
- Las características aumentan en complejidad en las **últimas** capas. Es decir, las **primeras capas** (las más cercanas a la capa de entrada) aprenden características muy **básicas**, como bordes y líneas, texturas o cómo se diferencian ciertos colores. Las **últimas capas** toman el mapa de características de la capa anterior como entrada y aprenden patrones más **complejos** de él.

Trabajar con la arquitectura ConvNet

Por ejemplo, si creamos un modelo de *reconocimiento facial*, la primera capa aprendería las *líneas, curvas y degradados más simples posibles*. La siguiente capa tomaría los mapas de características de la capa anterior y los usaría para aprender características más complejas, como el cabello y las cejas. La capa posterior aprendería características aún más complejas, como ojos, narices, oídos, etc.





Entrenamiento y optimización

No necesitamos un algoritmo nuevo para nuestro entrenamiento y optimización, podemos seguir usando la **retropropagación** y el descenso del **gradiente** para calcular el error, diferenciarlo de las capas anteriores y actualizar los pesos para acercarnos lo más posible a los óptimos globales.



MUCHAS GRACIAS...



GRACIAS....