



Módulo 7 – Fundamentos de Deep Learning

# Redes Convolutivas

Especialización en Ciencia de Datos

# Contenido



1. Redes Neuronales Convolutivas.
1. Implementación Keras-Tensorflow

# Redes Neuronales Convolutivas



KIBERNUM

# ¿Qué es una CNN?

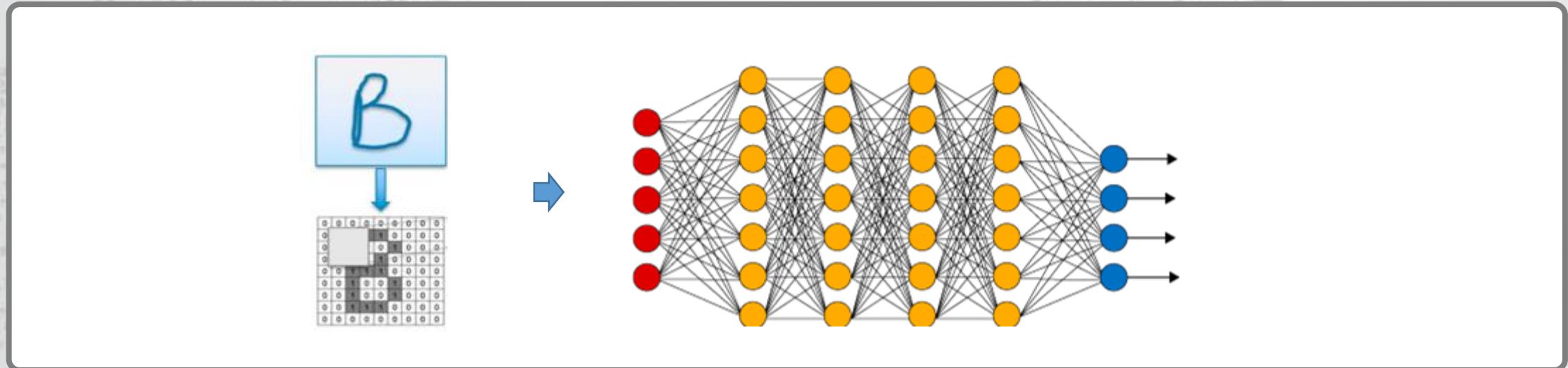
Una red neuronal convolucional (CNN, por sus siglas en inglés) es un tipo de red neuronal artificial que se utiliza principalmente para **procesamiento de imágenes y reconocimiento de patrones**.

Las CNN se llaman "convolucionales" porque utilizan una operación matemática llamada convolución para procesar la información de entrada. La convolución consiste en aplicar un filtro, que es una matriz de números, a la imagen de entrada para extraer características importantes, como bordes, contornos, texturas y patrones. Estas características son luego procesadas por capas de la red neuronal para hacer predicciones o clasificar la imagen.

Las CNN son muy útiles en aplicaciones como la clasificación de imágenes, la detección de objetos, la segmentación de imágenes, el reconocimiento de rostros y la detección de emociones. Debido a su capacidad para procesar grandes cantidades de datos, las CNN se han utilizado con éxito en campos como la medicina, la astronomía, la robótica y la ciencia de datos en general.

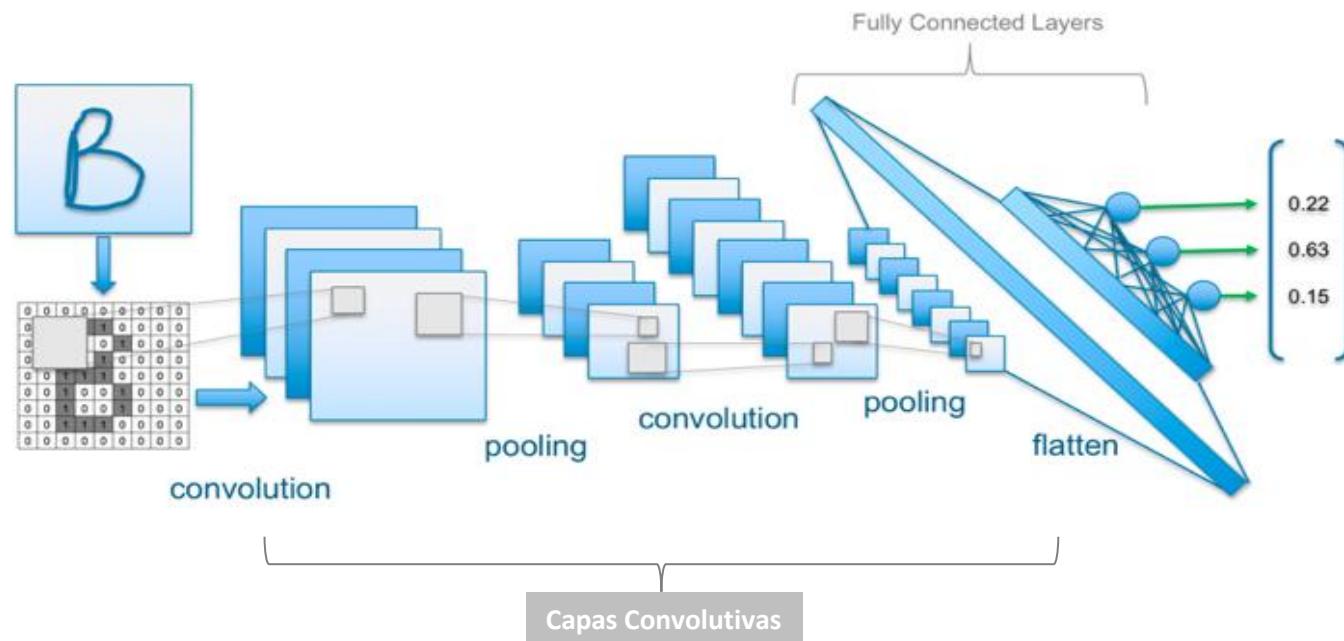
# Diferencias entre una CNN y una FCN

- La principal diferencia entre una red neuronal convolucional (CNN) y una red neuronal completamente conectada (fully connected o FCN) es la forma en que procesan los datos de entrada.
- En una red FCN, cada neurona de una capa está conectada a todas las neuronas de la capa siguiente, lo que significa que todas las características de entrada son procesadas simultáneamente para hacer una predicción. Por ejemplo, en una red FCN utilizada para el reconocimiento de dígitos escritos a mano, cada píxel de la imagen de entrada se conecta directamente a cada neurona de la primera capa oculta.



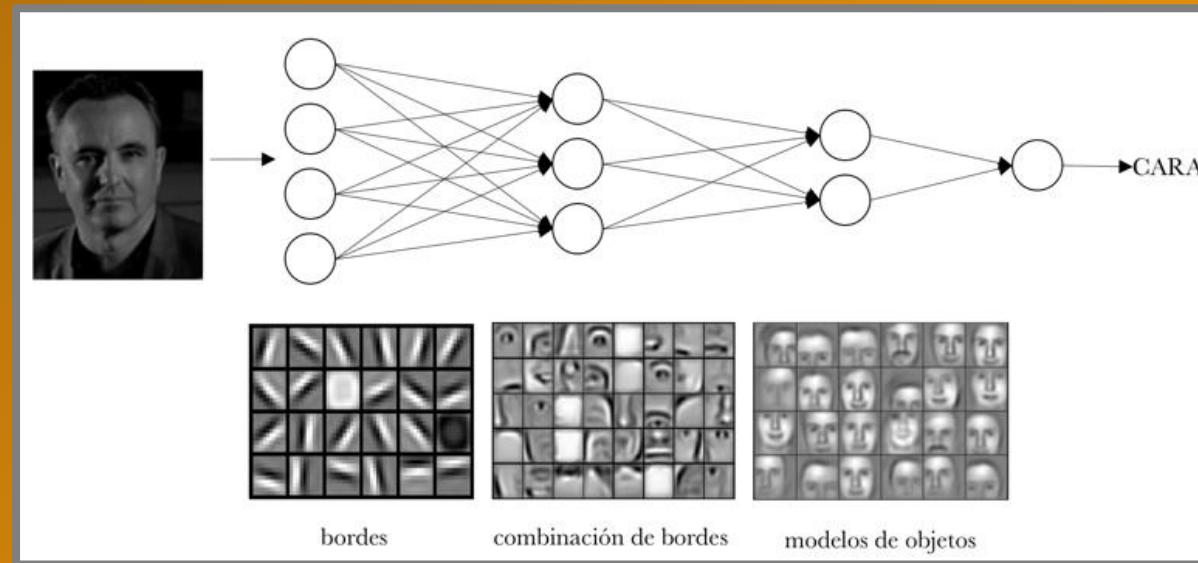
# Diferencias entre una CNN y una FCN

En cambio, en una CNN, las capas convolucionales procesan solo pequeñas regiones de la imagen de entrada a la vez, en lugar de la imagen completa, utilizando filtros que se desplazan sobre la imagen para extraer características locales. Además, las **capas convolucionales** están seguidas de **capas de pooling** que reducen la resolución de la imagen para disminuir la cantidad de parámetros de la red y evitar el sobreajuste. Finalmente, las capas completamente conectadas procesan las características extraídas por las capas convolucionales para hacer la predicción.



# Operación de Convolución

La convolución es una operación matemática que se utiliza para procesar imágenes y otros tipos de datos estructurados en redes neuronales convolucionales, permitiendo extraer características importantes de manera eficiente y con una cantidad menor de parámetros.



La convolución se utiliza en las CNN porque permite extraer características importantes de las imágenes de entrada, como bordes, contornos, texturas y patrones, de manera eficiente. Además, la convolución reduce la cantidad de parámetros de la red en comparación con las redes neuronales completamente conectadas, lo que ayuda a prevenir el sobreajuste y a mejorar el rendimiento de la red.

# Operación de Convolución

Para realizar el proceso de convolución, se utiliza lo que se denomina Filtro Convolutivo (Kernel), que es una matriz de valores numéricos.



Imagen de Entrada

\*

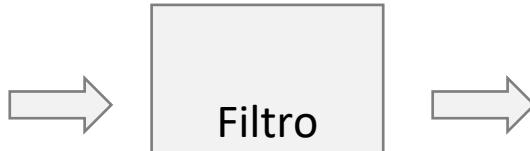
Filtro

=

Imagen de Salida

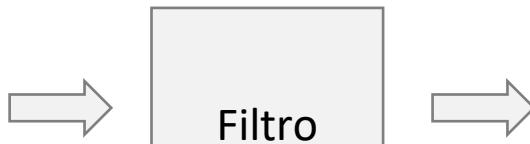
# Operación de Convolución

Algunos ejemplos:



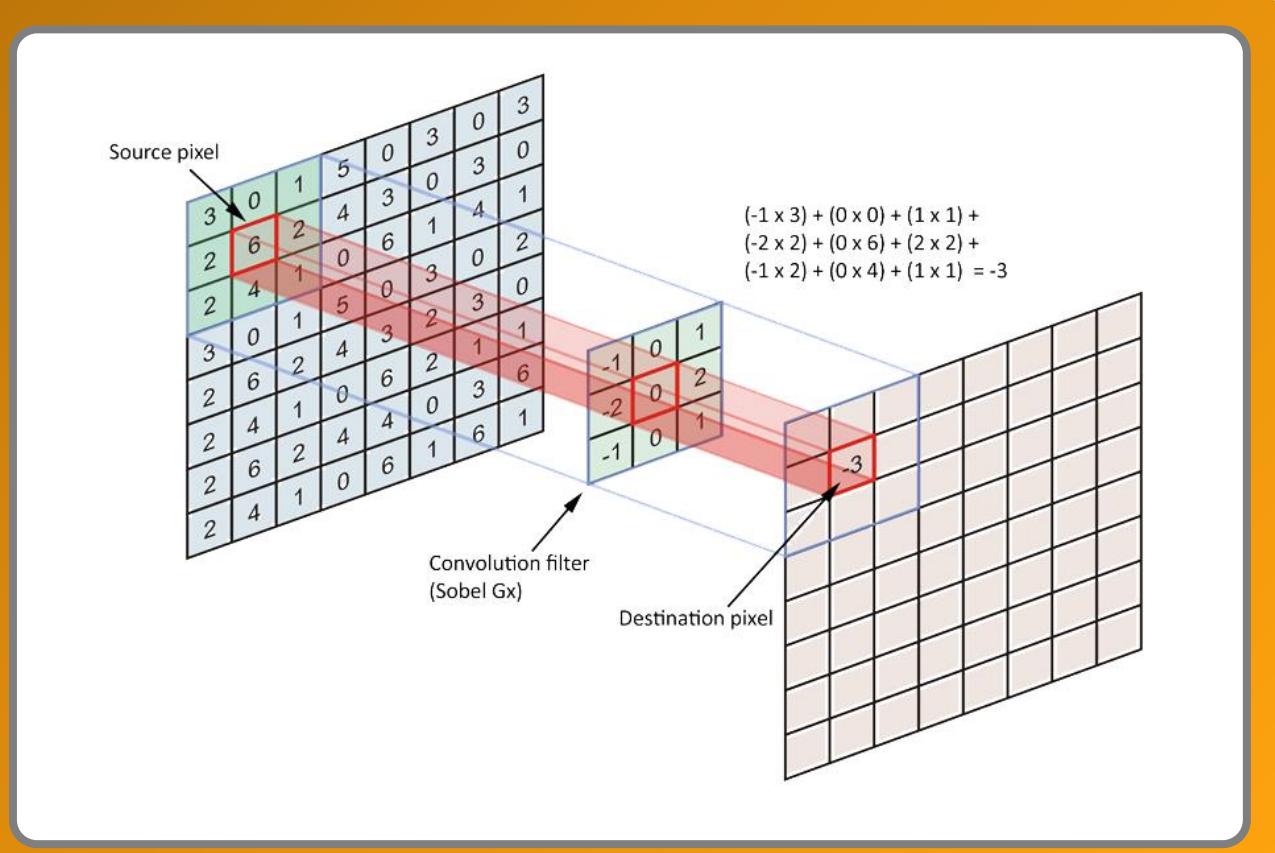
# Operación de Convolución

Algunos ejemplos:



# Operación de Convolución

La convolución es una operación matricial entre la matriz de entrada y el filtro o kernel convolucional, en donde se va recorriendo la matriz de entrada para obtener los valores en la matriz de salida. Nótese cómo los pixeles de entrada se transforman al aplicar el filtro.



# Operación de Convolución

De forma gráfica:

Imagen de entrada

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro

1	0
0	2

Salida

60		

$$\begin{array}{|c|c|} \hline 0 & 10 \\ \hline 20 & 30 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 2 \\ \hline \end{array}$$

=

$$1 * 0 + 0 * 10 + 0 * 20 + 2 * 30 = 60$$

(Multiplicación elemento a elemento)

# Operación de Convolución

De forma gráfica:

Imagen de entrada			
0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro	
1	0
0	2

Salida	
60	70

$$\begin{array}{cc} \begin{array}{|c|c|} \hline 10 & 10 \\ \hline 30 & 30 \\ \hline \end{array} & \times \end{array} \quad = \quad 1 * 10 + 0 * 10 + 0 * 30 + 2 * 30 = 70$$

(Multiplicación elemento a elemento)

# Operación de Convolución

De forma gráfica:

Imagen de entrada			
0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro	
1	0
0	2

Salida		
60	70	50

$$\begin{array}{cc} \begin{array}{|c|c|}\hline 10 & 0 \\ \hline 30 & 20 \\ \hline \end{array} & \times & \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 2 \\ \hline \end{array} \end{array} = 1 * 10 + 0 * 0 + 0 * 30 + 2 * 20 = 70$$

(Multiplicación elemento a elemento)

# Operación de Convolución

De forma gráfica:

Imagen de entrada

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro

1	0
0	2

Salida

60	70	50
60		

$$\begin{bmatrix} 20 & 30 \\ 10 & 20 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$= 1 * 20 + 0 * 30 + 0 * 10 + 2 * 20 = 60$$

(Multiplicación elemento a elemento)

# Operación de Convolución

De forma gráfica:

Imagen de entrada

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro

1	0
0	2

Salida

60	70	50
60	70	

$$\begin{bmatrix} 30 & 30 \\ 20 & 20 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} = 1 * 30 + 0 * 30 + 0 * 20 + 2 * 20 = 70$$

(Multiplicación elemento a elemento)

# Operación de Convolución

De forma gráfica:

Imagen de entrada

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro

1	0
0	2

Salida

60	70	50
60	70	50

$$\begin{bmatrix} 30 & 20 \\ 20 & 10 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$= 1 * 30 + 0 * 20 + 0 * 20 + 2 * 10 = 50$$

(Multiplicación elemento a elemento)

# Operación de Convolución

De forma gráfica:

Imagen de entrada

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro

1	0
0	2

Salida

60	70	50
60	70	50
20		

$$\begin{bmatrix} 10 & 20 \\ 0 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$= 1 * 10 + 0 * 20 + 0 * 0 + 2 * 5 = 20$$

(Multiplicación elemento a elemento)

# Operación de Convolución

De forma gráfica:

Imagen de entrada

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro

1	0
0	2

Salida

60	70	50
60	70	50
20	30	

$$\begin{array}{|c|c|} \hline 20 & 20 \\ \hline 5 & 5 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 2 \\ \hline \end{array}$$

$$= 1 * 20 + 0 * 20 + 0 * 5 + 2 * 5 = 30$$

(Multiplicación elemento a elemento)

# Operación de Convolución

De forma gráfica:

Imagen de entrada

0	10	10	0
20	30	30	20
10	20	20	10
0	5	5	0

Filtro

1	0
0	2

Salida

60	70	50
60	70	50
20	30	20

$$\begin{bmatrix} 20 & 10 \\ 5 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

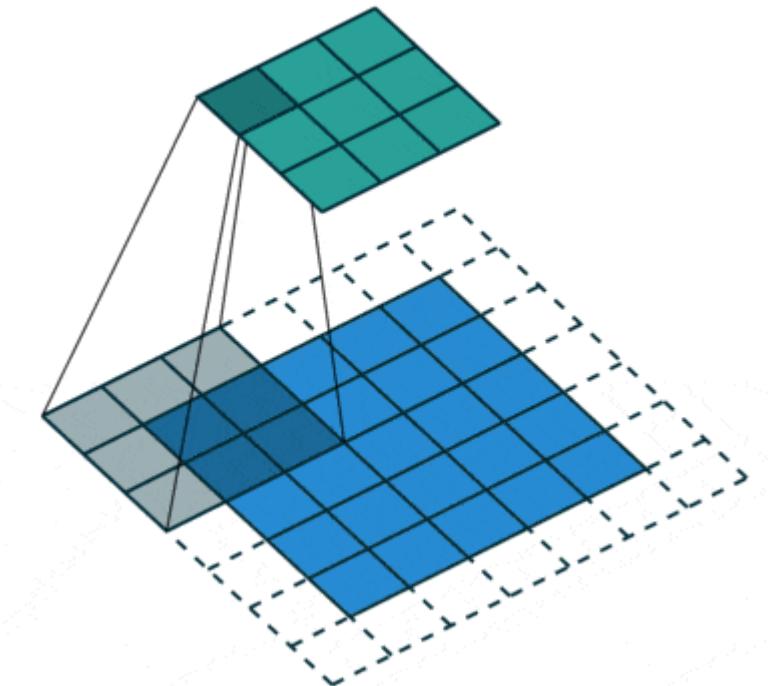
$$= 1 * 20 + 0 * 10 + 0 * 5 + 2 * 0 = 20$$

(Multiplicación elemento a elemento)

# Operación de Convolución

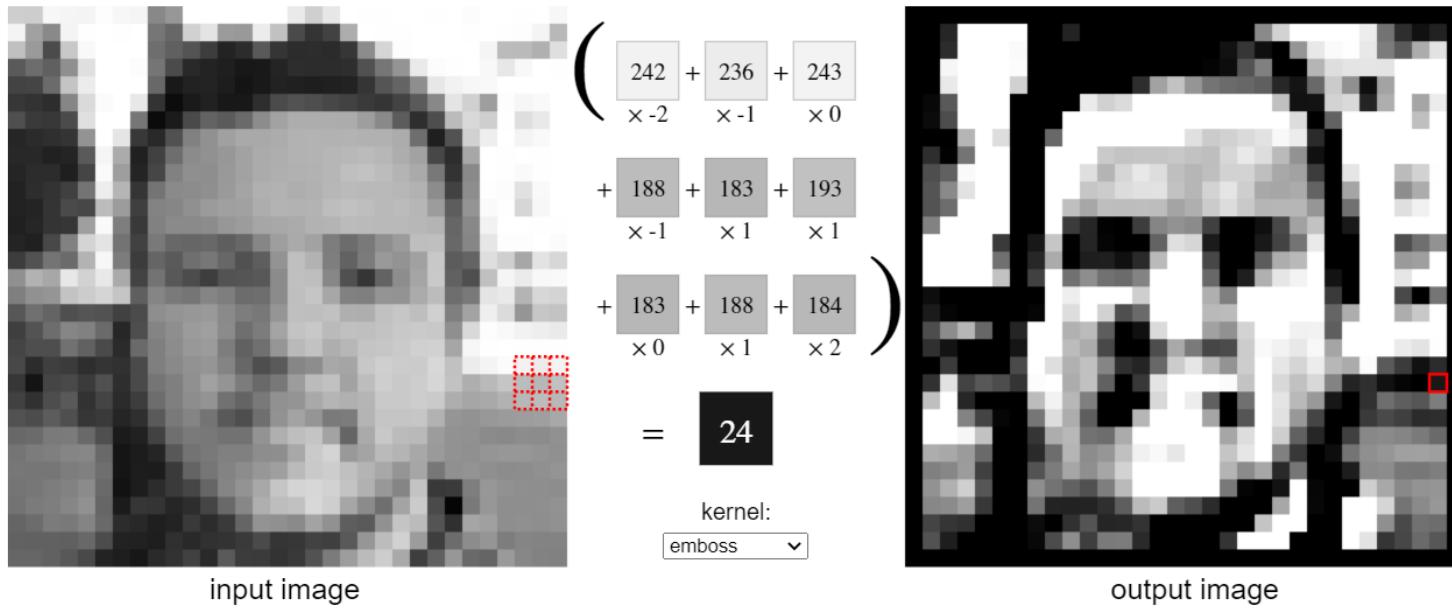
La mayoría de las veces necesitamos que el tamaño de la imagen de entrada sea igual al tamaño de la imagen de salida . Pero cuando aplicamos una multiplicación de filtro, normalmente perdemos dimensiones. Puede ver la dimensión de salida en la imagen gif arriba donde la matriz  $5 * 5$  se reduce a  $3 * 3$  .

Para mantener intactas las dimensiones , es decir, para mantener las dimensiones de entrada frente a salida según las necesidades del usuario , utilizamos un concepto de **relleno**.



# Playground Filtros Convolutivos

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.



One subtlety of this process is what to do along the edges of the image. For example, the top left corner of the input image only has three neighbors. One way to fix this is to extend the edge values out by one in the original image while keeping our new image the same size. In this demo, we've instead ignored those values by making them black.

Here's a playground were you can select different kernel matrices and see how they effect the original image or build your

<https://setosa.io/ev/image-kernels/>

# 1D, 2D y 3D

Cuando se trata de datos 1D, 2D o 3D, la única diferencia en convoluciones es la forma de la matriz de entrada seguida de la forma de los filtros que se ajustan en consecuencia según las dimensiones.

2-D



Image

3D



Image

3-D

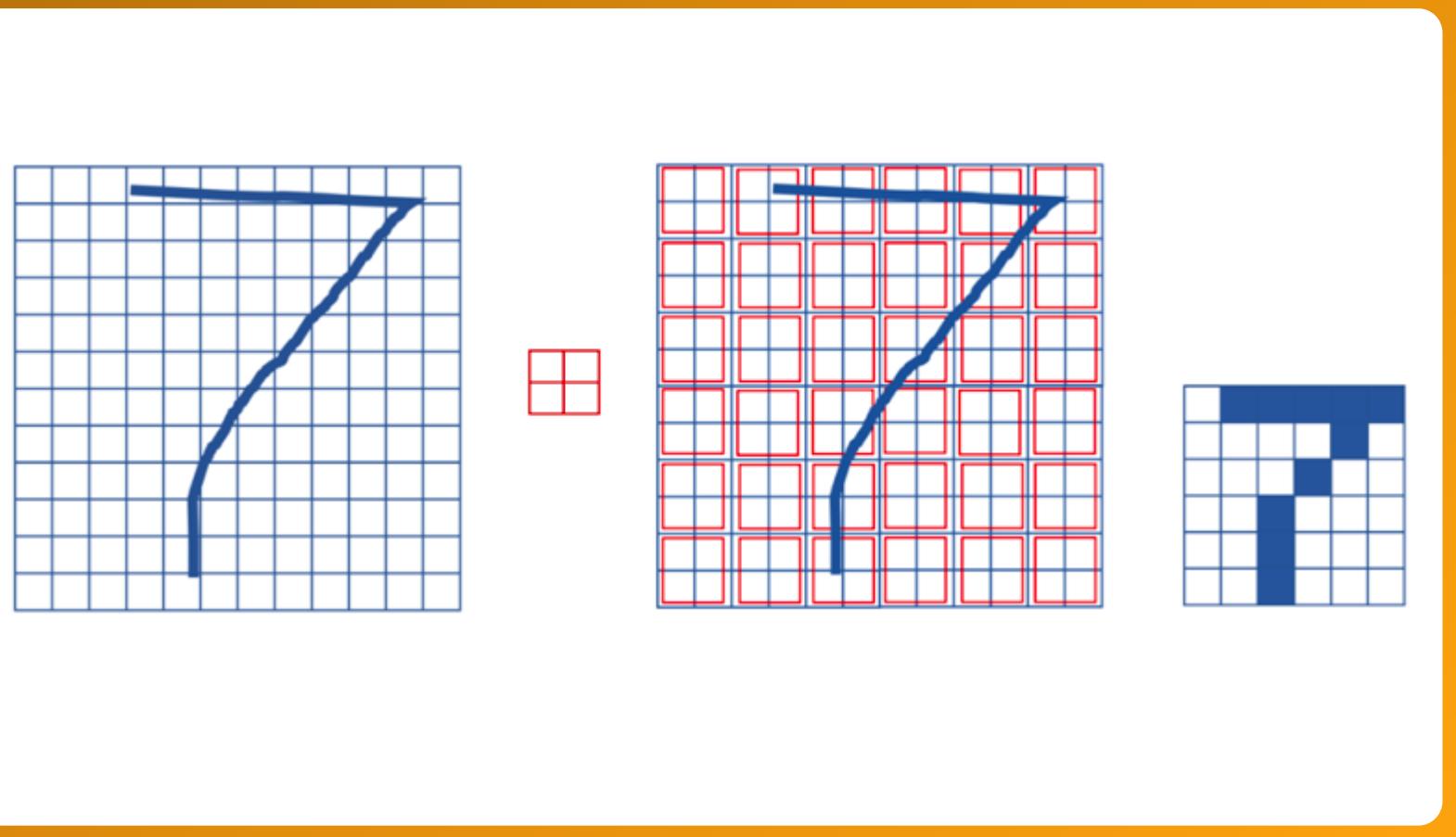
Imágenes en escala de grises son representadas por matrices 2D

Imágenes en color son representadas por matrices 3D, puesto que la tercera dimensión es el canal de color (R, G, B)

# Operación de Pooling

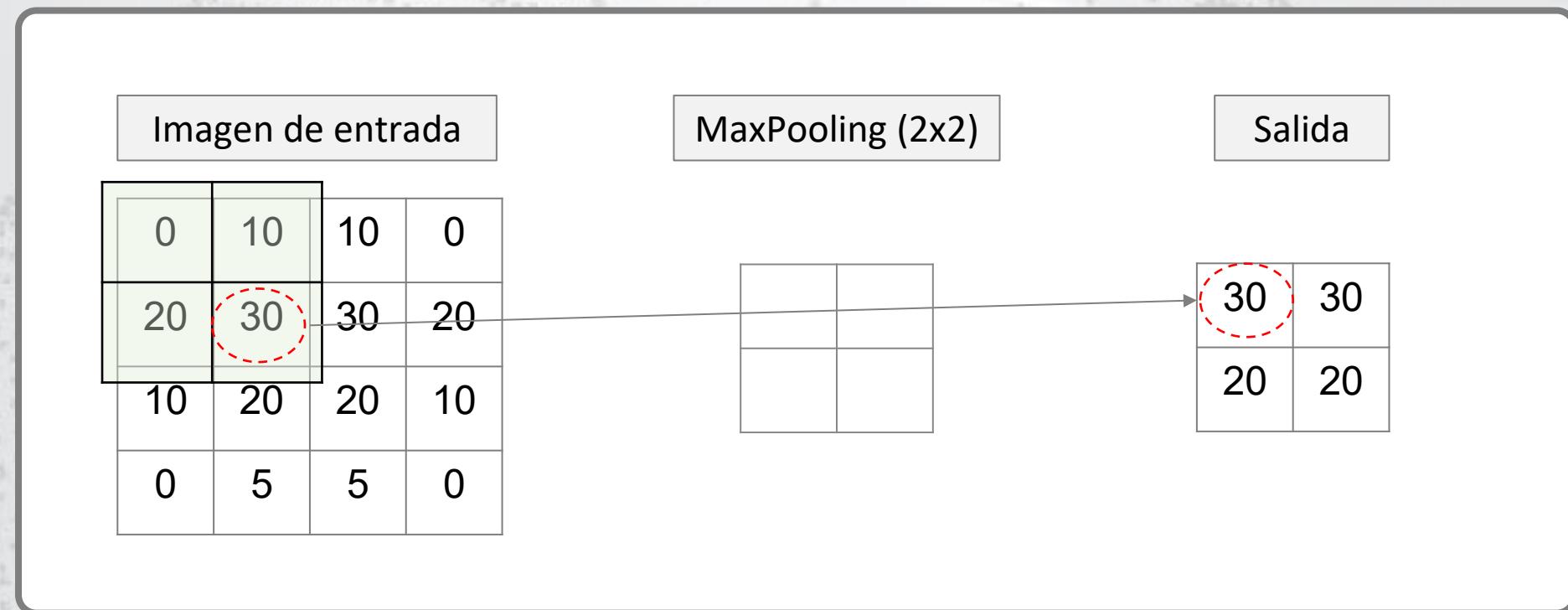
Pooling (también conocido como subsampling) es una técnica comúnmente utilizada en la capa de reducción de una red neuronal convolucional (CNN) para reducir la dimensionalidad de los mapas de características obtenidos por las capas convolucionales anteriores.

El resultado de la operación de pooling es una versión reducida del mapa de características original con menos dimensiones espaciales y, por lo tanto, menos información. Sin embargo, esto puede ser beneficioso para reducir el costo computacional y la complejidad del modelo, y también puede ayudar a prevenir el sobreajuste en algunos casos.



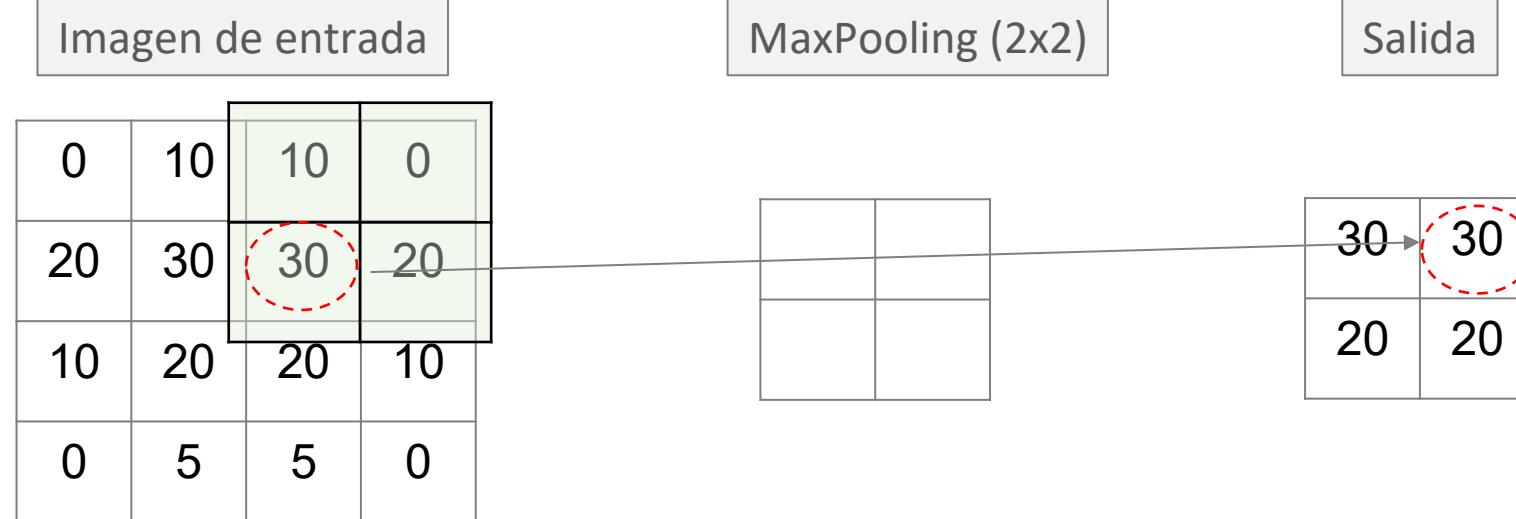
# MaxPooling

**Max Pooling** considera el número mayor al recorrer la imagen con la matriz de pooling, obteniendo así, una matriz de resultado de menor dimensión.



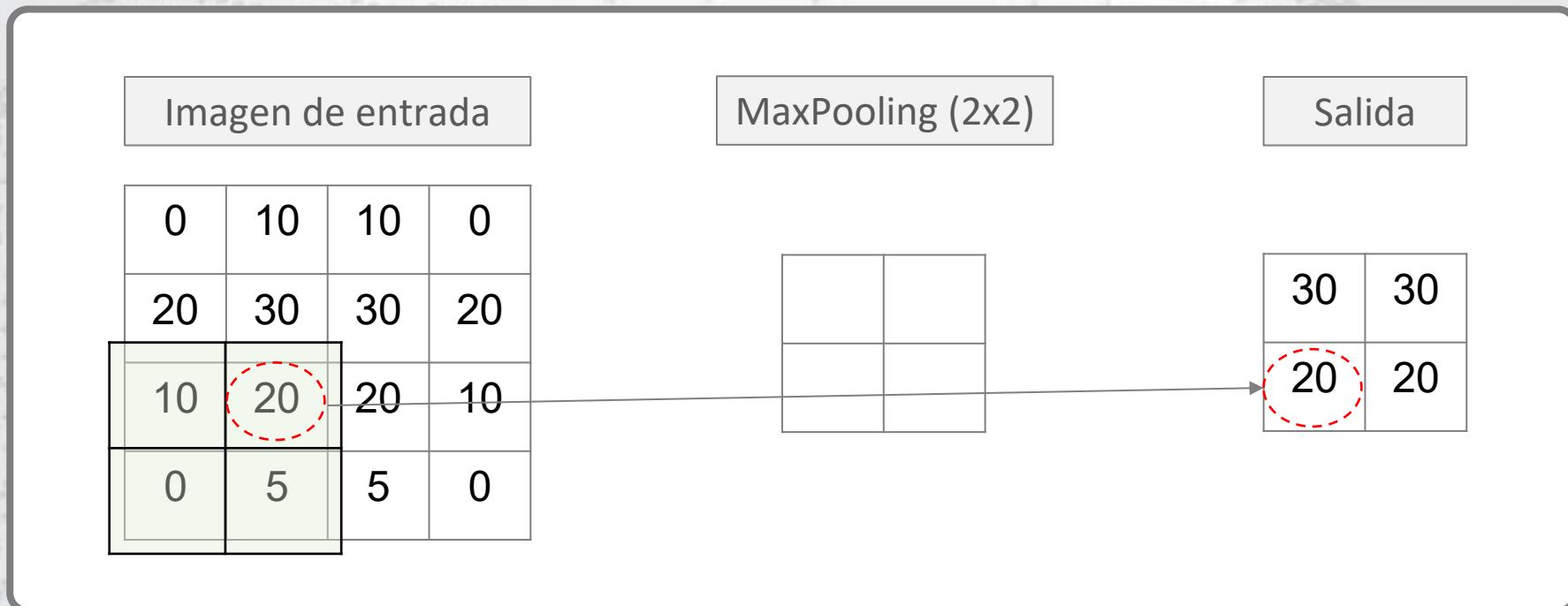
# MaxPooling

**Max Pooling** considera el número mayor al recorrer la imagen con la matriz de pooling, obteniendo así, una matriz de resultado de menor dimensión.



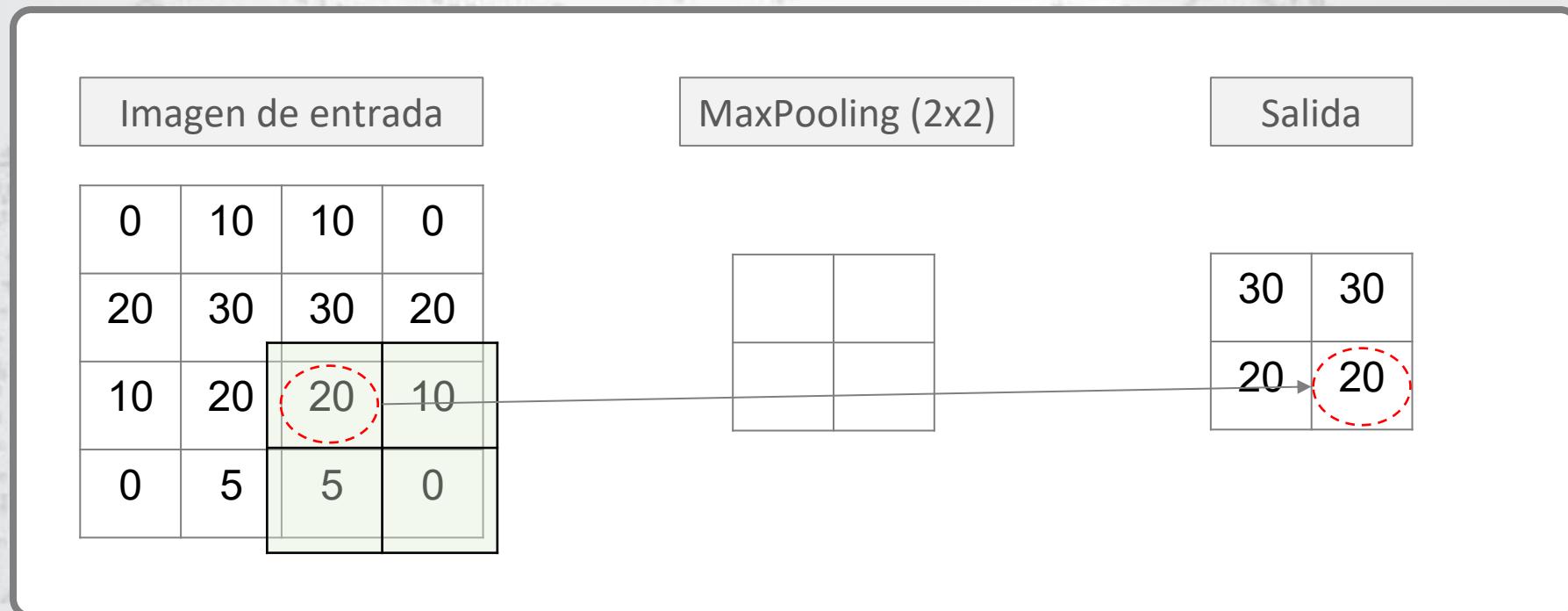
# MaxPooling

Max Pooling considera el número mayor al recorrer la imagen con la matriz de pooling, obteniendo así, una matriz de resultado de menor dimensión.



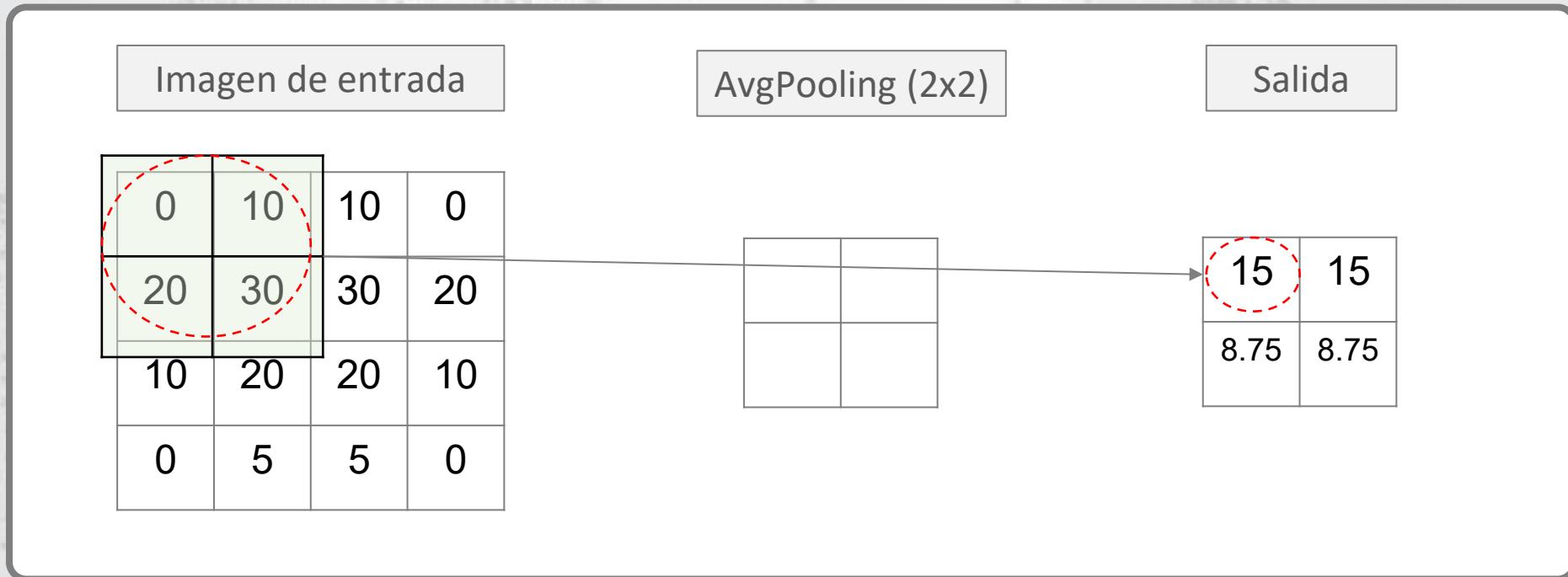
# MaxPooling

Max Pooling considera el número mayor al recorrer la imagen con la matriz de pooling, obteniendo así, una matriz de resultado de menor dimensión.



# AvgPooling

**Avg Pooling** considera el promedio de los valores considerados al recorrer la imagen con la matriz de pooling, obteniendo así, una matriz de resultado de menor dimensión.



# Operación de Pooling

La agrupación nos ayuda de 2 maneras:

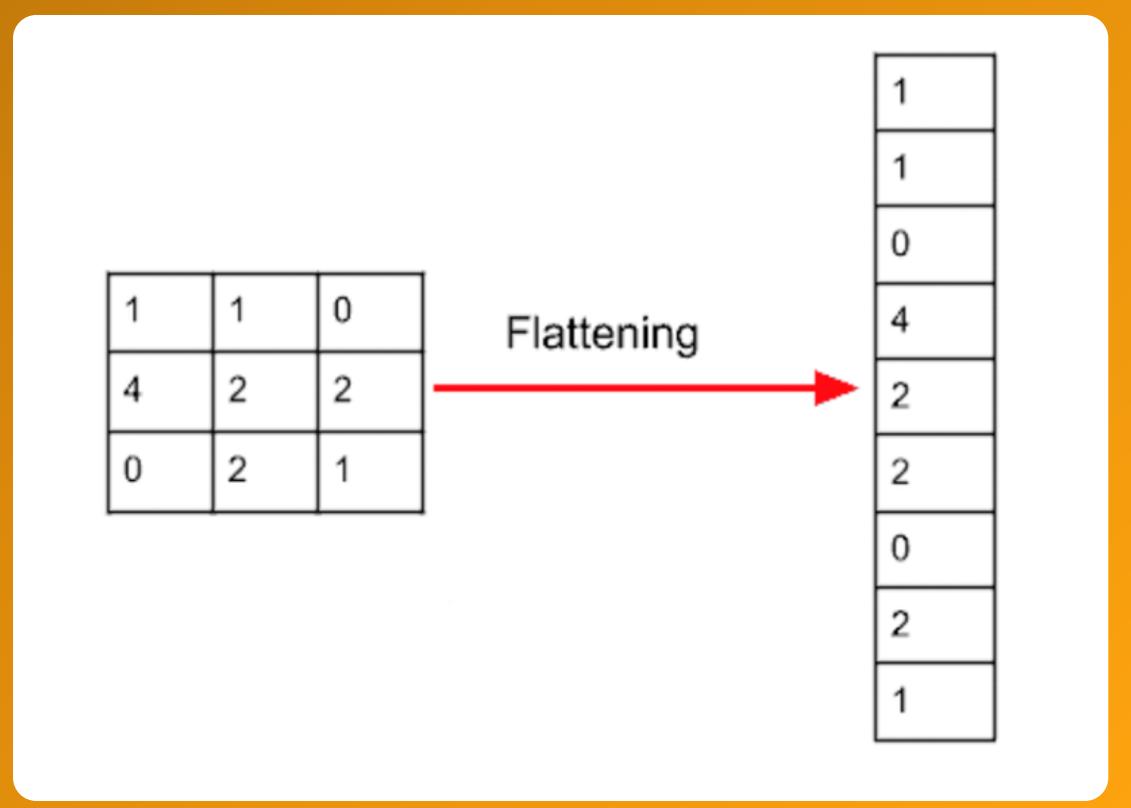
- Reducir la dimensión de entrada y conserve solo las características importantes.
- Ayuda a identificar una imagen independientemente de su posición en una imagen.

Ambas imágenes corresponden a la letra 'A'. Pero 'A' no está en la misma ubicación. La agrupación ayuda a cubrir esto y, sin embargo, identifica dónde está 'A'. **Este es un golpe maestro ya que ayuda a identificar cualquier imagen independientemente de cualquier posición.**



# Aplanamiento de Capas

Esta capa convierte una capa tridimensional en la red en un vector unidimensional para adaptarse a la entrada de una capa totalmente conectada para la clasificación. Por ejemplo, un tensor de  $5 \times 5 \times 2$  se convertiría en un vector de tamaño 50. Las capas convolucionales anteriores de la red extraían las características de la imagen de entrada, pero ahora es el momento de clasificar las características. Usamos la función softmax para clasificar estas características, lo que requiere una entrada unidimensional. Es por eso que la capa plana es necesaria.

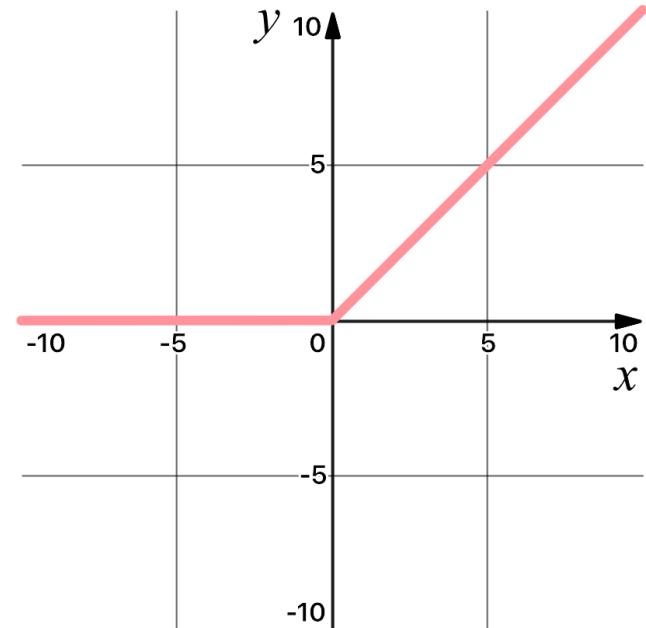


# Funciones de Activación - ReLU

## ReLU

Parte de la razón por la que estas innovadoras CNN pueden lograr una precisión tan tremenda es por su **no linealidad**. ReLU aplica la no linealidad muy necesaria en el modelo. La no linealidad es necesaria para producir límites de decisión no lineales, de modo que la salida no se pueda escribir como una combinación lineal de las entradas. Si no estuviera presente una función de activación no lineal, las arquitecturas CNN profundas se convertirían en una sola capa convolucional equivalente, que no funcionaría tan bien.

La función de activación de ReLU se usa específicamente como una función de activación no lineal, a diferencia de otras funciones no lineales como Sigmoid porque **se ha observado empíricamente que las CNN que usan ReLU son más rápidas de entrenar que sus contrapartes**.



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

# Funciones de Activación - Softmax

## Softmax

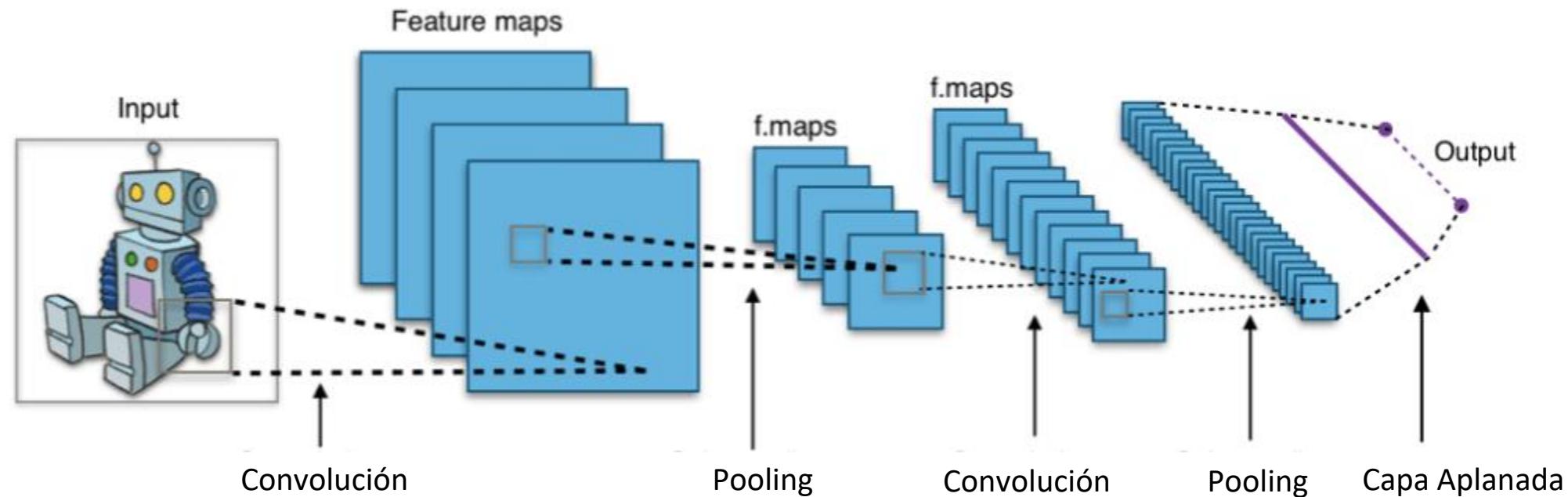
Una operación softmax tiene un propósito clave: asegurarse de que las salidas de la CNN sumen 1. Debido a esto, las operaciones softmax son útiles para **escalar las salidas del modelo en probabilidades**.

Quizás esté pensando cuál es la diferencia entre la normalización estándar y softmax; después de todo, ambos reescalan los logits entre 0 y 1. Recuerde que la retropropagación es un aspecto clave del entrenamiento de redes neuronales: queremos que la respuesta correcta tenga la "señal" más grande. Al usar softmax, estamos "aproximando" efectivamente a argmax mientras ganamos diferenciabilidad. El cambio de escala no pesa el máximo significativamente más que otros logits, mientras que softmax sí lo hace. En pocas palabras, softmax es un argmax "más suave".

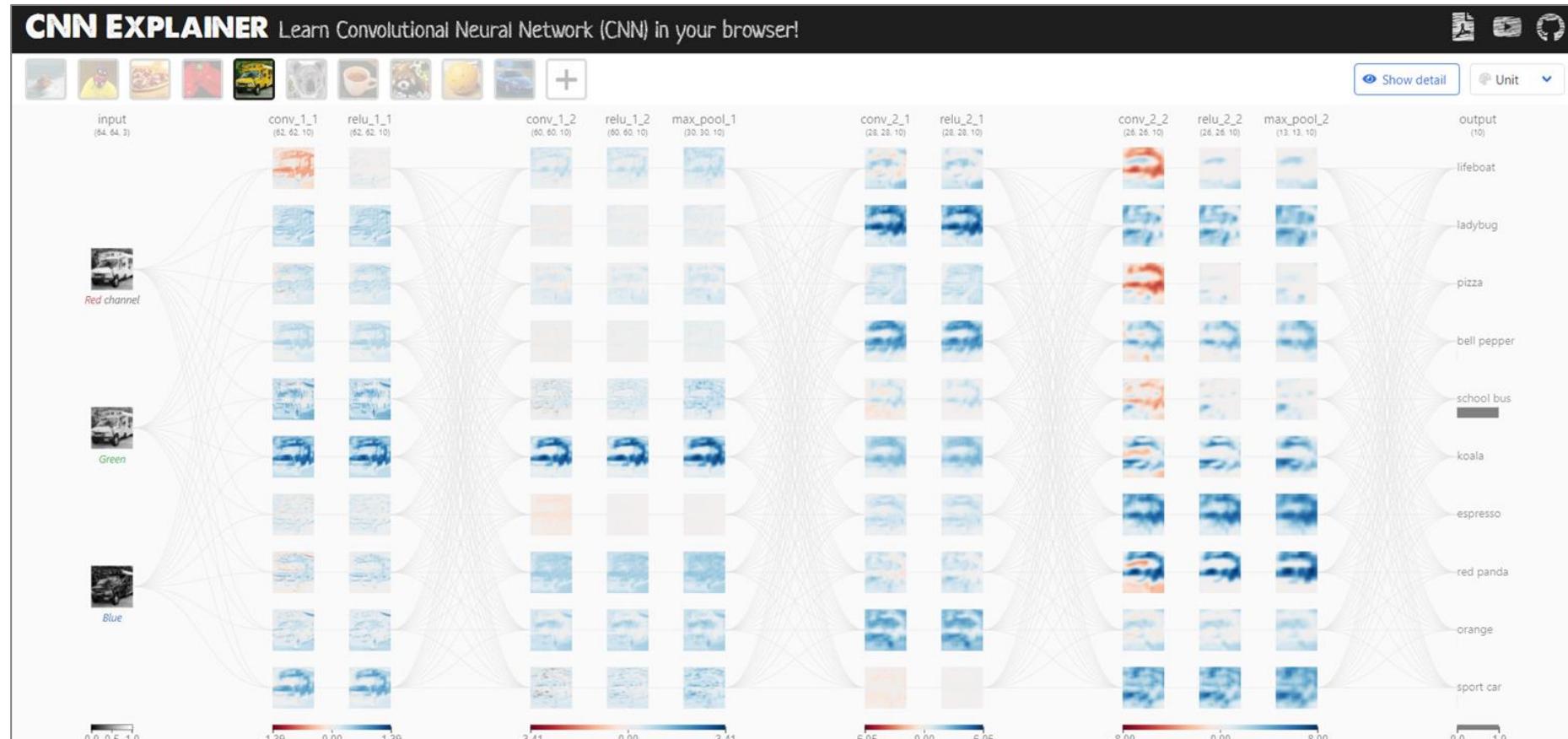
$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

# Arquitectura CNN

A continuación, un esquema completo de la arquitectura CNN:



# Playground



<https://poloclub.github.io/cnn-explainer/>

## Dudas y consultas



KIBERNUM



# Fin Presentación



KIBERNUM