

## Redes Neuronales Convolucionales

Las redes neuronales convolucionales son una arquitectura diseñada para trabajar con imágenes. La idea es ir "escaneando" la imagen de a partes.

Supongamos la siguiente imagen de  $6 \times 6$  pixeles:

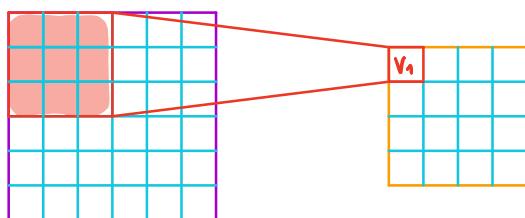
$x_1^1$	$x_1^2$	$x_1^3$	$\dots$	
$x_2^1$	$x_2^2$	$x_2^3$		
:	$\ddots$			

La idea es definir un **filtro** que va a recorrer toda la imagen. Supongamos que tenemos un filtro de  $3 \times 3$ . Este filtro tiene los pesos que se aprenden:

$w_1^1$	$w_1^2$	$w_1^3$
$w_2^1$	$\dots$	
:		

→ Filtro de  
 $3 \times 3$

Ahora, al pasar el filtro por la imagen lo que sucede es lo siguiente:

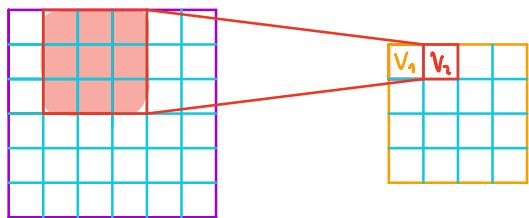


$v_1$  es igual al producto punto del filtro con el segmento analizado de la imagen

bias ↗

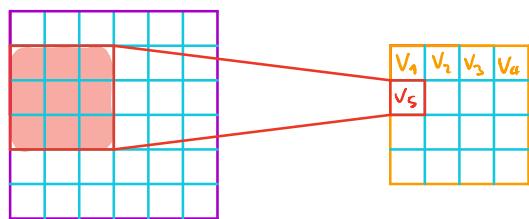
$$v_1 = w_1^1 \cdot x_1^1 + w_1^2 \cdot x_1^2 + w_1^3 \cdot x_1^3 + w_2^1 \cdot x_2^1 + w_2^2 \cdot x_2^2 + w_2^3 \cdot x_2^3 + w_3^1 \cdot x_3^1 + w_3^2 \cdot x_3^2 + w_3^3 \cdot x_3^3 + b$$

Así, vamos a generar como resultado una matriz de  $4 \times 4$  al ir desplazando el filtro:



$$V_2 = w_1^1 \cdot x_2^1 + w_1^2 \cdot x_3^1 + w_1^3 \cdot x_4^1 + \\ w_2^1 \cdot x_2^2 + w_2^2 \cdot x_3^2 + w_2^3 \cdot x_4^2 + \\ w_3^1 \cdot x_2^3 + w_3^2 \cdot x_3^3 + w_3^3 \cdot x_4^3 + b$$

Así, en algún momento tendremos:



Y continuamos hasta tener 16 valores.

Ojo! Aquí lo que estamos haciendo es pesar de una matriz de  $6 \times 6$  a otra de  $4 \times 4$ . Además al calcular los valores  $v_1, v_2, \dots, v_{16}$ , el producto punto más bias lo vamos a pesar por una función de activación (e.g. ReLU).

¿Y las imágenes a color?

Para trabajar con imágenes a color en general trabajamos con RGB (red, green, blue), por lo que vemos la

Imagen como una matriz de 3 dimensiones. Si superponemos una imagen de  $10 \times 10$ , una instancia para la red se vería así:

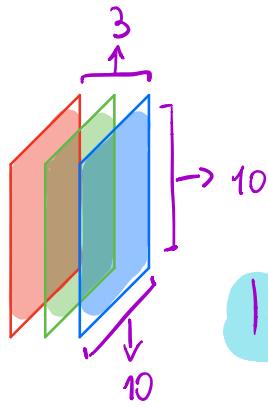
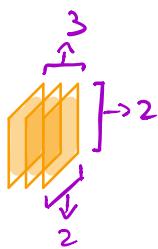


Imagen:  $10 \times 10 \times 3$

Y un filtro de  $2 \times 2$  en realidad se ve así:



Filtro:  $2 \times 2 \times 3$

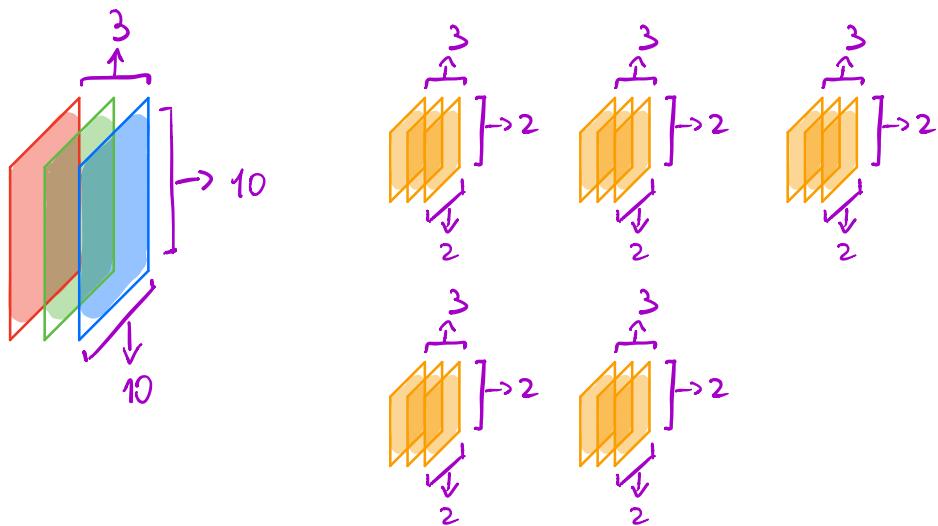
(12 pesos más un bias)

La operación que se hace sigue siendo un producto punto, pero en 3 dimensiones. Así, con la imagen y el filtro del ejemplo, generamos una matriz de  $9 \times 9 \times 1$ .

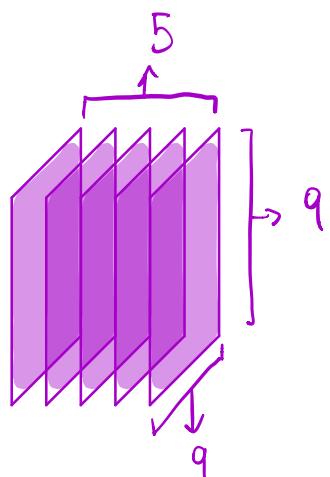
### Agregando más filtros

La idea de un filtro es que aprende a reconocer algo específico de la imagen, por ejemplo, líneas verticales, horizontales, bordes curvos, etc. Así, una **Capa Convolucionel** se compone de varios filtros, y cada uno debería descubrir cosas distintas.

Por ejemplo, si tenemos la misma imagen de  $10 \times 10$  y 5 filtros de  $2 \times 2$ :



Cada filtro ve e genera una matriz de  $9 \times 9 \times 1$ . Entonces el resultado de la capa convolucional (de 5 filtros de  $2 \times 2$ ) es una matriz de  $9 \times 9 \times 5$ .



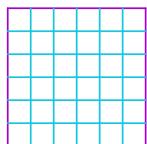
Aquí, cada matriz de  $9 \times 9$  viene de un filtro

Ojo! La capa convolucional anterior tiene que aprender  $2 \times 2 \times 3 \times 5 = 60$  pesos y 5 bias.

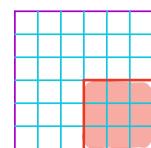
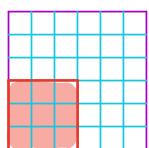
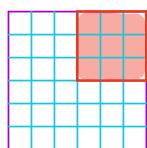
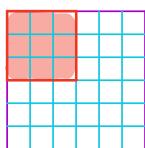
### Zero padding y Stride

Dos elementos importante de las capas convolucionales son el zero padding y el stride. Vamos a explicar ambos.

**Stride:** en los ejemplos supusimos que el filtro se movía de uno en uno. El "stride" es un hiperparámetro que controla el tamaño del salto. Supongamos una imagen de  $6 \times 6$ :



Si consideramos un filtro de  $3 \times 3$  y un stride igual a 3, el filtro hará el siguiente recorrido:



El salto fue de 3 en 3

Y se generará una matriz de  $2 \times 2$ .

**Padding:** en los ejemplos que hemos visto, la capa convolucionel disminuye el alto y largo de la matriz. La técnica de zero padding nos permite no disminuir el alto y largo del output agregando 0s al borde de la imagen. Supongemos una imagen de  $4 \times 4$  y un filtro de tamaño  $3 \times 3$ . El resultado será una matriz de  $2 \times 2$ , pero al agregar un padding de tamaño 1:

A diagram showing a 4x4 grid of zeros. To its right is a vertical column of zeros, labeled "padding". An arrow points from the text "Imagen de 4x4 más una columna de padding." to this column. The entire 5x5 grid is labeled "Imagen de 4x4 más una columna de padding."

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Ahora, el resultado del filtro de  $3 \times 3$  será una matriz de  $4 \times 4$ .

A modo de resumen, si una capa convolucionel tiene:

- K filtros
- Los filtros son de  $F \times F$
- El stride es de tamaño S
- El padding es de tamaño P

- La imagen es de  $W_1 \times H_1 \times D_1$

Entonces, el output será de tamaño  $W_2 \times H_2 \times D_2$  con:

$$- W_2 = \frac{(W_1 - F + 2P)}{S} + 1$$

$$- H_2 = \frac{(H_1 - F + 2P)}{S} + 1$$

$$- D_2 = K$$

Ojo! Los filtros pueden no ser cuadrados, pero por simplicidad lo expresamos así.

## Pooling

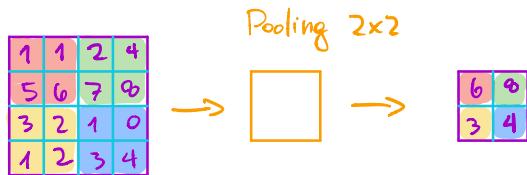
Las capas de pooling suelen venir después de las capas convolucionales para reducir dimensiones y para evitar el overfitting. Funcionan así:

Supongamos la siguiente imagen con los siguientes valores por pixel:

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Si instanciamos un filtro de pooling de tamaño  $2 \times 2$  y stride 2, el filtro se moverá como vimos antes, pero el valor retornado

no es el producto punto, sino el elemento máximo:



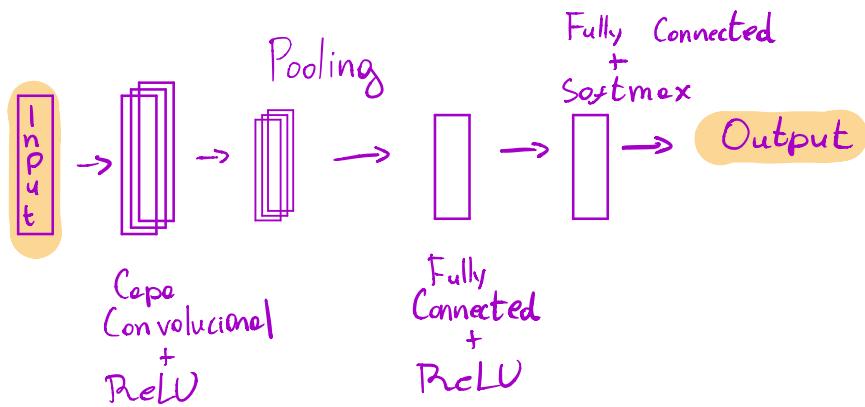
Así, si tenemos una matriz de  $224 \times 224 \times 64$  (que posiblemente es el resultado de una capa convolucional) y la pasemos por una capa de pooling con un filtro de  $2 \times 2$  y stride 2, el resultado es de tamaño  $112 \times 112 \times 64$ , porque hicimos "pooling" de cada una de las 64 matrices de  $224 \times 224$ .

Ojo! Hay más funciones, aparte de `max(·)`, para hacer pooling, como por ejemplo, sacar el promedio.

### Categorizando imágenes

Ahora, para clasificar una imagen, lo que hacemos es conectar el output de una capa convolucional (+ pooling) a una capa fully connected. Esto es, los elementos de la matriz

tridimensional son pasados por una capa de flatten (por ej. una matriz de  $3 \times 4 \times 5 = 60$  queda como una instancia de 60 features) y se conectan a una capa fully connected. Luego, para clasificar se conecta la capa fully connected a una capa de output.



Ojo! para encontrar los pesos y bias de las capas fully connected y de los filtros de la capa convolucional usamos Backpropagation.