

Foundations of Convolutional Neural Networks

Desafíos que se encuentran en Computer Vision:

Image Classification



Neural Style Transfer



Object detection

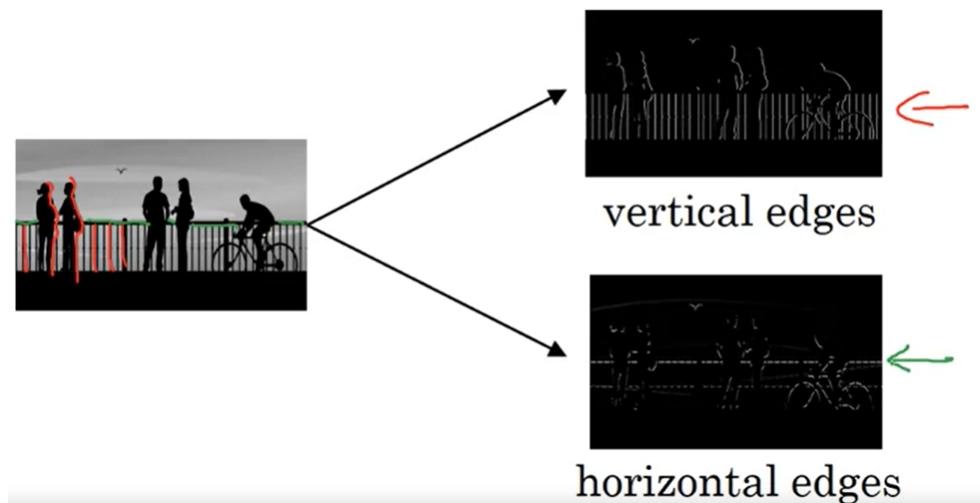


Andrew Ng

El problema de tratar imágenes con redes totalmente conectadas (fully connected) es que si se trabaja con imágenes grandes (1000x1000x3), la entrada tendría 3.000.000 elementos, y suponiendo que la primera capa tuviera 1000 neuronas, tendría $3.000.000 * 1000$ parámetros solamente en la primera capa.

- **Edge detection:**

Las primeras capas de una red neuronal detecta bordes, luego detectan partes de objetos, y por último detectan objetos enteros. Por ejemplo, para detectar objetos en una imagen, primero se detectan los bordes horizontales y verticales.



Para detectar ejes, debo convolucionar la imagen con un filtro. La disposición de los números en la matriz de filtro es la que determina qué tipo de eje se desea filtrar.

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6

"convolution"

*

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

3x3
filter

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4x4

Nota: En Python se puede utilizar conv_forward. En Tensorflow se usa tf.nn.conv2d y en keras se usa Conv2D.

Acá hay un ejemplo más clarificador sobre por qué ese filtro obtiene los bordes verticales

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6x6

↓ ↓

*

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

3x3

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

↑
↓

Andrew Ng

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

Vertical

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

Horizontal

La idea general es que las redes neuronales convolucionales aprenden los valores de estos filtros en función de las imágenes del entrenamiento.

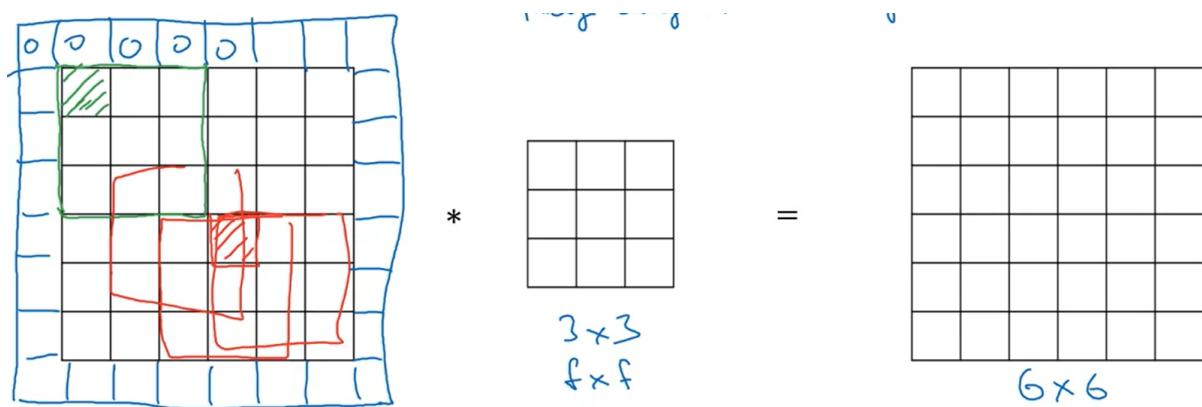
En la convolución, si tengo una imagen de $n \times n$ y un filtro de $f \times f$, el resultado de la convolución será:

$$(n + 2p - f + 1) \times (n + 2p - f + 1)$$

Un ejemplo puede ser en las imágenes pasadas, tenía una matrix de 6×6 y un filtro de 3×3 por lo tanto el resultado de la convolución será de $(6-3+1) \times (6-3+1) = 4 \times 4$.

• Padding

Un problema de filtrar las imágenes de esta forma, es que los píxeles de los bordes no influyen tanto en el resultado final, ya que los píxeles en los bordes pasan por mucho menos convoluciones que los píxeles en el medio de la imagen. Para evitar esto, se realiza un pad de la imagen. Esto quiere decir que se le agregan ceros en los bordes.



Ahora, el resultado tendrá un tamaño de:

$$(n + 2p - f + 1) \times (n + 2p - f + 1)$$

donde p es la cantidad de filas/columnas se agregan por lado.

• Valid and Same convolutions

En las convoluciones del tipo Valid, se realiza la operación como vimos anteriormente, pero en las convoluciones del tipo Same, se paddea la entrada automáticamente para que la salida tenga el mismo tamaño que la entrada. El pad que aplica será:

$$p = \frac{f-1}{2}$$

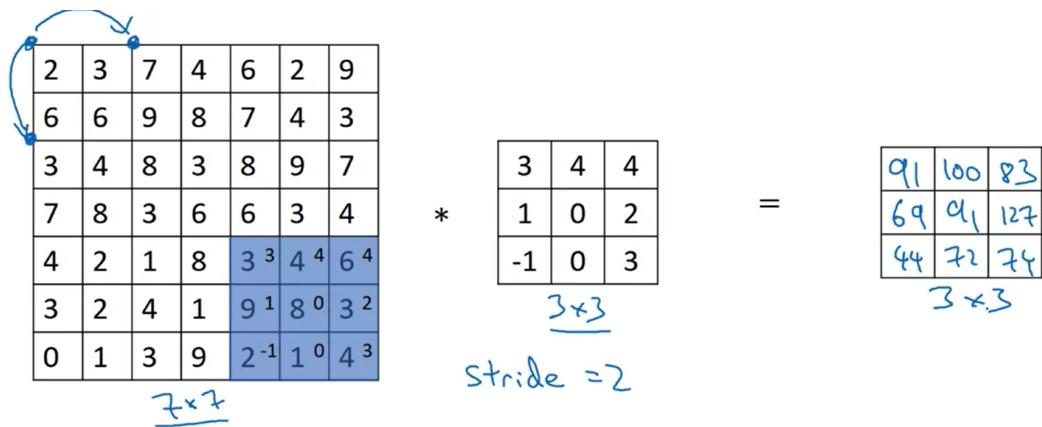
Por lo general, f es un número impar.

• Stride

En las convoluciones anteriores, el solapamiento del filtro sobre el input se corría de 1 fila a la vez, y luego 1 columna a la vez. Este salto (stride) puede ser de mas de una fila/columna. Esto modificará el tamaño de salida considerablemente, ya que se realizan muchas menos operaciones.

Tomando en cuenta el padding, la dimensión de salida será:

$$(\frac{(n+2p-f)}{s} + 1) \times (\frac{(n+2p-f)}{s} + 1)$$



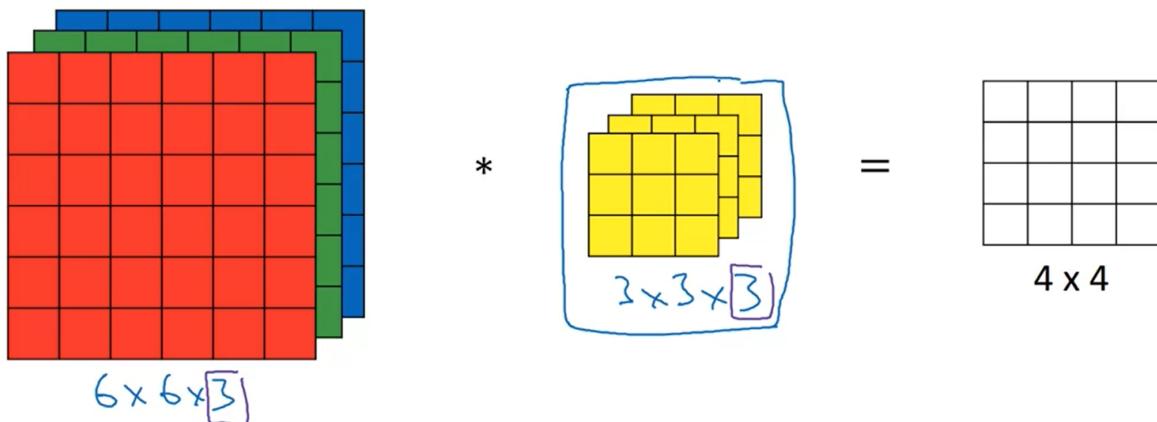
Si el tamaño no da un entero, se toma el entero menor.

• Convolutions over volume

Hasta ahora trabajamos con imágenes en 2D, pero en realidad, las imágenes RGB tienen 3 dimensiones, una por cada color (Red, Green, Blue). Por lo tanto las entradas serán de:

$$n \times n \times nc$$

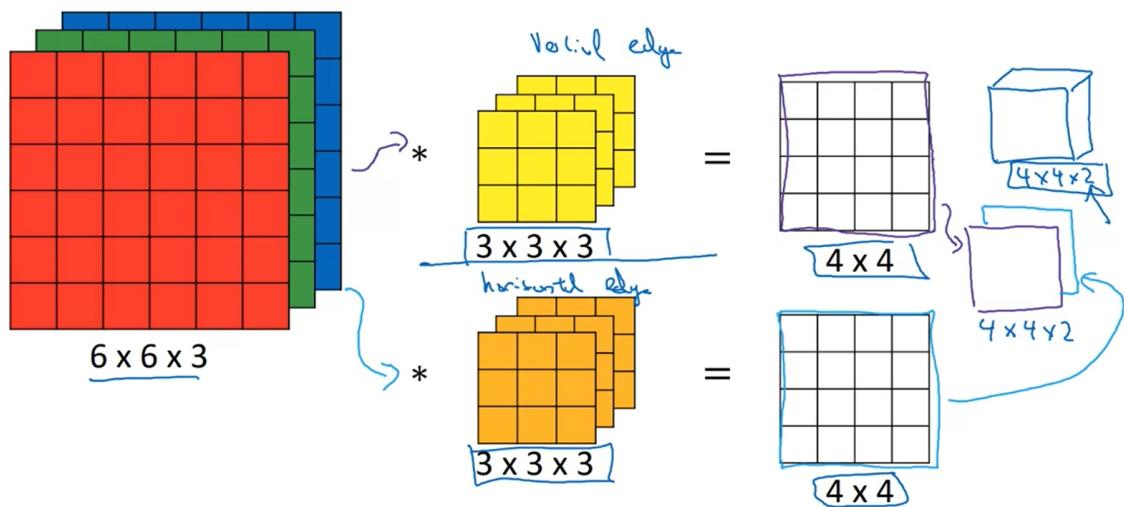
donde h es la altura, w el ancho y ch la cantidad de canales o colores (siempre son 3).



Las dimensiones de salida siguen estando asociadas por las ecuaciones vistas anteriormente, solo que se hace el cálculo por dimensión. Ya que el filtro debe tener la misma cantidad de canales que la entrada, la salida tendrá

$$(n - f + 1) \times (n - f + 1) \times (nc - nc + 1) = (n - f + 1) \times (n - f + 1) \times (1)$$

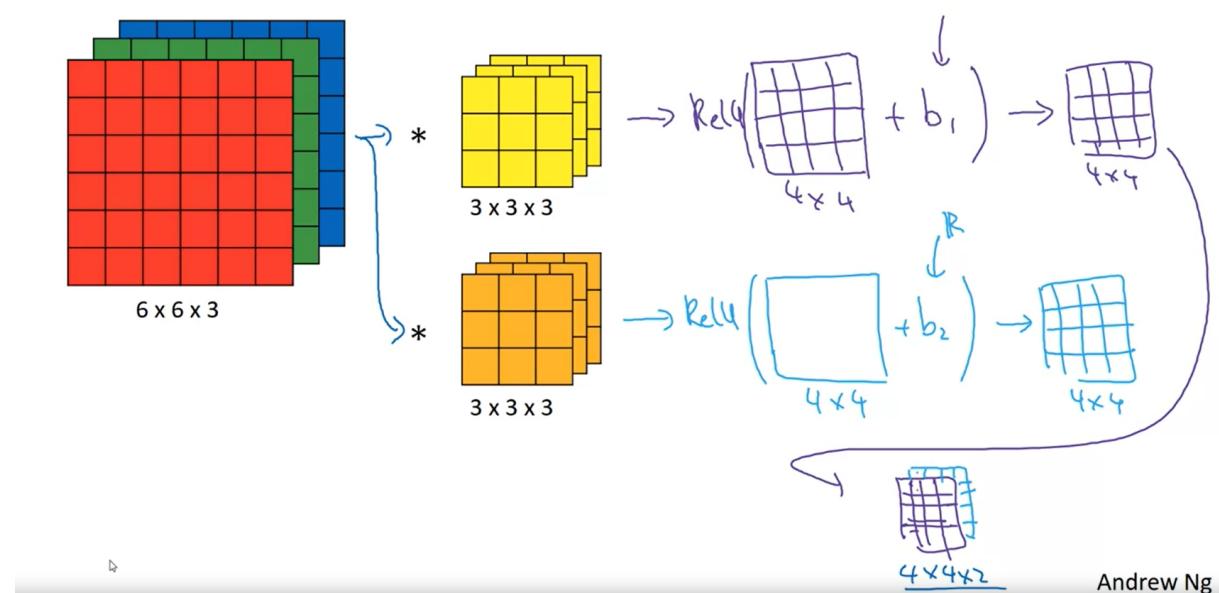
A su vez, también se pueden aplicar varios filtros a la imagen (uno a la vez) y apilarlos en la matriz resultante



Por lo tanto, el resultado de múltiples filtrados será:

$$(n - f + 1) \times (n - f + 1) \times (\#filters)$$

- One layer of a CNN



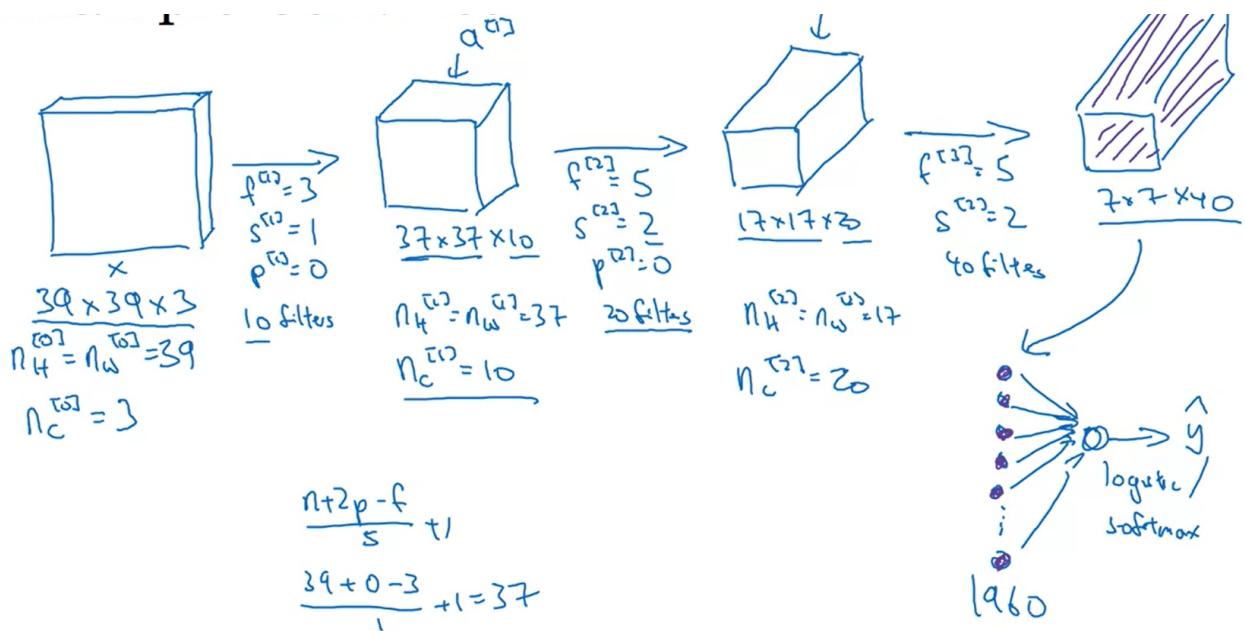
La cantidad de pesos será:

$$f \cdot f \cdot nc \cdot \#filters$$

Y contando el bias por filtro, la cantidad de parámetros que hay en una capa convolucional es de:

$$f \cdot f \cdot nc \cdot \#filters + \#filters$$

• Simple Convolutional Network Example

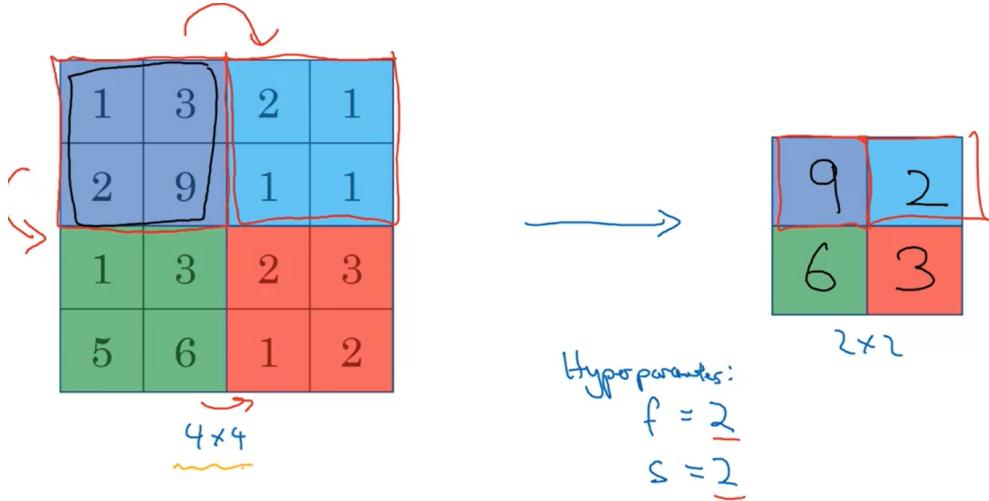


Por lo general, hay 3 tipos de capas en una CNN:

- Convolution (CONV)
- Pooling (POOL)
- Fully connected (Fc)

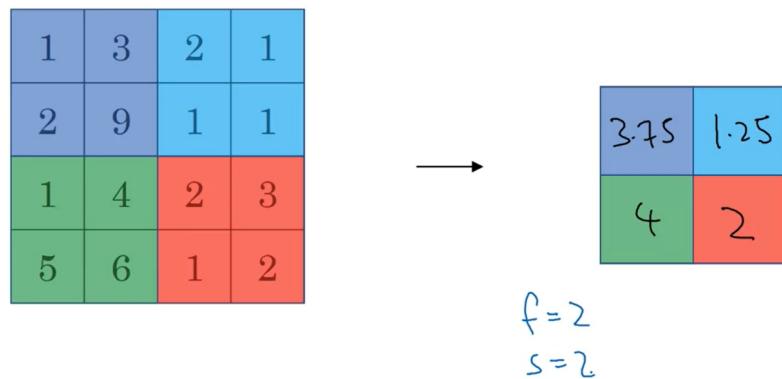
• Pooling layer

En las pooling layers se suele utilizar el Max pooling, que es básicamente particionar la entrada en submatrices, y tomar el máximo de cada submatriz. Los hiperparámetros de estas capas son el **tamaño de las submatrices (f)**, y el **salto que se realiza entre submatrices (stride (s))**

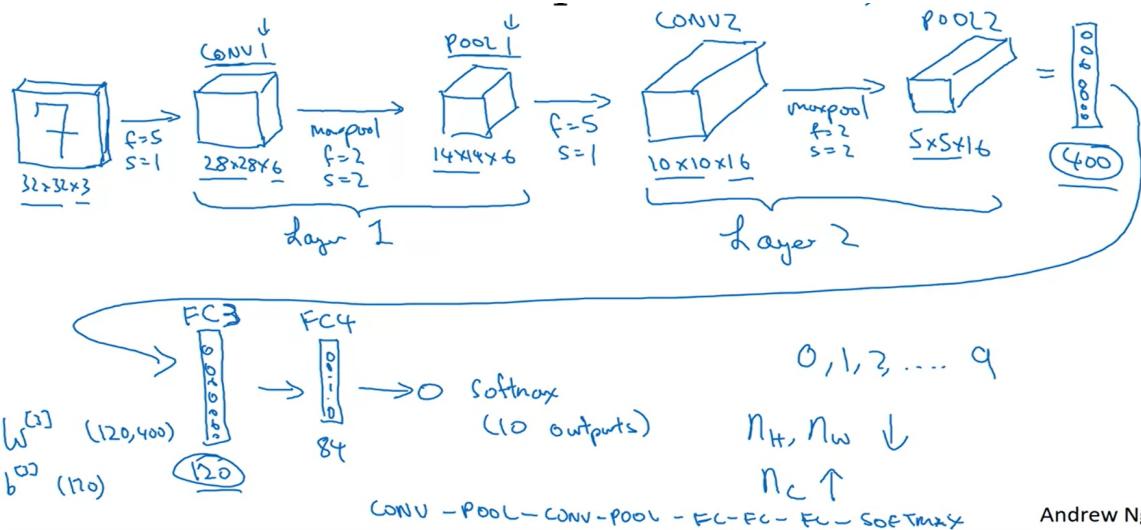


Esta operación reduce la dimensionalidad de la entrada también. Algo interesante es que este tipo de capa no tiene parámetros que se actualizan o que la red aprende, si no que simplemente tiene hiper parámetros (f y s).

El otro método de pooling es el average pooling, el cual en vez de tomar el máximo, toma el promedio de los valores por submatriz.



Ejemplo: LeNet-5



	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 <i>a^{3x3}</i>	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208 ↙
POOL1	(14,14,8)	1,568	0 ↙
CONV2 (f=5, s=1)	(10,10,16)	1,600	416 ↙
POOL2	(5,5,16)	400	0 ↙
FC3	(120,1)	120	48,001 ↘
FC4	(84,1)	84	10,081 ↘
Softmax	(10,1)	10	841

• Why Convolutions?

Reducen muchísimo la cantidad de parámetros que la red debe actualizar para aprender, por lo cual evita el overfitting que podría tener si se utiliza una Fully Connected.

Características importantes:

- **Parameter sharing:** Un detector de características (como lo es un detector de bordes verticales) que es útil en una parte de la imagen probablemente sea útil en otra parte de la imagen
- **Sparsity of connections:** En cada capa, cada elemento de salida depende solamente de pocos elementos de entrada (cosa que en una red fully connected no sucede, cada salida depende de todas las entradas).