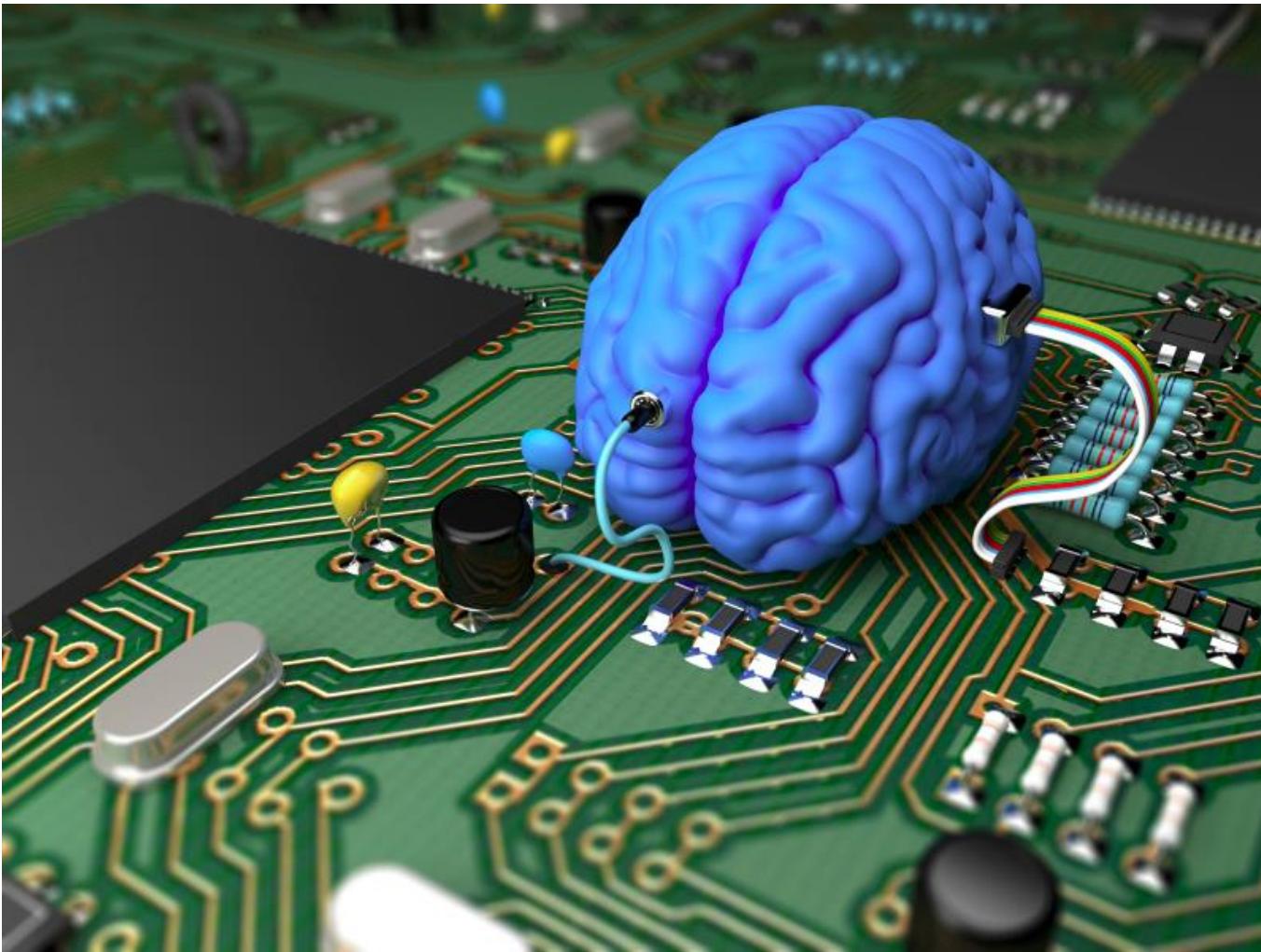


# Machine learning I : ConvNets



# En capítulos anteriores

- Como dividir el dataset de entrenamiento
- Revisión de varianza y sesgo
- Regularización: L2, dropout, data augmentation
- Minibatch y stochastic gradient descent
- Algoritmos de optimización: Momento, RMSProp y Adam

# Visión Artificial

- Las redes convolucionales han revolucionado el campo de la visión artificial.



Clasificación



Detección de objetos



Transferencia de estilo

# Densas vs Convolucionales

La principal diferencia entre una red neuronal *densa* y una red *convolucional*:

con le reti neuronali dense si perde la correlazione tra i pixel, ad esempio in un'immagine con 9 pixel (3x3) questi diventano i neuroni di entrata ma si perde la struttura

- Las redes densas aprenden **patrones globales** del espacio de características de input. En el caso de las imágenes utiliza **todos los pixels de la imagen**.
- Las redes convolucionales aprenden **patrones locales**. En el caso de las imágenes estos patrones los encuentra en **pequeñas ventanas (filtros) en 2D**.

# Principales características de las ConvNets

- **Los patrones que aprenden son invariante a translaciones**

- Si reconoce un cierto patrón en la esquina derecha de una imagen, la ConvNet lo encontrará en cualquier otro sitio
- Una red densa tendría que aprender el patrón de nuevo si apareciera en otra posición

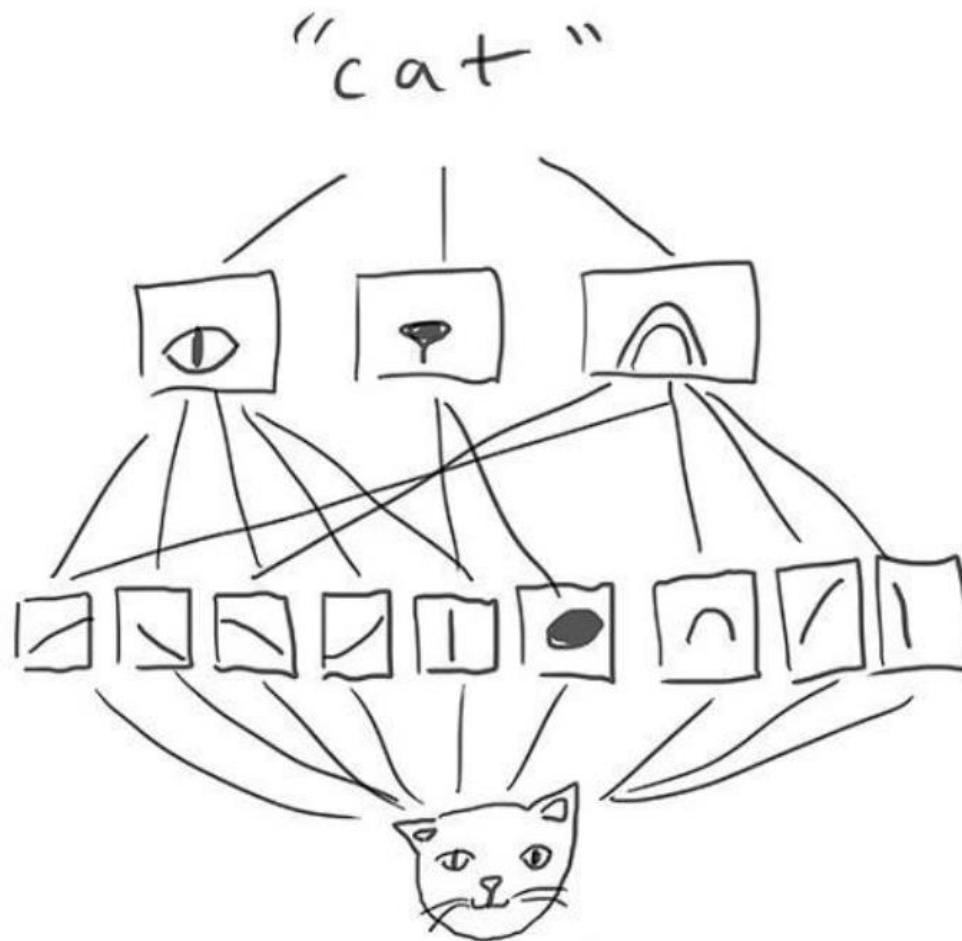
Por este motivo, las ConvNets:

- Muy eficientes para procesar imágenes
- Menos imágenes para entrenar ya que tienen mayor poder de generalización

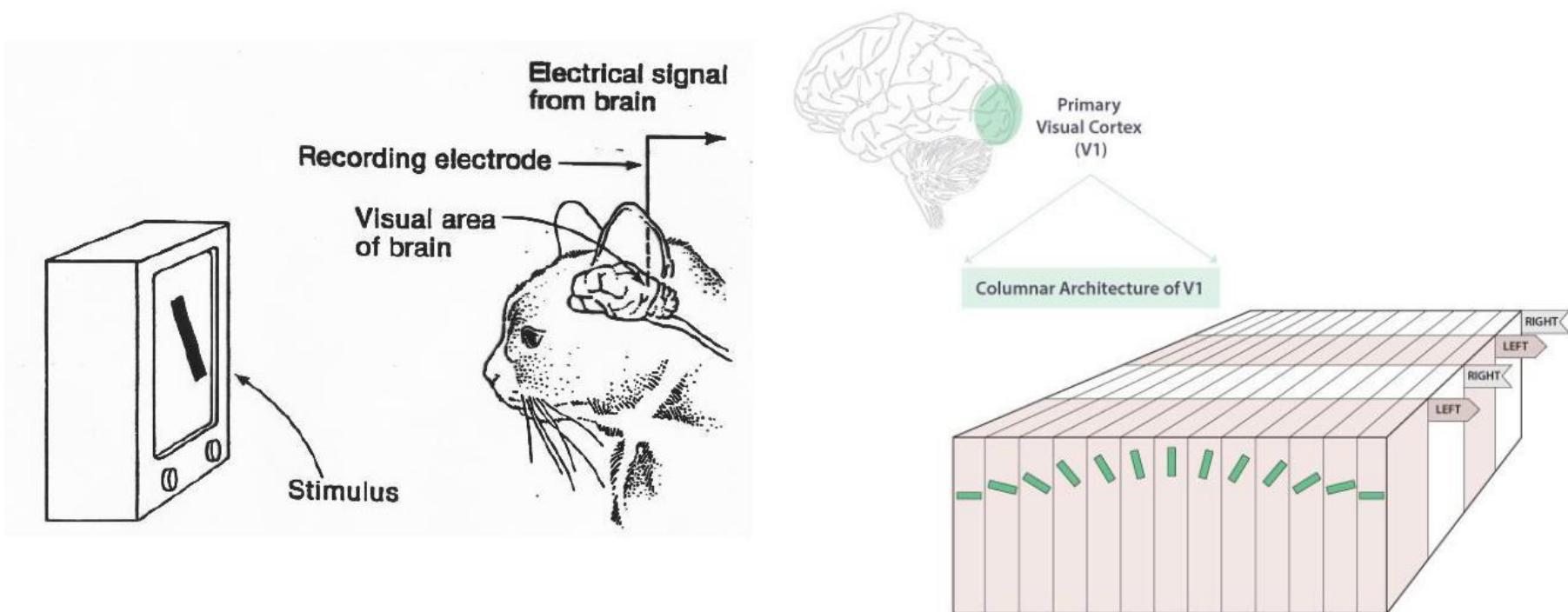
- **Pueden aprender la jerarquía espacial de las imágenes**

- La primera capa convolucional aprenderá a extraer esquinas y bordes
- La segunda los combinará para crear conceptos más complejos

# Jerarquía de características



# Experimento de Hubel y Wiesel



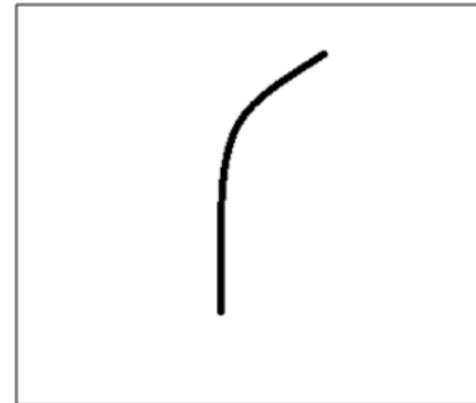
© Knowing Neurons <http://knowingneurons.com>

- Dos cosas en común con las redes usadas en Computer Vision:
  - Neuronas (filtros) especializados en orientaciones (representaciones) determinadas
  - Complejidad creciente para crear la representación final

# Capa Convolucional

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

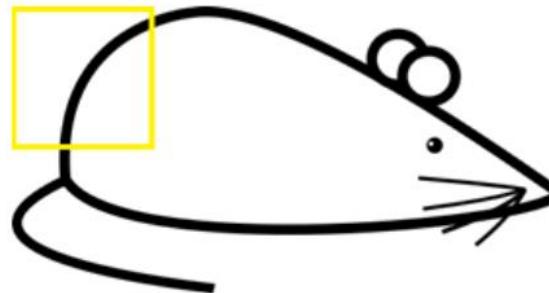
Pixel representation of filter



Visualization of a curve detector filter

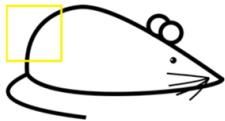


Original image



Visualization of the filter on the image

# Capa Convolucional



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

\*

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

Pixel representation of filter

$$\text{Multiplicación y suma} = (50*30) + (50*30) + (50*30) + (20*30)+(50*30) = 6600$$

# Capa Convolucional



Visualization of the filter on the image

0	0	0	0	0	0	0	0
0	40	0	0	0	0	0	0
40	0	40	0	0	0	0	0
40	20	0	0	0	0	0	0
0	50	0	0	0	0	0	0
0	0	50	0	0	0	0	0
25	25	0	50	0	0	0	0

Pixel representation of receptive field

\*

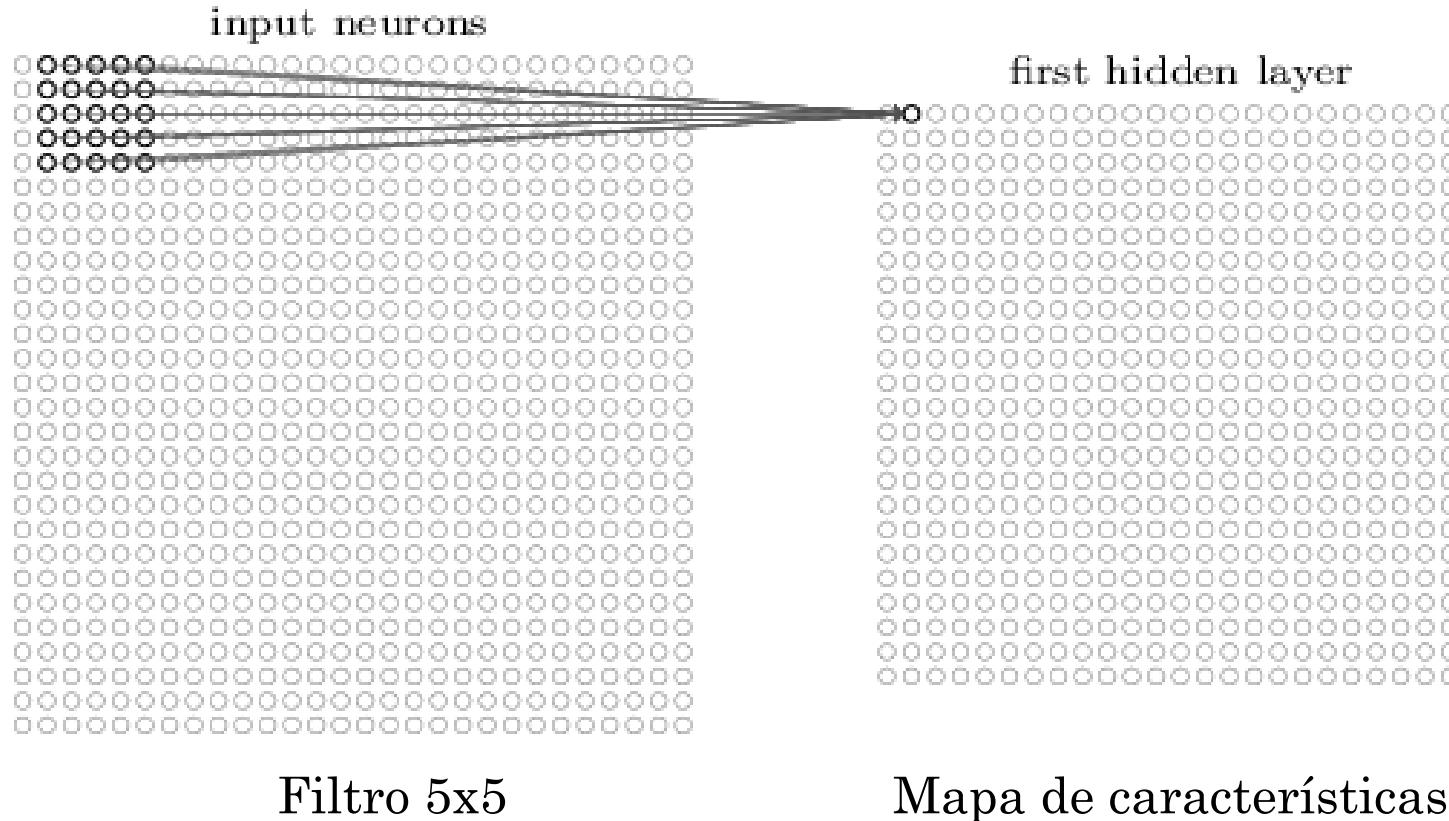
0	0	0	0	0	0	30	0
0	0	0	0	30	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

Multiplicación y suma = 0

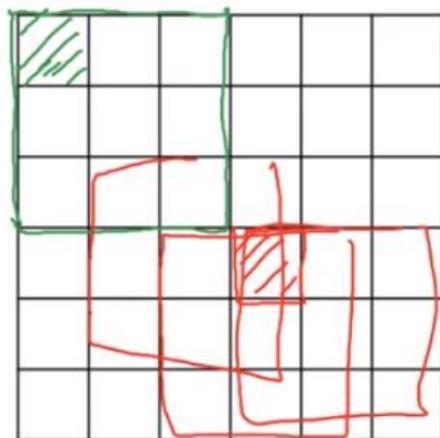
# Capa Convolucional

Imagen en blanco y negro

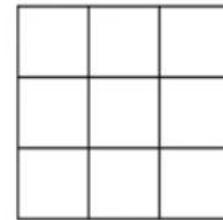


# Padding

- Uno de los problemas de las capas convolucionales es que pueden hacer que la imagen mengüe demasiado.
- Si tienes muchas capas la imagen va a acabar siendo muy pequeña.
- Otro problema es que la información de los píxeles de las esquinas y los bordes van a estar “infra-representados” en los mapas de características

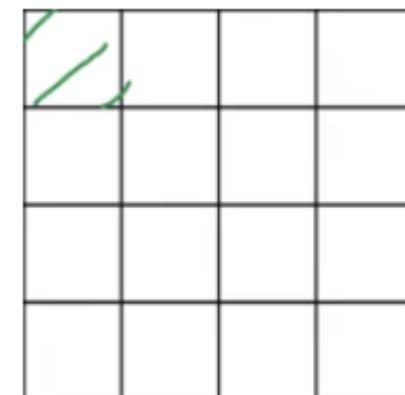


\*



$3 \times 3$   
 $f \times f$

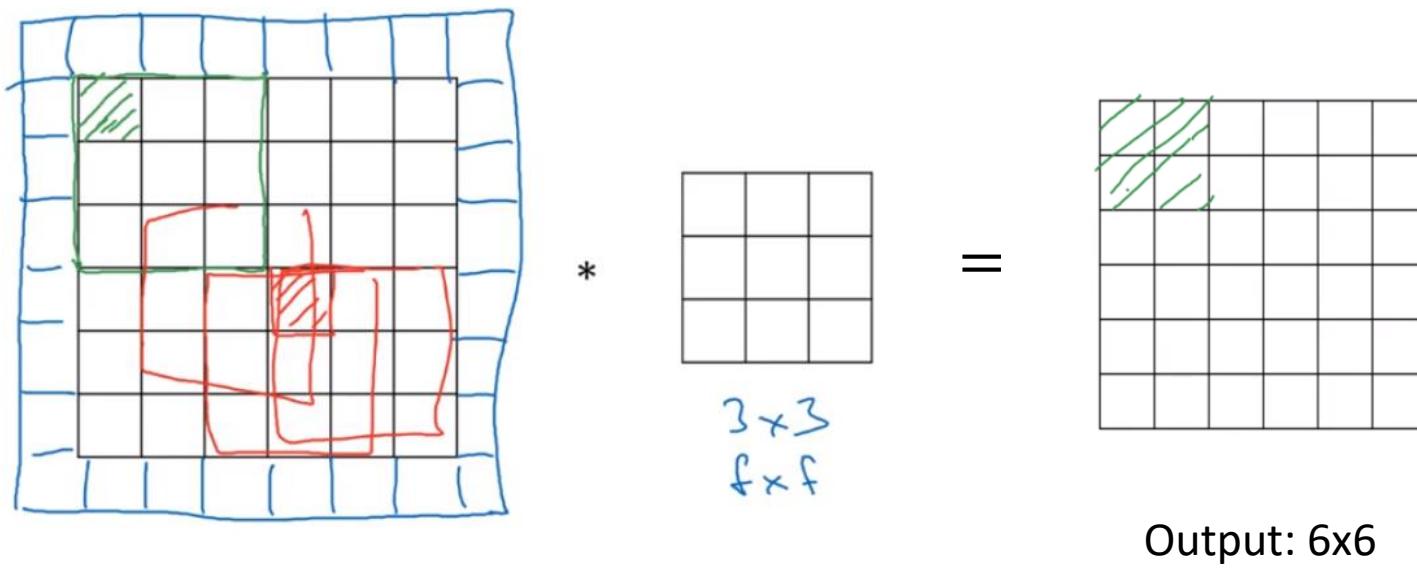
=



Output: 4x4

# Padding

$p=1$



# Padding

- “Valid” convolution: Sin padding

$$n \times n * f \times f \rightarrow n - f + 1 \times n - f + 1$$

$$6 \times 6 * 3 \times 3 \rightarrow 4 \times 4$$

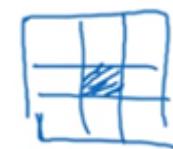
- “Same” convolution: Hacer padding hasta que el input y el output sean del mismo tamaño

$$n \times n * f \times f \rightarrow n + 2p - f + 1 \times n + 2p - f + 1$$

$$n + 2p - f + 1 = n \rightarrow p = \frac{f-1}{2}$$

El tamaño de los filtros ( $f$ ) se suele elegir impar: 3x3, 5x5, etc...

Además eso permite tener un pixel central



# Strided convolution

- En lugar de ir moviendo el filtro en pasos de 1 píxel lo hacemos en pasos de  $s$  pixels

$n \times n$  image       $f \times f$  filter

padding  $p$

stride  $s$

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \quad \times \quad \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

Si no es un número entero se redondea  
al entero más cercano por abajo

# Capa Convolucional

¿Y si la imagen es en color?

## Filtro 0

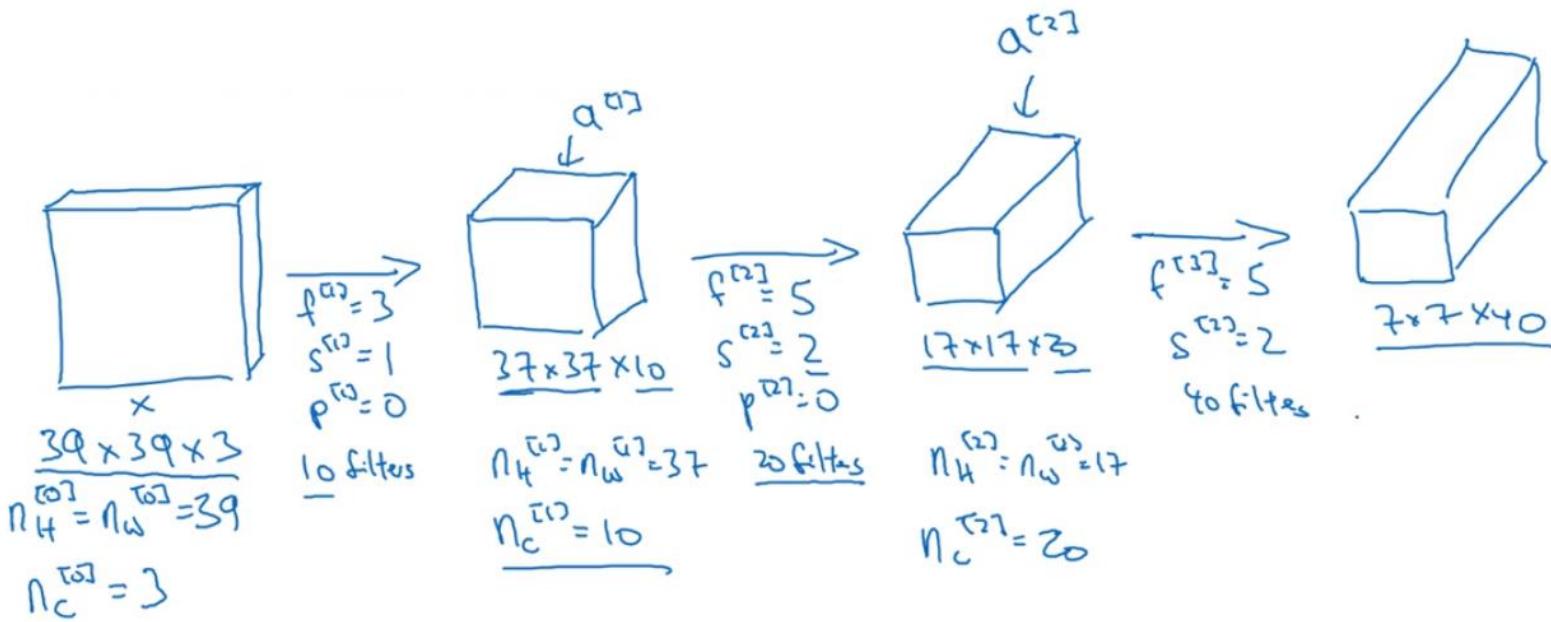
Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)	Output Volume (3x3x2)
$x[:, :, 0]$	$w0[:, :, 0]$	$o[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -4 \\ -1 & -2 & -5 \\ 3 & 1 & -1 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 2 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & -1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & -1 \end{bmatrix}$
$x[:, :, 1]$	$w0[:, :, 1]$	$o[:, :, 1]$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & -2 \\ 4 & 7 & -3 \\ 2 & 2 & 2 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$	$b0[:, :, 0]$	$b0[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$b1[:, :, 0]$	$b1[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		
$x[:, :, 2]$		
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$		
$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		

28 parámetros ( $3 \times 3 \times 3 + 1$ )

## Filtro 1

Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)	Filter W1 (3x3x3)	Output Volume (3x3x2)
$x[:, :, 0]$	$w0[:, :, 0]$	$w1[:, :, 0]$	$o[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -4 \\ -1 & -2 & -5 \\ 3 & 1 & -1 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 2 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & 1 \\ 0 & -1 & 0 \\ -1 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & -2 \\ 4 & 7 & -3 \\ 2 & 2 & 2 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -4 \\ -1 & -2 & -5 \\ 3 & 1 & -1 \end{bmatrix}$
$x[:, :, 1]$	$w0[:, :, 1]$	$w1[:, :, 1]$	$o[:, :, 1]$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 1 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & -2 \\ 4 & 7 & -3 \\ 2 & 2 & 2 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & -2 \\ 4 & 7 & -3 \\ 2 & 2 & 2 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -4 \\ -1 & -2 & -5 \\ 3 & 1 & -1 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -4 \\ -1 & -2 & -5 \\ 3 & 1 & -1 \end{bmatrix}$
$x[:, :, 2]$	$w0[:, :, 2]$	$w1[:, :, 2]$	$o[:, :, 2]$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & -2 \\ 4 & 7 & -3 \\ 2 & 2 & 2 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & -2 \\ 4 & 7 & -3 \\ 2 & 2 & 2 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -4 \\ -1 & -2 & -5 \\ 3 & 1 & -1 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & -4 \\ -1 & -2 & -5 \\ 3 & 1 & -1 \end{bmatrix}$
$x[:, :, 3]$	$b0[:, :, 0]$	$b1[:, :, 0]$	$b0[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$	$b0[:, :, 0]$	$b1[:, :, 0]$	$b0[:, :, 0]$
$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$	$b0[:, :, 0]$	$b1[:, :, 0]$	$b0[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$b0[:, :, 0]$	$b1[:, :, 0]$	$b0[:, :, 0]$
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$b0[:, :, 0]$	$b1[:, :, 0]$	$b0[:, :, 0]$

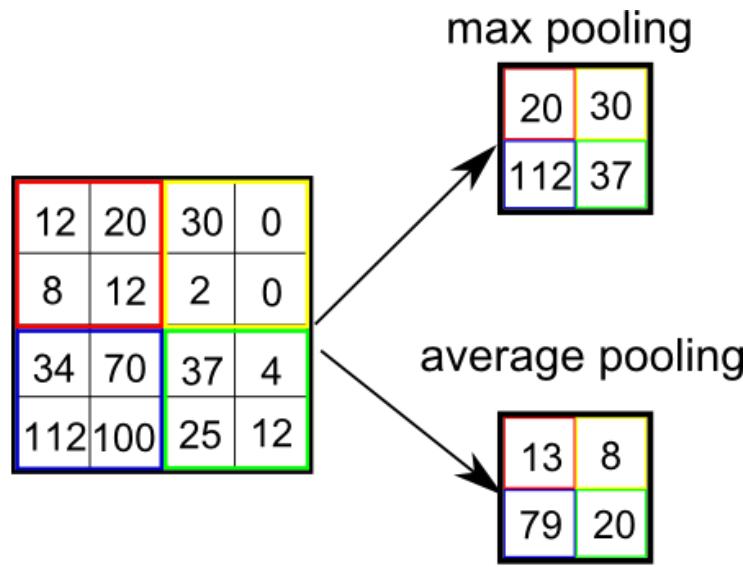
# Ejemplo



- El plano frontal del paralelepípedo depende de las variables de convolución (p,s,f...)
- La profundidad es igual al tamaño del filtro en cada capa

# Capa de Pooling

- La capa de pooling sirve para reducir la dimensionalidad de los mapas de características



## Hiperparámetros:

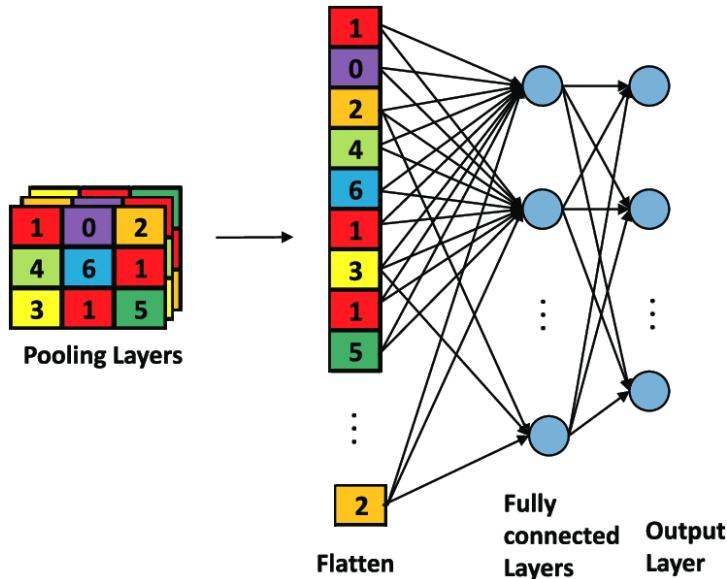
- Tamaño del filtro ( $f$ )
- Stride ( $s$ )
- Average o Max
- Padding ( $p$ )

Típicamente:  $f=2$ ,  $s=2$

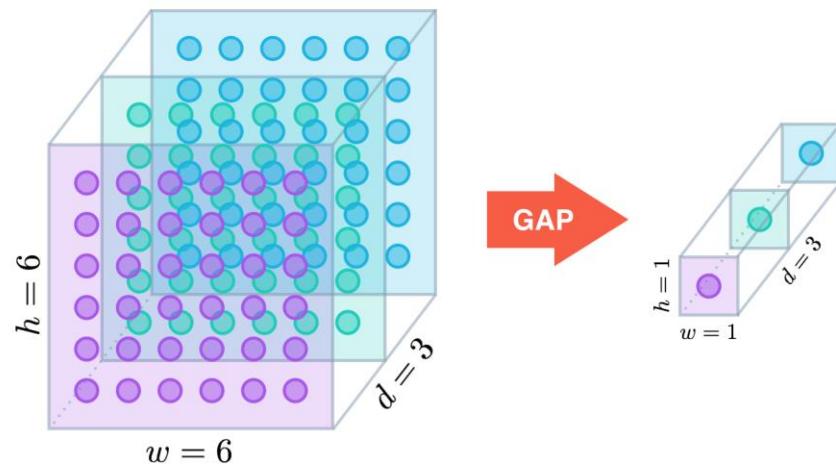
- El pooling no tiene ningún parámetros para aprender

# ¿Cómo pasar de la parte convolucional a la clasificación?

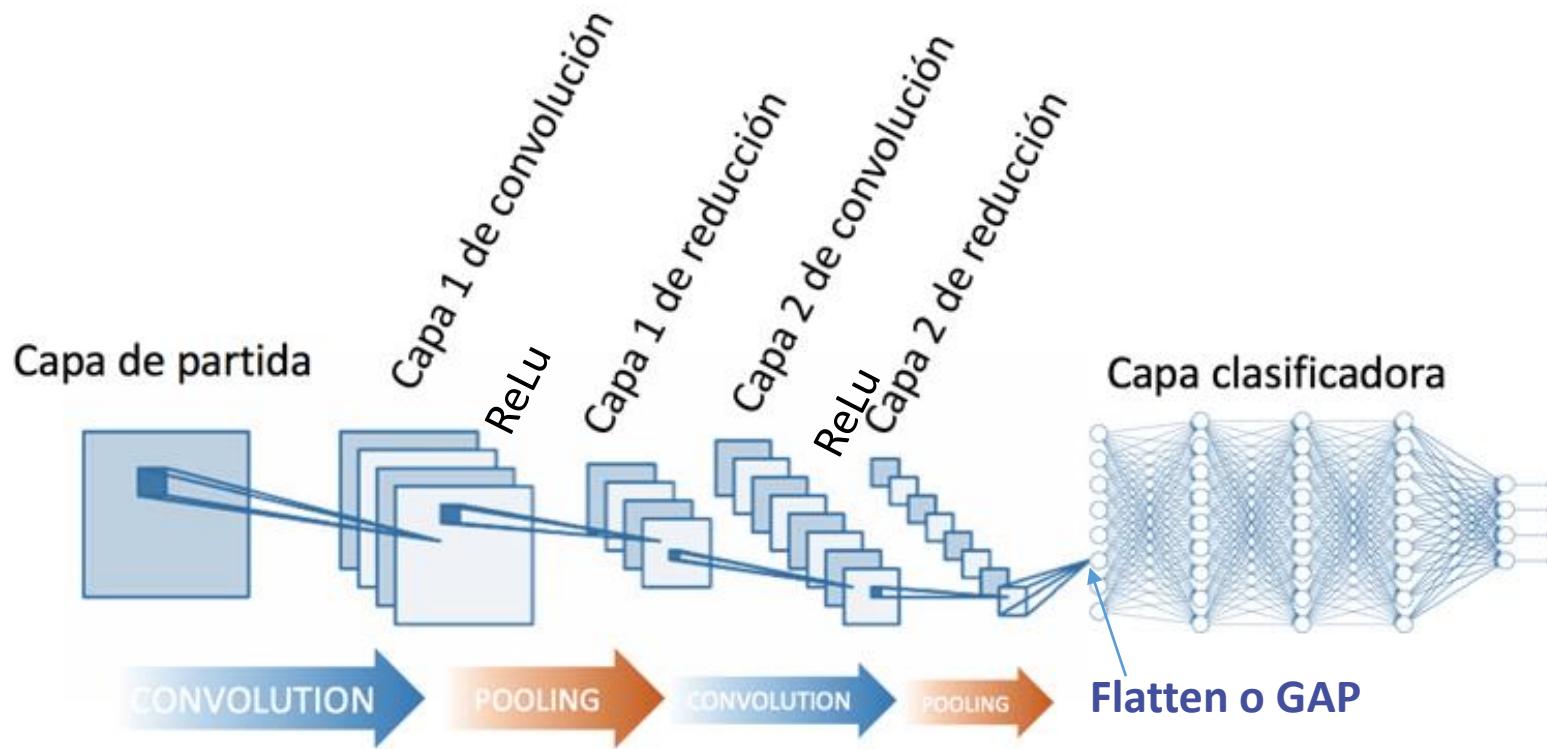
Flatten



Global Average Pooling (GAP)



# Juntándolo todo...



Vamos a contar como **una capa** el conjunto de **convolución + reducción** (hay autores que consideran que son dos capas) ya que solo la capa de convolución tiene parámetros para aprender

# Ejercicio

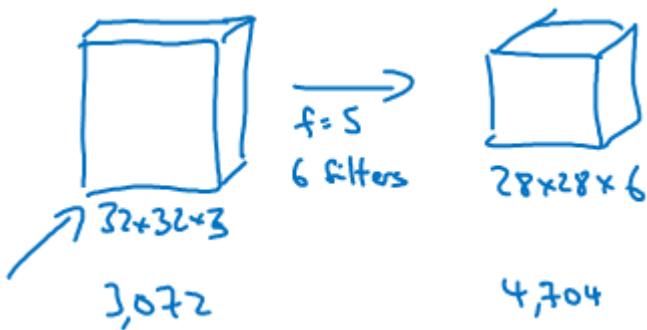
simile esame

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))) ((3x3)x1)x32 + 32
model.add(layers.MaxPooling2D((2, 2))) non apprende nessun parametro perché calcola semplicemente il massimo,
riduzione dimensionalità
model.add(layers.Conv2D(64, (3, 3), activation='relu')) entrano, dal precedente, 32 mappe di caratteristiche, 64 filtri
((3x3)x32)x64 + 64
l'1 è sempre il bias
il filtro ha lo stesso numero di canali dell'entrata
```

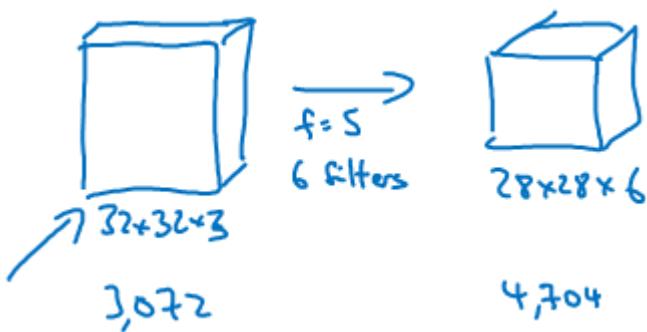
immagine di 28x28 pixel, 1 matrice, cioè bianco e nero

¿Cuántos parámetros hay en cada capa?

# ¿Por qué convolución?

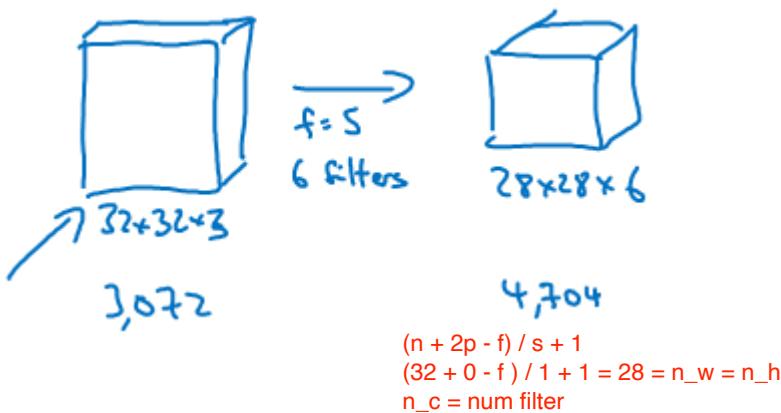


# ¿Por qué convolución?



$5 \times 5 \times 3 + 1 = 76$  parámetros por filtro  
76x6 filtros = 456 parámetros

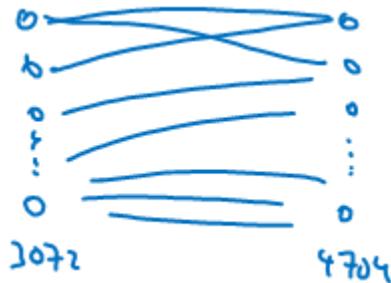
# ¿Por qué convolución?



$n_c$  filters =  $n_c$  input

$5 \times 5 \times 3 + 1 = 76$  parámetros por filtro  
76x6 filtros = 456 parámetros

Con una fully connected:



$3072 \times 4704 = 14.5$  millones de parámetros

# ¿Por qué tan pocos parámetros?

## Compartición de parámetros:

Si encuentras un filtro óptimo para detectar, por ejemplo, bordes verticales, te va a ser útil en toda la imagen

$$\begin{array}{|c|c|c|c|c|c|} \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline \end{array}$$

Con solo 9 parámetros sacamos 16 valores y “peinamos” toda la imagen

## Escasez de conexiones (connections sparsity):

Para calcular cada uno de los valores en el mapa de características solo nos hacen falta unos pocos valores de la imagen

# ¿Por qué tan pocos parámetros?

## Compartición de parámetros:

Si encuentras un filtro óptimo para detectar, por ejemplo, bordes verticales, te va a ser útil en toda la imagen

$$\begin{array}{|c|c|c|c|c|c|} \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline 10 & 10 & 10 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline 0 & 30 & 30 & 0 \\ \hline \end{array}$$

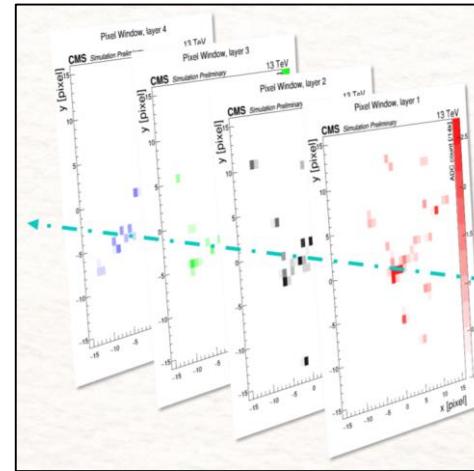
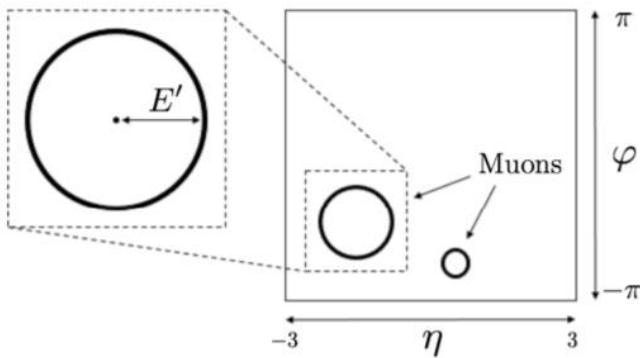
Con solo 9 parámetros sacamos 16 valores y “peinamos” toda la imagen

## Escasez de conexiones (connections sparsity):

Para calcular cada uno de los valores en el mapa de características solo nos hacen falta unos pocos valores de la imagen

# ¿Para qué se utilizan?

## Events classification

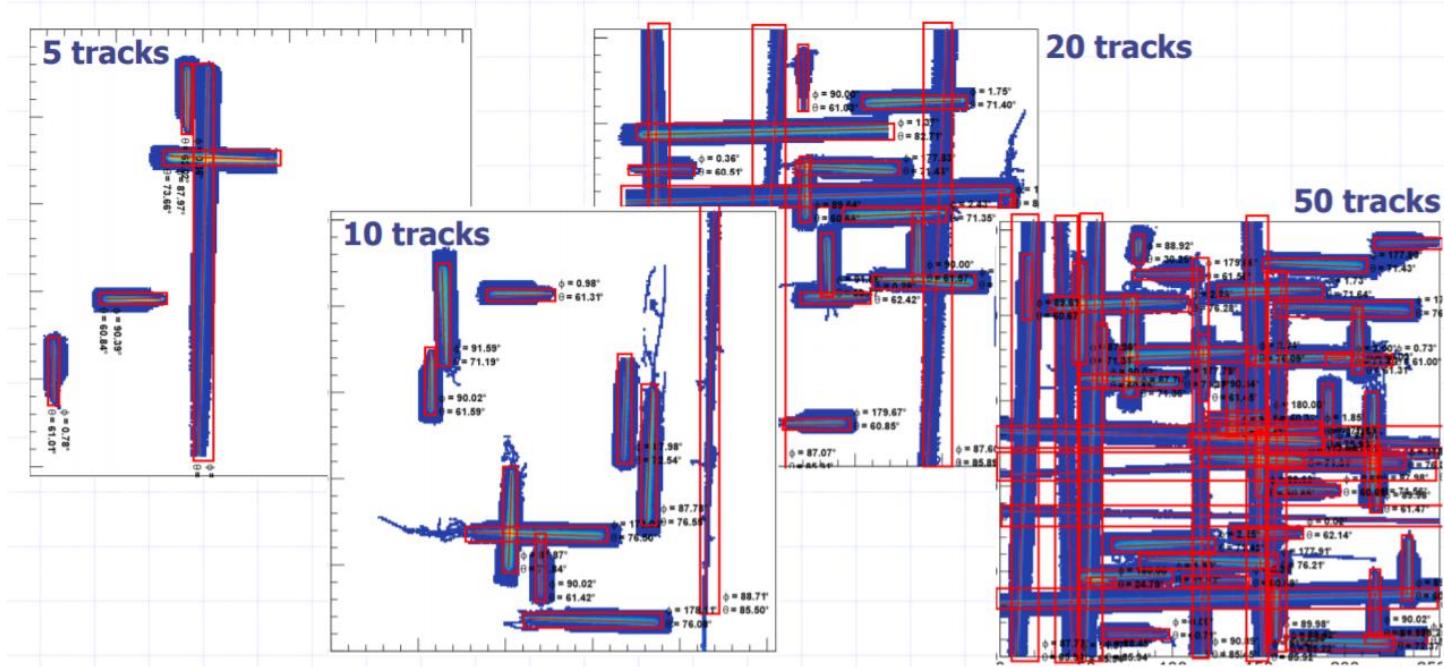


<https://arxiv.org/pdf/1708.07034.pdf>

See slides [here](#)



# ¿Para qué se utilizan?



# Descripción automática de las imágenes



<https://cs.stanford.edu/people/karpathy/deepimagesent/>



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



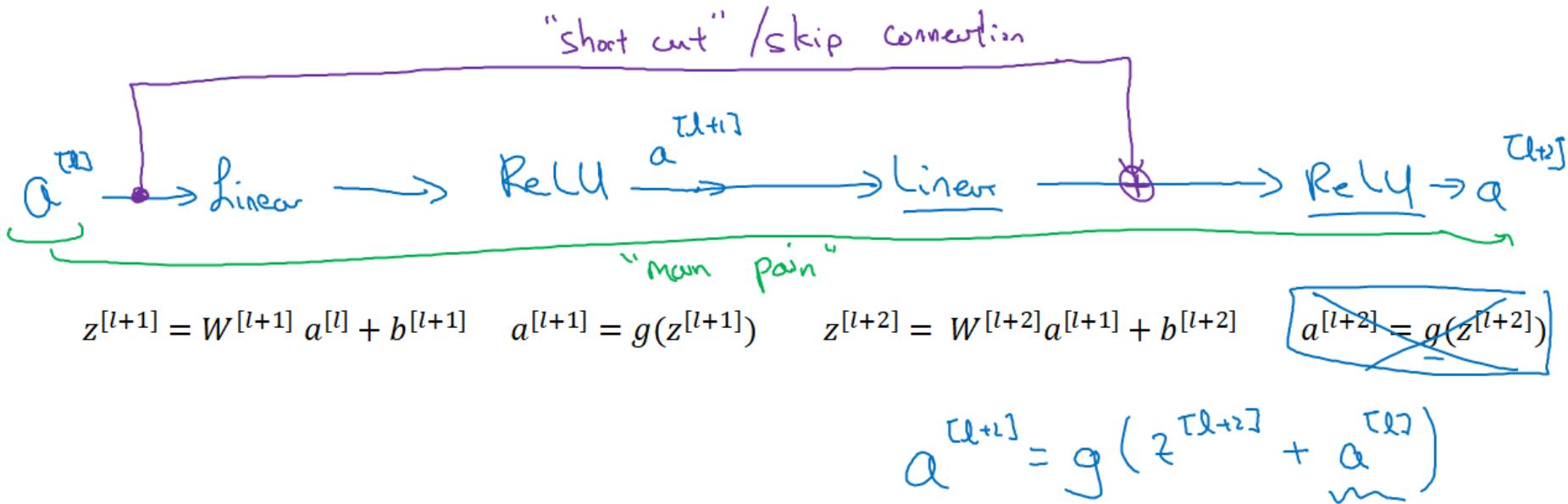
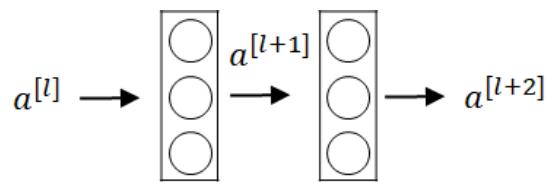
"two young girls are playing with lego toy."



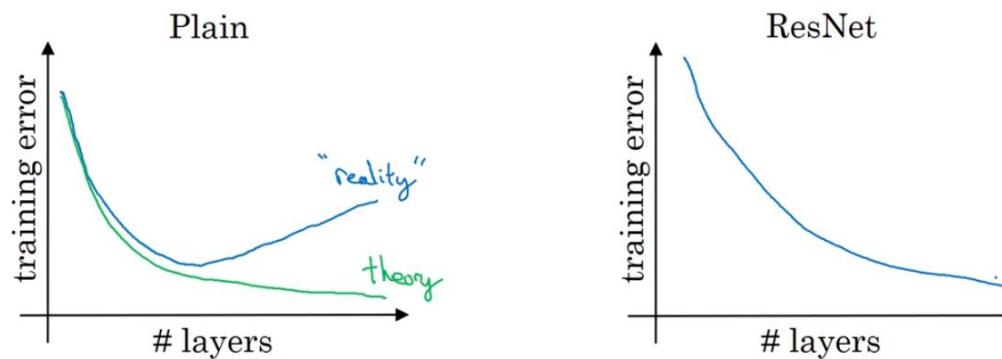
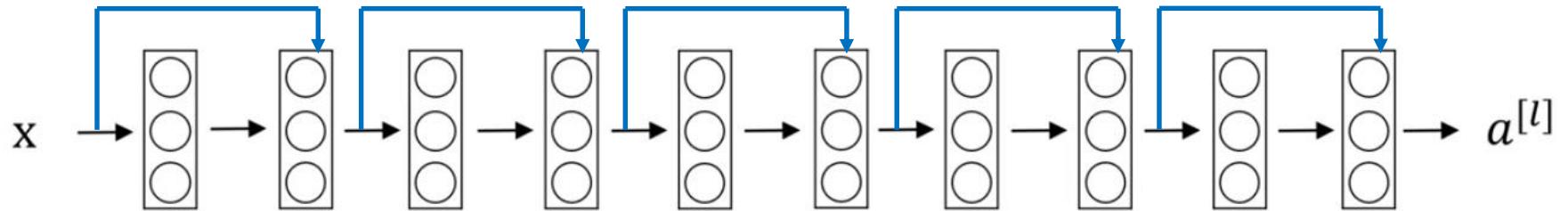
"boy is doing backflip on wakeboard."

<https://aistudio.google.com/starter-apps>

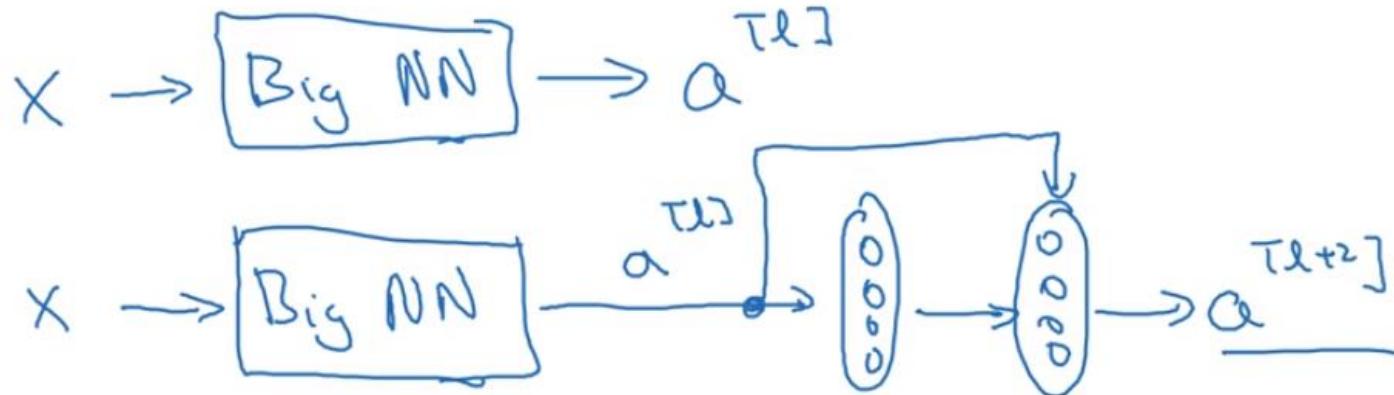
# Bloques residuales



# ResNet



# ¿Por qué funcionan las ResNet?



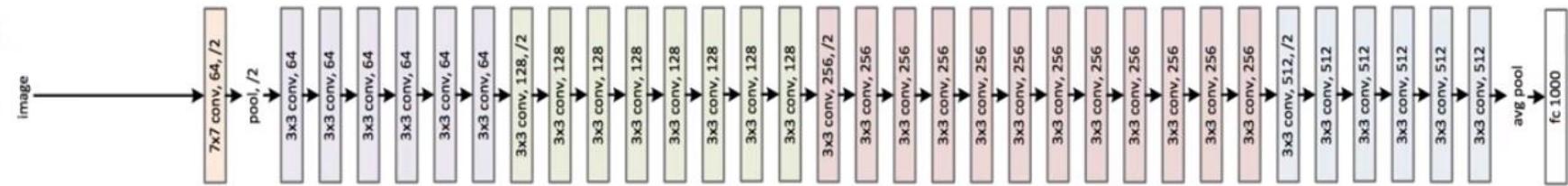
ReLU.  $a \geq 0$

$$\begin{aligned} a^{[l+2]} &= g(\underline{z^{[l+2]}} + \underline{a^{[l]}}) \\ &= g(\underline{w^{[l+2]} a^{[l+1]} + b^{[l+2]}} + \underline{a^{[l]}}) \end{aligned}$$

# ResNet

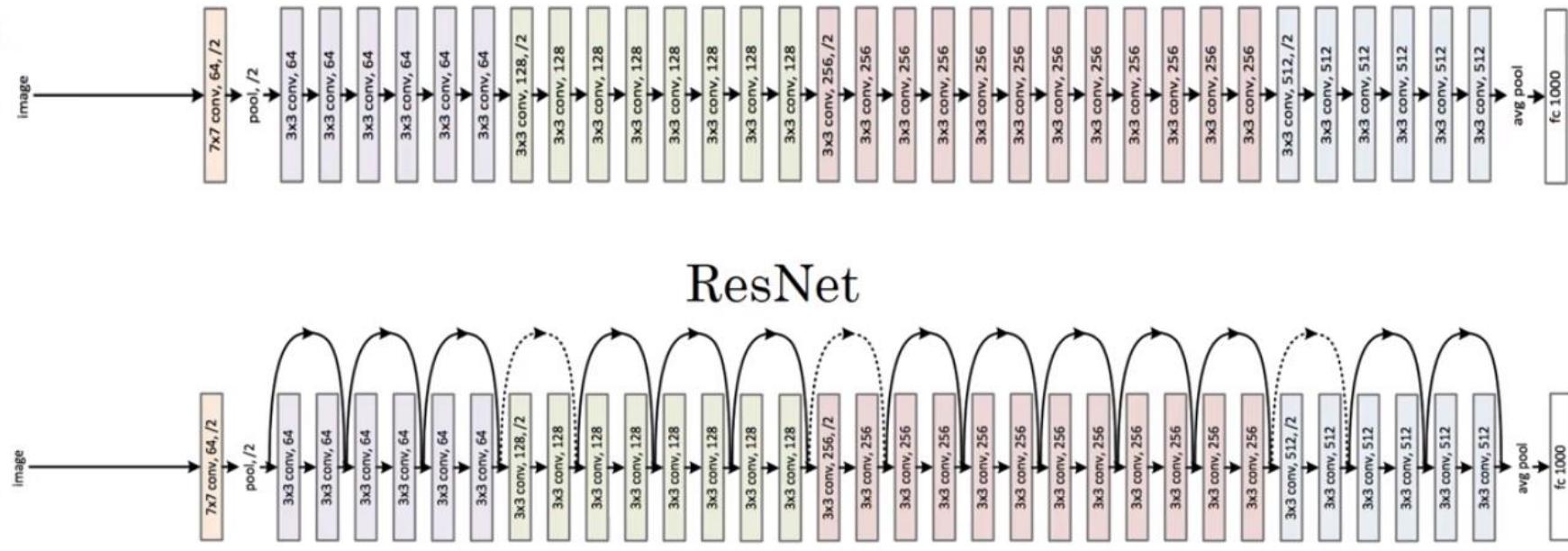
Plain

34-layer plain



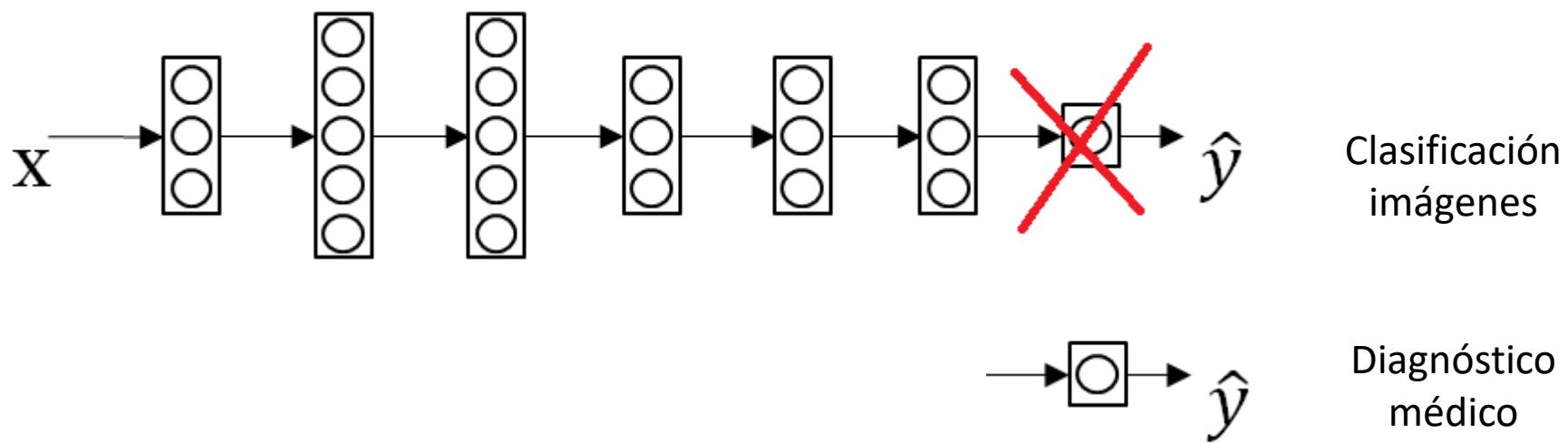
ResNet

34-layer residual

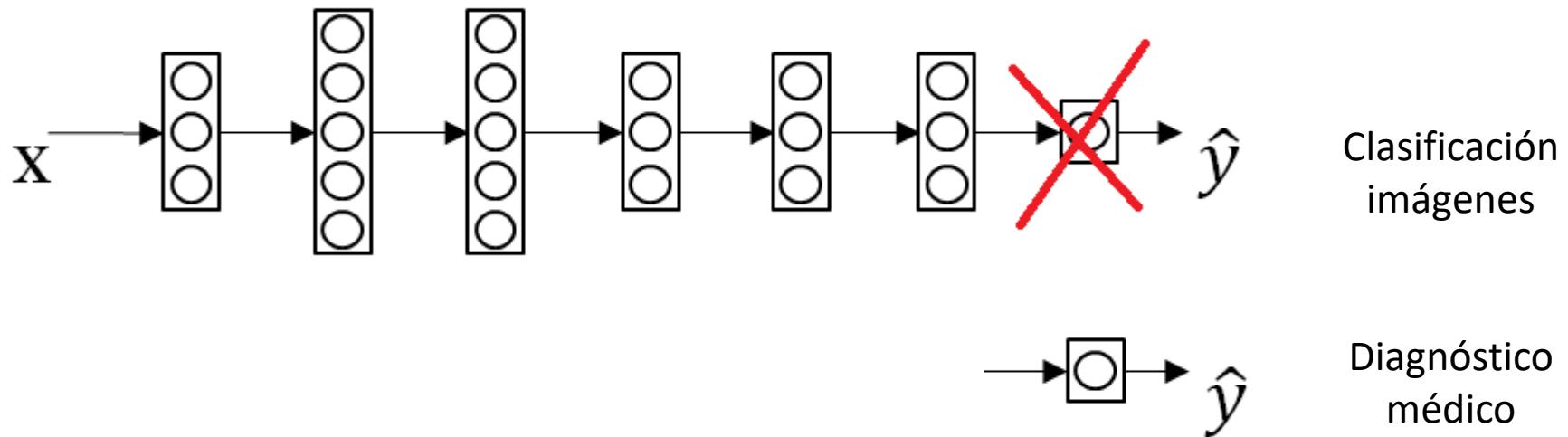


# Transfer Learning

- Una de las ideas más interesantes de Deep Learning es que uno puede aprovechar una red ya entrenada para un problema diferente
- Imaginemos que tenemos una red que se ocupa de clasificar imágenes
- Podemos eliminar la última capa y sus pesos y crear otra capa con pesos aleatorios



# Transfer Learning



- Reentrenamos la red usando el nuevo dataset de imágenes médica
- Si tienes pocas imágenes médicas: reentrenamos solo los pesos de la última capa (y dejar el resto fijo)
- Si tienes suficientes imágenes puedes reentrenar todos los pesos de la red neuronal → *fine tuning*

# ¿Cuándo hacer transfer learning?

## Transferencia A → B

- Cuando A y B tienen el mismo tipo de input
- Cuando tienes muchos más datos en A que en B
- Si sospechas que las características de bajo nivel de A pueden ser útiles para el problema B

# Images Style Transfer



Artistic Style Transfer (2015). [arxiv.org/abs/1508.06576](https://arxiv.org/abs/1508.06576) & <https://deepart.io>

# Video Style Transfer

Artistic style transfer for videos

Manuel Ruder  
Alexey Dosovitskiy  
Thomas Brox

University of Freiburg  
Chair of Pattern Recognition and Image Processing

# Neural Style Transfer



C



S



Transferencia de estilo

G

C – Contenido

S – Estilo

G – Imágen Generada

# Neural Style Transfer

- Definimos una función de coste
- Tendrá dos partes:

$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

$J_{\text{content}}$  → como de similar son G y C

$J_{\text{style}}$  → como de similares son S y G

$\alpha$  y  $\beta$  son dos hiperparámetros que podemos tunear para decidir si queremos que la imagen se parezca más a la original C o que siga el estilo S

# Neural Style Transfer



- Inicializa la imagen G aleatoriamente (i.e 100 x 100 x 3)



- Aplica gradient descent para minimizar J(G)

$$G = G - \frac{\partial J(G)}{\partial G}$$



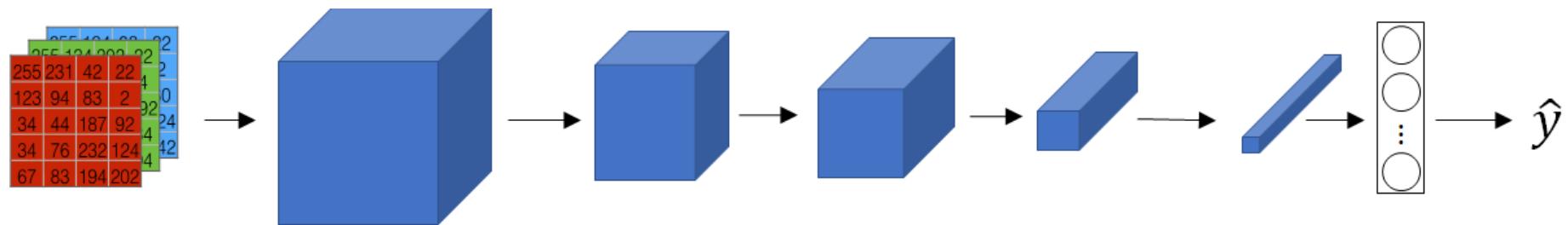
# Neural Style Transfer

- Función de coste del contenido:  $J_{content}$
- Cogemos una capa l en medio de la red (ni muy al principio ni muy al final)
- Usamos una red convolucional pre-entrenada
- Sean  $a^{l[C]}$  y  $a^{l[G]}$  el valor de la activación en la capa l de pasar la imagen inicial C y la generada G respectivamente
- Queremos que  $J_{content}$  mida como de parecidas son esas dos activaciones ya que si  $a^{l[C]}$  y  $a^{l[G]}$  son similares, las imágenes tendrán un contenido similar
- Así que definimos  $J_{content}$  como:

$$J_{content}(C, G) = \frac{1}{2} || a^{l(C)} - a^{l(G)} ||^2$$

# Neural Style Transfer

- ¿Qué es el estilo?

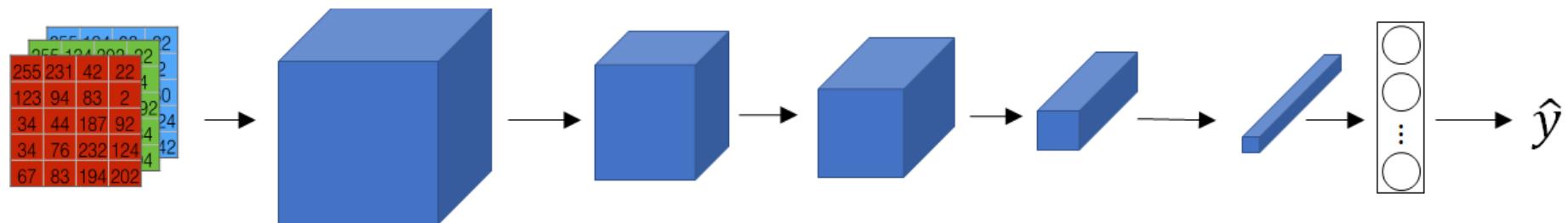


- Cogemos la capa 1 para medir el estilo

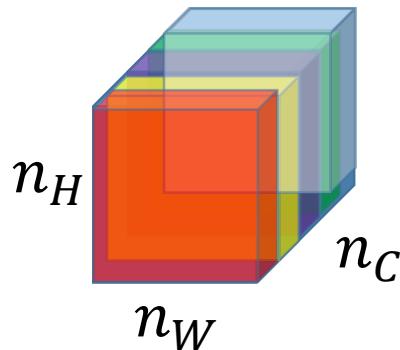
¿Cómo se os ocurre que podemos cuantificar cómo de parecidos son dos estilos?

# Neural Style Transfer

- ¿Qué es el estilo?

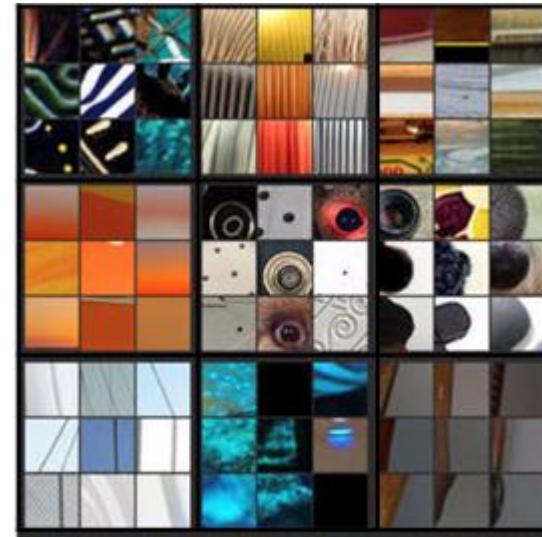
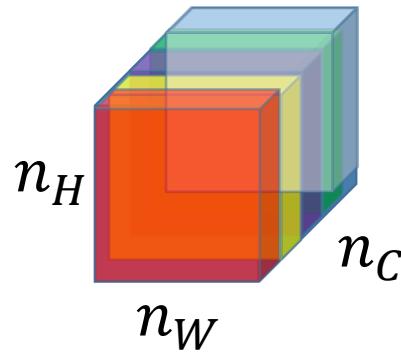


- Cogemos la capa 1 para medir el estilo
- Definimos el estilo como la **correlación entre diferentes canales**



¿Cómo de correlacionadas están las activaciones entre los diferentes canales?

# Neural Style Transfer



La correlación nos dice qué texturas ocurren juntas (correlación alta) y cuales no (correlación baja)

# Neural Style Transfer

- Sea  $a_{(i,j,k)}^{[l]}$  donde  $(i,j,k) \rightarrow (\text{ancho}, \text{alto}, \text{canal})$  de la capa l
  - $G_{kk'}^{[l]} = \sum_i \sum_j a_{ijk}^{[l]} a_{ijk'}^{[l]} \rightarrow$  Si estos pares de activaciones tienen un valor elevado G tendrá un valor elevado
  - Hacemos esto para la imagen S y G
- 
- $J(S,G) = \frac{1}{(2 n_h^{[l]} n_w^{[l]} n_c^{[l]})^2} || G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)} ||^2$

En realidad esto da igual porque tenemos el término  $\beta$

# Neural Style Transfer

$$J_{style}^{[l]}(S, G) = \frac{1}{\left(2n_H^{[l]} n_W^{[l]} n_C^{[l]}\right)^2} \sum_k \sum_{k'} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)})$$

Se obtienen mejores resultados si se define la  $J_{style}$  como la suma de todas las  $J_{style}$  en las distintas capas l

$$J_{style}(S, G) = \sum_l \lambda^{[l]} J_{style}^{[l]}$$

# Resumiendo

- ¿Qué son las capas convolucionales?  
¿Por qué funcionan bien con imágenes?
- Padding, striding...
- Capas de pooling → reduciendo la dimensionalidad
- Tipos de redes: ResNet
- Transferencia de aprendizaje
- Neural Style Transfer

# Generative Adversarial Networks

- Dos redes neuronales con distintos propósitos:
  - Generator
  - Discriminator
- El discriminador quiere detectar si una imagen es fake
- El generador quiere generar una imagen que engañe al discriminador
- El aprendizaje es un juego de suma cero de las funciones de pérdida de las dos redes.

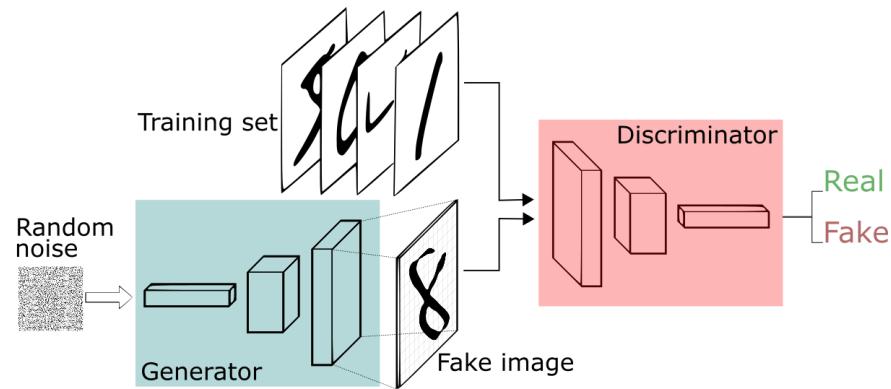
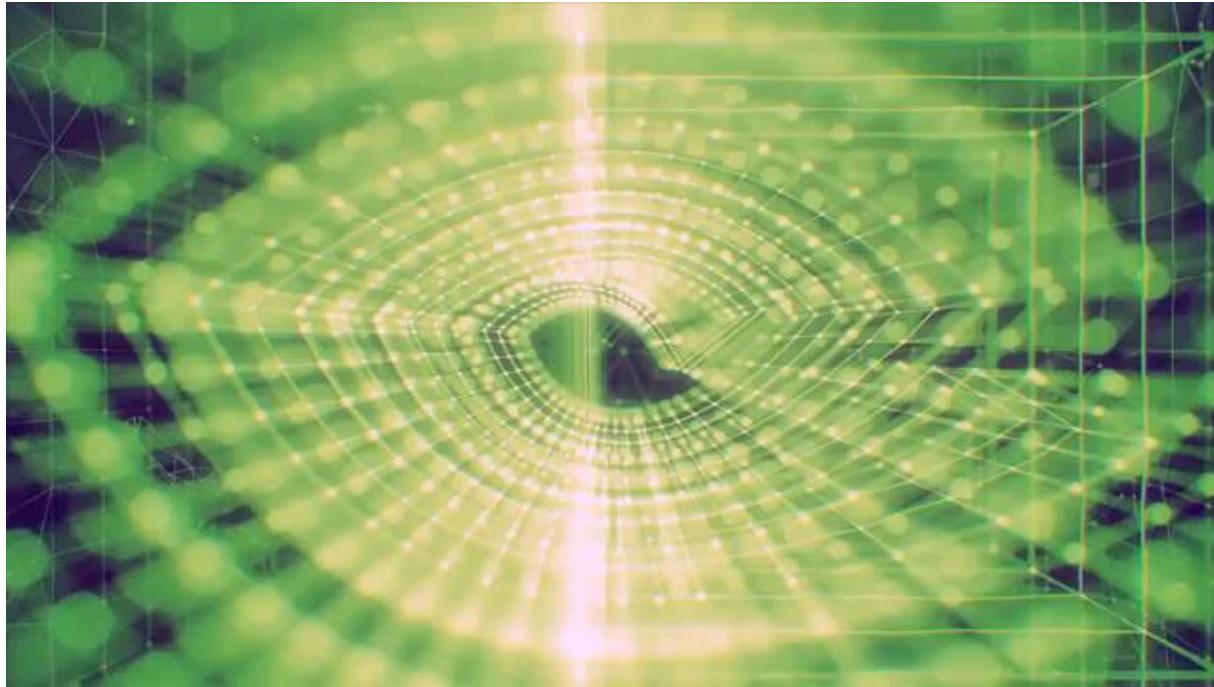
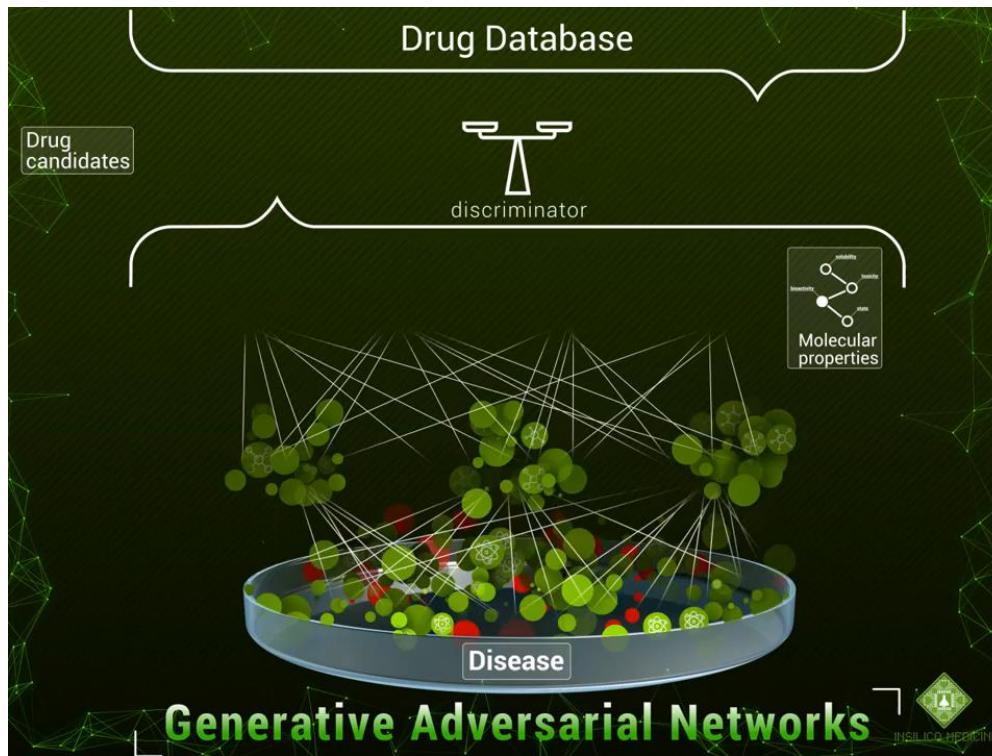


Image credit: [Thalles Silva](#)

# GAN for realistic faces generation



# Drug Discovery



# Self Driving Cars

## Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes

Tobias Pohlen, Alexander Hermans,  
Markus Mathias, Bastian Leibe

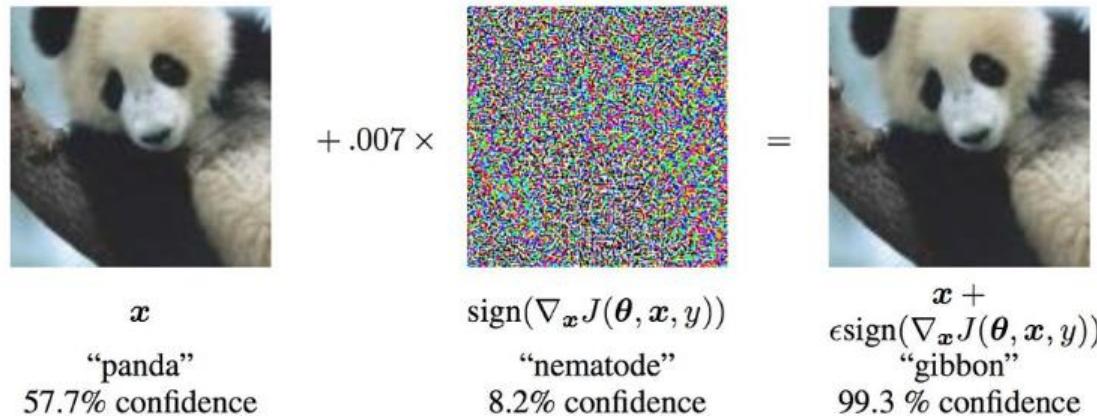
Visual Computing Institute, Computer Vision Group  
RWTH Aachen University



Each pixel in the image belongs to a certain category (car, moto, pedestrian, tree...)

# Vulnerabilities in vision systems

Un atacante que conoce el sistema de reconocimiento puede calcular un patrón ruidoso que estropea la performance.



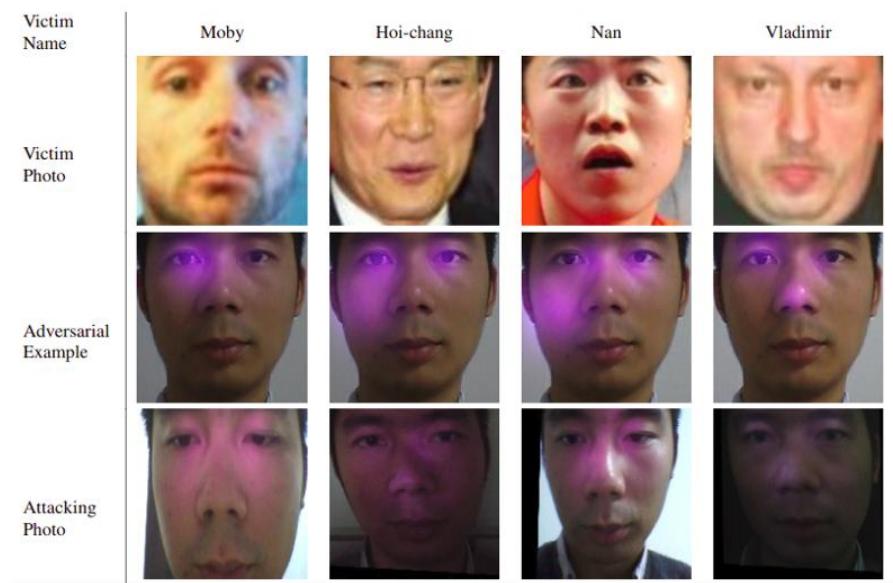
Goodfellow et al., "Explaining and harnessing adversarial examples", ICLR 2014

# Avoid Face Recognition

Hair style and make up



Infrared light



<https://cvdazzle.com/>

<https://arxiv.org/abs/1803.04683>







# Prácticas ConvNets: Reconociendo dígitos

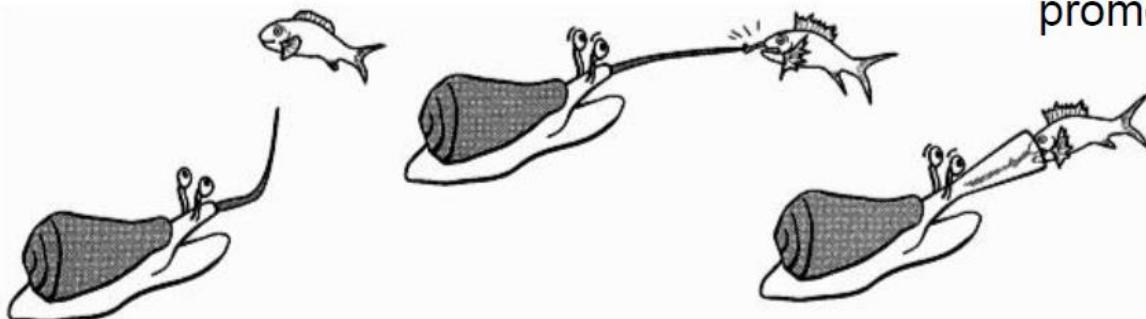
- Recordad que tenéis que subir a Moodle los Notebooks ejecutados para que pueda ver directamente los resultados
- Para argumentar vuestras respuestas podéis incluirlas como una celda de texto dentro del propio Notebook
- Para hacer las prácticas tenéis que descargar este repositorio (recordad que podéis hacerlas en datasciencehub, localmente o en google colab...esto ultimo os permitirá tener GPU):

<https://colab.research.google.com/drive/1HtjkSu-KOU9KHLAYmuidddeQtBDRO0RHc?usp=sharing>

- Documentación de Keras: [https://keras.io/getting\\_started/](https://keras.io/getting_started/)

# Backup

# Clasificando Conus



## •Training dataset

Colección de imágenes de expertos (68 especies | 1.5K imágenes) que cubren tres regiones diferentes:

- Región Panámica
- Región de África del Sur
- Atlántico Occidental y Mediterráneo

## •Resultados

Los resultados usando sets de imágenes de Google son prometedores

<http://conus.deep.ifca.es/>

# Clasificación de plantas



**Arquitectura**  
ResNet50  
**Framework**

Python con Lasagne/Theano

**Training dataset**

PlantNet (*6K especies | 250K imágenes*)

**Test datasets**

- Google Search Image (*3680 species | 36K imágenes*)
- Portuguese Flora (*1300 species | 15K imágenes*)
- iNaturalist (*3K especies | 300K imágenes*)

**Resultados**

Resultados útiles (Top1: 59% | Top5: 74%) para más de la mitad del dataset de test.

# Detección de objetos: Ventana móvil

Training set:



X

1

1

1

0

0

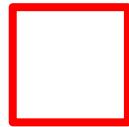
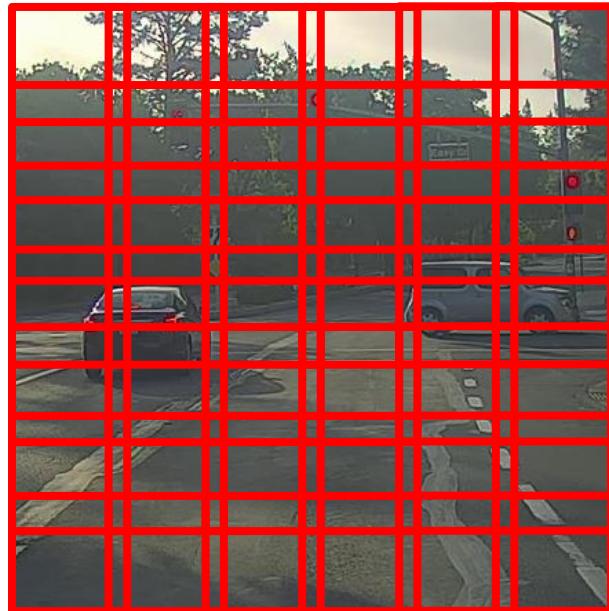
Con esto puedes entrenar una ConvNet que te diga si algo es o no un coche



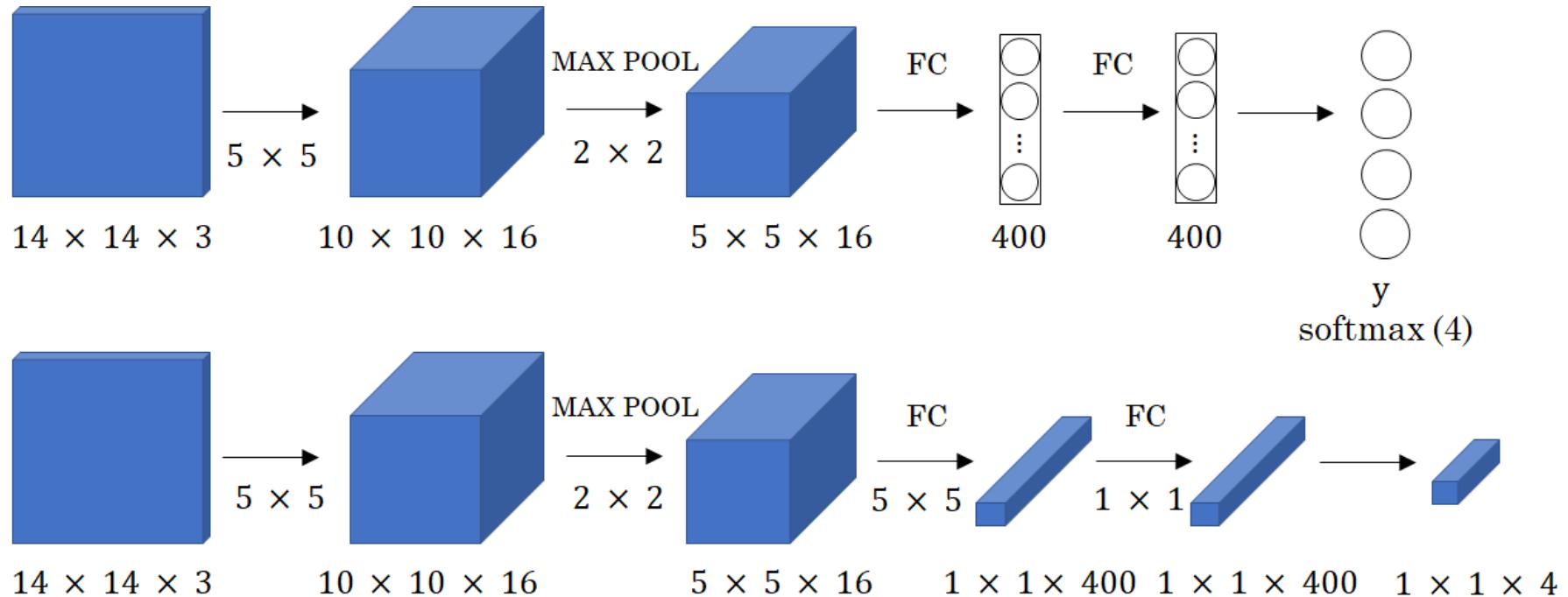
ConvNet

1

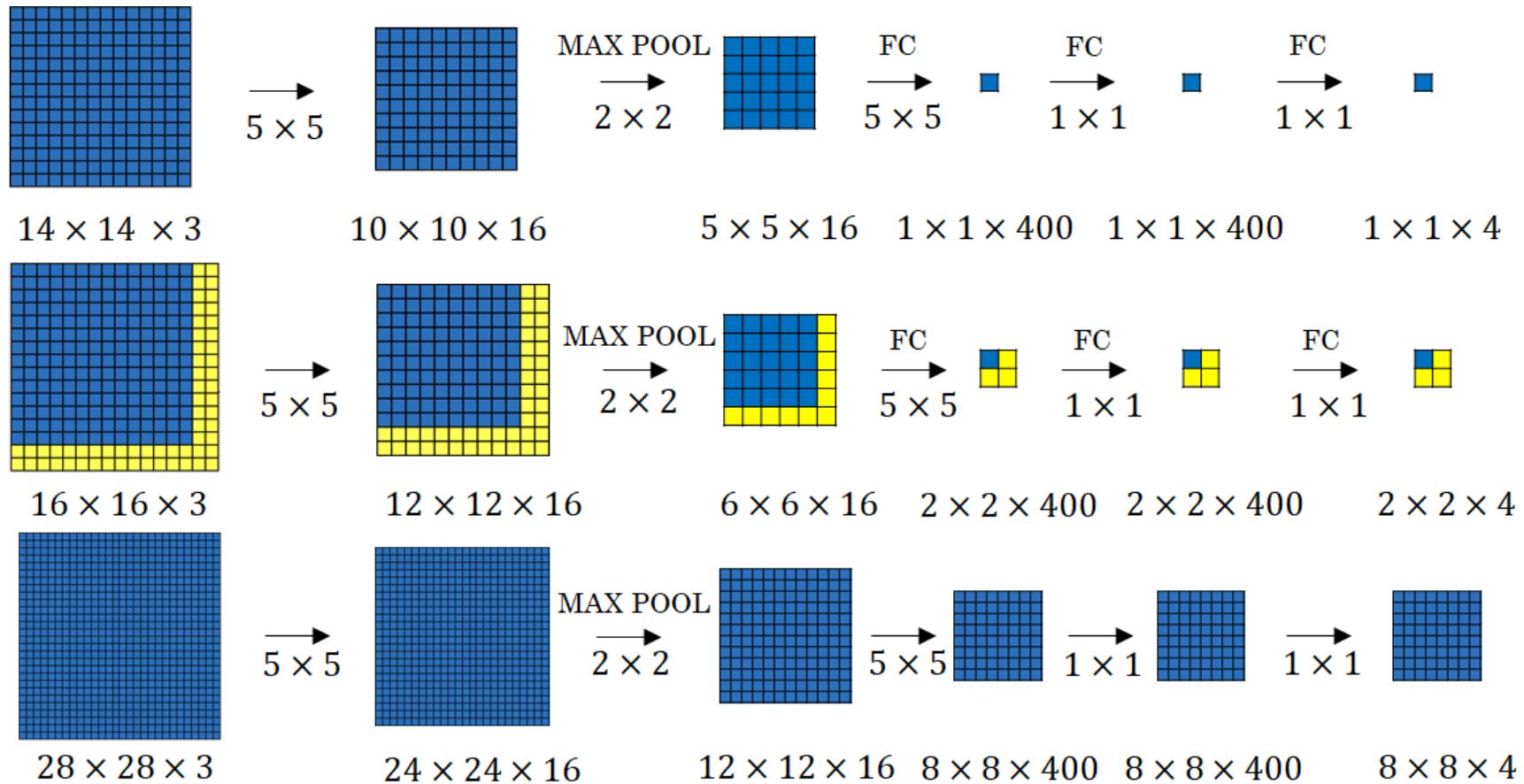
# Detección de objetos: Ventana móvil



# Cómo convertir FC en convolucional



# Implementación convolucional de la ventana móvil

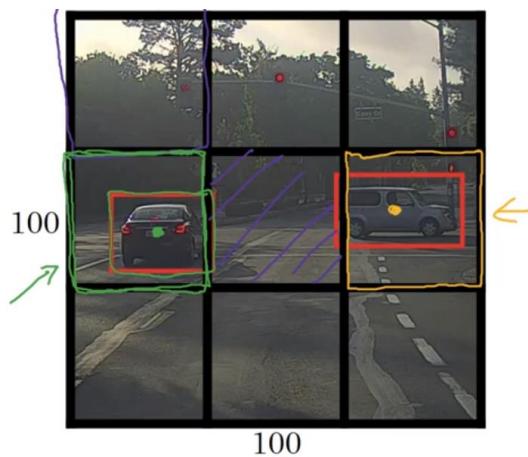


<https://arxiv.org/abs/1312.6229>

# YOLO

**Uno de los problemas del método anterior:** no estamos dando la posición del objeto con precisión

**Dividimos la imagen en celdas:** Aquí usamos 3x3 pero en un ejemplo real deberíamos tener mucha más granularidad



- Asumimos que solo hay un objeto por celda
- Etiquetas para cada una de las celdas:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

←

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

