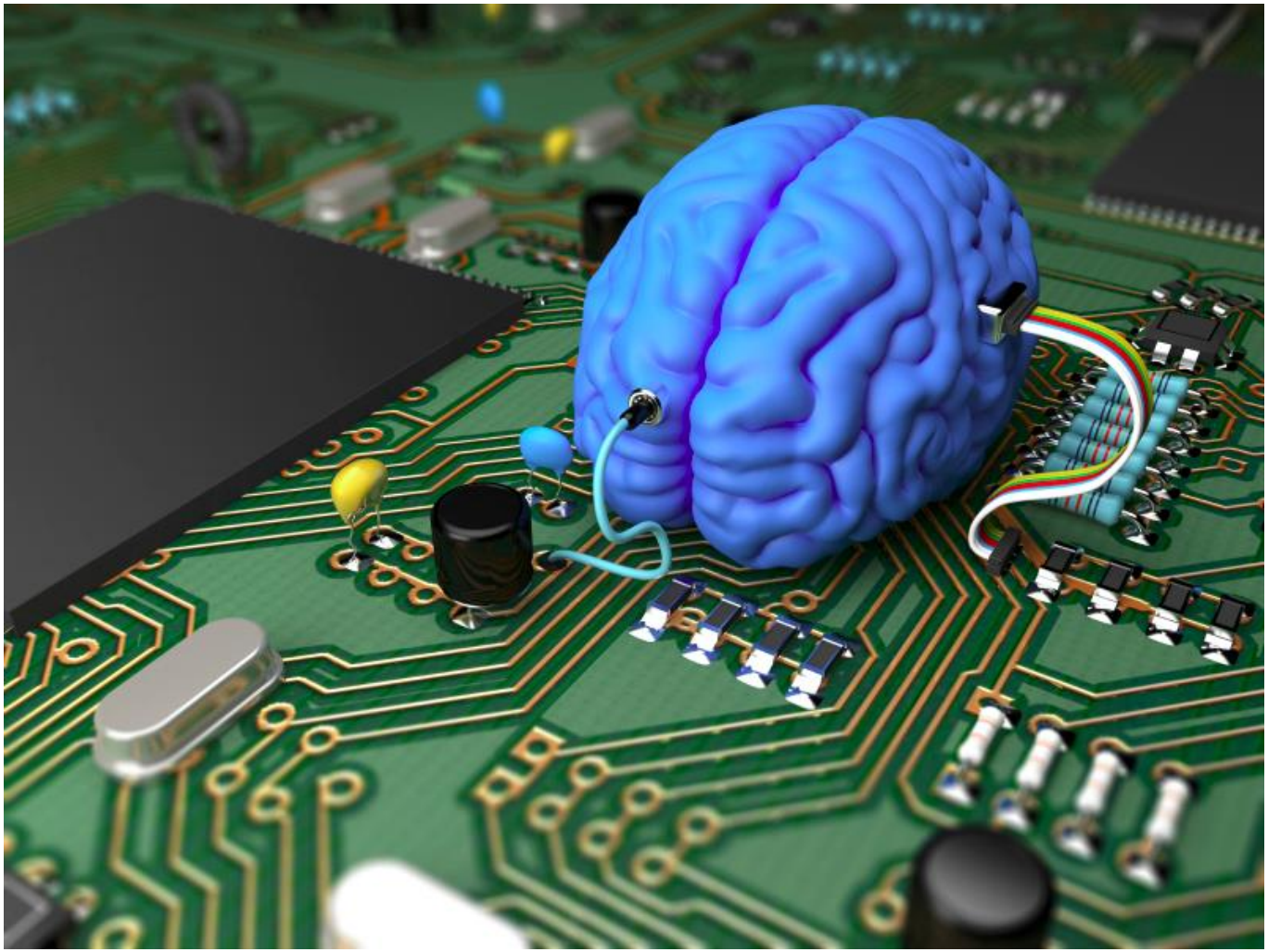
Machine learning I : ConvNets



Master Universitario Oficial **Data Science**
con el apoyo del
UC UNIVERSIDAD DE CANTABRIA · UIMP Universidad Internacional Menéndez Pelayo · CSIC

Lara Lloret Iglesias

Santander, Marzo 2019

En episodios anteriores...

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

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*:

- 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**.

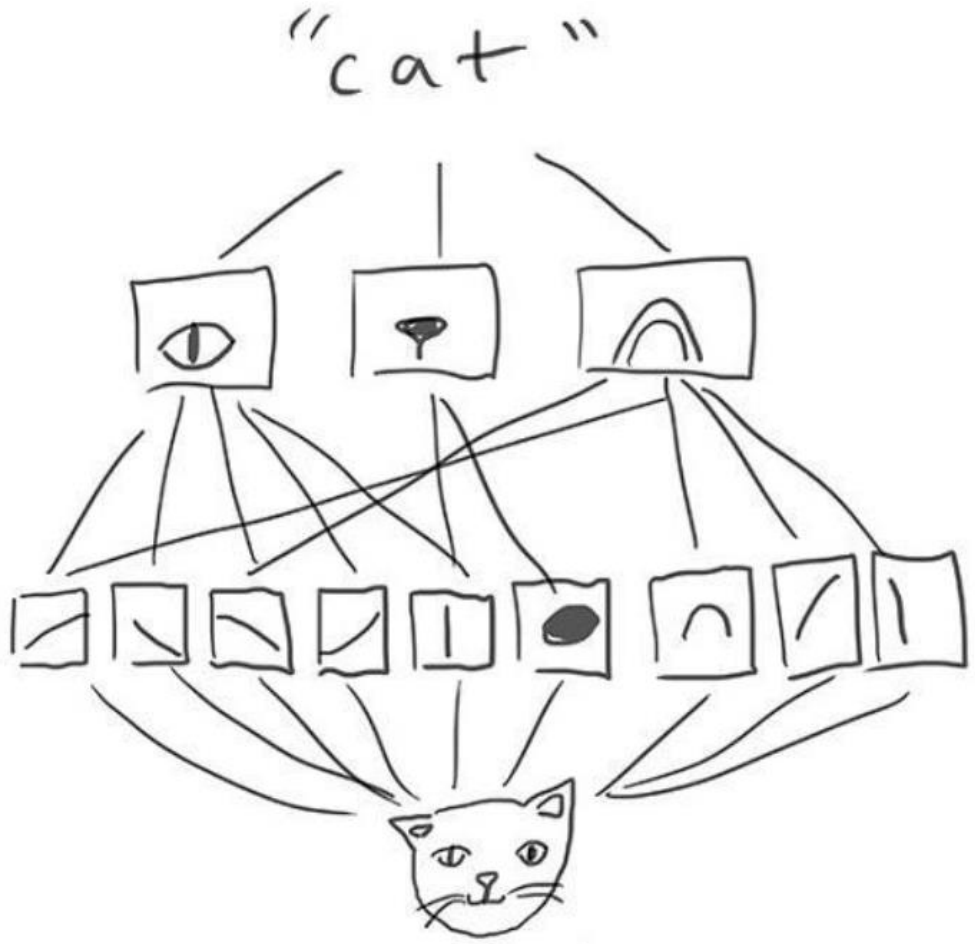
Principales características de las ConvNets

- **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

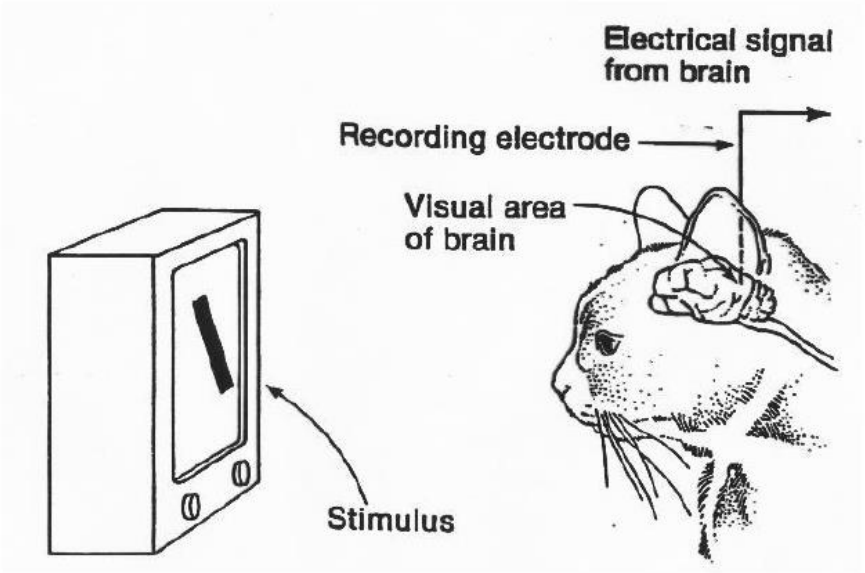
  Por este motivo, las ConvNets:
  - Muy eficientes para procesar imágenes
  - Menos imágenes para entrenar ya que tienen mayor poder de generalización

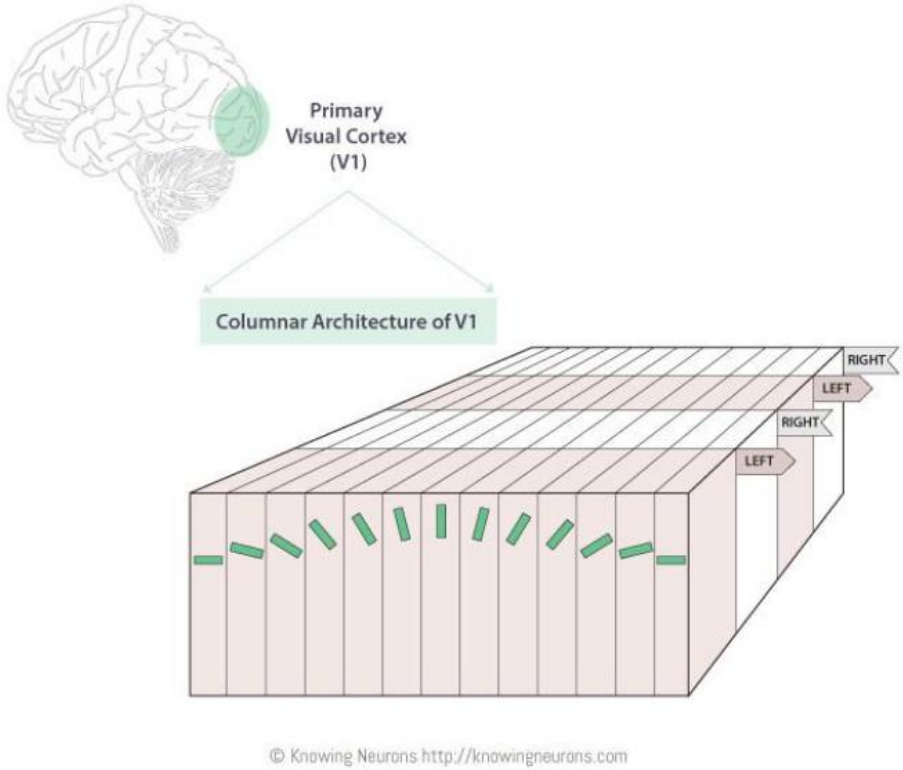
- **Pueden aprender la jerarquía espacial de las imágenes**
  - La primera capa convolucional aprenderá a extraer esquinas y bordes
  - La segunda los combinará para crear conceptos más complejos

Jerarquía de características



Experimento de Hubel y Wiesel



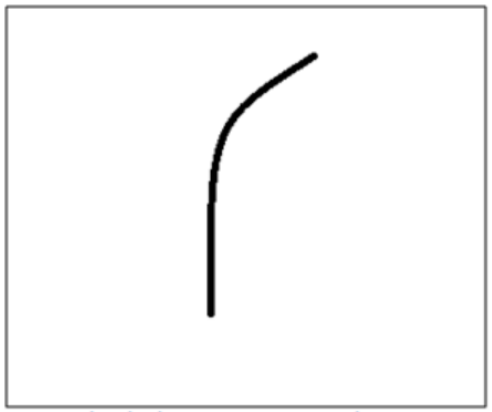


- Dos cosas en común con las redes usadas en Computer Vision:
  - Neuronas (filtros) especializados en orientaciones (representaciones) determinadas
  - Complejidad creciente para crear la representación final

Capa Convolucional

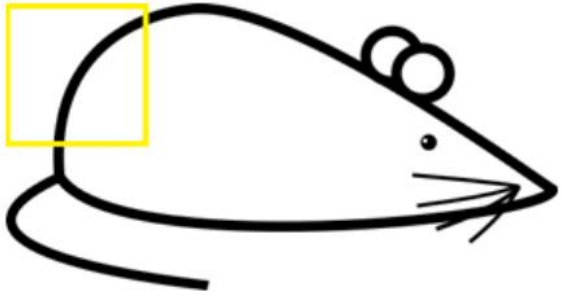
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



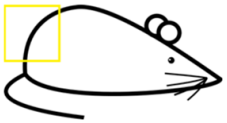
Visualization of a curve detector filter

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

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

Pixel representation of receptive field

*

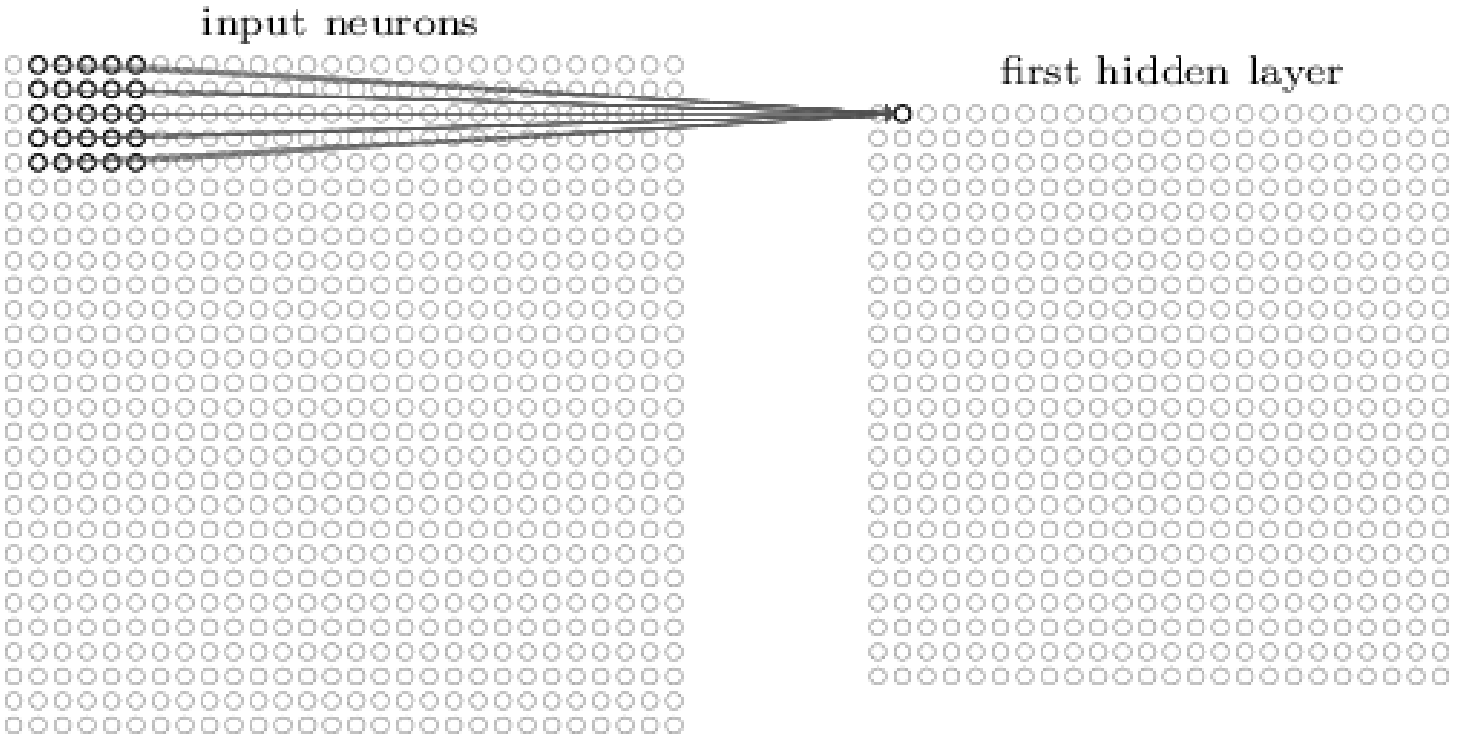
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplicación y suma = 0

Capa Convolucional

Imagen en blanco y negro

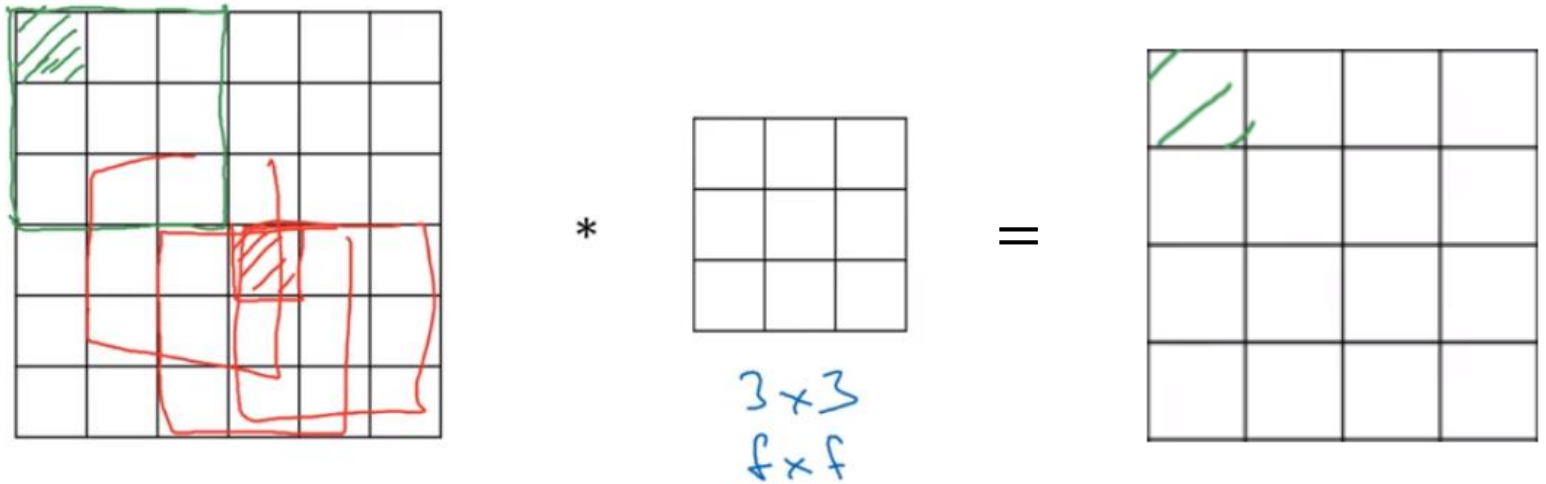


Filtro 5x5

Mapa de características

Padding

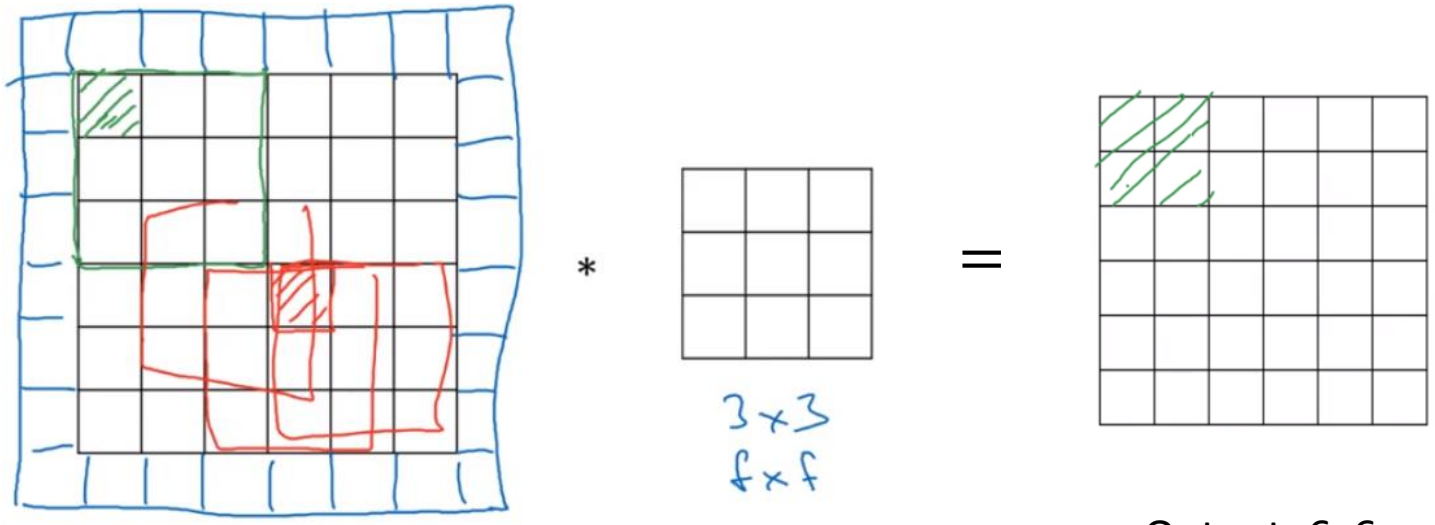
- Uno de los problemas de las capas convolucionales es que pueden hacer que la imagen mengüe demasiado.
- Si tienes muchas capas la imagen va a acabar siendo muy pequeña.
- Otro problema es que la información de los píxeles de las esquinas y los bordes van a estar "infra-representados" en los mapas de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



Output: 4x4

Padding

p=1



Output: 6x6

Padding

- "Valid" convolution: Sin padding

n **x** n * f **x** f → n – f + 1 **x** n – f + 1

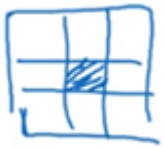
6 x 6 * 3 x 3 → 4 **x** 4

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

n **x** n * f **x** f → n+2p – f + 1 **x** 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 $\quad f \times f$ filter

padding p $\quad$ stride s

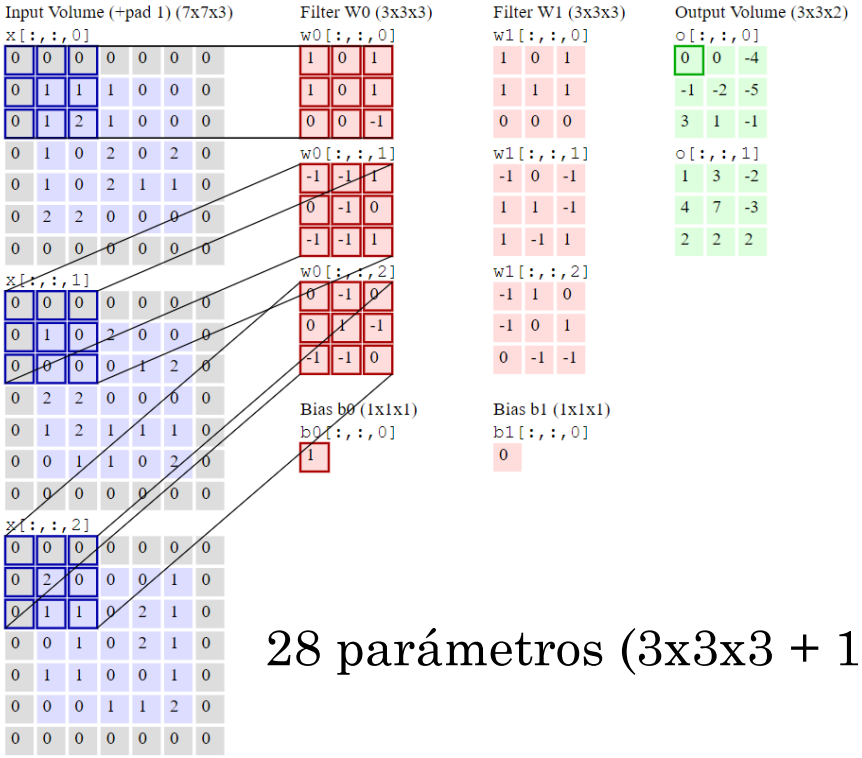
$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

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

Capa Convolucional

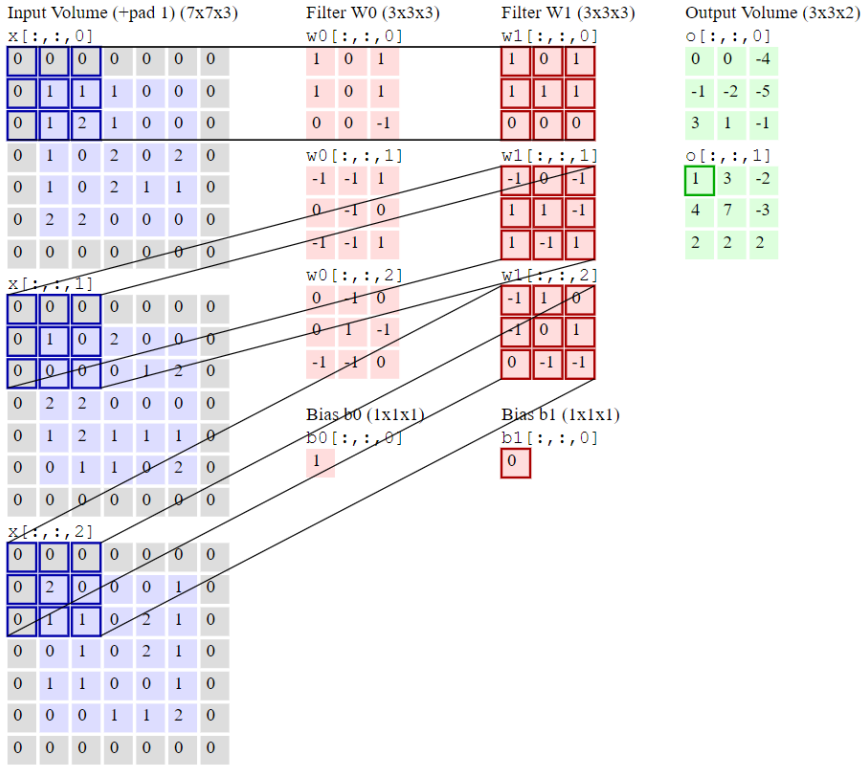
¿Y si la imagen es en color?

Filtro 0

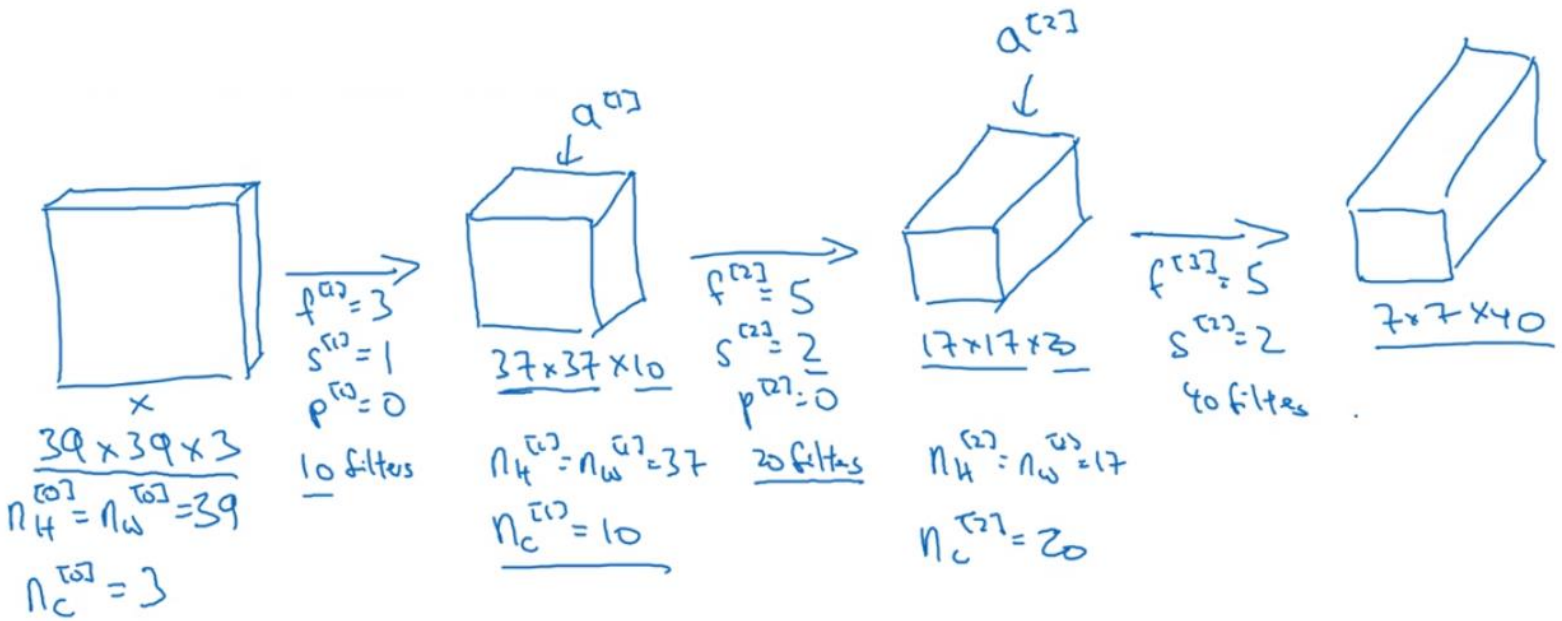


28 parámetros (3x3x3 + 1)

Filtro 1



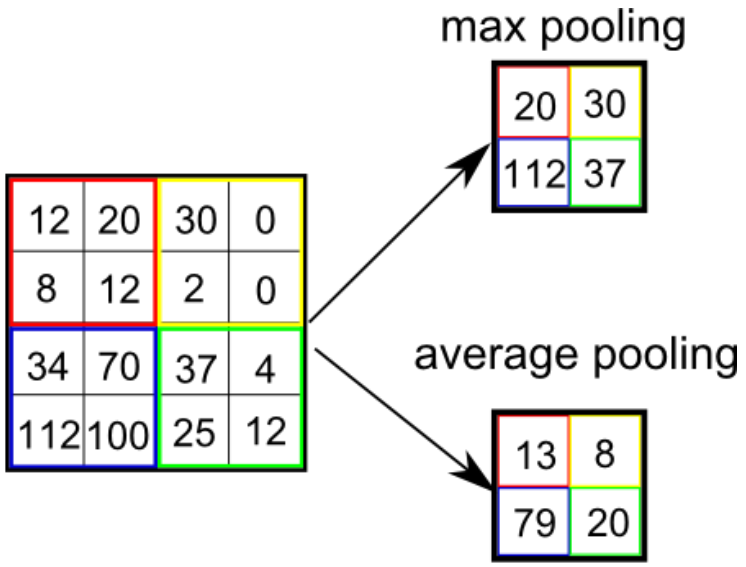
Ejemplo



- 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



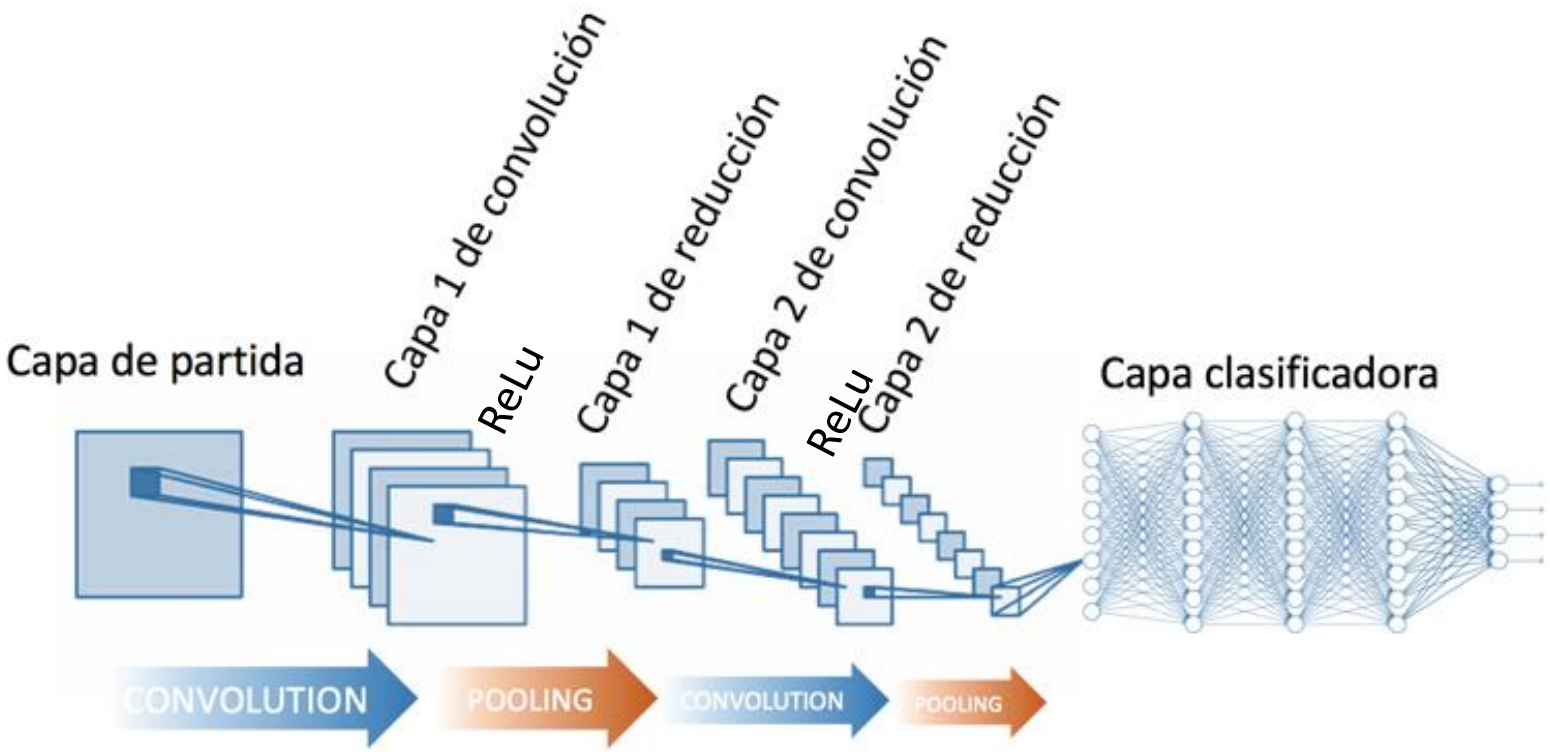
Hiperparámetros:

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

Típicamente: f=2, s=2

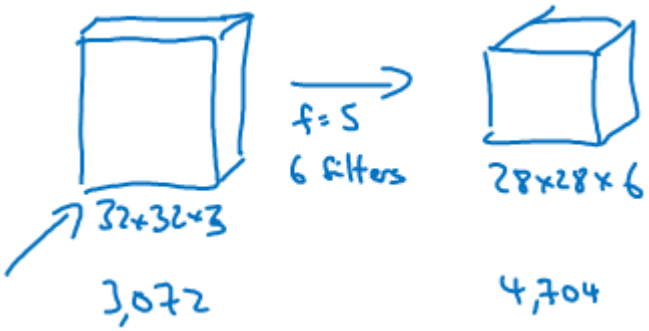
- El pooling no tiene ningún parámetros para aprender

Juntándolo todo...

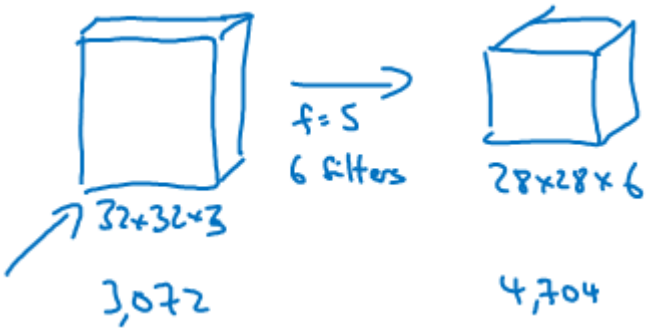


Vamos a contar como **una capa** el conjunto de **convolución + reducción** (hay autores que consideran que son dos capas) ya que solo la capa de convolución tiene parámetros para aprender

¿Por qué convolución?

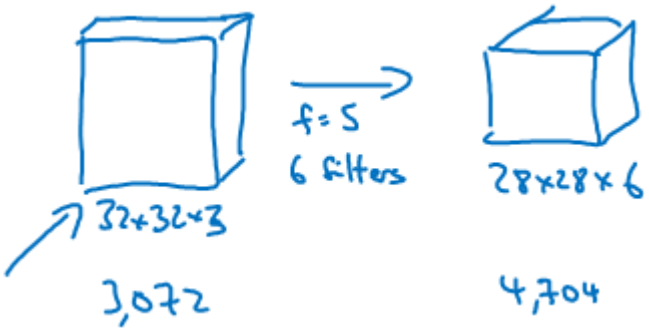


¿Por qué convolución?



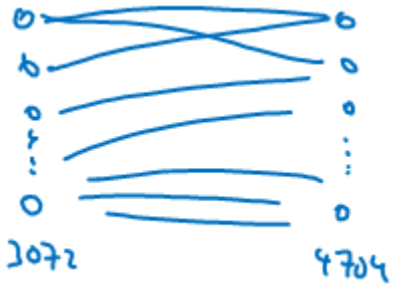
5x5x3+1=76 parámetros por filtro
76x6 filtros =456 parámetros

¿Por qué convolución?



5x5x3+1=76 parámetros por filtro
76x6 filtros =456 parámetros

Con una fully connected:



3072x4704 =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

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

$*$

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

$=$

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

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

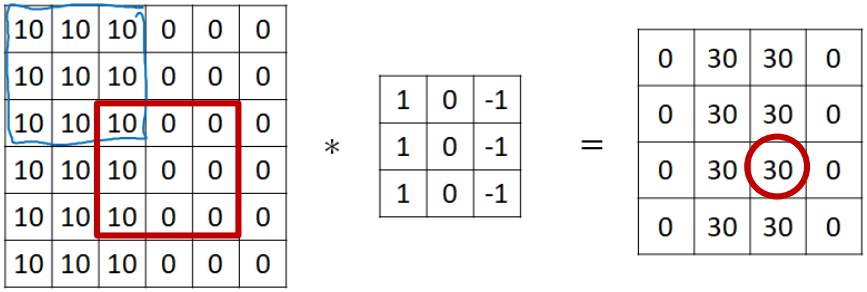
Escasez de conexiones (connections sparsity):

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



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

Ejercicio

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

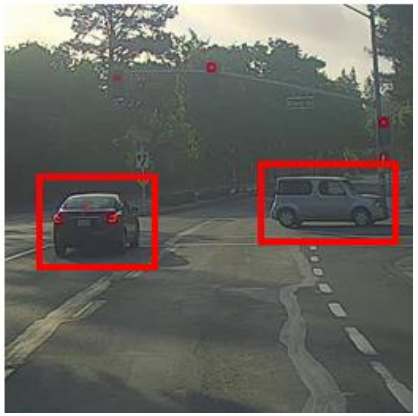
¿Cuántos parámetros hay en cada capa?

Localización de Objetos

Clasificación

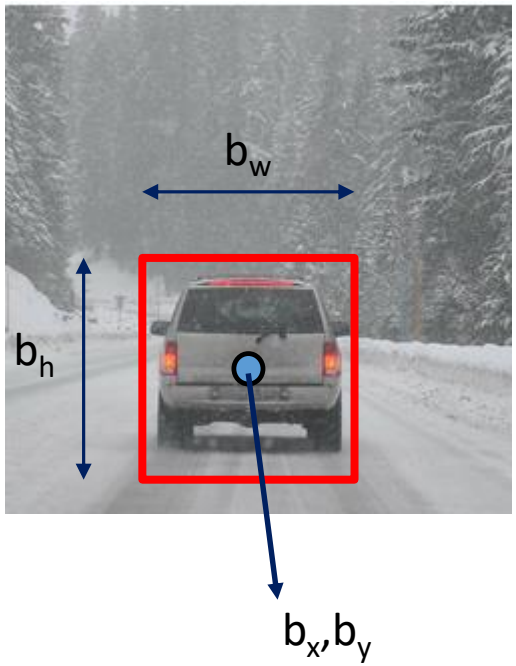
Clasificación con localización

Detección



Clasificación con localización

(0,0)



¿Como definimos la etiqueta *target* de nuestro problema (y)?

¿Hay objeto o no?

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

0 – peatón
1 – coche
2 – moto
3 – fondo

Clasificación con localización

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

$$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

Clasificación con localización

¿Cómo es la función de pérdida?

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Si $y_0 = 1$:

$$L(y, \hat{y}) = (y_0 - \hat{y}_0)^2 + (y_1 - \hat{y}_1)^2 + \cdots + (y_7 - \hat{y}_7)^2$$

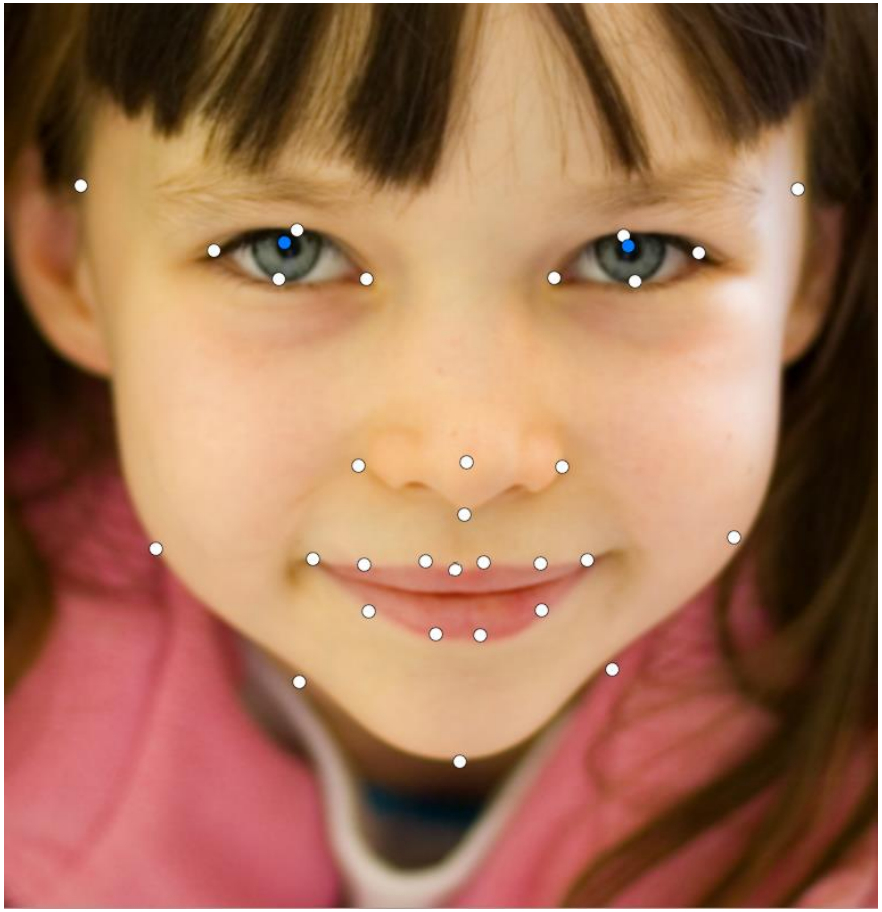
Si $y_0 = 0$:

$$L(y, \hat{y}) = (y_0 - \hat{y}_0)^2$$

Hemos usado el error cuadrático como función de pérdida, pero podrías usar cualquier otra

Detección de puntos de referencia

32 puntos de referencia



Detección de objetos: Ventana móvil

Training set:

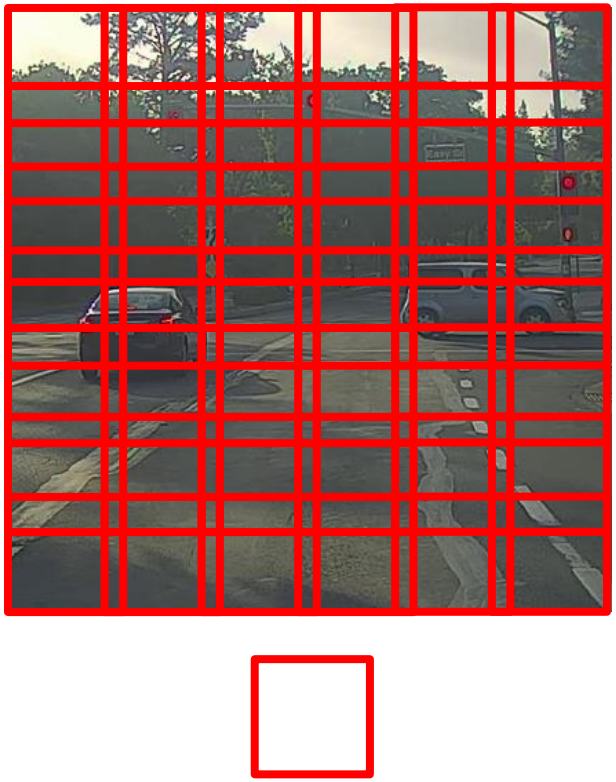
x	y
	1
	1
	1
	0
	0

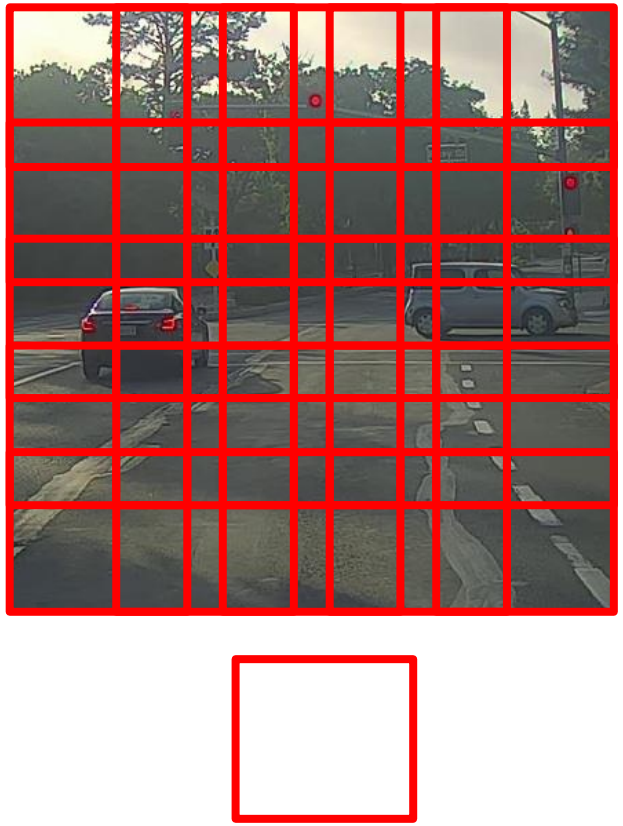
Con esto puedes entrenar una ConvNet que te diga si algo es o no un coche

ConvNet

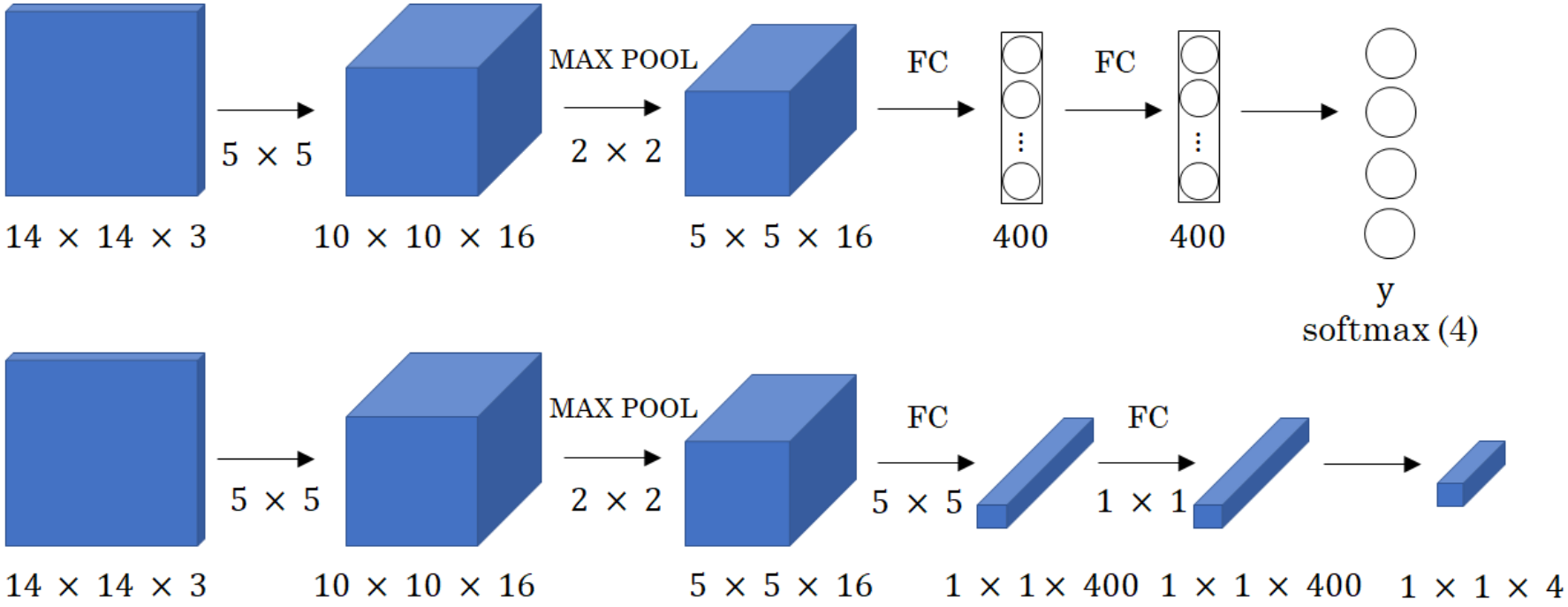
1

Detección de objetos: Ventana móvil

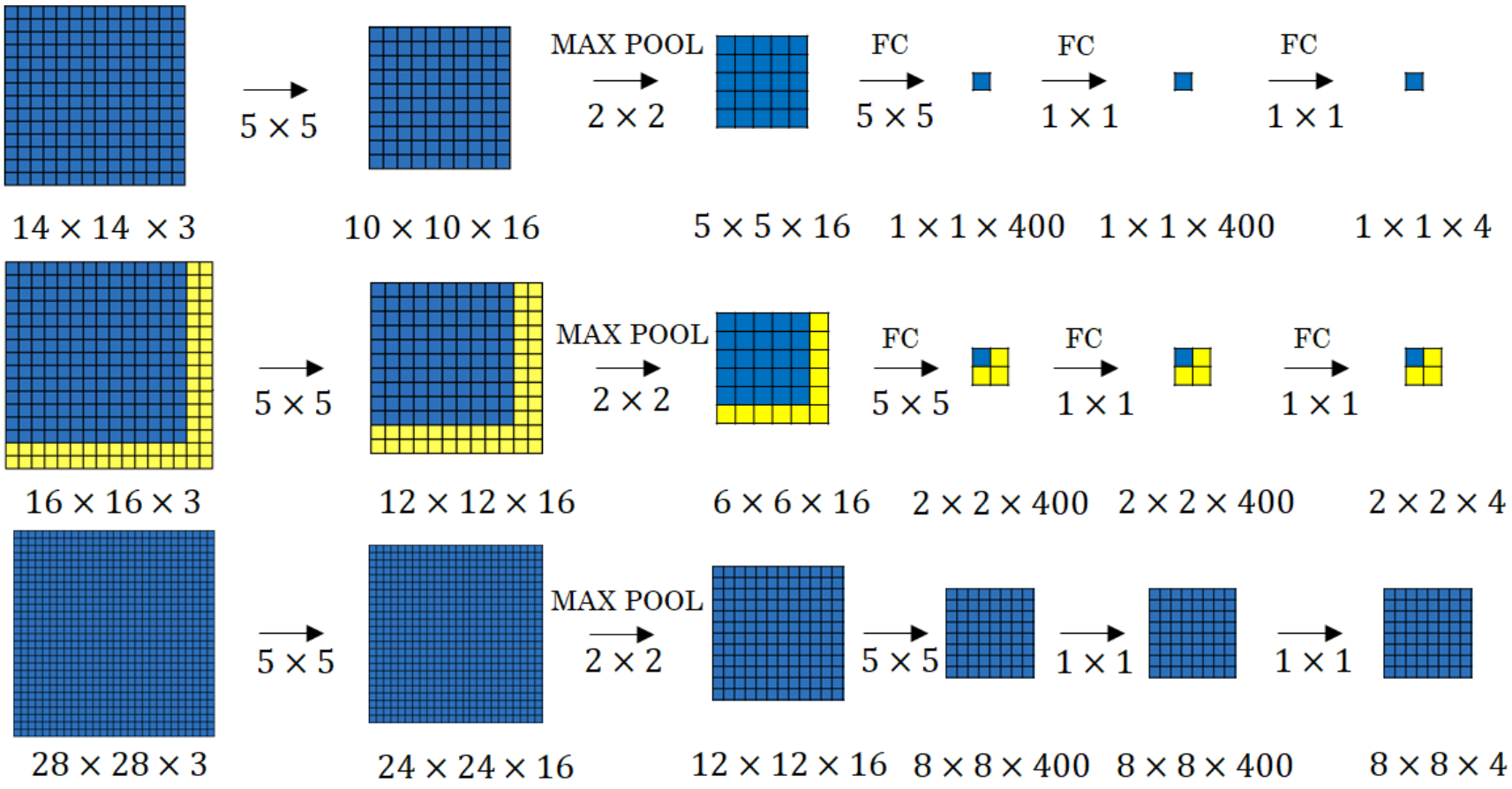




Cómo convertir FC en convolucional



Implementación convolucional de la ventana móvil

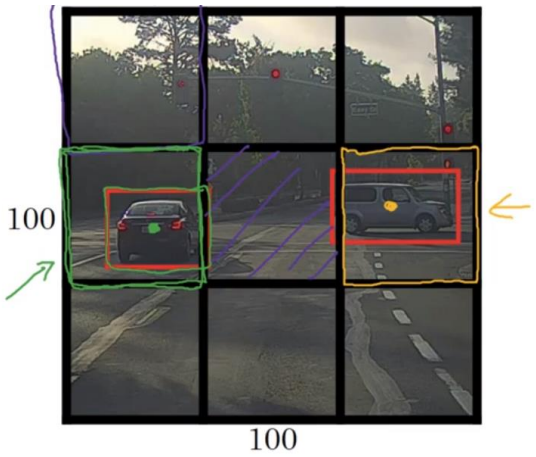


https://arxiv.org/abs/1312.6229

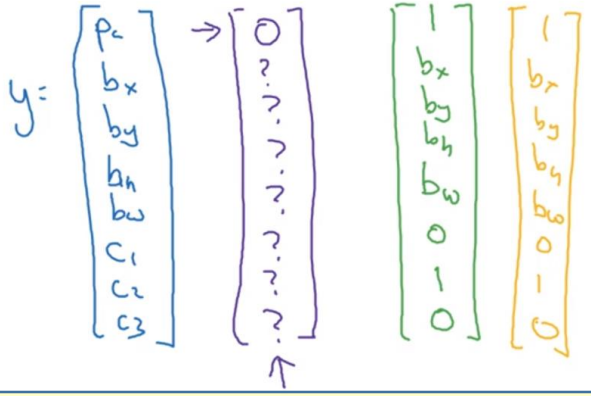
YOLO

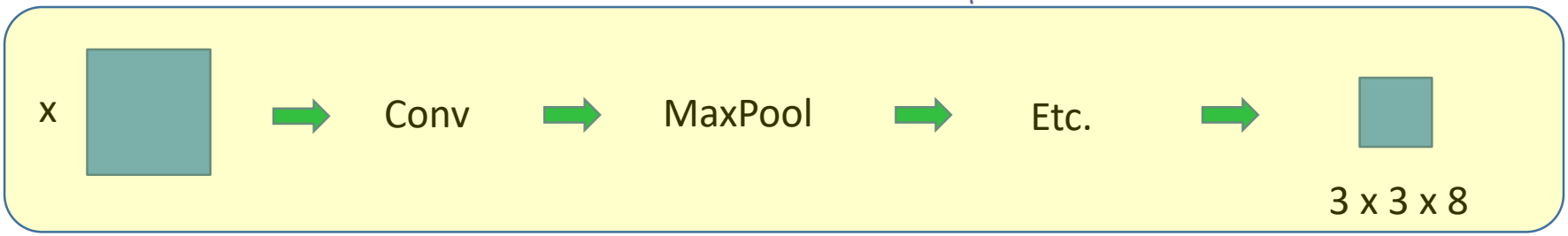
Uno de los problemas del método anterior: no estamos dando la posición del objeto con precisión

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



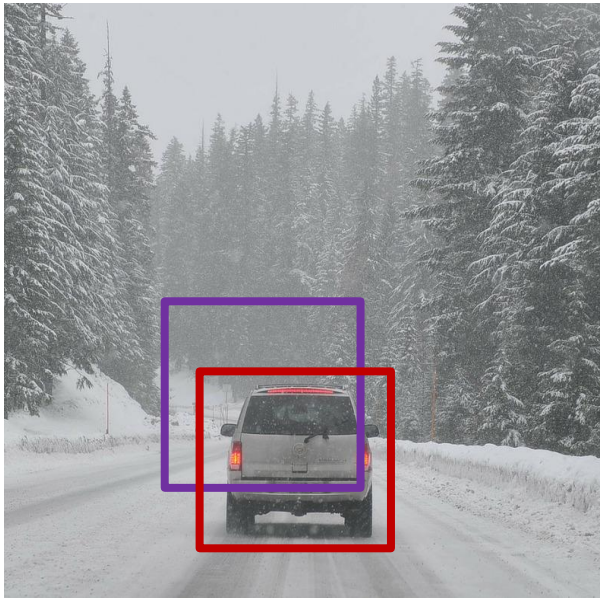
- Asumimos que solo hay un objeto por celda
- Etiquetas para cada una de las celdas:





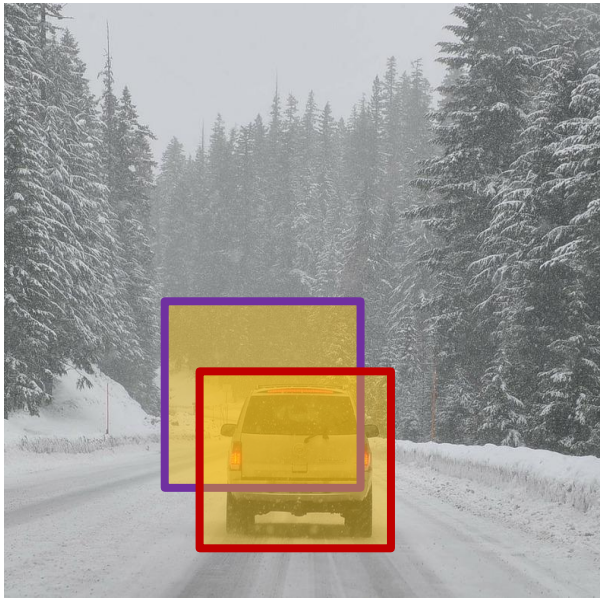
Cómo evaluar la localización

- Método IoU (Intersection over Union)



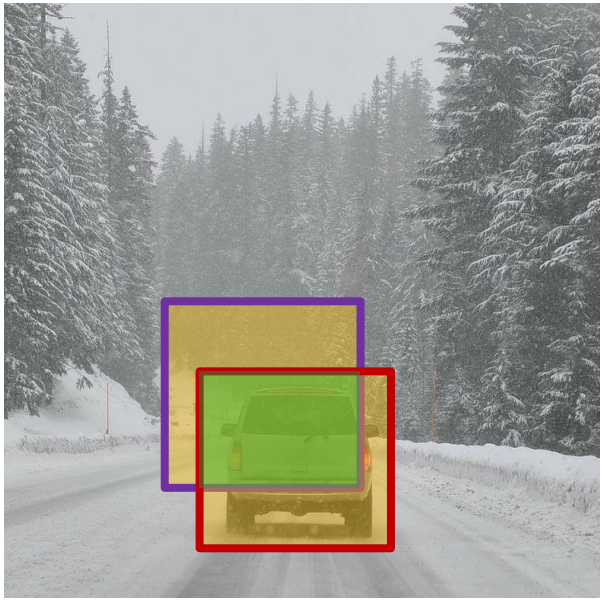
Cómo evaluar la localización

- Método IoU (Intersection over Union)



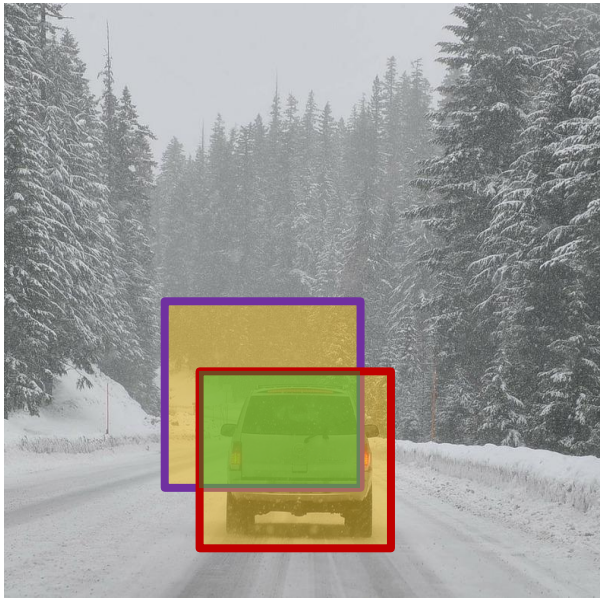
Cómo evaluar la localización

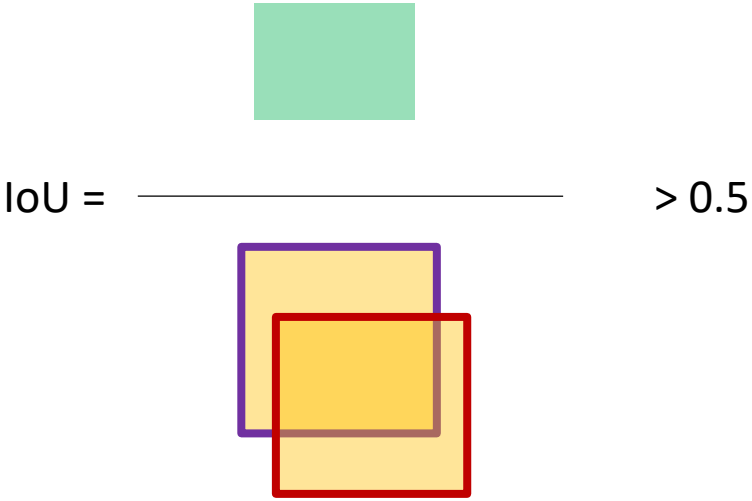
- Método IoU (Intersection over Union)



Cómo evaluar la localización

- Método IoU (Intersection over Union)

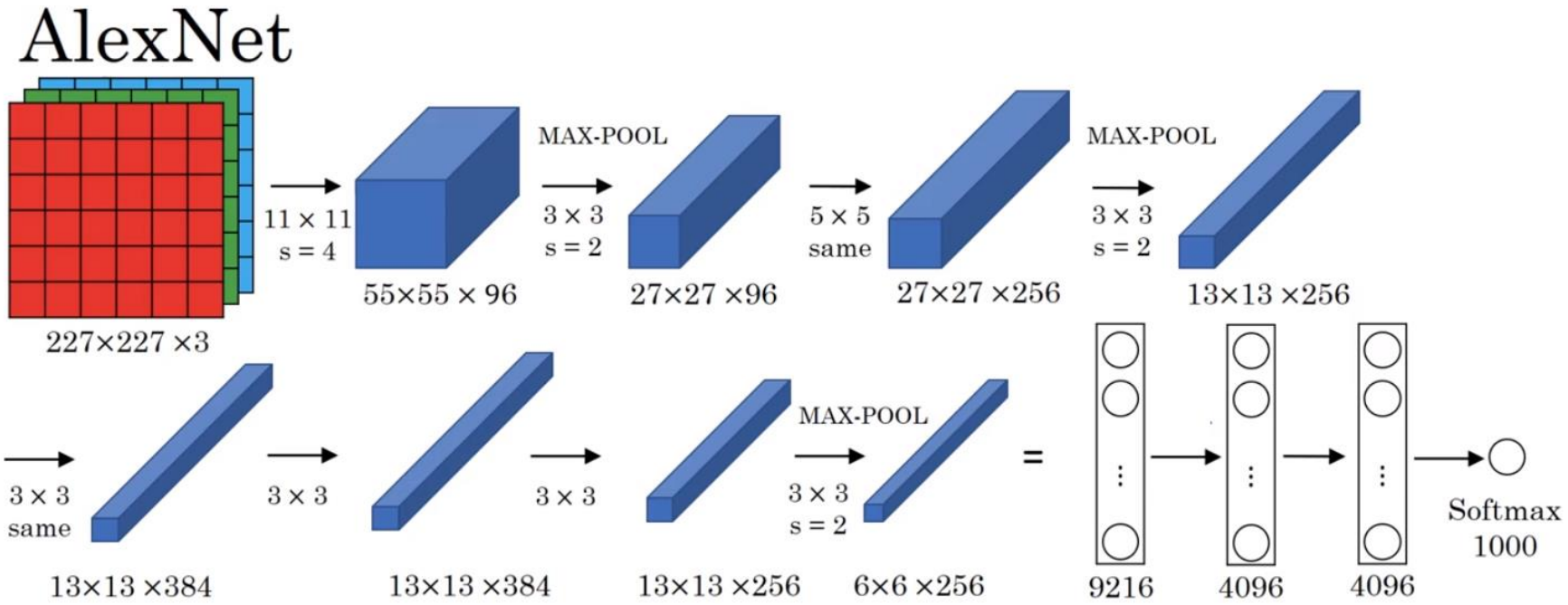




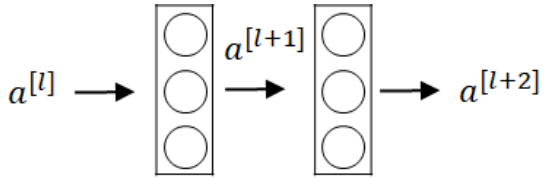
Tipos de redes

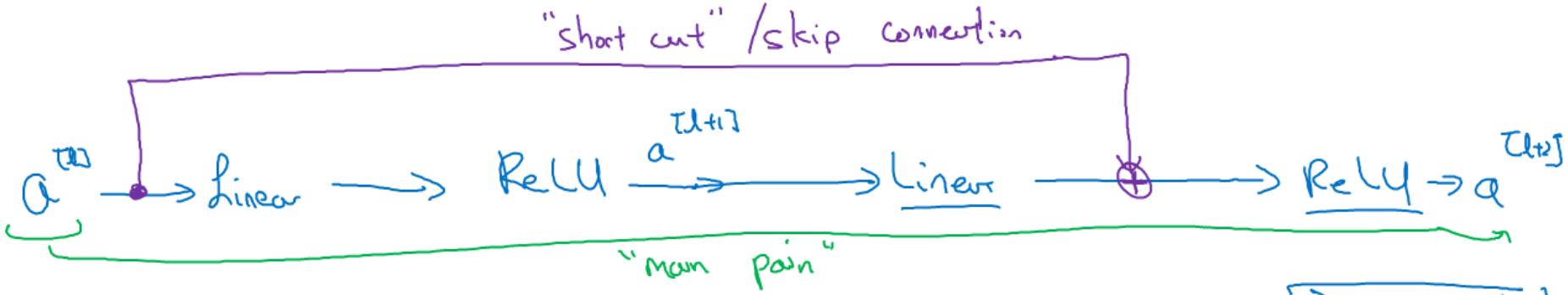
- Clásicas:
  - LeNet-5
  - AlexNet
  - VGG
- ResNet
- Inception

Ejemplo: AlexNet



Bloques residuales

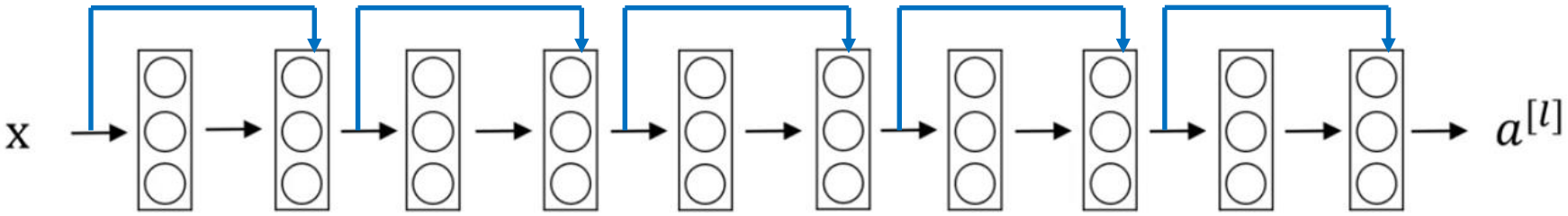


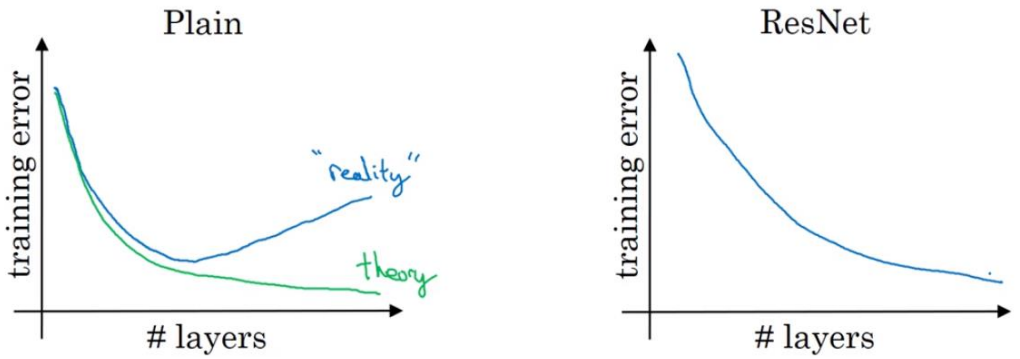


$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]} \qquad a^{[l+1]} = g(z^{[l+1]}) \qquad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]} \qquad a^{[l+2]} = g(z^{[l+2]})$$

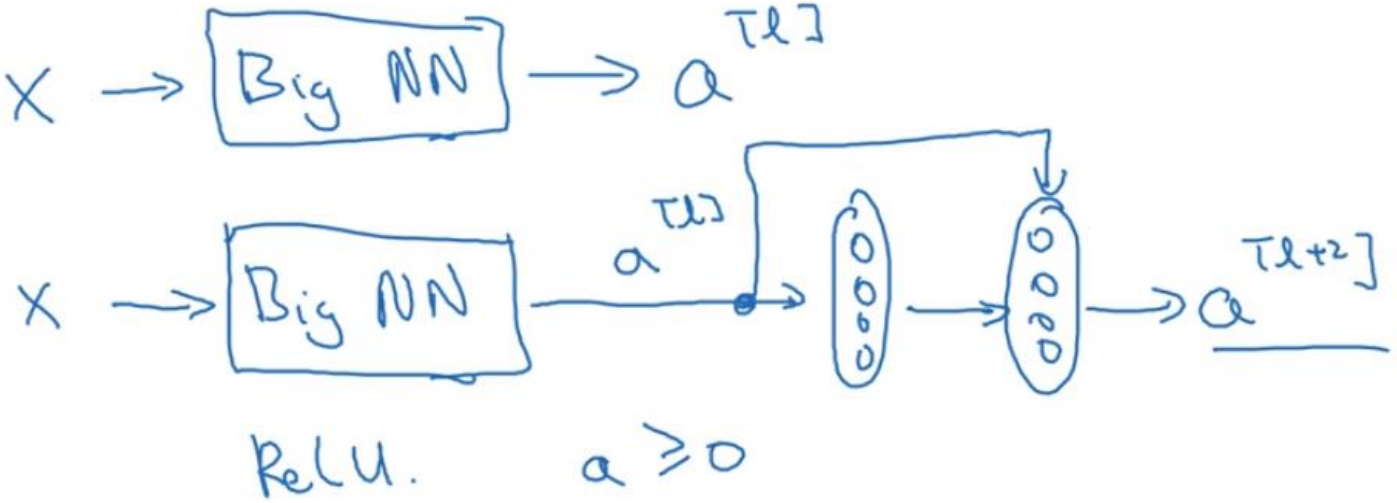
$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

ResNet


x
$a^{[l]}$


Plain
training error
"reality"
theory
layers
ResNet
training error
layers

¿Por qué funcionan las ResNet?

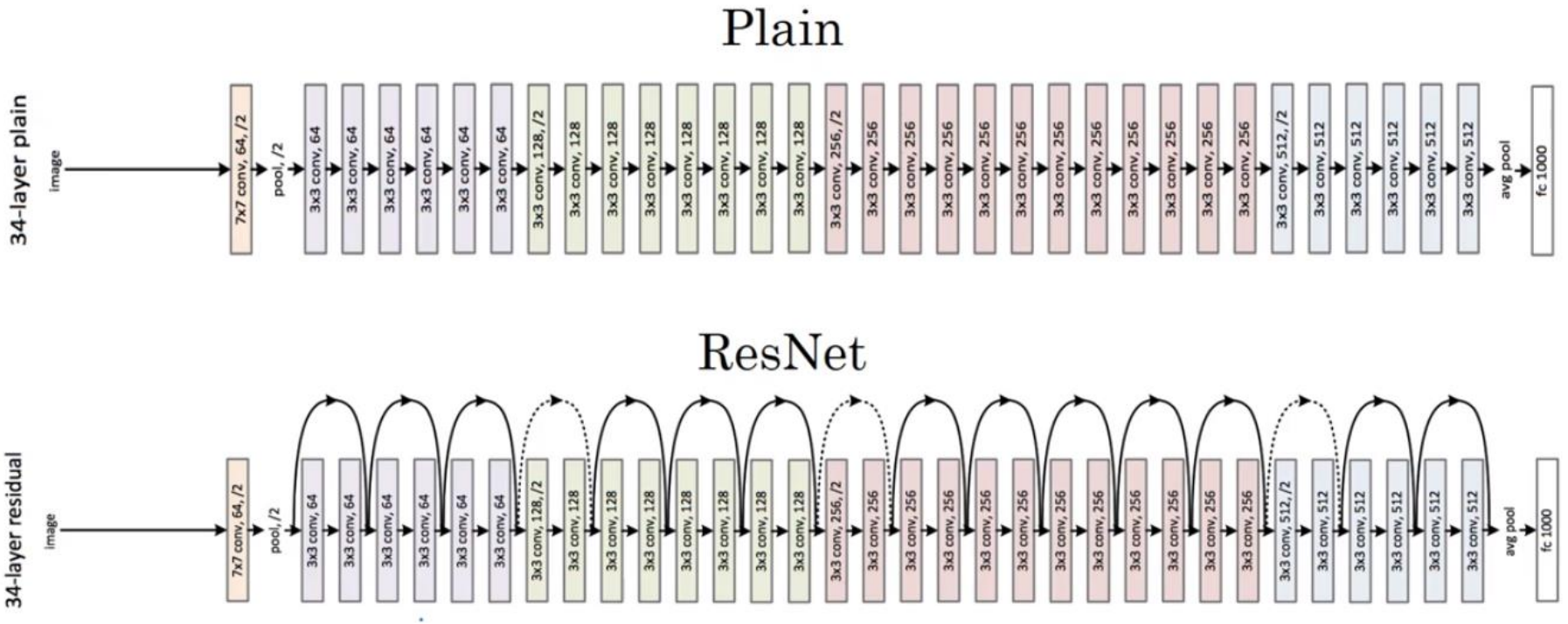


ReLU. $a \geq 0$

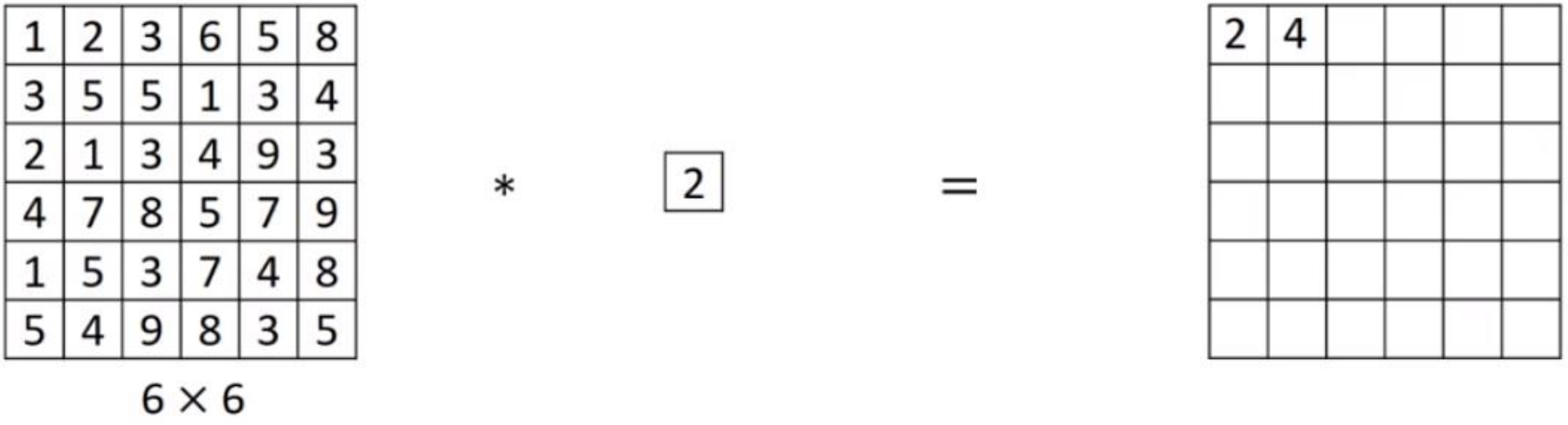
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

$$= g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

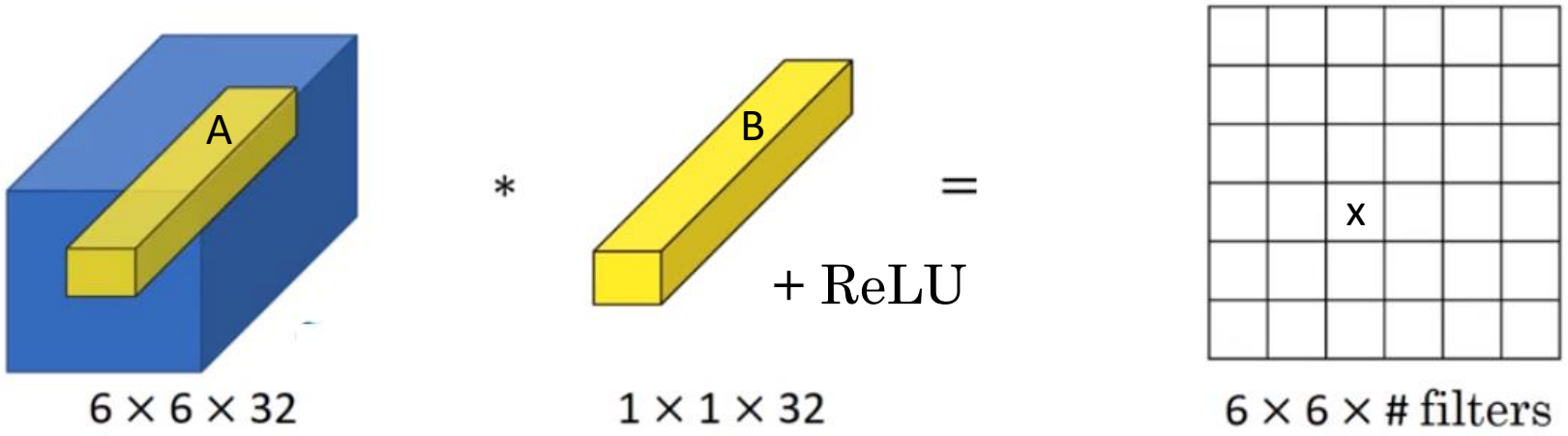
ResNet


Plain
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
ResNet
34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

Convolución 1x1



Convolución 1x1

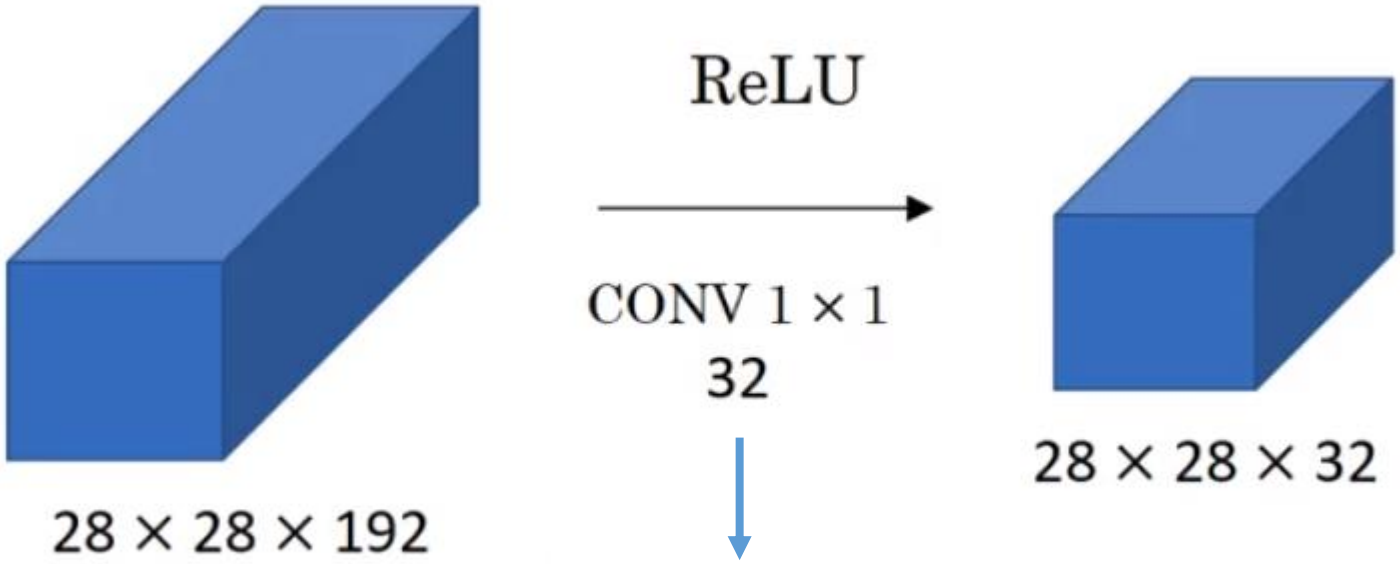


Coger los 32 valores de A, multiplicarlos elemento a elemento por los 32 valores de B, después aplicarle una función no lineal

Similar a aplicar una Fully Connected para cada bloque

También se conoce como Network in Network

Convolución 1x1



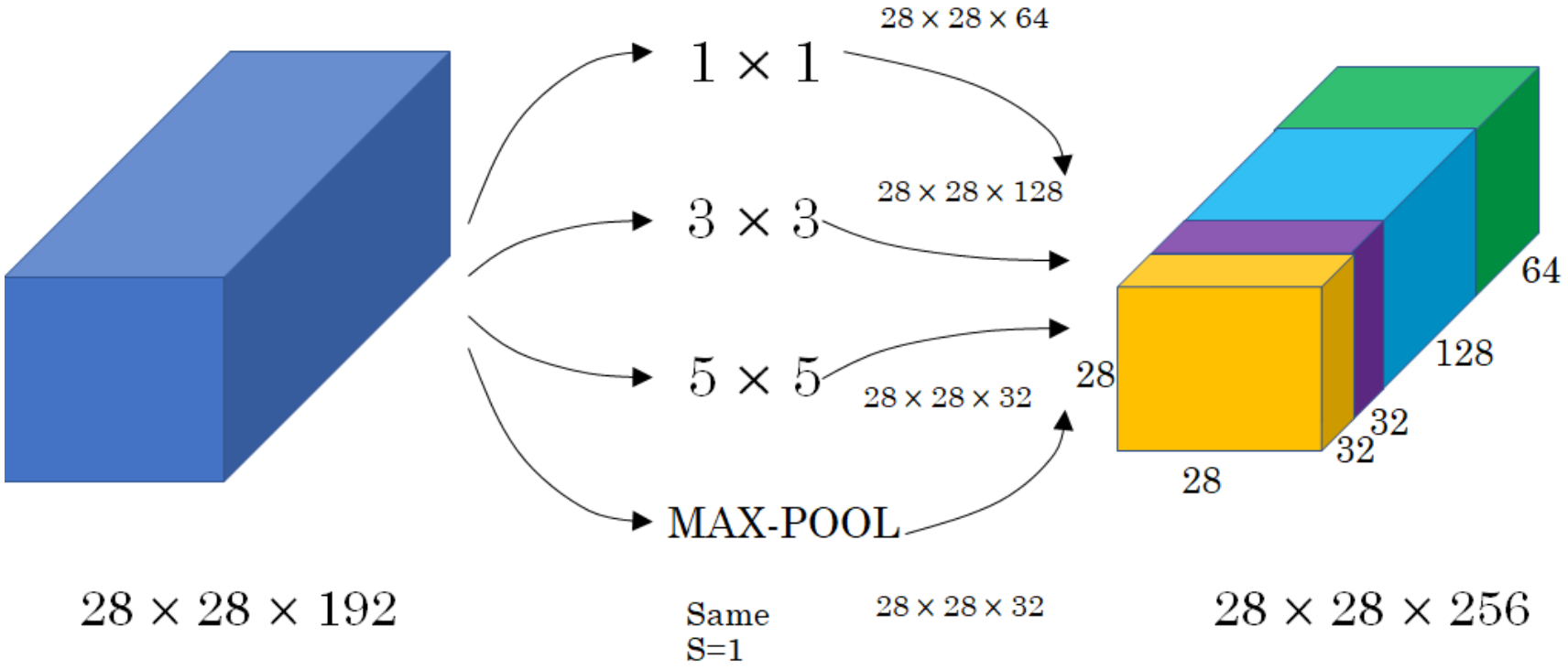
Cada filtro será 1x1x192

- Si quieres reducir la dimensión de ancho y largo → pooling layer
- Si quieres reducir el número de canales (profundidad) → filtros 1x1

Capa "inception"

- En lugar de elegir el tamaño del filtro, si usar o no pooling…

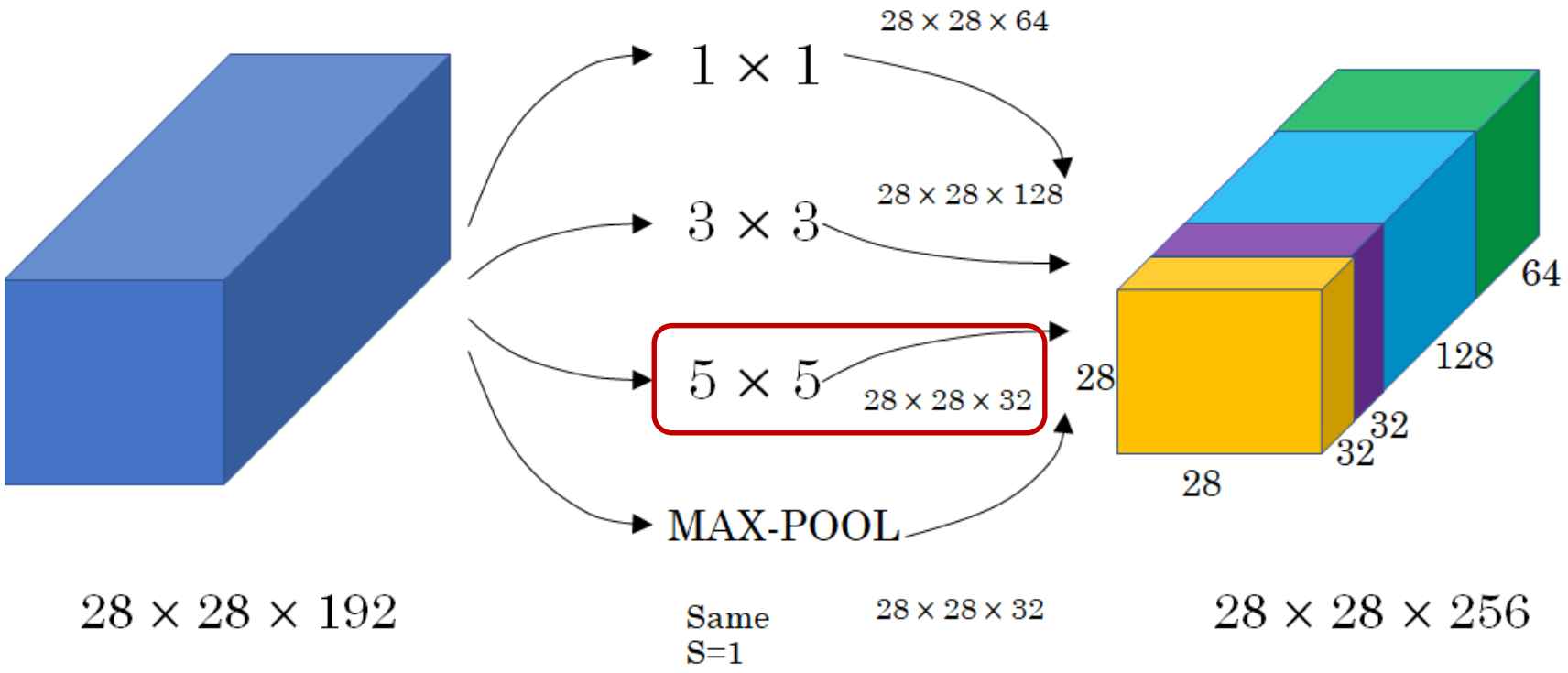
¿por qué no hacerlo todo a la vez? ☺



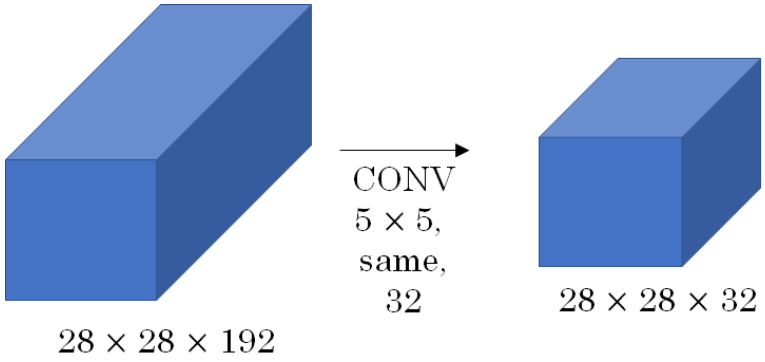
Capa "inception"

- En lugar de elegir el tamaño del filtro, si usar o no pooling…

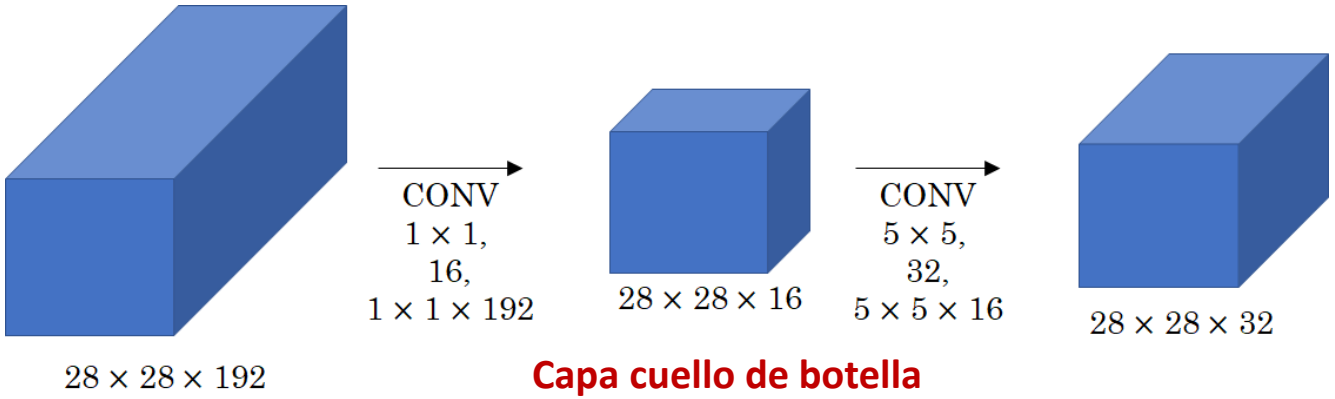
¿por qué no hacerlo todo a la vez? ☺



Inception



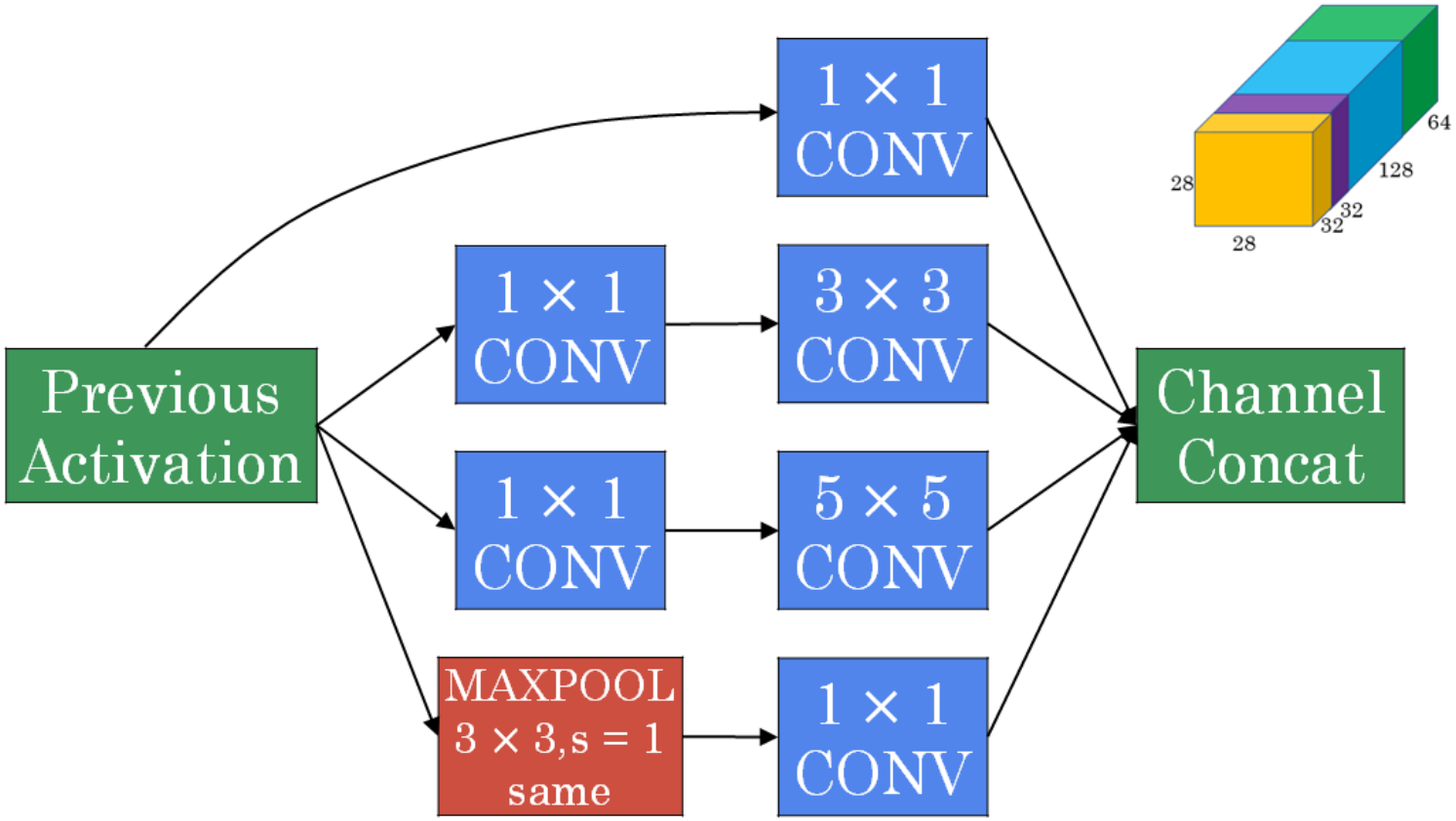
Número de multiplicaciones ~ 120 millones



Capa cuello de botella

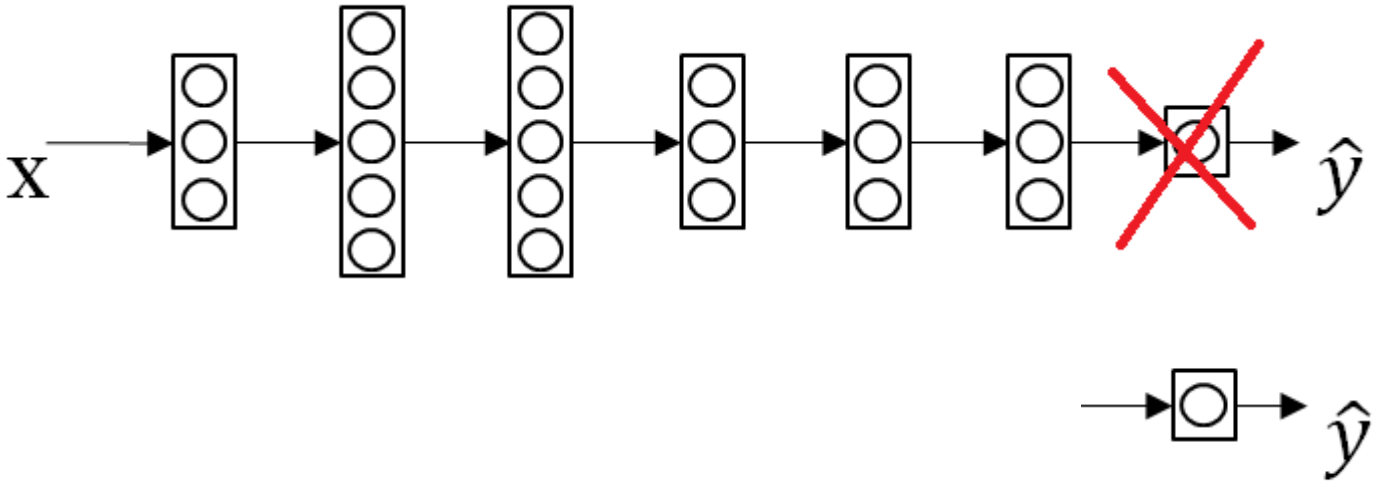
Número de multiplicaciones ~ 12 millones

Inception

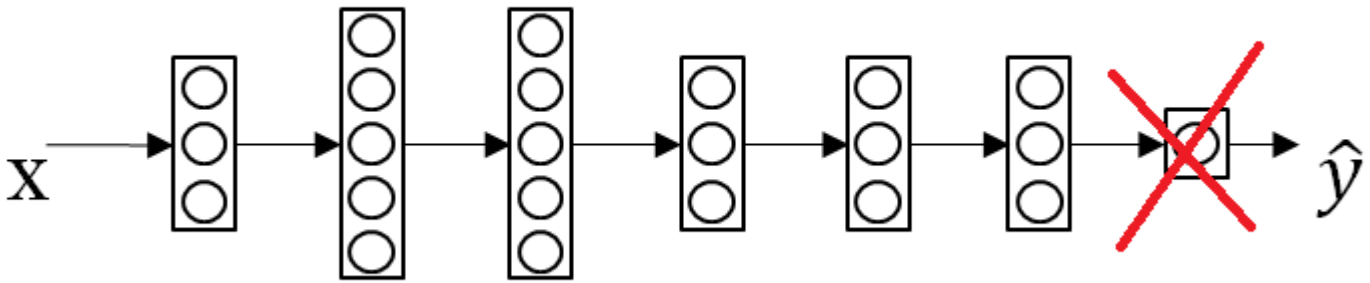

Previous Activation
1 × 1 CONV
1 × 1 CONV
3 × 3 CONV
1 × 1 CONV
5 × 5 CONV
MAXPOOL 3 × 3,s = 1 same
1 × 1 CONV
Channel Concat
64
128
28
32
32
28

Transfer Learning

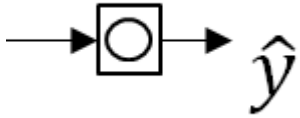
- Una de las ideas más interesantes de Deep Learning es que uno puede aprovechar una red ya entrenada para un problema diferente
- Imaginemos que tenemos una red que se ocupa de clasificar imágenes
- Podemos eliminar la última capa y sus pesos y crear otra capa con pesos aleatorios



Transfer Learning



Clasificación imágenes



Diagnóstico médico

- 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

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:

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

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

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

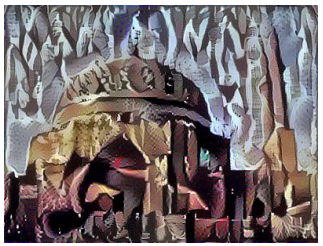
α y β son dos hiperparámetros que podemos tunear para decidir si queremos que la imagen se parezca más a la original C o que siga el estilo S

Neural Style Transfer

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

- Aplica gradient descent para minimizar J(G)

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



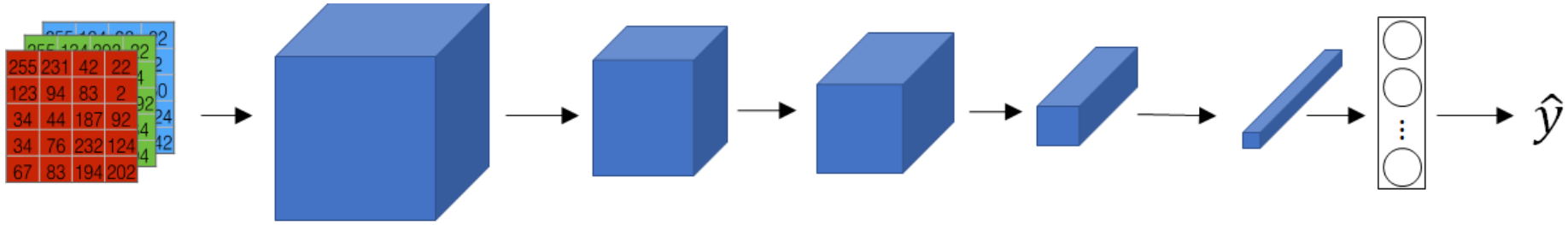
Neural Style Transfer

- 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:

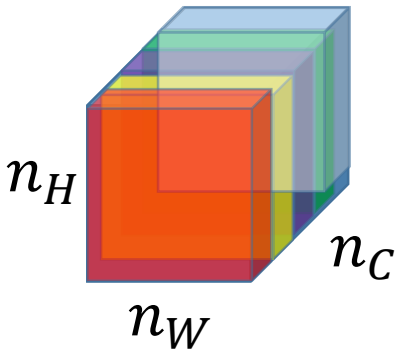
$$J_{content}(C,G) = \frac{1}{2} || \; a^{[l](C)} - a^{[l](G)} \; ||^2$$

Neural Style Transfer

- ¿Qué es el estilo?

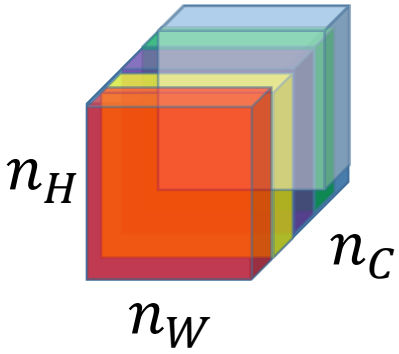


- Cogemos la capa l para medir el estilo
- Definimos el estilo como la **correlación entre diferentes canales**



¿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 $a^{[l]}_{(i,j,k)}$ donde (i,j,k) → (ancho, alto, canal) de la capa l

- $G^{[l]}_{kk`} = \sum_i \sum_j a^{[l]}_{ijk} a^{[l]}_{ijk`}$ → Si estos pares de activaciones tienen un valor elevado G tendrá un valor elevado

- Hacemos esto para la imagen S y G

- $J(S,G) = \frac{1}{(2\, n_h^{[l]} n_w^{[l]} n_c^{[l]})^2} || G^{[l](S)}_{kk`} - G^{[l](G)}_{kk`} ||^2$

En realidad esto da igual porque tenemos el término β

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, Inception

- Transferencia de aprendizaje

- Neural Style Transfer

Art Generator

http://193.146.75.223:5000/

https://github.com/laramaktub/artgenerator

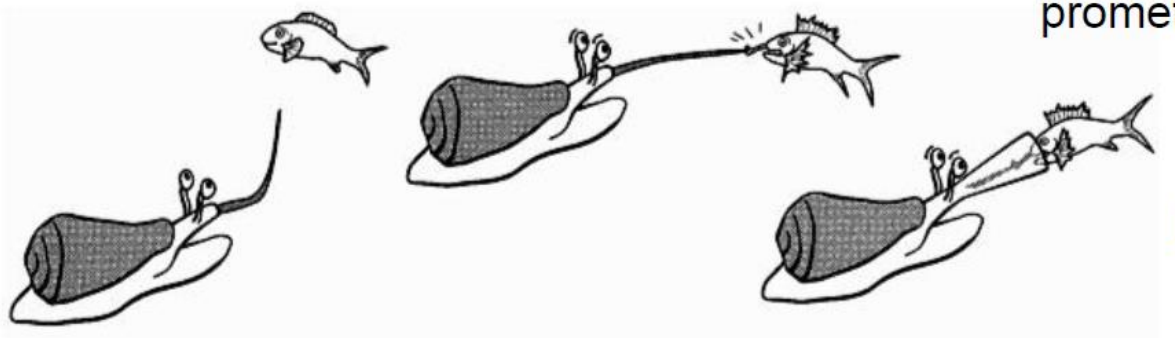
arXiv:1508.06576

Clasificando Conus

- **Training dataset**

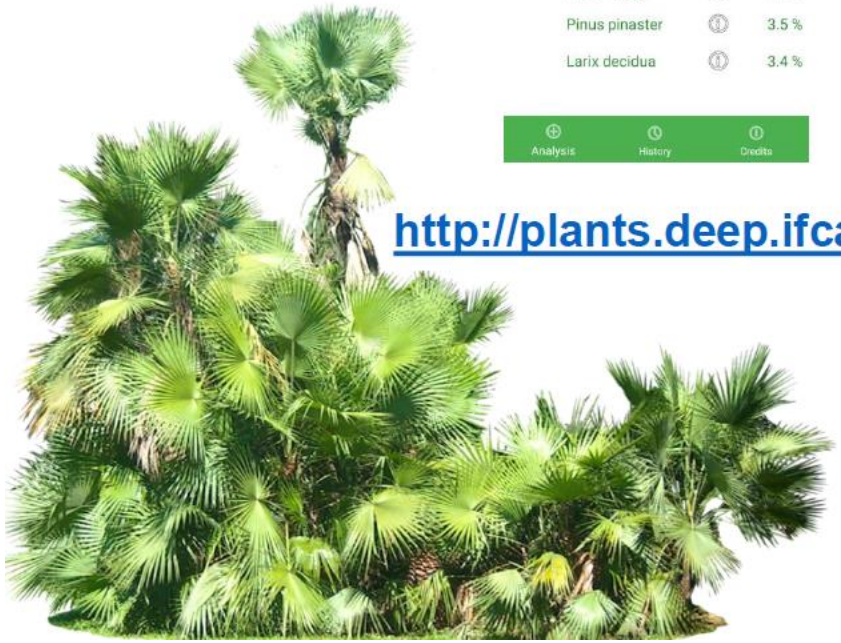
  Colección de imágenes de expertos (*68 especies | 1.5K imágenes*) que cubren tres regiones diferentes:
  - Región Panámica
  - Región de África del Sur
  - Atlántico Occidental y Mediterráneo
- **Resultados**

Los resultados usando sets de imágenes de Google son prometedores



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

Clasificación de plantas



http://plants.deep.ifca.es/

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.

Backup

Solución

https://github.com/laramaktub/MachineLearningI

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```