

# Redes convolucionales

Programa ejecutivo de Inteligencia Artificial

Año de realización: 2019-2020

**Alberto Torres Barrán**  
[alberto.torres@icmat.es](mailto:alberto.torres@icmat.es)



# Introducción

# Repaso de Deep Learning

- Modelo principal: red profunda (**feed-forward**): composición de **proyecciones lineales** y **no-linealidades**

$$\begin{aligned} h^{(i+1)} &= Wz^{(i)} + b \\ z^{(i+1)} &= \sigma(h^{(i+1)}) \end{aligned}$$

- Al final: añadir coste apropiado para regresión o clasificación.
- Las redes neuronales se conocen desde mediados del siglo XX, pero su fuerte resurgimiento no fue hasta esta década:
  - Paralelización en tarjetas gráficas (**GPUs**).
  - Librerías de **diferenciación automática**.

# Diferenciación Automática (AD)

- ¿Cómo calcular el gradiente en una red profunda?
- **A mano:** no escala a nuevas arquitecturas, propenso a errores.
- **Diferenciación numérica:** acumulación de errores y elevado coste computacional.

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji} - \epsilon)}{2\epsilon} + O(\epsilon^2)$$

- **Diferenciación simbólica:** manipulación exacta de expresiones (mediante tablas de derivadas), pero explosión en la cantidad de términos:
- Surge la **diferenciación automática o algorítmica:** aplica diferenciación simbólica pero solo a expresiones simples, y al componerlas, actualiza los resultados numéricos parciales (que serán **exactos**)

# Optimizando mediante SGD

- Una vez hemos calculado el gradiente en un punto mediante AD, las opciones actuales más populares son
- **Descenso por el gradiente estocástico (SGD):**
  1. Muestrear mini-batch
  2. Propagar hacia delante (forward) y calcular función de pérdida
  3. Propagar hacia atrás, calculando los gradientes
  4. Actualizar los parámetros
- Una **epoca** consiste en repetir la iteración anterior hasta muestrear todos los datos

# Volviendo a Deep Learning

- Ya tenemos todos los ingredientes:
  - Conjuntos de datos muy grandes.
  - Redes neuronales como **aproximadores universales**.
  - Librerías para **diferenciación automática**: tensorflow, keras, pytorch...
  - Potentes CPUs o GPUs para **paralelizar a lo largo de cada ejemplo del mini-batch**.
- ¿Qué falta en muchas ocasiones?
  - Solo teóricamente las redes neuronales son totalmente expresivas.
  - Conviene añadir un **sesgo inductivo (inductive bias)** para ayudar al aprendizaje:
    - Imágenes, señales: invarianza a traslaciones, escala: **redes convolucionales**.
    - Texto, secuencias: sensibilidad al orden de los símbolos: **redes recurrentes**.

# Redes convolucionales

# Introducción

- Tipo de red neuronal para datos con topología similar a una rejilla
  1. 1D, series temporales, audio
  2. 2D, imágenes, datos espaciales
  3. 3D, video, datos espacio-temporales, meteorología
- Red convolucional: en al menos una capa se usan convoluciones en lugar de operaciones con matrices

# Motivación

Algunas aplicaciones de redes convolucionales

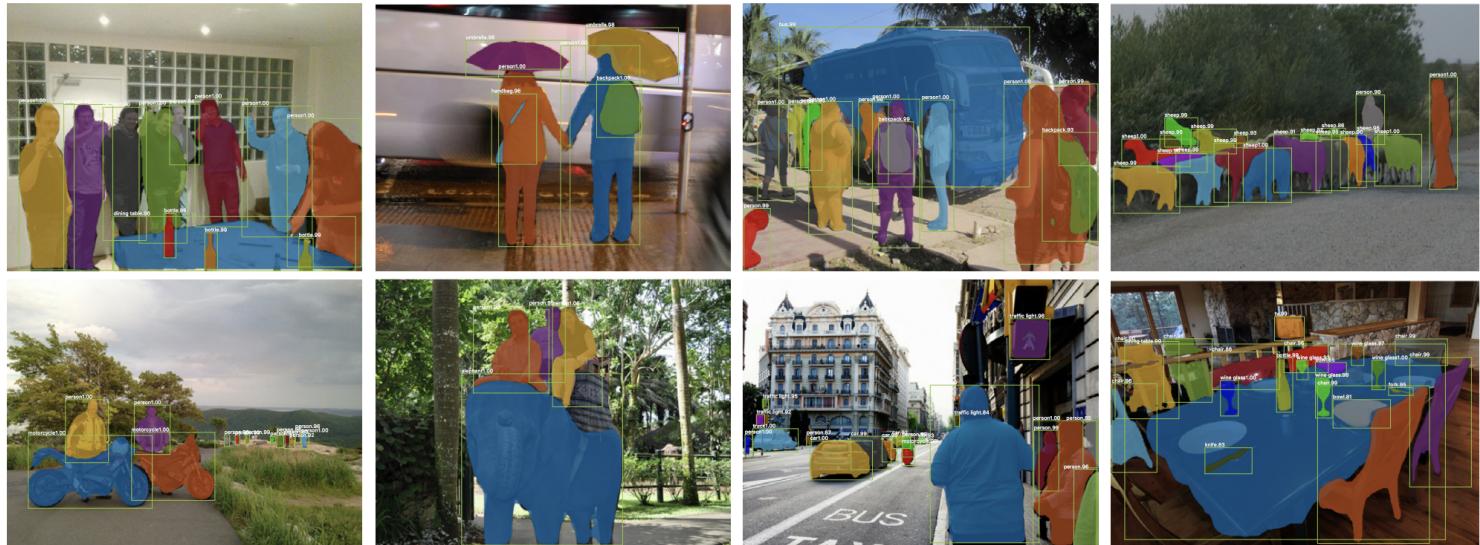
- Clasificación imágenes: por ej. detección cáncer de piel
- Detección de objetos
- Segmentación de objetos
- Transferencia de estilo
- Colorear imágenes

# Detección de objetos



Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 779-788).

# Segmentación de objetos



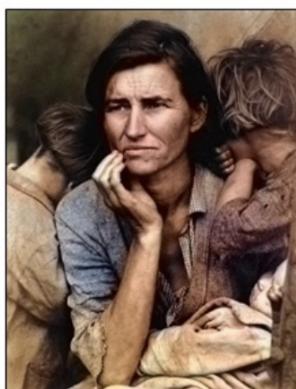
He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN." In Proceedings of the IEEE International Conference on Computer Vision, pp. 2961-2969. 2017

# Transferencia de estilo



Gatys, Leon A.; Ecker, Alexander S.; Bethge, Matthias. Image style transfer using convolutional neural networks. En Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 2414-2423.

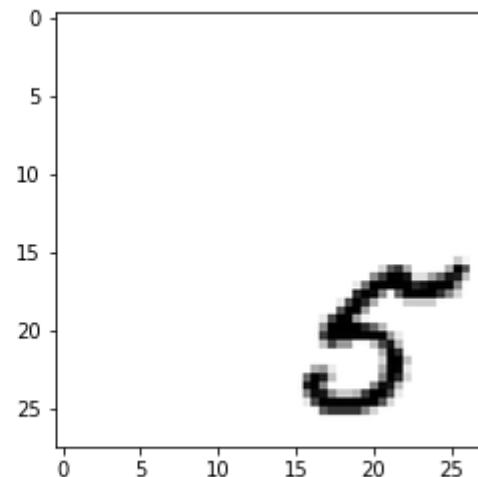
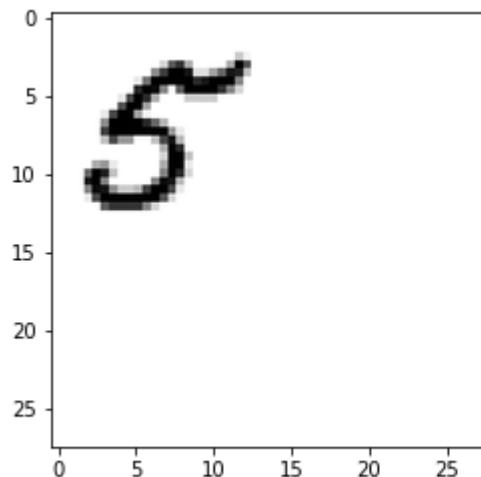
# Colorear imágenes



Zhang, Richard; Isola, Phillip; Efros, Alexei A. Colorful image colorization. En European conference on computer vision. Springer, Cham, 2016. p. 649-666.

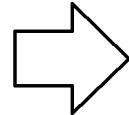
# Clasificar imágenes es complicado

- Cambios en iluminación, contraste, punto de vista
- O simplemente traslaciones:



# Aproximaciones tradicionales

- Crear variables de forma manual
- Preprocesar las imágenes: centrar, recortar, ...



Raschka, Sebastian. STAT 479: Deep Learning, Spring 2019. Introduction to Convolutional Neural Networks Part 1

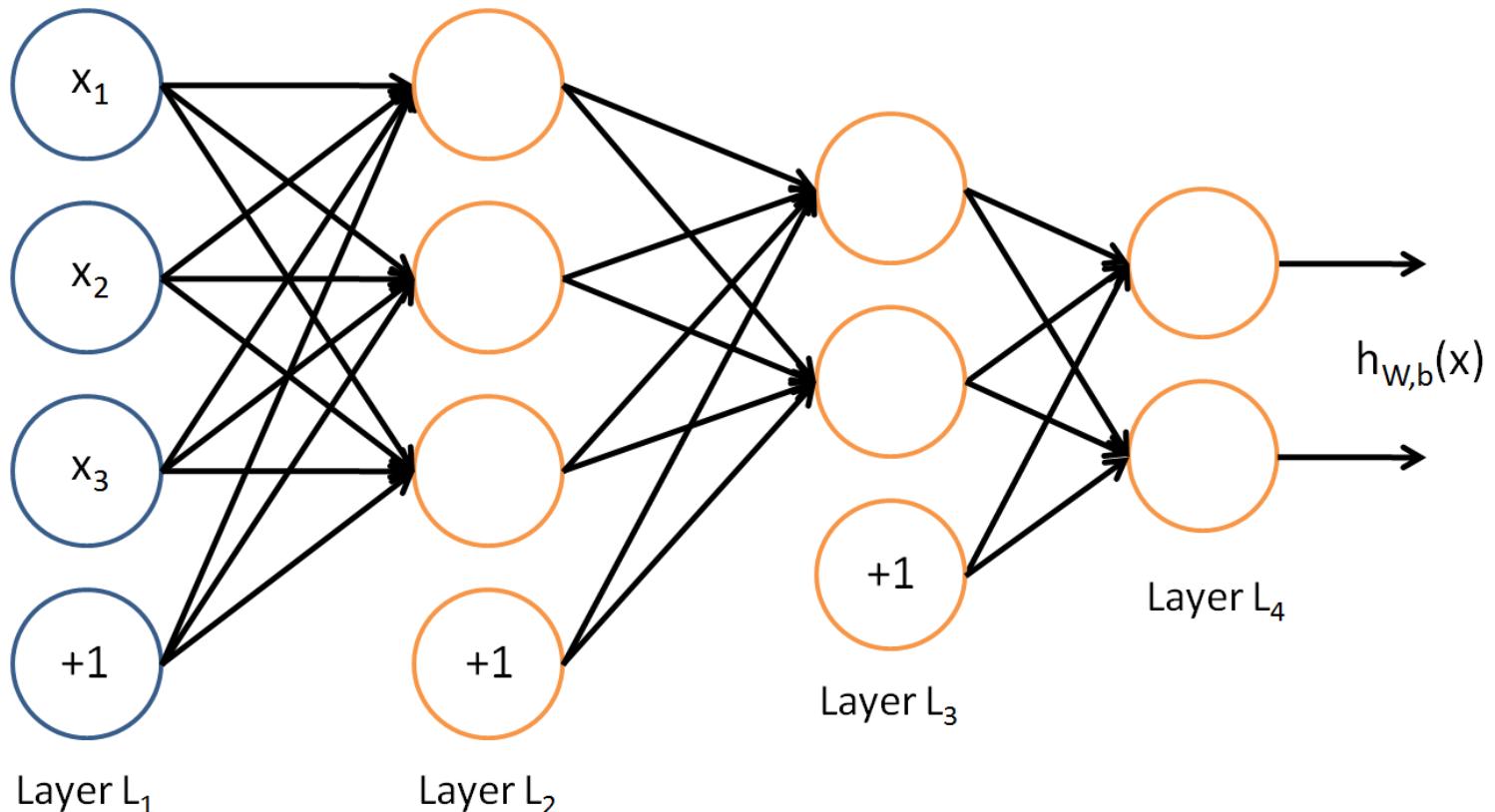
# Competición ImageNet

- Más de 14 millones de imágenes anotadas a mano
- Más de 20,000 categorías
- Desde 2010, competición anual de clasificación automática (ILSVRC)
  - únicamente 1000 categorías
  - en 2011, el mejor error era de aprox. 25%
  - en 2017, 29/38 equipos tenían un error menor del 5%

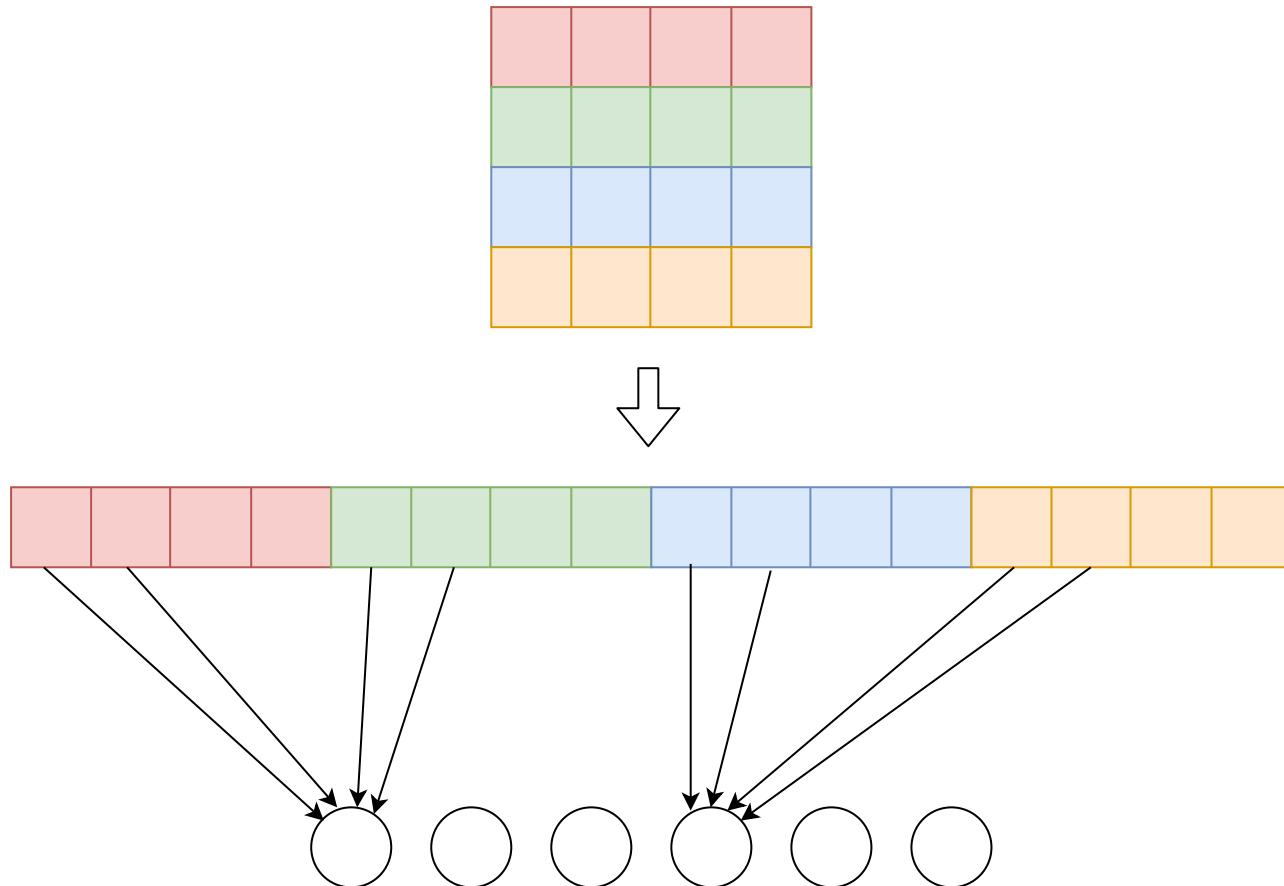
# Historia

1. En 1990, **Lecun et al.** usa una CNN para leer dígitos de códigos postales
  - una de las primeras aplicaciones reales de una red neuronal
  - más del 90% de tasa de acierto
2. En 2012, **Krizhevsky et al.** usan una CNN para ganar la competición ILSVRC2012
  - tasa de acierto (top 5), 15.3%
  - segundo mejor modelo, 26.2%
3. A partir de 2012 múltiples arquitecturas más complejas siguen reduciendo el error:
  - 2014: VGG-16 (7.3%), GoogleNet (6.7%)
  - 2015: Microsoft ResNet (3.57%)

# Conexiones densas (*fully connected*)



# Conexiones sparse (*locally connected*)



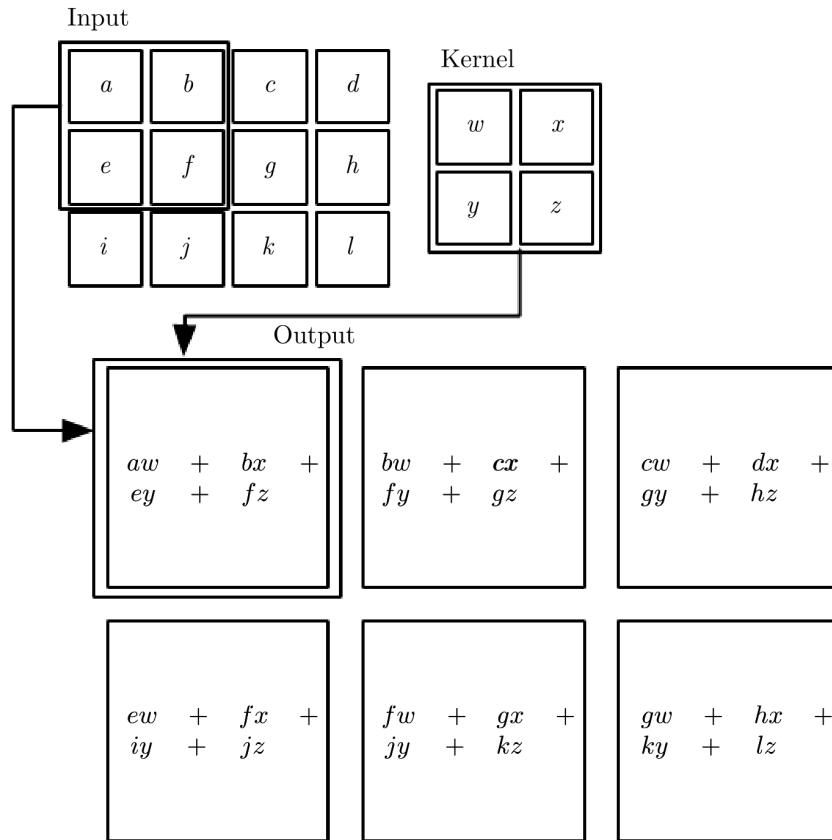
# Convolución en 2D

- $I$  es la matriz de entrada (2D)
- $K$  es el kernel (2D)

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

- La convolución o filtro se aplica a toda la imagen con los mismos pesos
- Se define con 4 parámetros:
  - *stride* o paso de la convolución
  - tamaño del kernel, generalmente cuadrado
  - *depth*, número de filtros o convoluciones distintas a aplicar
  - *padding*

# Ejemplo



Goodfellow et al. Deep Learning (2016)

# Motivación

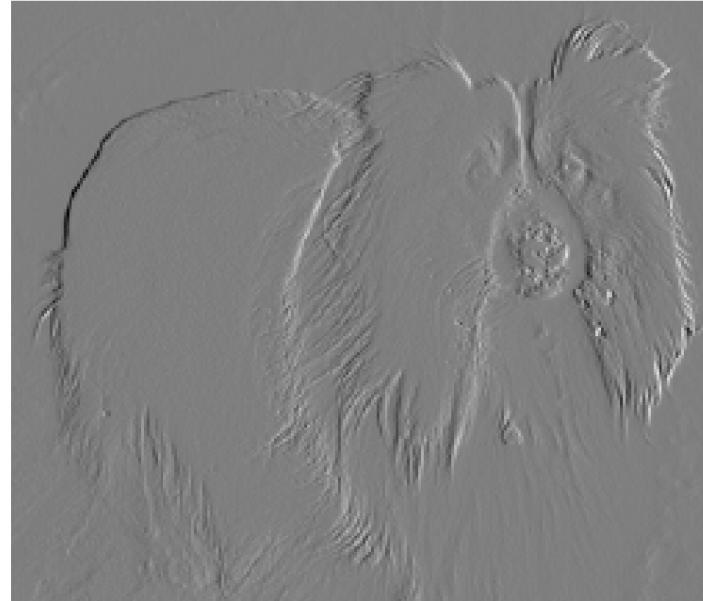
## 1. conexiones dispersas

- explotar estructura espacial
- detectar características locales (aristas, etc.)

## 2. compartición de pesos

- invariante frente a traslaciones
- reduce la cantidad de memoria necesaria

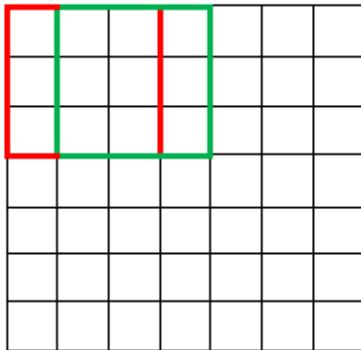
# Ejemplo características locales



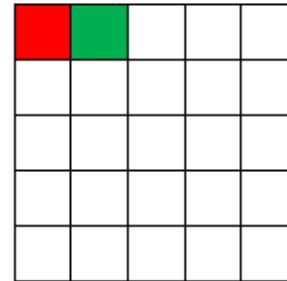
- Imagen de la derecha: restar a cada píxel su vecino por la izquierda
- Esta operación se puede representar de forma muy eficiente con una convolución

# *Stride (paso)*

7 x 7 Input Volume

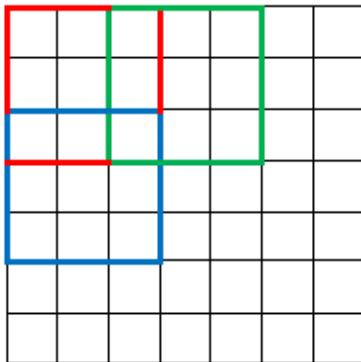


5 x 5 Output Volume

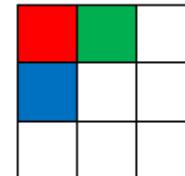


**stride = 1**

7 x 7 Input Volume



3 x 3 Output Volume

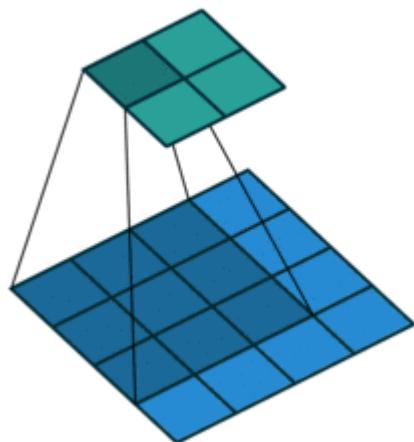


**stride = 2**

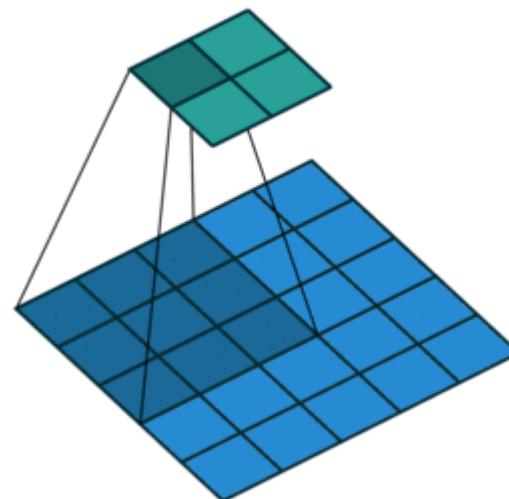
Fuente

# Ejemplos

- Entrada  $4 \times 4$
- Kernel  $3 \times 3$
- Stride 1
- Salida  $2 \times 2$

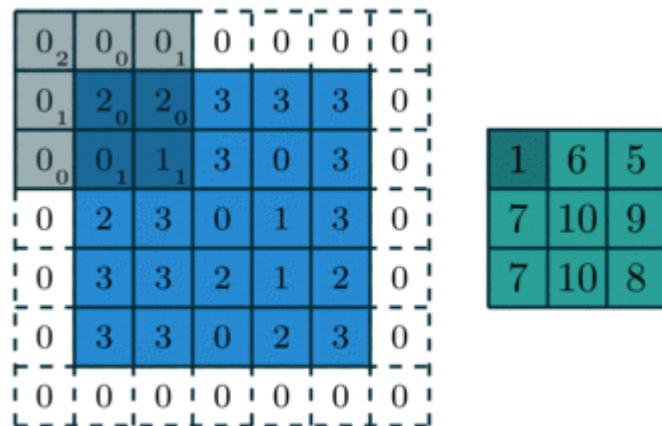


- Entrada  $5 \times 5$
- Kernel  $3 \times 3$
- Stride 2
- Salida  $2 \times 2$

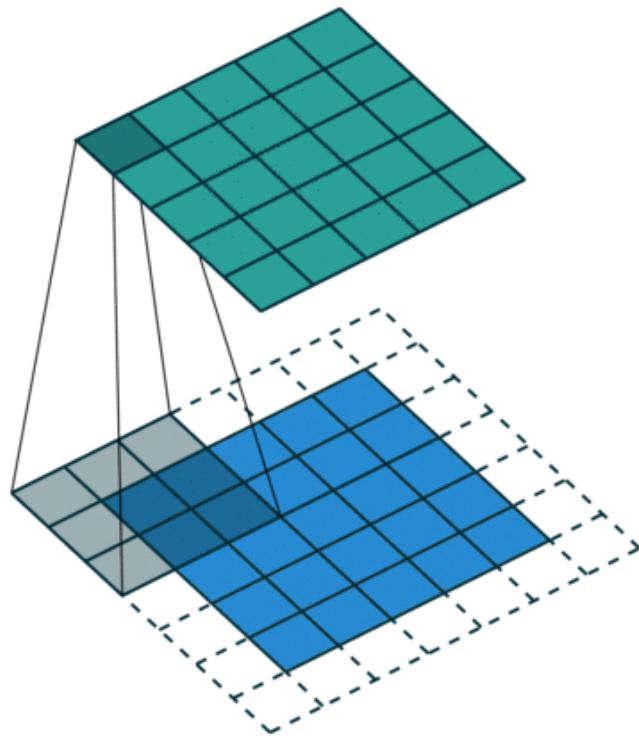


# Padding

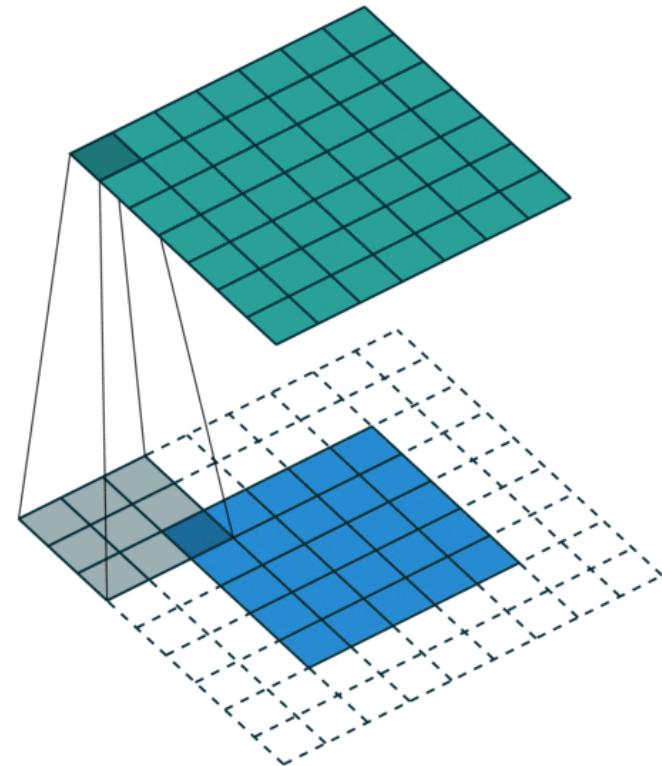
- En ocasiones se añade un *padding* de 0 al borde de la imagen:
  1. preservar el tamaño de la entrada
  2. cuando es necesario por la combinación de tamaño de entrada, tamaño de kernel y stride
- Ejemplo: entrada  $5 \times 5$ , kernel  $3 \times 3$  y *stride* 2



- Generalmente la salida tiene menor tamaño que la entrada
- Aplicando padding podemos hacer que tenga el mismo o incluso mayor
- Entrada  $5 \times 5$ , stride 1, kernel  $3 \times 3$

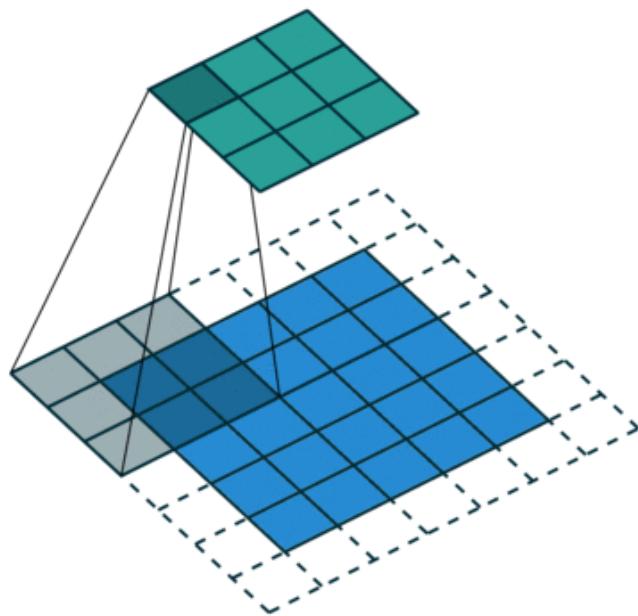


Salida  $5 \times 5$

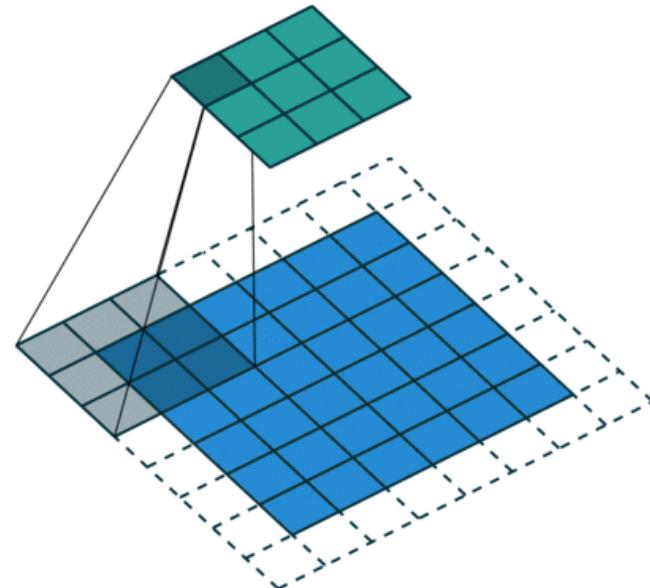


Salida  $7 \times 7$

- A veces el padding es necesario para poder aplicar el kernel
- Ejemplo: kernel  $3 \times 4$ , stride 2
- La salida tiene el mismo tamaño en ambos!



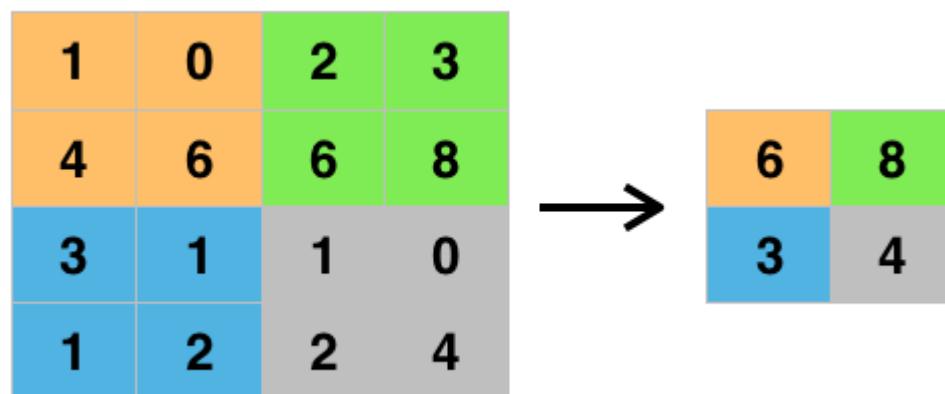
Entrada  $5 \times 5$



Entrada  $6 \times 6$

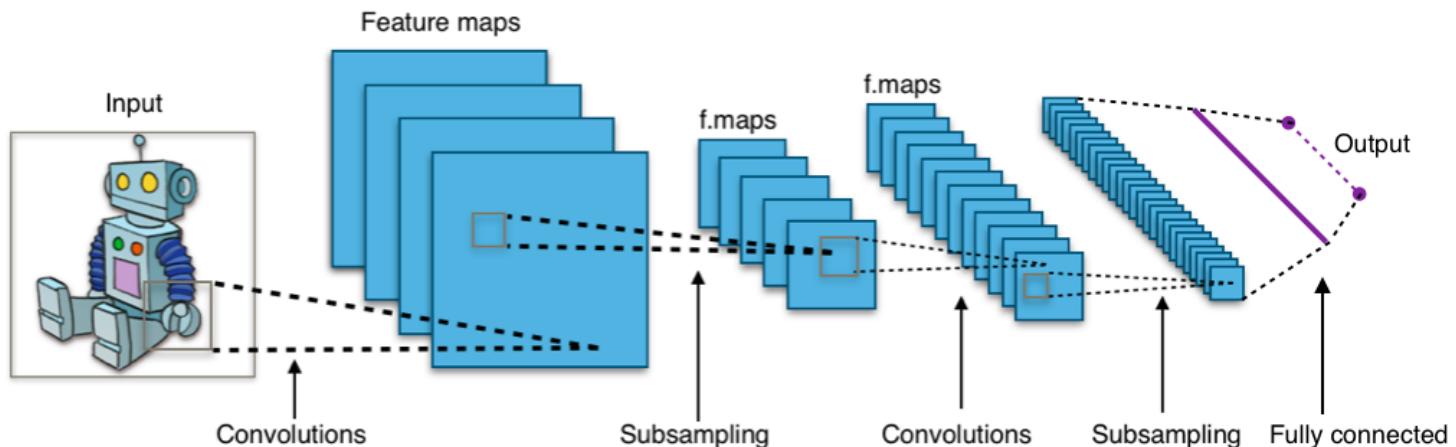
# Pooling

- Capa de submuestreo no lineal
- Previene sobreajuste, reduciendo número de parámetros
- Ayuda con la invarianza frente a traslaciones
- Útil cuando interesa conocer si una característica está o no, pero no su localización exacta  $\Rightarrow$  clasificación imágenes
- Más habitual: **max pooling**

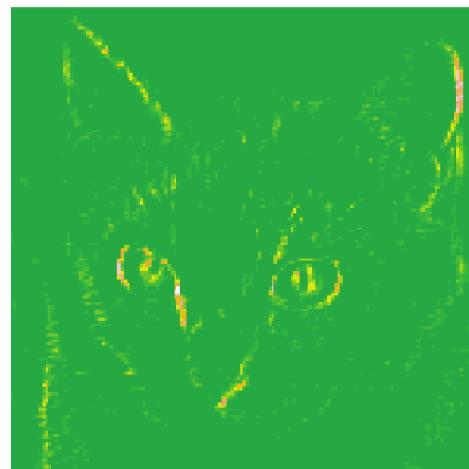
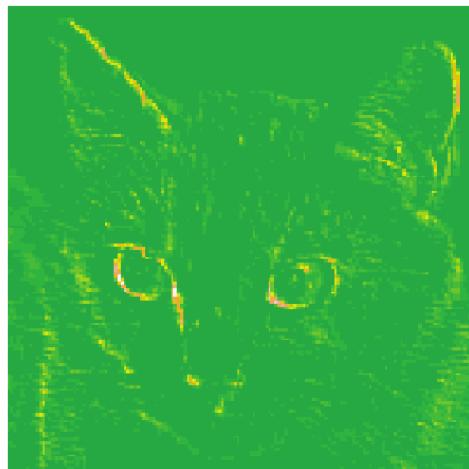


# Arquitectura típica

- Parámetros: número de *feature maps*, tamaño del kernel, stride
- Subsampling: max pooling
- Antes de las capas *fully connected*, hay que aplanar (*flatten*) la salida



# Visualizando activaciones



# Primeras capas

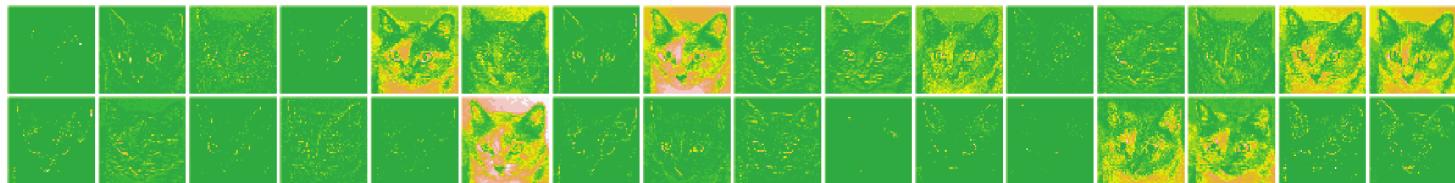


Figure 5.20 conv2d\_5

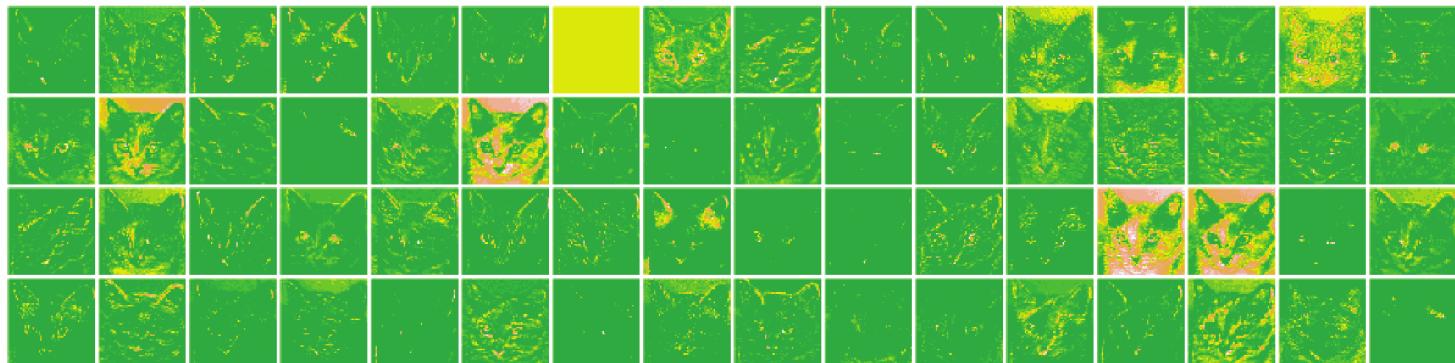


Figure 5.21 conv2d\_6

# Últimas capas

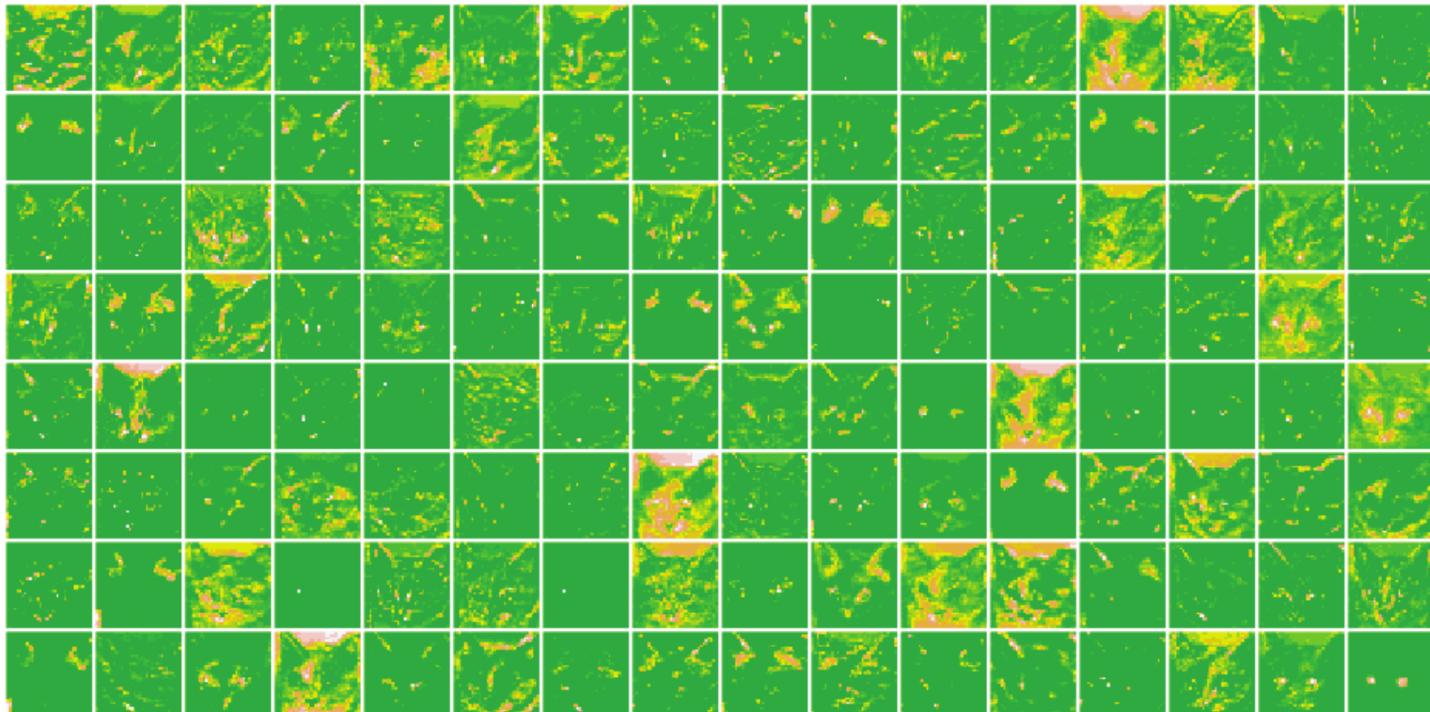


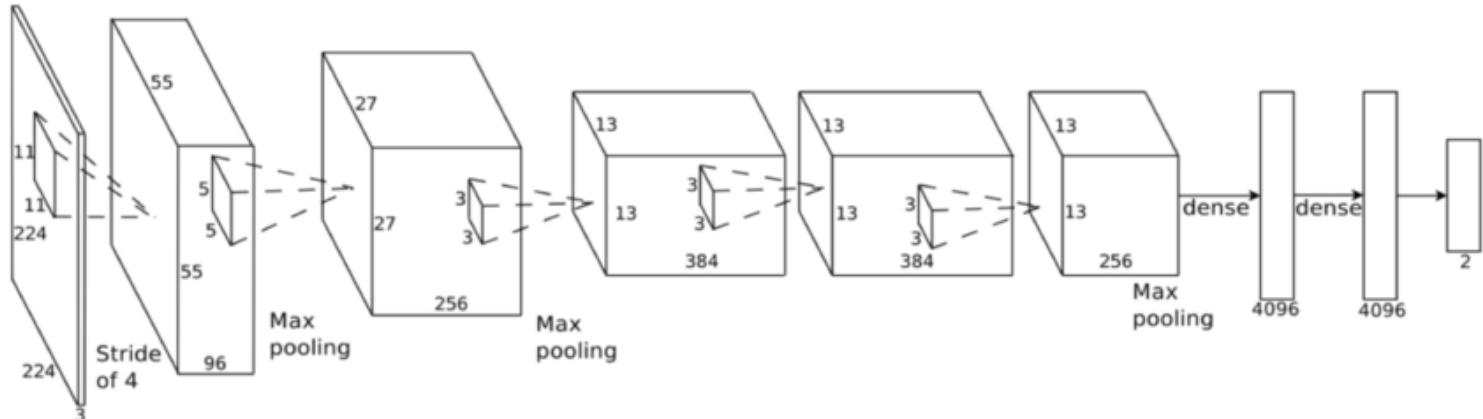
Figure 5.22 conv2d\_7

# CAM (class activation map)



Figure 5.32 Superimposing the class-activation heatmap on the original picture

# Ejemplo arquitectura: AlexNet



```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 96,
                kernel_size = c(11, 11),
                activation = 'relu',
                input_shape = c(224, 224, 3),
                strides = c(4, 4),
                padding = 'valid') %>%
  layer_max_pooling_2d(pool_size = c(2, 2),
                      strides = c(2, 2),
                      padding = 'valid') %>%
  ...
```

# Recursos adicionales

# Enlaces de interés

- <https://reddit.com/r/LearnMachineLearning>: nivel introductorio/medio
- <https://reddit.com/r/machinelearning>: discusiones sobre artículos y temas de actualidad
- <https://medium.com/topic/machine-learning>: artículos hacia audiencia general