

Redes neuronales convolucionales

Aprendizaje profundo

Departamento de Sistemas Informáticos

E.T.S.I. de Sistemas Informáticos - UPM

5 de marzo de 2024



Contexto actual

Las redes convolucionales (CNN) son una de las principales arquitecturas usadas



Tendencia de diferentes técnicas de deep learning a lo largo de los años

Tendencia de diferentes técnicas de deep learning a lo largo de los años.

Fuente: *An Introductory Review of Deep Learning for Prediction Models With Big Data*

Son la técnica predominante a la hora de procesar *imágenes* y *datos tabulares*

¹ Emmert-Streib, F., Yang, Z., Feng, H., Tripathi, S., & Dehmer, M. (2020). [An introductory review of deep learning for prediction models with big data](#). Frontiers in Artificial Intelligence, 3, 4.

Motivación

Surgen para adaptar las redes neuronales al tratamiento de imágenes

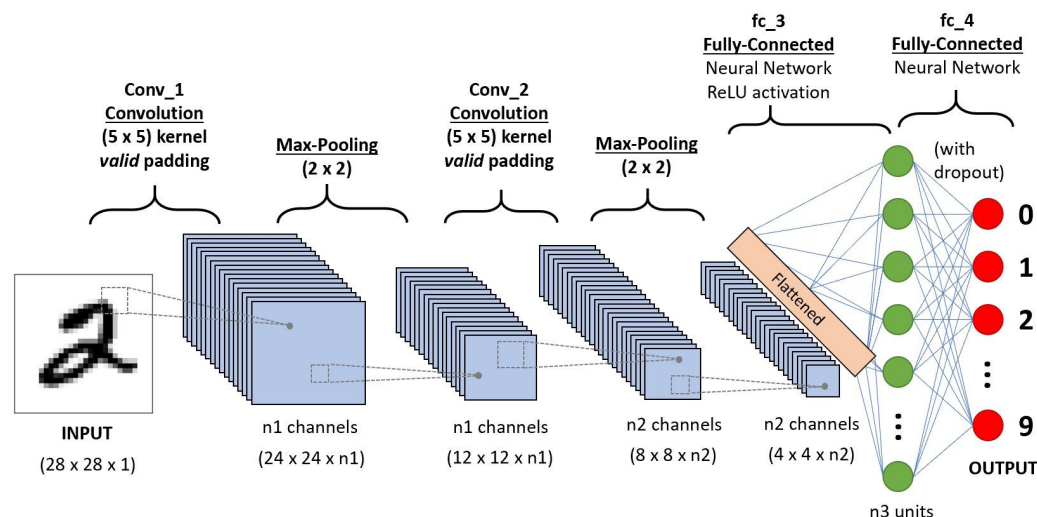
- Aprovechan las características de los datos espaciales para reducir el número de parámetros de la red
- Aprenden la invarianza de los datos, lo que les permite generalizar mejor
- Son capaces de extraer características jerárquicas de los datos, ayudando a identificar patrones complejos

Se apoyan en la operación de ***convolución*** para procesar los datos

Arquitectura de una CNN

Se divide en dos partes para el proceso de datos espaciales:

1. **Extracción de características:** Capas de convolución, *pooling* y normalización
2. **Inferencia:** Capas densas (MLP)



Arquitectura de una CNN. Fuente: [Analytics Vidhya](#)


Operación de convolución

¿Qué es una operación de convolución?


En nuestro contexto definiremos la convolución como operación que **procesa una matriz numérica manteniendo las relaciones espaciales de la misma**

- Se aplica un **filtro** (o **kernel**) a la matriz de entrada produciendo una salida denominada **mapa de características**
- En visión artificial, se han utilizado tradicionalmente para producir efectos


Ejemplo de convolución: Desenfoque tipo caja

 Ejemplo de desenfoque tipo caja
Ejemplo de desenfoque tipo caja.

Ejemplo de convolución: Desenfoque tipo caja

 Ejemplo de desenfoque gaussiano
Ejemplo de desenfoque gaussiano.

Ejemplo de convolución: Desenfoque tipo caja

 Ejemplo de desenfoque realce de bordes
Ejemplo de desenfoque realce de bordes.

Operación de convolución (I)

Producto escalar de una matriz con un **filtro (kernel)** que se desplaza por ella



Operación de convolución

Operación de convolución con filtro 2D sobre imagen de un único canal. Fuente: [Analytics Vidhya](#)

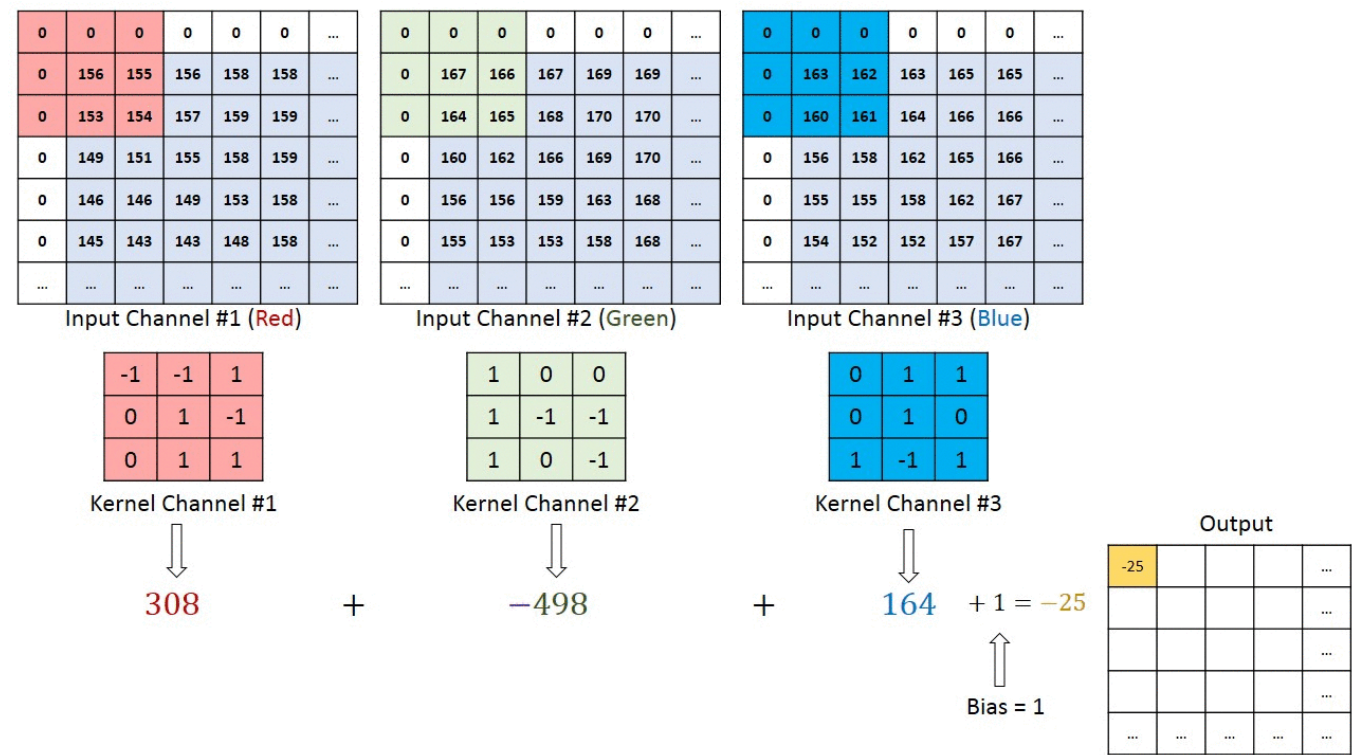
Dos elementos fundamentales:

- **Matriz de entrada:** Dos (e.g. imagen en escala de grises) o tres dimensiones (e.g. imagen a color)
- **Filtro:** Ancho y alto determinado, mientras que coincide en profundidad con la matriz de entrada

El filtro recorre la matriz de entrada haciendo el producto escalar en cada posición

Operación de convolución (II)

La región que el filtro (kérnel) es capaz de observar se denomina **campo receptivo**



Operación de convolución sobre una imagen de 3 canales. Fuente: [SaturnCloud](#)

Convolución en redes neuronales

¿Y si en lugar de filtros preconfigurados, los «aprendemos»?

- Esa es la idea detrás de las redes neuronales convolucionales
- Una **convolución neuronal** cambia los valores del núcleo por neuronas con sus propios pesos

 Convolución en redes neuronales

Pesos asociados a un filtro 2D

Activación de capas convolucionales

Tras la convolución, el resultado pasa por una función de activación no lineal



Convolución en redes neuronales

Proceso completo de obtención de mapa de características

La salida de la operación se denomina **mapa de características** del filtro

Hiperparámetros de la capa convolucional

Un poco de implementación

La capa `Conv2D` de Keras configura cada una de las capas convolucionales

```
tf.keras.layers.Conv2D(  
    filters=...,  
    kernel_size=...,  
    strides=...,  
    padding=...,  
    activation=...,  
)
```

Estos son los más comunes, presentes en prácticamente en cualquier framework

filters

Número de filtros que se aplicarán a la imagen

- Cada filtro es un conjunto de pesos que se aplican a la imagen
- Cada filtro produce un mapa de características

 Número de filtros

Capa convolucional de cinco filtros, que darán lugar a cinco mapas de características

kernel_size

Especifica el tamaño del filtro que se deslizará sobre la imagen

- Especificado como una tupla de dos enteros, (alto, ancho)
- El tamaño del filtro determina el campo receptivo de la capa



Tamaño de kernel

Filtros de tamaño 3×3 y 5×5 .

strides

Determina el salto que dará el filtro al deslizarse sobre la imagen

- Especificado como tupla de dos enteros, (alto, ancho)
- Si no se especifica, el salto es de 1 en ambas direcciones



Stride de 2x2

Salto *stride* de 2×2 para el deslizamiento del filtro. Fuente: [Towards Data Science](#)

padding

Indica cómo se rellena la imagen para que el filtro pueda deslizarse por los bordes

- `valid` : No se rellena la imagen (por defecto)
- `same` : Se rellena la imagen con ceros para que el tamaño del mapa de características sea el mismo que el de la entrada

padding same

Padding establecido como `same` , rellinando la imagen por los bordes para que el mapa de características resultante tenga el mismo tamaño que la entrada.

Fuente: [Towards Data Science](#)

activation

Define la función de activación que se aplicará a la salida de la convolución

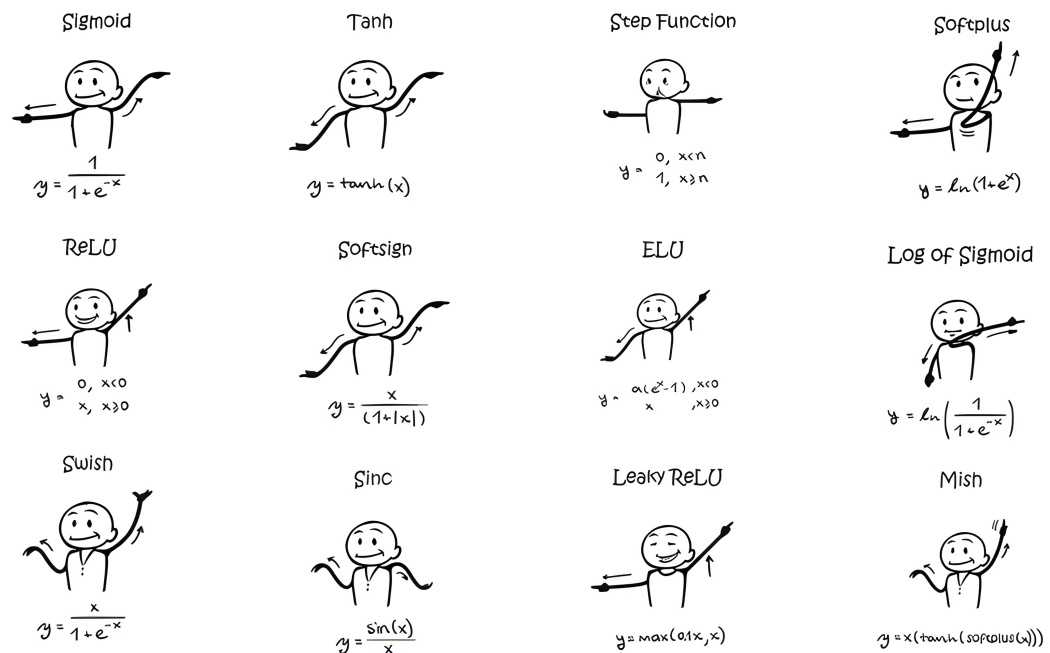


Figura 5. Algunas funciones de activación comunes.

Upsampling y downsampling

Cambios de dimensionalidad

A la hora de diseñar una red convolucional, las capas que cambian las dimensiones de la información son fundamentales

- ***Downsampling***: Reducción de la resolución espacial
- ***Upsampling***: Aumento de la resolución espacial

Las capas de convolución por defecto realizan *downsampling* de dos maneras:

- Mediante el uso del parámetro `strides`
- Mediante el uso de `padding` en la imagen (el de tipo `valid`)

Pero en DL a veces necesitamos muchas capas de convolución, y a la vez mantener, aumentar o disminuir la resolución espacial

Reducción dimensional con *pooling*

El *pooling* es una operación que reduce la resolución espacial de la imagen

padding same

Diferentes operaciones de *pooling* sobre la misma matriz. Fuente: [Towards Data Science](#)

Filtro que devuelve un valor de cada región de la imagen

- **Max pooling:** Se toma el valor máximo de una región
- **Average pooling:** Se toma el valor promedio de una región
- **Global pooling:** Se toma el valor máximo o promedio de toda la imagen

Aumento dimensional con *upsampling*

El *upsampling* es una operación que aumenta la resolución espacial de la imagen

 padding same

Ejemplo de *upsampling* mediante la técnica *bed of nails*. Fuente: [Towards Data Science](#)

Existen una amplia multitud de técnicas

- **Vecinos cercanos:** Se copia el valor de un píxel a toda la región generada
- **Interpolación:** Se rellenan los valores faltantes con valores interpolados de los píxeles vecinos
- ***Bed of nails*:** Se rellenan los valores faltantes con ceros

Strides para el cambio dimensional

Es otra alternativa para reducir la dimensión de la entrada

- Si el *stride* es mayor que 1, el filtro se desplaza más rápido por la imagen
- La salida de la convolución es más pequeña que la entrada



Stride de 2x2

Un *stride* de 2×2 se puede usar como *downsampling* «inteligentes». Fuente: [Towards Data Science](#)

La principal ventaja respecto al *pooling* es que se aprenden los pesos de los filtros

- Podemos decir que se usa un filtro «inteligente»

Filtros 1×1

Se usan en ocasiones para reducir la dimensionalidad de la imagen¹

- Disminuye la cantidad de canales (menos complejidad y cálculos),
- Aplicación de operaciones no lineales sin alterar el tamaño de la imagen, y
- Cada canal de cada píxel como entrada de una red neuronal, así aprenden transformaciones complejas a nivel de canal.



Stride de 2×2

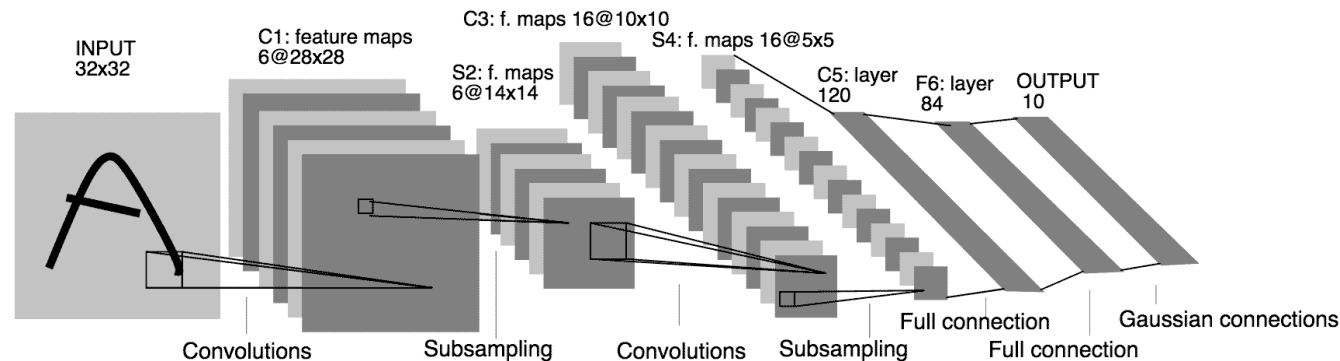
Un *stride* de 2×2 se puede usar como *downsampling* «inteligentes». Fuente: [Towards Data Science](#)

¹ Lin, M., Chen, Q., & Yan, S. (2013). [Network in network](#). arXiv preprint arXiv:1312.4400.

Clasificación de dígitos con redes convolucionales.ipynb

Diferentes arquitecturas de CNN

Desarrollada por LeCun et al.¹ en 1998 para reconocer de dígitos escritos a mano



Arquitectura LeNet-5. Fuente: [Anatomies of Intelligence](#)

Es considerada el «Hola Mundo» del aprendizaje profundo

- **Arquitectura:** Dos capas convolucionales, dos capas *pooling* y tres capas densas
- **Principal problema:** *Vanishing gradients*.

¹ LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.. Proceedings of the

AlexNet - Arquitectura que popularizó las CNN

Desarrollada por Alex Krizhevsky et al.¹ en 2012, ganadora de ImageNet 2012



Arquitectura AlexNet

Arquitectura AlexNet. Fuente: [Medium](#)

- **Arquitectura:** Cinco convolucionales, 2 *pooling* y tres densas
- **Principal aportación:** Uso de *ReLU* y *dropout* para evitar el sobreajuste
- **Problema:** *Muchos parámetros*

Esta arquitectura ha inspirado el diseño de muchas arquitecturas posteriores

¹ Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. Advances in neural information processing systems, 25, 1097-1105.

GoogLeNet (Inception v1) - Bloques *inception*

Desarrollada por Szegedy et al.¹ en 2014, ganadora de ImageNet 2014



Arquitectura GoogLeNet

Arquitectura GoogLeNet. Fuente: [Medium](#)

- **Arquitectura:** 22 capas con bloques *inception* (principal aportación)
- **Problema:** *Complejidad de implementación*

¹ Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). *Going deeper with convolutions*. Proceedings of the IEEE conference on computer vision and pattern recognition, 1, 3.

Bloque *inception*

Estructura que permite la extracción de características a diferentes escalas

- Nos permite utilizar múltiples tipos de tamaño de filtro, en lugar de uno solo
- Luego se concatena el resultado de cada filtro y pasarlo a la siguiente capa
- Sucesivas versiones han ido añadiendo mejoras al bloque

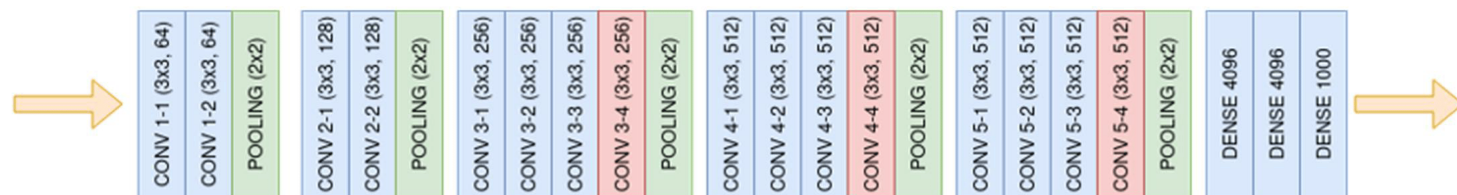
 Bloque *_inception_ v1*

Bloque *inception* v1. Fuente: [The Startup](#)

¹ Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). *Going deeper with convolutions*. Proceedings of the IEEE conference on computer vision and pattern recognition, 1, 3.

VGGNet - Arquitectura con muchas capas

Desarrollada por Simonyan y Zisserman¹ en 2014



Arquitectura VGGNet-19. Fuente: [Electronics \(MDPI\)](#)

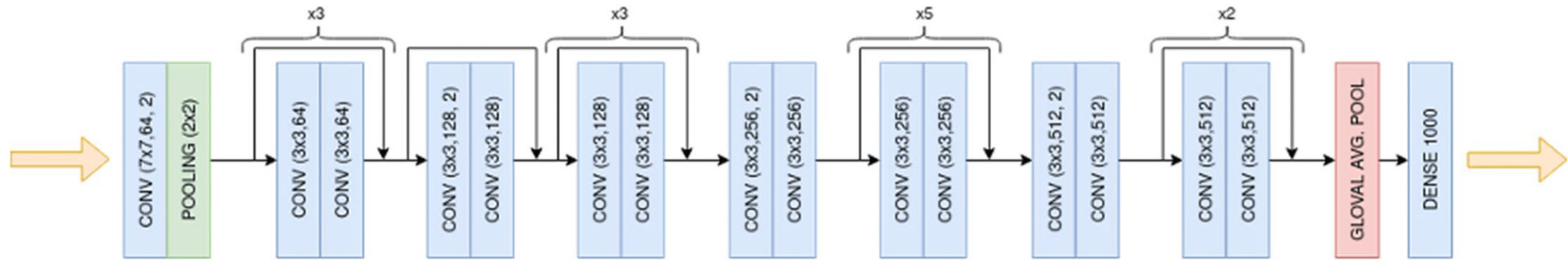
- **Arquitectura:** 16 convoluciones (o más) y 3 densas
- **Problema:** *Muchos parámetros, vanishing gradients*
- **Ventaja:** Fácil de entender y de implementar

¹ Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556.

ResNet - Redes residuales

Redes neuronales convolucionales

Desarrollada por He et al.¹ en 2015



Arquitectura ResNet-34. Fuente: [Medium](#)

- **Arquitectura:** 34+ capas convolucionales con bloques residuales
- **Principal aportación:** Conexiones residuales para evitar el *vanishing gradient*
- **Ventaja:** Permite el entrenamiento de redes muy profundas

¹ He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition*. Proceedings of the IEEE conference on computer vision and pattern recognition. 770-778.

Ejercicios sugeridos

- Clasificación alternativa
- Implementando LeNet

Licencia

Esta obra está licenciada bajo una licencia [Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional](#).

Puedes encontrar su código en el siguiente enlace:
<https://github.com/etsisi/Aprendizaje-profundo>