Facsímil 01

Los cimientos

Capítulo 08

CNNs y RNNs: redes para visión y secuencias

686f6c61 · 2026

Capítulo 08: CNNs y RNNs: redes para visión y secuencias

Entrando en el tema

Hasta ahora hemos hablado de redes neuronales en abstracto: capas que se conectan con capas, neuronas que reciben entradas y producen salidas. Esa arquitectura genérica, el MLP, funciona para muchos problemas, pero tiene un punto débil: trata todos los datos como una lista plana de números, sin aprovechar su estructura.

Una imagen no es una lista plana. Es una rejilla de píxeles donde lo que importa son los patrones locales: un ojo está rodeado de ceja y párpado, no de píxeles aleatorios dispersos por la imagen. Un texto no es una lista plana. Es una secuencia donde el significado de una palabra depende de las que vinieron antes y de las que vendrán después.

Dos arquitecturas nacieron para resolver estos dos problemas. Las CNNs para la visión. Las RNNs para las secuencias. Y aunque los Transformers las han desplazado parcialmente, entenderlas es entender por qué el Transformer fue revolucionario.

CNNs: cómo ve una máquina

Una *Convolutional Neural Network* (CNN) no mira una imagen píxel a píxel. Aplica **filtros**: pequeñas matrices (3×3, 5×5) que se deslizan por la imagen detectando patrones locales.[1]

Piensa en un filtro como una plantilla. Un filtro que detecta bordes horizontales tiene valores positivos arriba y negativos abajo. Cuando se desliza sobre una zona de la imagen donde hay un borde horizontal, produce un valor alto. Cuando pasa por una zona uniforme, produce un valor cercano a cero.

El flujo de una CNN es:

```
Imagen → Convoluciones → Pooling → Más convoluciones → Pooling → Clasificación
```

Convolución. El corazón de la CNN. Un filtro pequeño se desliza por toda la imagen. En cada posición, multiplica sus valores por los píxeles correspondientes y suma el resultado. Una capa convolucional típica tiene decenas o cientos de filtros, cada uno especializado en detectar un patrón distinto: bordes verticales, esquinas, texturas, colores específicos.[2]

Veamos un ejemplo concreto. Imagina una imagen de 5×5 píxeles en escala de grises y un filtro de 3×3 que detecta bordes verticales:

```
Imagen 5x5:       Filtro 3x3 (borde vertical):
 1  1  1  0  0      -1   0   1
 1  1  1  0  0      -1   0   1
 1  1  1  0  0      -1   0   1
 1  1  1  0  0
 1  1  1  0  0
```

El filtro se coloca en la esquina superior izquierda y multiplica elemento a elemento: (-1×1 + 0×1 + 1×1) + (-1×1 + 0×1 + 1×1) + (-1×1 + 0×1 + 1×1) = 0 + 0 + 0 = 0. En esa zona no hay borde vertical: todo son unos.

Ahora deslizamos el filtro a la derecha, donde están los ceros. En la zona de transición entre unos y ceros: (-1×1 + 0×1 + 1×0) + (-1×1 + 0×1 + 1×0) + (-1×1 + 0×1 + 1×0) = (-1) + (-1) + (-1) = -3. El valor absoluto alto indica un borde vertical.

Esto es lo que hacen decenas de filtros en paralelo, cada uno especializado en un patrón distinto. La red **aprende** los valores del filtro durante el entrenamiento: no los programa una persona, los descubre la retropropagación.

En notación compacta, la convolución de una imagen I con un filtro K de tamaño $k \times k$ produce, en cada posición (i, j), esta suma:

$$(I * K)_{i,j} = \sum_{m=1}^{k} \sum_{n=1}^{k} I_{i+m,\, j+n} \cdot K_{m,n}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
I	Imagen de entrada (o mapa de la capa anterior)	rejilla 5×5
K	Filtro o *kernel* que se desliza	matriz 3×3
k	Tamaño del filtro (lado)	3
(i, j)	Posición donde se apoya el filtro	esquina superior izquierda
$(I * K)_{i,j}$	Valor de salida en esa posición	0 en zona uniforme, −3 en un borde

En palabras: en cada posición se multiplica el filtro por la zona de imagen que tiene debajo y se suman todos los productos. Eso da un punto del mapa de salida; deslizando el filtro por toda la imagen se obtiene el mapa completo.

Dos hiperparámetros controlan cómo se desliza el filtro:

- **Stride (paso).** Cuántos píxeles avanza el filtro en cada salto. Con *stride* 1 se solapa mucho y la salida casi conserva el tamaño; con *stride* 2 salta de dos en dos y reduce la resolución a la mitad.
- **Padding (relleno).** Cuántos píxeles de ceros se añaden alrededor del borde. Sin *padding* la salida encoge y los bordes se procesan menos veces; con *padding* se conserva el tamaño.

El tamaño del mapa de salida conviene tenerlo a mano para que las dimensiones cuadren:

$$\text{salida} = \left\lfloor \frac{N - K + 2P}{S} \right\rfloor + 1$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
N	Tamaño de la entrada (lado)	5
K	Tamaño del filtro (lado)	3
P	*Padding*: píxeles añadidos por lado	0
S	*Stride*: paso del filtro	1
salida	Tamaño del mapa resultante (lado)	$(5 - 3 + 0)/1 + 1 = 3$

En palabras: una imagen de 5×5 con un filtro de 3×3, sin relleno y paso 1, produce un mapa de 3×3.

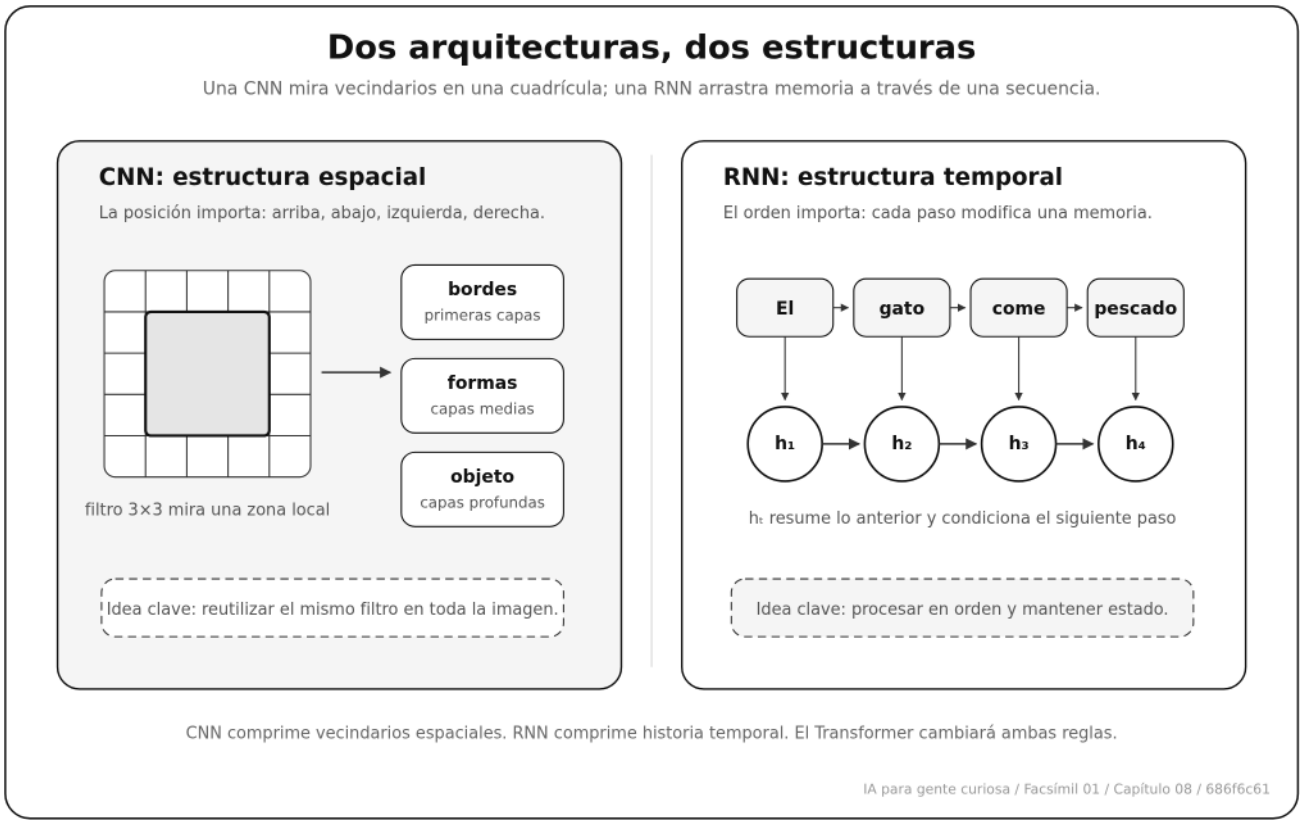
Pooling. Reduce la resolución espacial manteniendo la información importante. La operación más común es *max pooling*: divide la imagen en zonas (por ejemplo, 2×2) y se queda solo con el valor máximo de cada zona. Esto hace la red más eficiente (menos píxeles que procesar) y más robusta (invariante a pequeños desplazamientos: si el gato se mueve dos píxeles a la derecha, el *max pooling* probablemente siga detectándolo).

De patrones simples a conceptos. Las primeras capas detectan bordes y colores. Las capas intermedias combinan bordes en formas y texturas. Las capas profundas reconocen objetos: ojos, ruedas, letras. Es la misma jerarquía que vimos en el capítulo 5, pero especializada para datos visuales.

Canales y mapas de características. El ejemplo anterior usa una imagen en escala de grises, con un solo canal. Una imagen en color tiene **tres canales** (rojo, verde y azul), y el filtro opera sobre los tres a la vez: un filtro 3×3 sobre una imagen RGB es en realidad un volumen 3×3×3. Además, una capa convolucional no aplica un único filtro, sino decenas o cientos, y cada uno produce su propio **mapa de características** (*feature map*). Así, una capa

puede recibir 3 canales y entregar 64 mapas, que la capa siguiente tratará como sus 64 canales de entrada. Esa es la razón por la que las CNNs profundas tienen tantos parámetros pese a que cada filtro individual es diminuto.

Skip connections (ResNet). En 2015, ResNet introdujo una idea simple pero revolucionaria: atajos que permiten a la información saltarse capas.[3] En lugar de aprender la transformación completa, la capa aprende el «residuo»: la diferencia entre la entrada y la salida deseada. Si la transformación óptima es no hacer nada (la identidad), la capa puede aprender pesos cercanos a cero. Esto resolvió el problema de entrenar redes muy profundas y permitió arquitecturas de más de cien capas.



Relevancia actual. Las CNNs siguen siendo muy importantes en visión por computador porque aprovechan una estructura real de las imágenes: píxeles cercanos suelen estar relacionados. Ese sesgo inductivo las hace eficientes en detección, segmentación, clasificación y despliegues *edge*. Pero ya no son la única familia fuerte: los *Vision Transformers* tratan una imagen como parches y usan atención, y han demostrado que con suficientes datos y cómputo pueden competir o superar a las CNNs en muchas tareas.[4] En generación de imágenes, además, conviven U-Nets convolucionales y modelos de difusión basados en Transformers, como DiT.[5] La decisión de ingeniería no es “CNN vieja, Transformer nuevo”, sino qué estructura de datos, coste, volumen de entrenamiento y latencia tienes.

RNNs y LSTMs: cómo procesar secuencias

Antes de los Transformers, si querías que una red procesara texto, audio o series temporales, usabas una *Recurrent Neural Network* (RNN). Su idea es elegante: procesar la secuencia elemento a elemento, manteniendo un **estado oculto** que resume todo lo visto hasta ahora.[6]

```
Token 1 → RNN (estado oculto) → Token 2 + estado → RNN (actualiza estado) → Token 3 + estado...
```

Cuando la RNN lee la palabra «El», actualiza su estado oculto para reflejar que ha visto un artículo. Cuando lee «gato», el estado ahora codifica «El gato». Cuando llega a «es», el estado refleja «El gato es». Y así, token a token, el estado oculto acumula el contexto de toda la secuencia.

Esa actualización del estado tiene una fórmula, la ecuación que define una RNN:

$$h_t = \tanh(W_h\, h_{t-1} + W_x\, x_t + b)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x_t	Entrada en el paso t (el *embedding* del token actual)	el vector de «gato»
h_{t-1}	Estado oculto del paso anterior, la memoria acumulada	resumen de «El»
h_t	Nuevo estado oculto, ya actualizado con x_t	resumen de «El gato»
W_x, W_h	Matrices de pesos aprendidas (para la entrada y para el estado)	se ajustan al entrenar
b	Vector de sesgo	se ajusta al entrenar
$\tanh$	Activación que mantiene el estado acotado	salida entre −1 y 1

En palabras: el nuevo estado mezcla la entrada actual con el estado anterior, lo pasa por una activación y produce la memoria actualizada. Fíjate en que W_h y W_x son las mismas en todos los pasos: por eso, en el *backward*, el gradiente se multiplica una y otra vez por W_h, lo que explica el desvanecimiento o la explosión que veremos más abajo.

El problema es la memoria. Cuando llegas al token 500, el estado oculto ya ha sido sobrescrito cientos de veces. La información del token 1 se ha diluido hasta desaparecer. Es como el juego del teléfono: el mensaje se degrada con cada transmisión.[7]

LSTM: memoria con compuertas

La *Long Short-Term Memory* (LSTM) resuelve este problema con un mecanismo ingenioso: compuertas. En lugar de un solo estado oculto que se sobrescribe, la LSTM tiene:

- Una **compuerta de olvido**: decide qué información antigua descartar.
- Una **compuerta de entrada**: decide qué información nueva almacenar.
- Una **compuerta de salida**: decide qué parte del estado actual exponer como salida.

Estas compuertas son pequeñas redes neuronales que aprenden cuándo abrir y cuándo cerrar. Si el modelo encuentra una palabra clave al principio de un documento que será relevante quinientos tokens después, la compuerta de olvido puede decidir conservarla. Si encuentra información irrelevante, la descarta.

Gracias a las LSTMs, las RNNs pudieron capturar dependencias bastante más largas que una RNN simple. No conviene convertir esto en una cifra universal: que un modelo recuerde 100, 500 o más pasos depende de datos, arquitectura, tamaño del estado, entrenamiento y tarea. Lo importante es la forma del límite: la información tiene que pasar por una cadena temporal, y cada paso puede degradar la señal.

GRU: la hermana pequeña de la LSTM

La *Gated Recurrent Unit* (GRU), presentada en 2014, simplifica la LSTM: tiene solo dos compuertas (reinicio y actualización) en lugar de tres, y fusiona el estado oculto con la memoria de largo plazo en un solo vector.[8] Con menos parámetros que la LSTM, la GRU entrena más rápido y a menudo rinde igual de bien. Es la opción preferida cuando los recursos son limitados o cuando la LSTM no aporta una mejora significativa.

Bidireccional: mirar al pasado y al futuro

Una RNN estándar solo ve el pasado: cuando procesa la palabra 5, conoce las palabras 1 a 4 pero no tiene ni idea de lo que vendrá en la 6. Para muchas tareas (como etiquetar categorías gramaticales o traducir) el contexto futuro es tan importante como el pasado.

Las **RNNs bidireccionales** resuelven esto con dos RNNs en paralelo: una procesa la secuencia de izquierda a derecha y la otra de derecha a izquierda. Sus estados ocultos se concatenan, dando a cada posición acceso tanto al contexto anterior como al posterior. Son más caras computacionalmente pero notablemente más precisas en tareas donde el contexto completo importa.

El gradiente en las RNNs: un problema amplificado

El desvanecimiento del gradiente es especialmente dañino en las RNNs. En una red *feedforward* de 10 capas, el gradiente se atenúa 10 veces. En una RNN que procesa 100 tokens, el gradiente se atenúa 100 veces (porque la red se despliega en el tiempo y cada paso

temporal equivale a una capa). Es como si tuvieras una red de 100 capas, pero donde todas comparten los mismos pesos. Cada paso hacia atrás en el tiempo multiplica el gradiente por la misma matriz de pesos. Si los valores propios de esa matriz son menores que 1, el gradiente se desvanece exponencialmente. Si son mayores que 1, **explota**, produciendo actualizaciones caóticas.

Las LSTMs y GRUs mitigan esto con sus compuertas, que permiten que el gradiente fluya a través del estado de memoria con menos atenuación. Pero no eliminan el cuello de botella: la información sigue viajando paso a paso. Para secuencias muy largas, muchas tareas dejaron de intentar "mejorar la memoria recurrente" y pasaron a usar atención, recuperación o arquitecturas híbridas. Eso es exactamente lo que hizo el Transformer para lenguaje.

RNN vs LSTM vs Transformer

CARACTERÍSTICA	RNN	LSTM	TRANSFORMER
Procesamiento	Secuencial	Secuencial	Paralelo
Memoria/contexto práctico	Limitada por estado oculto y gradiente	Mejor que RNN simple, pero sigue siendo secuencial	Ventana explícita de atención; su tamaño depende del modelo y del coste
Paralelizable	No	No	Sí (aprovecha GPUs)
Velocidad	Lenta	Lenta	Rápida

El Transformer ganó por dos razones.[9] Primero, **paralelismo**: mira todos los tokens de una ventana a la vez. Una RNN tiene que procesar el token 1 antes que el 2, y el 2 antes que el 3. Un Transformer procesa la ventana simultáneamente, aprovechando GPUs masivamente paralelas. Segundo, **atención directa**: en lugar de comprimir todo el contexto en un estado oculto de tamaño fijo, cada token puede «mirar» directamente a cualquier otro token de la ventana. Si la palabra 500 necesita información de la palabra 1, la atención le da un camino directo. Ese camino no es gratis: la atención estándar crece de forma cuadrática con la longitud de la ventana, por eso los contextos largos requieren ingeniería adicional.

En el día a día

Aunque los Transformers dominan el procesamiento de lenguaje, las CNNs y RNNs siguen siendo relevantes. La decisión de ingeniería rara vez es elegir la familia más moderna; es elegir la que mejor encaja con la estructura del dato, el volumen disponible y el presupuesto de cómputo y latencia.

CNNs en visión. Cuando la tarea es clasificar imágenes, detectar objetos o segmentar y tienes pocos datos etiquetados, poco cómputo o un despliegue en dispositivo, una CNN sigue siendo la primera candidata razonable. El motivo es su sesgo inductivo: asume que los píxeles cercanos están relacionados y que el mismo patrón puede aparecer en cualquier zona, así que reutiliza filtros y necesita muchos menos ejemplos que un modelo sin esa suposición. Esa misma economía de parámetros la hace barata de entrenar y de servir. La restricción que manda aquí es el coste y el volumen de datos. Solo cuando dispones de muchísimos ejemplos, infraestructura de entrenamiento amplia y quieres atención global o unificar visión con lenguaje, un *Vision Transformer* entra en la conversación, porque sin ese volumen de datos su falta de sesgo inductivo se paga en sobreajuste.

CNNs, U-Nets y DiT en generación de imágenes. En generación no eliges una sola familia sino una combinación. Las U-Nets convolucionales han sido el bloque central de los modelos de difusión porque su estructura de codificador y decodificador con atajos preserva el detalle espacial fino que una imagen necesita. Al escalar a más datos y más cómputo, los Transformers de difusión como DiT se vuelven competitivos porque escalan mejor con el tamaño del modelo. La decisión depende del presupuesto de entrenamiento y de la resolución objetivo, así que conviene mirar la arquitectura concreta del modelo que usas en lugar de asumir una por defecto.

LSTMs en series temporales. Para predicción de ventas, mantenimiento predictivo o análisis de señales con secuencias cortas o medias, una LSTM sigue siendo muy competitiva y a menudo preferible. La restricción dominante es la relación entre coste y ganancia: la atención del Transformer crece de forma cuadrática con la longitud de la secuencia, y ese coste solo se justifica si de verdad necesitas relacionar posiciones muy lejanas. Para predecir la demanda de mañana a partir de unos cientos de puntos, un modelo recurrente entrena más rápido, es más barato de mantener y rinde igual o mejor que un Transformer enorme.

RNNs en dispositivos pequeños. Cuando la restricción que manda es la memoria y el consumo, una RNN simple o una GRU consume mucha menos RAM y cómputo que un Transformer, que debe guardar las activaciones de toda la ventana de atención. En un microcontrolador o un dispositivo *edge* con presupuesto de energía ajustado, procesar la señal paso a paso con un estado oculto pequeño puede ser la única opción que cabe en el hardware. Aquí la elegancia secuencial de la RNN, que en lenguaje era un lastre frente al paralelismo, se convierte en su mayor ventaja.

Por qué debería importarte

Estas arquitecturas no son reliquias históricas que estudias por cultura general. Son las piezas que explican **por qué** los Transformers son como son.

Cada decisión de diseño del Transformer es una respuesta a una limitación de las RNNs. La atención existe porque el estado oculto de una RNN no bastaba para contexto largo. El paralelismo existe porque el procesamiento secuencial era demasiado lento. La capacidad de

mirar posiciones lejanas dentro de una ventana explícita existe porque las LSTMs, incluso con compuertas, seguían obligando a que la información viajara paso a paso por la secuencia.

Entender de dónde venimos es entender hacia dónde vamos. Y en ingeniería, a veces la herramienta adecuada no es la más moderna, sino la que mejor se adapta a tus restricciones de recursos.

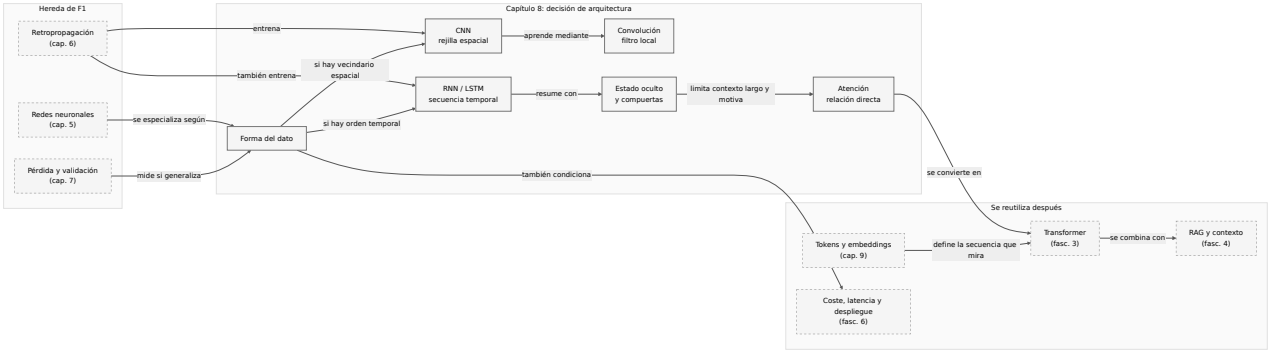
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Usar un Transformer para todo	Para una serie temporal de 100 puntos, una LSTM puede ser más rápida, más barata e igual de precisa que un Transformer. Usar la arquitectura más grande no siempre mejora la decisión.	Evalúa la complejidad de tu problema antes de elegir la arquitectura. Si es secuencial y corto, prueba RNN/LSTM primero.
Aplicar CNNs a datos no espaciales	Las CNNs asumen que los datos tienen estructura local (píxeles cercanos están relacionados). Aplicarlas a datos tabulares donde las columnas no tienen orden espacial no aporta ventajas sobre un MLP.	CNNs para imágenes y datos con estructura de rejilla. MLPs para datos tabulares. Transformers para secuencias largas.
Ignorar el preprocesamiento en CNNs	Una CNN espera imágenes de un tamaño fijo. Si le pasas imágenes de tamaños variados sin redimensionar, los resultados serán inconsistentes.	Normaliza el tamaño de las imágenes, escala los valores de píxel a [0,1] o [-1,1] y aplica *data augmentation* durante el entrenamiento.
Asumir que las RNNs «entienden» el orden	Las RNNs procesan en orden, pero eso no garantiza que capturen dependencias a largo plazo. Una RNN simple puede fallar en capturar que la primera palabra de un párrafo es el sujeto de la última oración.	Si necesitas dependencias a largo plazo, usa LSTMs, GRUs o, directamente, Transformers.

Cómo encaja todo

Este mapa se lee como una decisión de arquitectura, no como una línea histórica. Venimos de redes, gradientes y capas; aquí aprendemos que la forma del dato importa. Si la estructura es espacial, una CNN aprovecha vecindarios. Si la estructura es temporal, una RNN o LSTM arrastra estado. Si la secuencia es larga, necesitamos atención, recuperación o arquitecturas híbridas.

La decisión que enseña el capítulo no es memorizar siglas, sino justificar una familia por contrato de datos, coste y despliegue. Esa misma forma de pensar reaparece cuando elijas embeddings, modelos locales, RAG o arquitecturas de Transformer.


Hereda de F1
Retropropagación (cap. 6)
Redes neuronales (cap. 5)
Pérdida y validación (cap. 7)
Capítulo 8: decisión de arquitectura
entrena
también entrena
se especializa según
mide si generaliza
Forma del dato
si hay vecindario espacial
si hay orden temporal
CNN rejilla espacial
aprende mediante
Convolución filtro local
RNN / LSTM secuencia temporal
resume con
Estado oculto y compuertas
limita contexto largo y motiva
Atención relación directa
también condiciona
se convierte en
Se reutiliza después
Tokens y embeddings (cap. 9)
define la secuencia que mira
Transformer (fasc. 3)
se combina con
RAG y contexto (fasc. 4)
Coste, latencia y despliegue (fasc. 6)

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
CNN	Red neuronal convolucional. Usa filtros que se deslizan por la entrada para detectar patrones locales. Especializada en imágenes y datos con estructura de rejilla.
Convolución	Operación que aplica un filtro sobre una entrada, deslizándolo por todas las posiciones y calculando la similitud en cada punto.
Filtro (kernel)	Matriz pequeña de pesos aprendidos que se desliza por la entrada para detectar un patrón concreto como un borde o una textura.
Mapa de características	Salida que produce un filtro al recorrer toda la entrada; indica dónde aparece el patrón que ese filtro detecta.
Stride	Paso del filtro: cuántos píxeles avanza en cada salto. Un paso mayor reduce la resolución de la salida.
Padding	Relleno de ceros alrededor del borde de la entrada que sirve para conservar el tamaño de la salida tras la convolución.
Pooling	Operación que reduce la resolución espacial manteniendo la información relevante. *Max pooling* toma el valor máximo de cada zona.
RNN	Red neuronal recurrente. Procesa secuencias elemento a elemento, manteniendo un estado oculto que resume el contexto previo.
Estado oculto	Vector que resume la información de todos los elementos anteriores de una secuencia en una RNN.
LSTM	Variante de RNN con compuertas de olvido, entrada y salida. Permite aprender dependencias a más largo plazo que una RNN simple.
GRU	Variante de RNN más ligera que la LSTM, con solo dos compuertas. Entrena más rápido y suele rendir de forma comparable.
ViT (Vision Transformer)	Arquitectura que divide una imagen en parches y los procesa con atención en lugar de convoluciones.
Atención	Mecanismo que permite a cada elemento de una secuencia mirar directamente a cualquier otro dentro de una ventana, sin pasar por un estado recurrente.

Antes de pasar página

- [ ] ¿Puedo explicar cómo una CNN detecta patrones en una imagen? (Si no, vuelve a «CNNs: cómo ve una máquina».)
- [ ] ¿Entiendo qué hace el *pooling* y por qué es útil? (Si no, vuelve a la sección de CNNs.)
- [ ] ¿Sé cómo procesa una RNN una secuencia y cuál es su principal limitación? (Si no, vuelve a «RNNs y LSTMs».)
- [ ] ¿Puedo explicar por qué el Transformer superó a las RNNs? (Si no, vuelve a la tabla comparativa.)
- [ ] ¿Sé justificar una arquitectura por forma de datos, coste y despliegue? (Si no, vuelve a «En el día a día».)

En resumen

IDEA FUERZA	DETALLE
Las CNNs detectan patrones locales en imágenes mediante filtros que se deslizan por la entrada.	De bordes en las primeras capas a objetos completos en las profundas. *Pooling* reduce dimensionalidad y añade invarianza.
Las RNNs procesan secuencias manteniendo un estado oculto. Las LSTMs añaden compuertas para recordar a más largo plazo.	Mejoran el problema, pero siguen comprimiendo historia en estado recurrente y procesando paso a paso.
El Transformer ganó por paralelismo y atención directa.	Procesa una ventana completa a la vez y permite que cualquier token mire directamente a otro dentro de esa ventana, pagando coste de memoria y cómputo.

Para saber más

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. y Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. En *Proceedings of EMNLP* (pp. 1724-1734). https://doi.org/10.3115/v1/D14-1179

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. y Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. En *International Conference on Learning Representations*. https://openreview.net/forum?id=YicbFdNTTy

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org

He, K., Zhang, X., Ren, S. y Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90

Hochreiter, S. y Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Krizhevsky, A., Sutskever, I. y Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. En *Advances in Neural Information Processing Systems 25* (pp. 1097-1105). https://papers.nips.cc/paper/4824

LeCun, Y., Bengio, Y. y Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. https://doi.org/10.1038/nature14539

Peebles, W. y Xie, S. (2023). Scalable diffusion models with Transformers. En *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 4195-4205). https://doi.org/10.1109/ICCV51070.2023.00387

Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). https://papers.nips.cc/paper/7181-attention-is-all-you-need

Notas

1. Krizhevsky, A., Sutskever, I. y Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. En *Advances in Neural Information Processing Systems 25* (pp. 1097-1105). https://papers.nips.cc/paper/4824. AlexNet demostró que las CNNs profundas, entrenadas con GPUs, podían superar ampliamente a los métodos tradicionales de visión por computador, iniciando la revolución del *deep learning* en visión. ↩

2. LeCun, Y., Bengio, Y. y Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. https://doi.org/10.1038/nature14539. Los autores describen cómo las capas convolucionales aprenden jerarquías de características visuales, desde bordes simples en las primeras capas hasta objetos complejos en las profundas. ↩

3. He, K., Zhang, X., Ren, S. y Sun, J. (2016). Deep residual learning for image recognition. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90. Las conexiones residuales permiten entrenar redes de más de cien capas sin degradación del rendimiento, resolviendo el problema del desvanecimiento del gradiente en arquitecturas muy profundas. ↩

4. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. y Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. En *International Conference on Learning Representations*. https://openreview.net/forum?id=YicbFdNTTy. El trabajo popularizó ViT: dividir una imagen en parches y procesarlos con un Transformer. ↩

5. Peebles, W. y Xie, S. (2023). Scalable diffusion models with Transformers. En *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 4195-4205). https://doi.org/10.1109/ICCV51070.2023.00387. DiT muestra cómo sustituir bloques U-Net por Transformers escalables en modelos de difusión. ↩

6. Hochreiter, S. y Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735. Hochreiter y Schmidhuber introdujeron la LSTM para resolver el problema del desvanecimiento del gradiente en RNNs, permitiendo que las redes recurrentes aprendieran dependencias a largo plazo por primera vez. ↩

7. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org. El capítulo 10 aborda en detalle las RNNs, el problema del desvanecimiento del gradiente en secuencias largas y las arquitecturas con compuertas como LSTM y GRU. ↩

8. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. y Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. En *Proceedings of EMNLP* (pp. 1724-1734). https://doi.org/10.3115/v1/D14-1179. La GRU fue introducida como una alternativa más simple y computacionalmente eficiente a la LSTM, con rendimiento comparable en muchas tareas. ↩

9. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). https://papers.nips.cc/paper/7181-attention-is-all-you-need. Los autores demostraron que eliminar la recurrencia y usar solo atención permitía entrenar modelos más rápido (por el paralelismo) y con mejor calidad (por la capacidad de atender a cualquier posición de la secuencia). ↩