Deep Learning para Business Analytics

Día 04 · Tema 2 — CNN para visión

De los píxeles a la decisión de retail.

Eduardo C. Garrido-Merchán

Profesor Propio Adjunto, Métodos Cuantitativos, ICADE

ecgarrido@comillas.edu

Madrid, Día 04

INDITEX ZARA MERCADONA amazon NVIDIA Google

ejemplos del día

COMILLAS UNIVERSIDAD PONTIFICIA ICAI ICADE CIHS

Lo que vamos a cubrir hoy.

1. Por qué no basta con un MLP

2. La convolución a mano

3. Arquitecturas canónicas

4. Transfer learning con Hugging Face

5. Caso retail del día

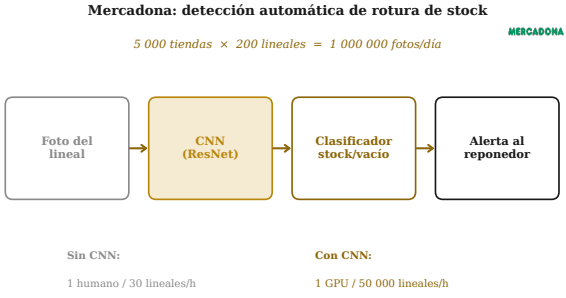
Lo que vas a saber hacer:

1. Calcular a mano la convolución de un kernel 3×3 sobre un parche.
2. Justificar por qué un MLP no escala a imágenes y una CNN sí.
3. Fine-tunear una ResNet preentrenada en Hugging Face en 20 líneas.

Parte 1 / 5

Por qué no basta con un MLP

Mercadona sabe si falta stock en el lineal con una foto.



El problema. 🛒 5 000 tiendas, 200 lineales cada una. Revisar la rotura de stock con personal humano cuesta **33 000 h/día**.

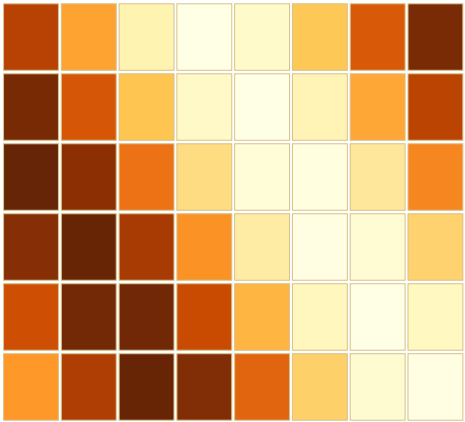
La solución. 📸 Una cámara por pasillo + una CNN que clasifica “stock OK” vs. “estante vacío” en 20 ms por imagen.

El ahorro. Reponedores alertados en tiempo real. Reducción de **20–30 %** en ventas perdidas por rotura, según estimaciones del sector.

Idea clave. La inspección visual automática es el caso canónico de CNN en retail.

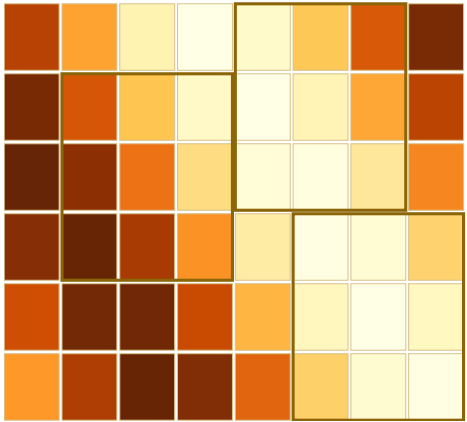
La imagen tiene estructura espacial que un MLP ignora.

MLP sobre 8×6 píxeles: 48 entradas distintas, 48 pesos por neurona.



cada píxel tiene su propio peso

CNN: un mismo filtro 3×3 recorre la imagen.



los mismos 9 pesos en todas las posiciones

Tres propiedades que motivan la CNN.

Localidad. 👁️ Los píxeles cercanos importan entre sí, los lejanos casi no. La CNN solo conecta vecinos.

Pesos compartidos. 🔗 El mismo detector de bordes vale en la esquina superior izquierda y en la inferior derecha. Una CNN aprende un filtro *una vez* y lo aplica en toda la imagen.

Composición jerárquica. 📚 Bordes → formas → partes → objetos. Las capas profundas reusan lo que aprenden las superficiales.

Idea clave. Una CNN tiene varios órdenes de magnitud menos parámetros que un MLP equivalente, y por eso entrena con datos finitos.

Parte 2 / 5

La convolución a mano

El kernel es una lupa que recorre la imagen buscando bordes.

El kernel es una lupa: busca UN patrón local en toda la imagen.

Imagen 6×6 con borde vertical

9	9	7	1	0	1
8	7	9	0	2	2
9	8	7	2	2	1
9	8	9	2	1	0
9	9	9	2	1	0
7	8	8	1	2	0

Kernel 3×3

+1	+0	-1
+1	+0	-1
+1	+0	-1

3	**21**	**19**	-1
1	**19**	**20**	1
2	**19**	**21**	5
-1	**20**	**22**	5

Activación 4×4

valores altos = borde detectado

La idea. 🔍 Un kernel 3×3 es una plantilla de 9 números. Se desliza por toda la imagen, celda a celda.

Qué detecta. En cada posición calcula la *correlación local* entre la plantilla y el parche de la imagen. Si la plantilla dice "borde vertical" $(+1, 0, -1)$, activa fuerte donde hay un borde.

Muchos kernels. Una capa CNN aprende 32–512 kernels en paralelo. Cada uno detecta **un patrón distinto**: bordes, esquinas, texturas.

IDEA CLAVE. La clave es que el kernel se reusa en toda la imagen: pesos compartidos.

Ejemplo a mano: convolución 3×3 sobre una imagen 5×5.

La celda destacada de la salida = suma elem-a-elem del parche resaltado y el kernel.

Imagen 5×5 (input)

1	**0**	**2**	1	0
0	**1**	**3**	2	1
2	**1**	**0**	1	2
1	0	1	0	1
0	2	1	0	1

Kernel 3×3 (detector de bordes verticales)

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

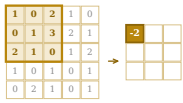
Salida 3×3 (feature map)

-2	-2	2
-1	-1	0
1	2	-2

Las nueve posiciones, fotograma a fotograma.

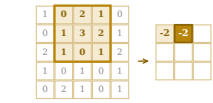
El kernel [1, 0, −1] recorre la imagen: nueve ventanas, nueve cuentas, nueve celdas de salida.

Paso 1 de 9 · ventana en fila 1, columna 1



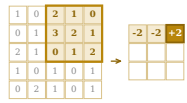
(1+0+2) − (2+3+0) = -2

Paso 2 de 9 · ventana en fila 1, columna 2



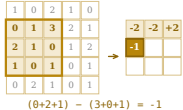
(0+1+1) − (1+2+1) = -2

Paso 3 de 9 · ventana en fila 1, columna 3



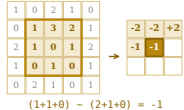
(2+3+0) − (0+1+2) = +2

Paso 4 de 9 · ventana en fila 2, columna 1



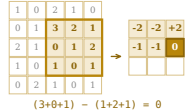
(0+2+1) − (3+0+1) = -1

Paso 5 de 9 · ventana en fila 2, columna 2



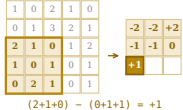
(1+1+0) − (2+1+0) = -1

Paso 6 de 9 · ventana en fila 2, columna 3



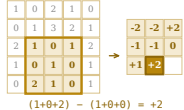
(3+0+1) − (1+2+1) = 0

Paso 7 de 9 · ventana en fila 3, columna 1



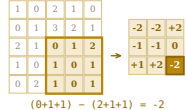
(2+1+0) − (0+1+1) = +1

Paso 8 de 9 · ventana en fila 3, columna 2



(1+0+2) − (1+0+0) = +2

Paso 9 de 9 · ventana en fila 3, columna 3



(0+1+1) − (2+1+1) = -2

Cada viñeta es un fotograma: la ventana dorada avanza una celda a la derecha, y al llegar al borde baja una fila. La cuenta de abajo es la suma de la columna izquierda menos la de la derecha (la central va multiplicada por 0).

Versión en vídeo: `slides/anim/conv_ejemplo.mp4`

Cálculo paso a paso, en texto.

Imagen (input). 📸 Matriz 5×5 de la slide anterior, en escala de grises.

Kernel. 🔍 Detector de bordes verticales: $K = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$.

Posición destacada (esquina superior izquierda). El parche 3×3 es $\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \end{pmatrix}$. Multiplicamos celda a celda con K y sumamos:

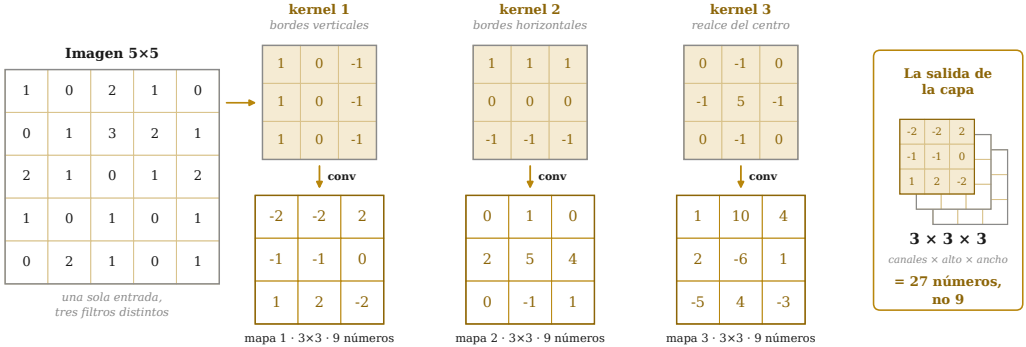
$$1 \cdot 1 + 0 \cdot 0 + 2 \cdot (-1) + 0 \cdot 1 + 1 \cdot 0 + 3 \cdot (-1) + 2 \cdot 1 + 1 \cdot 0 + 0 \cdot (-1) \;=\; \textbf{-2}.$$

Recorrido. ⚙️ Se desliza el kernel un paso a la derecha y se repite. Hay $3 \times 3 = 9$ posiciones (con *stride* 1, *padding* 0).

Idea clave. Una capa convolucional son muchos kernels en paralelo, cada uno destacando un patrón local distinto.

Tres kernels sobre la misma imagen dan tres mapas, no uno.

En la LeNet-5 del día, conv1 tiene 6 kernels 5×5 y devuelve 6 × 28 × 28 = 4.704 números; conv2, con 16 kernels, 16 × 10 × 10 = 1.600.



La cuenta. Un kernel deja 9 números. Tres kernels dejan $3 \times 3 \times 3 = 27$: la salida de una capa convolucional es un *volumen* de canales, uno por filtro, y el siguiente kernel ya no ve una imagen plana sino ese montón de matrices.

Idea clave. Cuando en las formas veas $6 \times 28 \times 28$, el 6 son los kernels de esa capa: tantos mapas como filtros, y por eso al avanzar los mapas se estrechan pero se multiplican.

De la matriz de juguete a una foto de tienda de verdad.

Sobre una foto real es exactamente la misma operación, repetida miles de veces.

1. Una foto es una matriz de números

64 × 64 píxeles de brillo (0 = negro, 255 = blanco)

2. La ventana 3×3 hace la misma cuenta de la slide anterior

221	164	135	125	152	195	147	109
238	223	227	180	180	154	120	122
231	159	**114**	**155**	**243**	206	178	223
219	118	**106**	**180**	**239**	168	197	236
161	113	**124**	**222**	**236**	160	227	211
109	120	120	151	163	155	235	180
123	121	120	111	110	117	126	124
121	121	122	124	125	123	119	120

kernel 3×3

+1	+0	-1
+1	+0	-1
+1	+0	-1

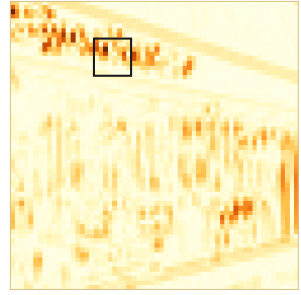
borde vertical

`(114+106+124) − (243+239+236) = -374`

la columna derecha es mucho más clara que la otra:

|respuesta| grande → aquí hay un borde vertical

3. Repetida en las 62 × 62 posiciones



mapa de activación: oscuro = borde vertical detectado

Versión en vídeo: `slides/anim/conv_foto.mp4` · el kernel recorre las 2.116 posiciones en 14 segundos

Pooling: reducir resolución y ganar invariancia.

Max-pooling: en cada bloque 2×2, gana el máximo (aquí 6).

Entrada 4×4

6	**3**	1	2
4	**5**	0	1
8	1	2	7
0	2	3	4

Max-pooling 2×2 → 2×2

6	2
8	7

Convolución a mano: imagen 4×4, kernel 3×3.

Imagen 4×4

2	**0**	**1**	3
1	**1**	**2**	0
0	**3**	**1**	1
2	1	0	2

Kernel 3×3

1	0	-1
0	1	0
-1	0	1

Salida 2×2

3	-3
0	3

```
Pos (0,0): 2·1+0·0+1·(−1)+1·0+1·1+2·0+0·(−1)+3·0+1·1 = 3
Pos (0,1): 0·1+1·0+3·(−1)+1·0+2·1+0·0+3·(−1)+1·0+1·1 = -3
Pos (1,0): 1·1+1·0+2·(−1)+0·0+3·1+1·0+2·(−1)+1·0+0·1 = 0
Pos (1,1): 1·1+2·0+0·(−1)+3·0+1·1+1·0+1·(−1)+0·0+2·1 = 3
```

Datos. ⚙️ Imagen 4×4, kernel 3×3, stride = 1, padding = 0 ⇒ salida 2×2.

Pos (0,0).
$2\cdot1+0\cdot0+1\cdot(-1)+1\cdot0+1\cdot1+2\cdot0+0\cdot(-1)+3\cdot0+1\cdot1 = \mathbf{3}$.

Pos (0,1).
$0\cdot1+1\cdot0+3\cdot(-1)+1\cdot0+2\cdot1+0\cdot0+3\cdot(-1)+1\cdot0+1\cdot1 = \mathbf{-3}$.

Pos (1,0).
$1\cdot1+1\cdot0+2\cdot(-1)+0\cdot0+3\cdot1+1\cdot0+2\cdot(-1)+1\cdot0+0\cdot1 = \mathbf{0}$.

Pos (1,1).
$1\cdot1+2\cdot0+0\cdot(-1)+3\cdot0+1\cdot1+1\cdot0+1\cdot(-1)+0\cdot0+2\cdot1 = \mathbf{3}$.

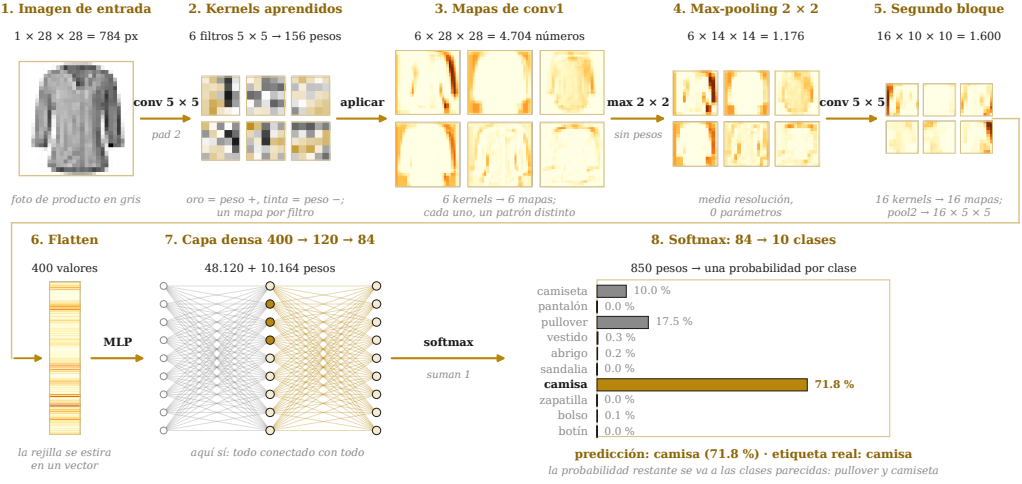
Idea clave. Con 4 posiciones ya se ve: el kernel activa donde los valores cambian de alto a bajo.

Parte 3 / 5

Arquitecturas canónicas

De los píxeles a la clase: el recorrido completo de una imagen.

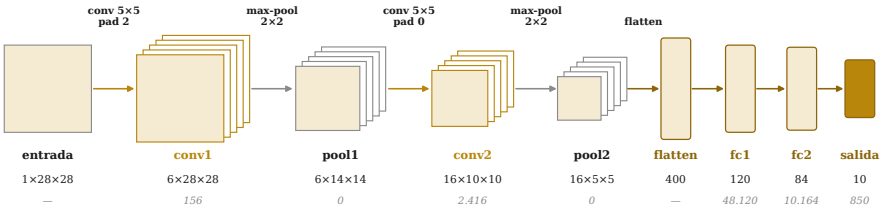
LeNet-5 entrenada para esta figura: 61.706 parámetros, accuracy de test 87.9 % en Fashion-MNIST



Kernels, mapas y probabilidades reales, no un esquema: salen de `slides/lenet_fashion.py`.

LeNet-5 entera cabe en 61.706 parámetros.

Regla de la forma: $n_{\text{out}} = (n + 2p - k)/s + 1$ conv1: (28 + 4 − 5)/1 + 1 = 28 conv2: (14 + 0 − 5)/1 + 1 = 10 abajo, los params de cada capa



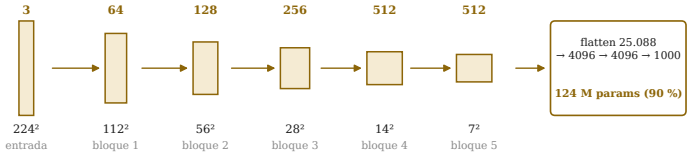
Cómo se lee. Cada bloque es un tensor: la altura es el tamaño espacial y el grosor del montón, el número de canales. Conv y pooling *estrechan* el mapa y *ensanchan* los canales hasta que la rejilla ($16 \times 5 \times 5$) cabe en un vector.

Idea clave. El extractor convolucional se lleva el 4 % de los pesos y hace el trabajo difícil; el 96 % restante está en la cabeza densa, que es la parte que sustituimos al hacer transfer learning.

VGG-16 y ResNet-50 son la misma receta, escalada.

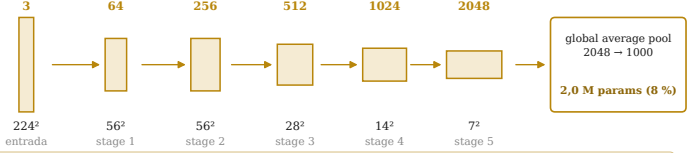
VGG-16 (2014): 13 conv 3×3 + 3 densas · 138 M params

toda la reducción con max-pool 2×2; la cabeza densa se come el 90 % de los pesos

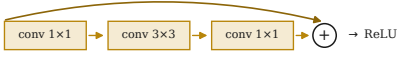


ResNet-50 (2015): 4 stages de bloques residuales · 25,6 M params

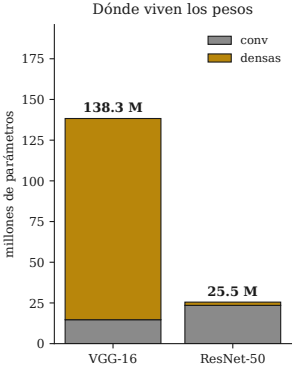
mismos bloques, mucho más profundos, y la cabeza densa sustituida por un promedio global



Bloque residual:



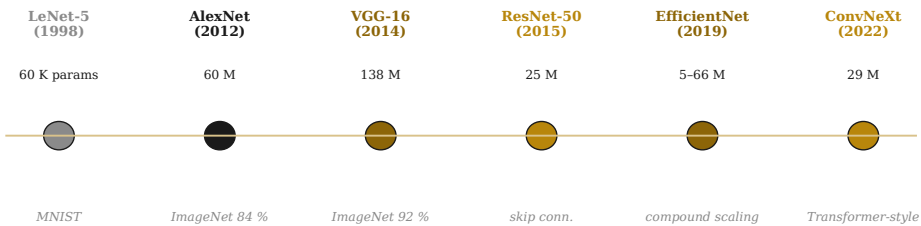
el atajo (skip connection) suma la entrada del bloque tal cual a su salida:
es lo que permite entrenar 50 capas sin que el gradiente se apague por el camino



ResNet-50 tiene 5,4× menos pesos que VGG-16 y acierta más

Veinticinco años de CNN, condensados.

Veinticinco años de arquitecturas convolucionales canónicas.

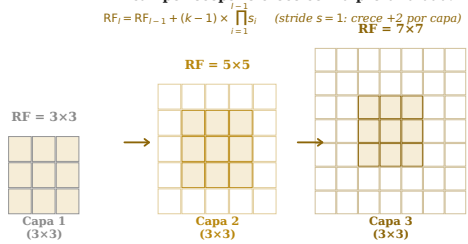


Concepto → aplicación de negocio en retail y operaciones.

Concepto	Aplicación	Entradas / Salidas	KPI
ResNet fine-tuned	Catálogo automático de moda (Inditex / Zara).	Foto → categoría (10–50 clases).	Top-1, t. etiquetado/h.
EfficientNet ligera	Control de calidad en planta (Mercadona).	Foto → OK / defecto.	Recall defecto, FP/día.
YOLO-style detección	Conteo en estante (Carrefour).	Imagen → cajas + clase.	Precisión recuento, rotura.
OCR (DocTR / Tesseract)	Facturas y recibos (BBVA, Endesa).	Imagen doc. → texto + campos.	Error campo, % rev. humana.
Segmentación U-Net	Imágenes satélite (Iberdrola).	Imagen → máscara uso suelo.	IoU, ha detectadas.

Idea clave. El mismo zoo de modelos resuelve cinco casos de negocio distintos; cambia la cabeza y el dataset.

El campo receptivo crece con la profundidad.



Fórmula. 📚 Con kernels $k \times k$ y stride s=1:

$$\mathrm{RF}_l = \mathrm{RF}_{l-1} + (k-1) \times \prod_{i=1}^{l-1} s_i$$

Ejemplo con 3×3. Capa 1: RF = 3. Capa 2: 3 + 2=5. Capa 3: 5 + 2=7.

Consecuencia. ⚙️ Para reconocer un objeto de 50 píxeles necesitamos ~25 capas 3×3. Esto explica por qué ResNet tiene 50+ capas: no es capricho.

Deep > wide. 🎯 Apilar muchas capas 3×3 es más eficiente en parámetros que usar pocas capas 7×7 (VGG lo descubrió en 2014).

Idea clave. La profundidad no es solo capacidad: es la única vía para cubrir objetos grandes.

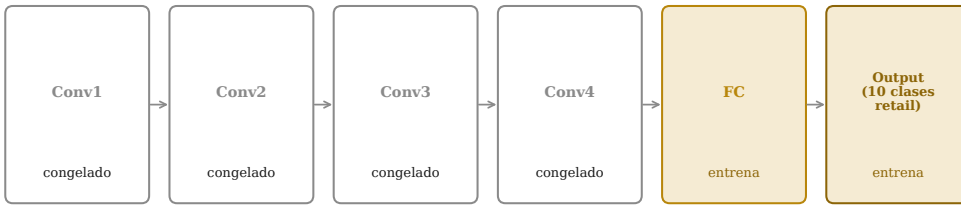
Parte 4 / 5

Transfer learning con Hugging Face

Transfer learning: hereda el backbone, entrena la cabeza.

Backbone preentrenado en ImageNet (1.4 M imágenes)

Cabeza nueva, ajustada en tu dataset (1.000 imágenes)



Receta exprés con `transformers` de Hugging Face.

```
from transformers import AutoImageProcessor, AutoModelForImageClassification
from datasets import load_dataset

proc = AutoImageProcessor.from_pretrained("microsoft/resnet-50")
model = AutoModelForImageClassification.from_pretrained(
    "microsoft/resnet-50", num_labels=10, ignore_mismatched_sizes=True)

ds = load_dataset("fashion_mnist")
# congelar todo el backbone
for p in model.resnet.parameters(): p.requires_grad = False

# trainer con cosine, weight_decay y early stopping (ver Día 03)
trainer.train()
```

Idea clave. Con 20 líneas y un GPU de Colab tienes un clasificador de moda razonable en 5 minutos.

Ejemplo a mano: cuánto se ahorra con transfer learning.

Sin transfer (entrenar desde cero). ⚠️ ResNet-50 tiene 25,5 M parámetros. Entrenar desde cero requiere ~1 M imágenes etiquetadas y ~24 h en una A100.

Con transfer (backbone congelado). 🚀 Solo entrenamos la cabeza (1.000–10.000 parámetros). Necesitamos ~1.000 imágenes y ~5 minutos en una T4 gratis de Colab.

Cálculo del coste estimado. 💰

$$\text{Coste desde cero} \approx 24\,\text{h} \times 3{,}0\,\$/\text{h} = \mathbf{72\ \$}; \quad \text{Coste con transfer} \approx 5\,\text{min} \times 0\,\$/\text{h} = \mathbf{0\ \$}.$$

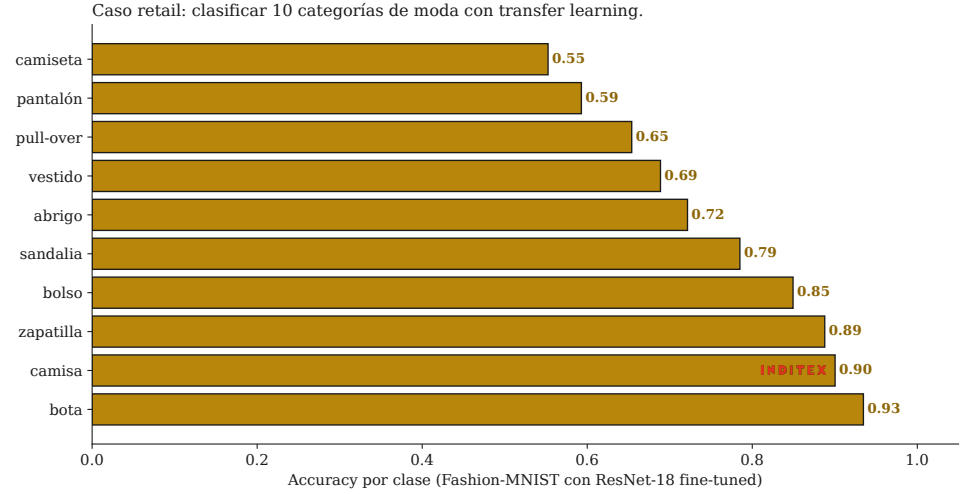
Lectura. 🌍 Hugging Face Hub es donde se publica gratis ese trabajo previo de 72 $, y tú lo reúsas.

Idea clave. Si tienes menos de 50.000 imágenes, transfer learning es casi siempre la elección correcta.

Parte 5 / 5

Caso retail del día

Fashion-MNIST con ResNet fine-tuned: accuracy por categoría.



Lo que el alumno se lleva a casa de este día.

Concepto 1. 📚 Una capa convolucional es muchos kernels pequeños que recorren la imagen sumando-pesando un parche local.

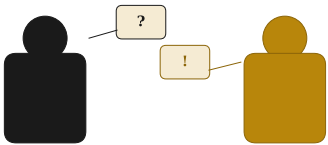
Concepto 2. 👁️ Pooling reduce resolución y aporta invariancia a pequeños desplazamientos.

Concepto 3. 🚀 En 2026, casi todo proyecto de visión empieza por descargar un modelo preentrenado de Hugging Face y entrenar la cabeza.

Para la práctica. Fine-tunear una ResNet sobre Fashion-MNIST o EuroSAT y comparar contra un MLP plano.

Próximo día. Día 05 (Tema 2.2): transfer learning a fondo con Hugging Face y casos BBVA / Repsol.

Pausa para debatir: ¿800 fotos bastan para una CNN propia?



La pregunta. 📸 Una pyme tiene 800 fotos de su producto. ¿Es razonable entrenar una CNN propia o forzar transfer learning?

Posición A. 🚀 Forzar transfer: 800 ejemplos no llegan para una CNN desde cero.

Posición B. 🏭 Entrenar propia si la marca lo justifica: la métrica baja, pero la auditoría sube.

Reglas. 5 minutos, dos turnos de palabra por bando, móviles boca abajo. Yo modero, no participo.

Hasta el Día 05.

Más transfer learning, OCR y casos bancarios.

Eduardo C. Garrido-Merchán

ecgarrido@comillas.edu