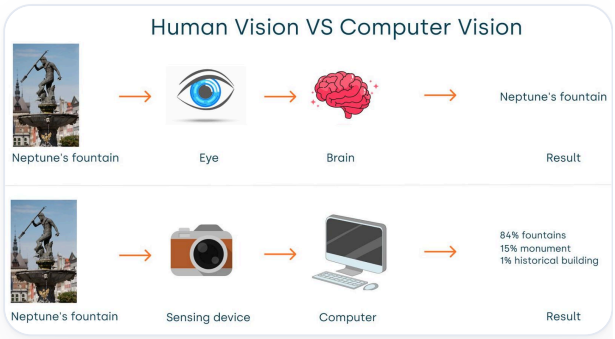
Enseñando a las máquinas a ver

La Visión Artificial es la disciplina de la inteligencia artificial que busca capacitar a las máquinas para **extraer información significativa y comprender el mundo a partir de imágenes y vídeos digitales**, imitando la percepción humana.



La brecha semántica: ¿cómo lo ve un humano vs. una máquina?

Visión humana

Inmediatamente identificamos «un gato naranja durmiendo en un sofá» y comprendemos su contexto.

Visión computacional

Una máquina percibe una compleja **matriz de millones de píxeles**, cada uno con valores numéricos RGB como [255, 128, 64…].

Las imágenes digitales son matrices de píxeles. La visión artificial aprende representaciones que permiten asociar esos valores con objetos, relaciones o regiones.

Tareas principales en visión artificial

Clasificación

Identifica qué objetos están presentes en una imagen.

Detección de objetos

Localiza y clasifica los objetos individuales dentro de un recuadro delimitador.

Segmentación

Define el contorno exacto de cada objeto a nivel de píxel.

Detección de anomalías

Identifica eventos o datos inusuales que no coinciden con un patrón esperado.

Estimación de pose

Localiza puntos clave (articulaciones) en personas o animales para comprender su postura.

Otras

OCR, estimación de profundidad, superresolución, keypoint matching…

¿Qué salida necesitas?

La misma imagen admite tareas distintas.

Quieres contar coches y localizar cada uno.

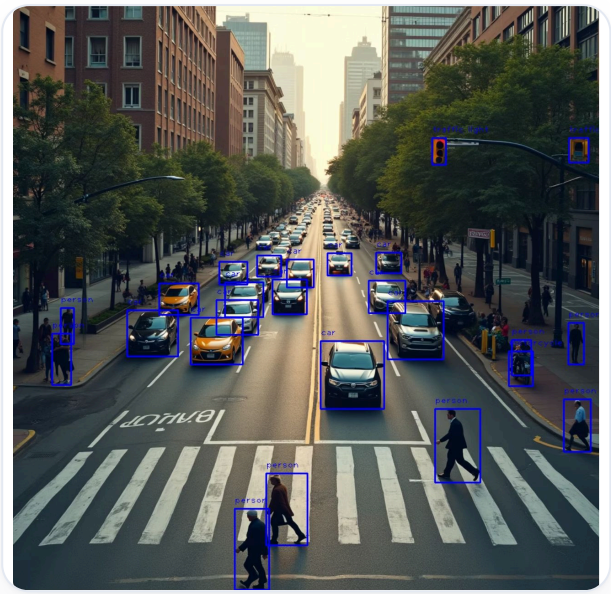
- A Una etiqueta para toda la imagen.
- B Una caja y una categoría por objeto.
- C Una puntuación de confianza para la imagen.

Respuesta B. Exacto: la detección localiza cada instancia y permite contarla.

La clasificación da una etiqueta global. La detección devuelve cajas y categorías por instancia. La segmentación delimita píxeles. Otros problemas incluyen anomalías, pose, OCR, profundidad y superresolución.

1. Detección de objetos

La detección de objetos va más allá de la simple clasificación. Su **objetivo principal es localizar y clasificar múltiples objetos simultáneamente** dentro de una imagen, proporcionando para cada uno una caja delimitadora, su categoría y un score de confianza.

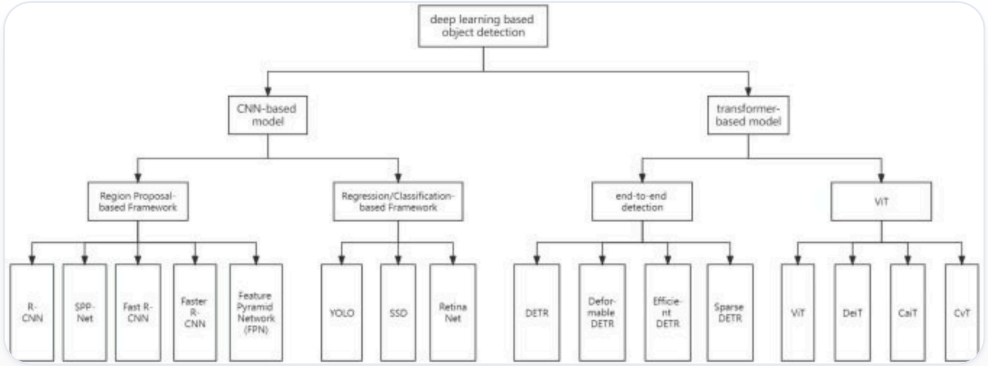


Diferencias clave:

- **Clasificación:** identifica qué hay en la imagen (una etiqueta global).
- **Detección:** localiza cada objeto con cajas y etiquetas múltiples.
- **Segmentación:** delimita el contorno exacto píxel a píxel.
- **Detección de anomalías:** hay algo raro en esta imagen.

Tipos de arquitecturas de Object Detection

Hay muchos tipos de arquitecturas, estas son las familias más importantes.



La rama basada en CNN se divide en frameworks con propuestas de región (R-CNN, SPP-Net, Fast R-CNN, Faster R-CNN, FPN) y frameworks de regresión o clasificación directa (YOLO, SSD, RetinaNet). La rama basada en transformer agrupa detección extremo a extremo (DETR, Deformable DETR, Efficient DETR, Sparse DETR) y arquitecturas ViT (ViT, DeiT, CaiT, CvT).

Evolución histórica de los detectores

Los modelos de detección de objetos han evolucionado significativamente en la última década, nos centraremos en tres grandes etapas arquitectónicas.

01 **Two-Stage (2014-2016)**

R-CNN, Fast R-CNN, Faster R-CNN: arquitecturas de dos etapas que primero proponen regiones candidatas y luego las clasifican. Precisos pero complejos y lentos.

02 **One-Stage (2015 →)**

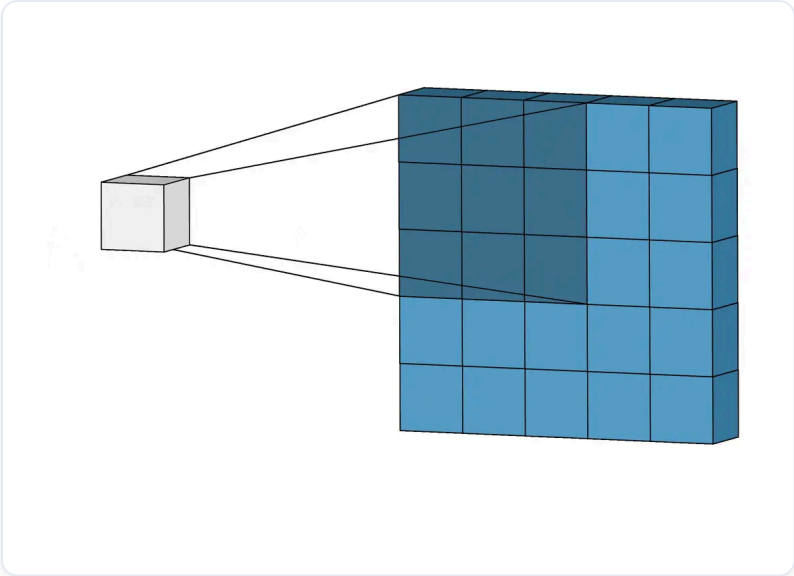
YOLO revoluciona el campo: una sola red neuronal procesa la imagen completa en tiempo real, prediciendo cajas y clases simultáneamente mediante una cuadrícula.

03 **Transformers (2020 →)**

DETR y RF-DETR: pipelines limpios basados en atención, sin anchors predefinidos ni supresión de no-máximos (NMS). Arquitectura más simple y generalizable.

Redes neuronales convolucionales (CNNs)

Las Redes Neuronales Convolucionales (CNNs) son una parte muy importante de la visión artificial moderna. Están diseñadas para procesar datos visuales directamente, utilizando capas especializadas que detectan características como bordes, texturas y patrones en las imágenes de manera jerárquica.



Este enfoque permite a las CNNs aprender representaciones complejas de las imágenes, lo que las hace excepcionalmente potentes para estas tareas.

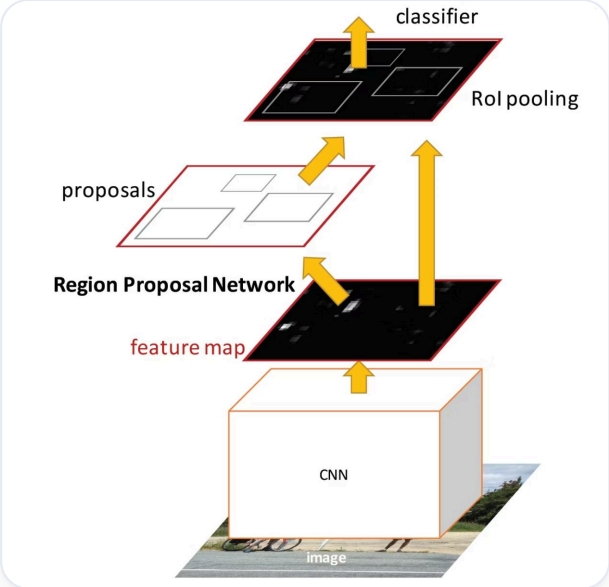
Las CNN incorporan localidad e invariancia aproximada a traslaciones. Siguen siendo componentes útiles en muchos detectores; el contexto global y la atención son decisiones de arquitectura, no una carencia universal de todas las CNN.

Two-Stage: precisión a costa de complejidad

Los detectores de dos etapas, como la familia R-CNN, dividen el problema en **dos fases secuenciales**: primero generan propuestas de regiones candidatas (lugares donde podría haber objetos), y luego clasifican y refinan cada propuesta.

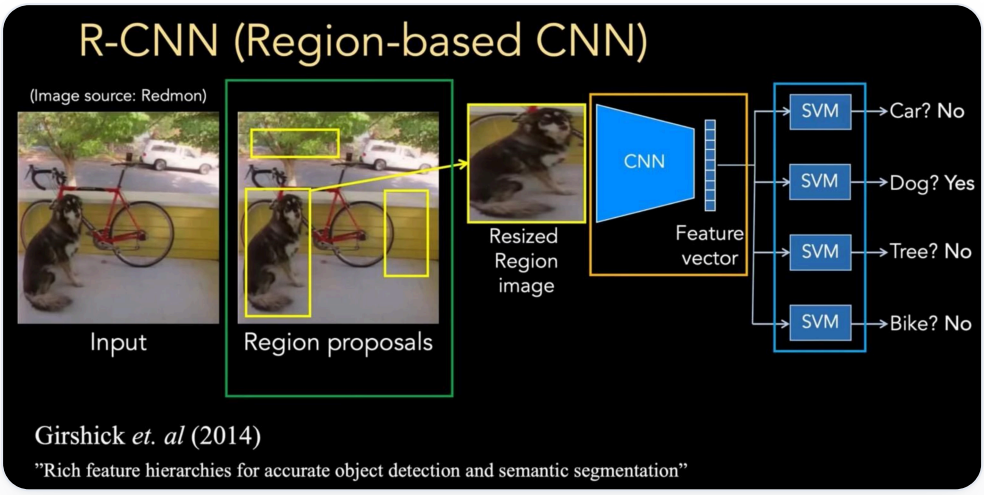
ROI Pooling acelera el proceso al extraer features de forma eficiente, pero el pipeline sigue siendo computacionalmente costoso. La generación inicial de regiones puede proponer cientos de candidatos, ralentizando la inferencia.

Trade-off característico: excelente precisión en benchmarks, pero latencia elevada para aplicaciones en tiempo real. Ideales cuando la exactitud es más importante que la velocidad. Difícil de optimizar.



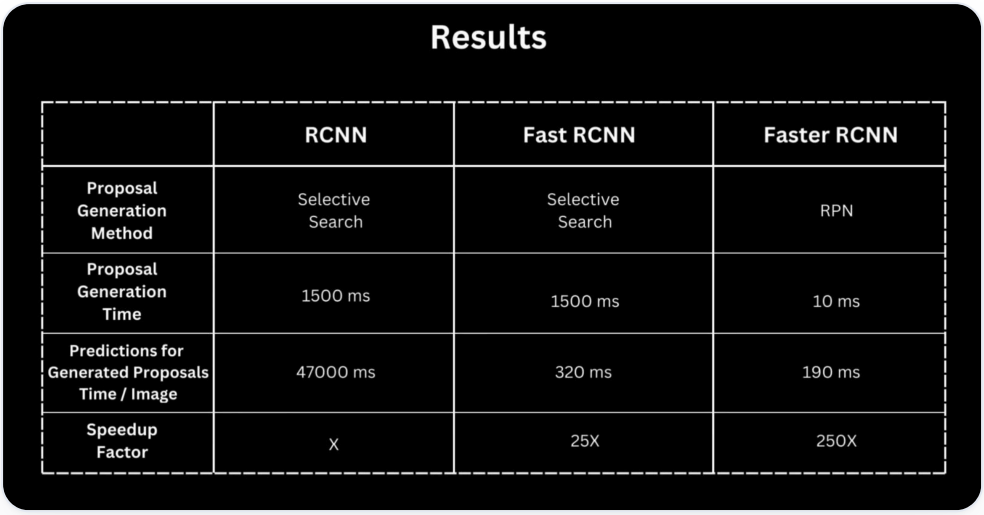
R-CNN

Region proposal + CNN + clasificador.



Girshick et al. (2014), «Rich feature hierarchies for accurate object detection and semantic segmentation».

Comparación de velocidad de RCNNs



El salto está en la generación de propuestas: Selective Search tarda 1500 ms y la RPN de Faster R-CNN, 10 ms.

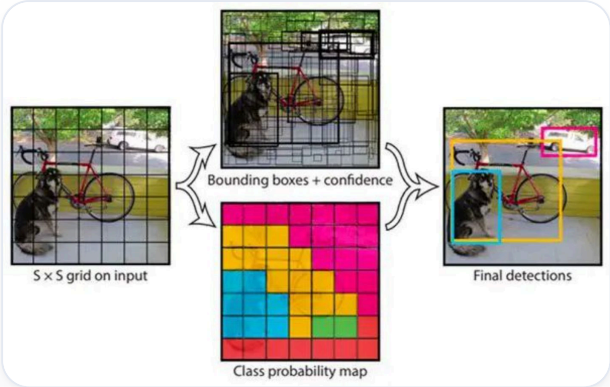
La tabla compara tres generaciones. Método de propuestas: Selective Search en RCNN y Fast RCNN, RPN en Faster RCNN. Tiempo de generación: 1500 ms, 1500 ms y 10 ms. Predicción sobre las propuestas por imagen: 47000 ms, 320 ms y 190 ms. Factor de aceleración: X, 25X y 250X.

YOLO: detección como regresión directa

YOLO (You Only Look Once) reformula la detección como un **problema de regresión único**. La imagen se divide en una cuadrícula S×S, donde cada celda predice directamente las cajas delimitadoras y las probabilidades de clase de los objetos cuyo centro cae en ella.

Ventaja clave: una única pasada hacia adelante por la red permite velocidades de procesamiento en tiempo real (30+ FPS), manteniendo precisión competitiva. Esta arquitectura one-stage elimina la necesidad de generar propuestas de región por separado.

- Cada celda predice múltiples cajas con sus scores de confianza.
- Las predicciones ocurren simultáneamente para toda la imagen.
- Ideal para aplicaciones que requieren latencia mínima.



La arquitectura YOLO: velocidad y simplicidad

Cómo el modelo procesa la imagen en una sola pasada («You Only Look Once»).

01

La estrategia de la cuadrícula (S×S Grid)

YOLO divide la imagen de entrada en una cuadrícula. Si el centro de un objeto cae en una celda específica, esa celda es la responsable de detectarlo.

02

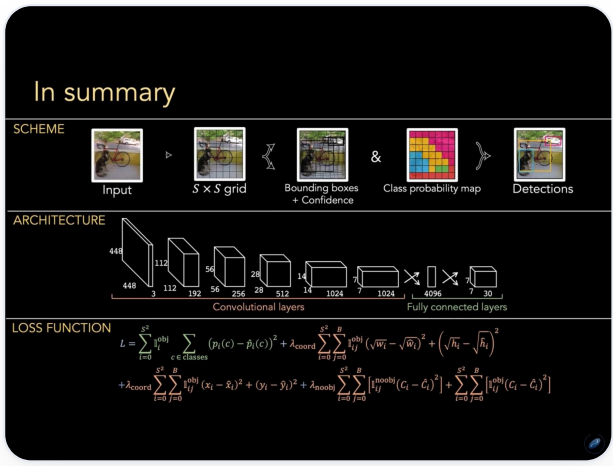
Predicción simultánea (bounding boxes + clases)

Cada celda predice recuadros (bounding boxes) con su ubicación, tamaño y nivel de confianza, y la categoría del objeto (mapa de probabilidad), todo al mismo tiempo.

03

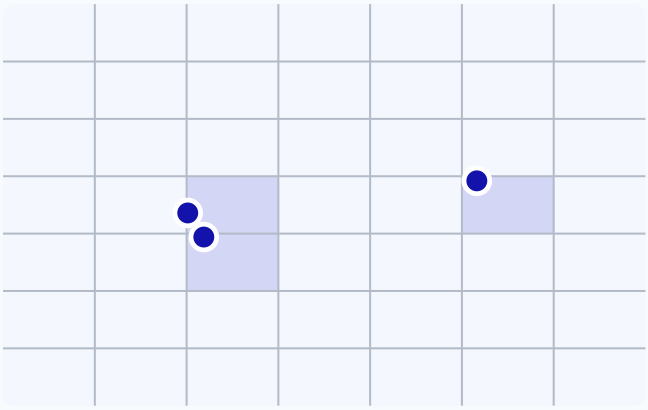
Arquitectura unificada (CNN)

Una única Red Neuronal Convolucional (CNN) procesa la imagen completa, conectando directamente los píxeles de entrada con las coordenadas y clases de salida, eliminando etapas intermedias complejas.



Una celda se responsabiliza de un objeto

Baja la cuadrícula y mira qué pasa cuando dos centros caen en la misma celda.



Cada punto es el centro de un objeto. YOLOv1 usaba una cuadrícula de 7 × 7.

Objetos con celda propia

3 de 3

Cada objeto cae en una celda distinta: la cuadrícula puede hacerse cargo de todos.

Esta página recoge el estado con una cuadrícula de 7 × 7, la que usaba YOLOv1. Al bajar a 6 × 6 o menos, los centros de los dos coches de la izquierda caen en la misma celda y esa celda solo puede responsabilizarse de uno: es la limitación más citada de YOLOv1 con objetos juntos o pequeños. Subir S reparte mejor los centros en general, pero no lo garantiza: con 8 × 8 y con 12 × 12 esos dos centros vuelven a caer juntos, porque lo que decide es dónde quedan las líneas de la rejilla. Además, más celdas multiplican las predicciones y el coste. Las versiones posteriores atacan el problema con predicción multiescala y, más tarde, con enfoques anchor-free.

Una sola pasada, una sola red

Comprueba en qué se diferencia un detector de una etapa.

¿Qué describe la idea de un detector de una etapa?

- A Propone regiones y las clasifica en un segundo recorrido.
- B No necesita imágenes de entrenamiento.
- C Predice localización y categoría en una misma ruta de detección.

Respuesta C. Exacto: esa es la idea central, aunque los detalles cambian entre versiones.

Un detector de una etapa predice cajas y clases en una única pasada sobre toda la imagen, sin una fase previa que proponga regiones candidatas.

Evolución de la familia YOLO (2016-2024)

La serie YOLO ha transformado la detección de objetos, pasando de un concepto novedoso a un ecosistema robusto y versátil.

2016 · YOLOv1

El origen de la detección en tiempo real: un solo vistazo para detectar objetos, superando a los métodos de dos etapas.

2018 · YOLOv3

Estableció el estándar con predicción multiescala y el backbone Darknet-53, optimizando la detección de objetos pequeños.

2020 · YOLOv4

Maximizó la velocidad con CSPDarknet y «Bag of Freebies» (por ejemplo, Mosaic), optimizado para rendimiento en C++.

2020 · YOLOv5

Democratizó YOLO con su implementación en PyTorch, destacando por su facilidad de uso y accesibilidad.

2022 · YOLOv7

Alcanzó el estado del arte en velocidad y precisión para dispositivos Edge con la innovadora arquitectura E-ELAN.

2023 · YOLOv8

Cambio a Anchor-Free y unificó la API para detección, segmentación y estimación de pose. Publicado por ultralytics.

2024 · YOLOv9

Introdujo PGI y GELAN para resolver la pérdida de información en redes profundas sin coste computacional adicional.

Por qué las CNNs no son suficientes

Aunque las CNNs revolucionaron la visión artificial, presentan limitaciones arquitectónicas que restringen su capacidad para tareas más complejas, especialmente en escenarios con objetos distantes o relaciones espaciales no evidentes.

Campo receptivo limitado

Las CNNs procesan información en parches locales. Para entender relaciones a larga distancia, necesitan muchas capas, lo que aumenta la profundidad del modelo y dificulta la propagación de gradientes. Esto es ineficiente para capturar el contexto global de una imagen.

Componentes Ad-Hoc

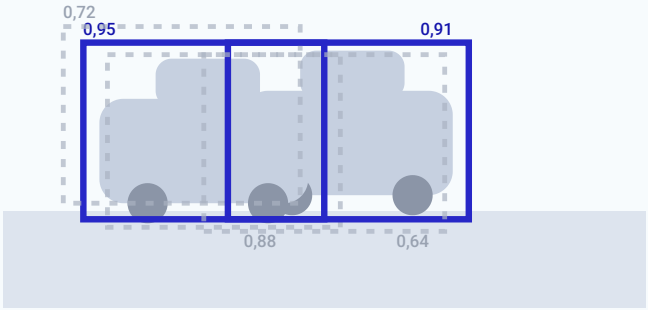
Dependen de elementos manuales y heurísticos como los «anchors» predefinidos (cajas de diferente tamaño y aspecto) y el proceso de Supresión de No Máximos (NMS) para eliminar redundancia. Estos componentes añaden complejidad, hiperparámetros y son difíciles de optimizar globalmente.

Sesgo inductivo fuerte

Asumen una fuerte localidad espacial e invarianza a traslación. Esto es útil para objetos que aparecen en cualquier lugar de la imagen, pero dificulta la modelización de relaciones abstractas o estructuras que no se ajustan a patrones de vecindad directa.

El umbral de NMS es un hiperparámetro

Baja el umbral y desaparece un coche entero; súbelo y vuelven los duplicados.



Cinco cajas candidatas sobre dos coches. El detector las propone todas antes de filtrar.

Cajas que quedan

2

Suprimidas

3

Una caja por coche: el umbral limpia los duplicados sin borrar objetos.

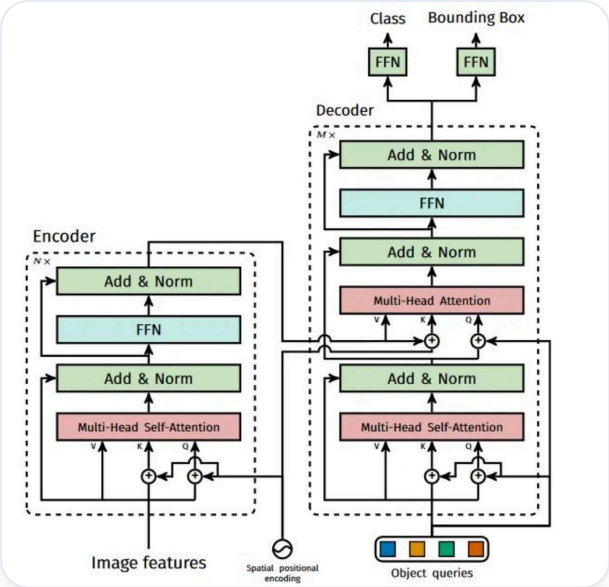
Esta página recoge el estado con umbral 0,50. El algoritmo recorre las cajas de mayor a menor puntuación: cada superviviente suprime a las que solapan con ella por encima del umbral. Con 0,20 la caja del primer coche llega a suprimir la del segundo, porque comparten un 25 % de área, y se pierde un objeto real. Con 0,80 apenas se suprime nada y quedan las cinco cajas. No hay un valor universalmente correcto: depende de cuánto se solapen los objetos del dominio, y por eso DETR y RF-DETR eliminan este paso mediante la asignación uno a uno durante el entrenamiento.

DETR: Transformers para detección

DETR (DEtection TRansformer) introduce un **paradigma completamente nuevo**: utiliza una arquitectura encoder-decoder con «object queries» aprendibles que predicen un conjunto fijo de objetos directamente.

Pipeline simplificado: un backbone (como ResNet) extrae features visuales, el encoder Transformer las procesa globalmente, y el decoder utiliza N queries (típicamente 100) para predecir N potenciales objetos, cada uno con su caja y clase.

- Sin hiperparámetros de NMS que ajustar.
- Sin anchors dependientes del dataset.
- Arquitectura end-to-end diferenciable y limpia.



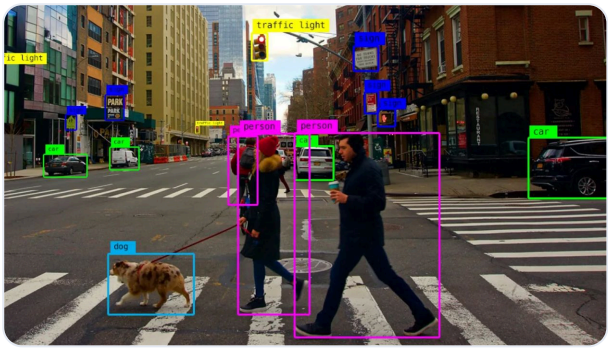
Fin del NMS: matching húngaro

Una de las mayores innovaciones de DETR es la eliminación del proceso de Supresión de No Máximos (NMS) gracias al matching húngaro, simplificando radicalmente el pipeline de detección, un algoritmo de asignación bipartita que encuentra la correspondencia óptima entre predicciones y objetos reales durante el entrenamiento. Esto elimina la necesidad de anchors predefinidos y supresión de no-máximos (NMS).

YOLO (pre-NMS): predicciones redundantes

Predice miles de cajas candidatas y necesita un proceso de filtrado (NMS) para eliminar las duplicadas.

DETR: predicciones directas y únicas



Asigna una predicción única a cada objeto real utilizando un algoritmo de matching húngaro basado en un coste global.

Una predicción por objeto

La asignación bipartita ocurre durante el entrenamiento, no en la inferencia.

¿Para qué sirve la asignación bipartita en DETR durante el entrenamiento?

- A Relacionar predicciones y objetos reales uno a uno al calcular la pérdida.
- B Aplicar NMS a las cajas de cada imagen al predecir.
- C Crear automáticamente etiquetas para entrenar.

Respuesta A. Exacto: define qué objetivo supervisa cada predicción.

El coste combina clase y geometría. Al forzar una correspondencia uno a uno, el modelo aprende a no duplicar predicciones y deja de necesitar NMS al inferir.

RF-DETR

RF-DETR representa un avance significativo al integrar eficientemente la potencia de los Transformers con innovaciones clave, superando a menudo a otros modelos en velocidad y precisión. Su diseño permite una detección de objetos en tiempo real, adaptable a diversas tareas.

Backbone DINOv2 pre-entrenado

Utiliza un potente vision transformer pre-entrenado para un reconocimiento robusto de patrones visuales, lo que facilita una rápida adaptación y reduce la necesidad de grandes volúmenes de datos para nuevas tareas.

Atención deformable eficiente

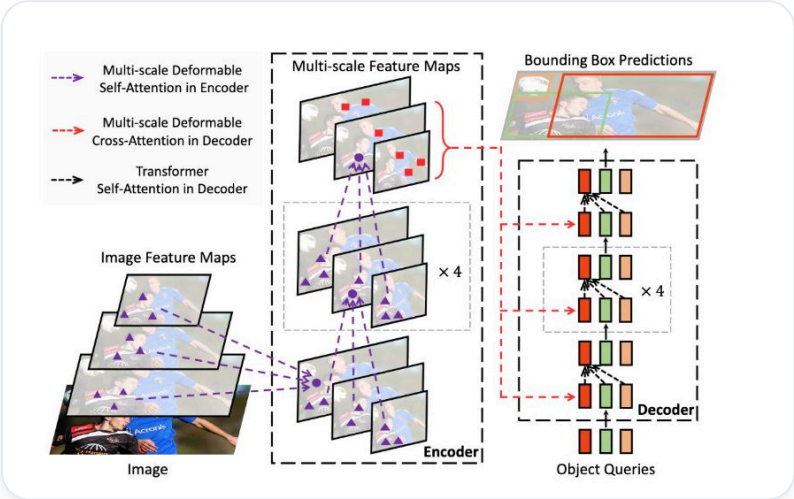
A diferencia de la autoatención estándar, este mecanismo se centra en un pequeño conjunto de puntos de muestreo clave, permitiendo procesar mapas de características de alta resolución de manera eficiente y mejorando la detección de objetos pequeños.

Pipeline modular y sin anchors

Su diseño flexible elimina la necesidad de anchors predefinidos y la supresión de no-máximos (NMS), simplificando la arquitectura y haciéndola más generalizable y fácil de optimizar para distintas aplicaciones.

RF-DETR paso a paso

El funcionamiento de RF-DETR se basa en una arquitectura de transformer diseñada para la detección de objetos, procesando la información visual a través de una serie de pasos secuenciales y refinamientos.



01 **Combinación de features multiescala**

RF-DETR integra features provenientes de diferentes escalas de resolución de la imagen. Esto se realiza antes de que los datos lleguen al encoder del Transformer, asegurando una comprensión contextual rica desde el principio.

02 **Alineación con atención deformable en el encoder**

El encoder del Transformer procesa estas features multiescala y utiliza un mecanismo de atención deformable. Este enfoque permite alinear y enfocar eficientemente las características más relevantes, ajustándose dinámicamente.

03 **Interacción de queries con Cross-Attention**

Las «object queries» interactúan con los niveles de features del encoder a través de un mecanismo de cross-attention. Esto permite que cada query acceda a información relevante de diversas escalas para proponer ubicaciones.

04 **Refinamiento de predicciones en el decoder**

El decoder de RF-DETR consta de múltiples capas que refinan iterativamente las predicciones de los objetos. Cada capa toma las predicciones de la anterior y las mejora.

05 **Salida final de bounding boxes**

Finalmente, el modelo genera directamente las bounding boxes para cada objeto, sin la necesidad de post-procesamiento como NMS, lo que agiliza el proceso de detección.

Por qué NO usamos ultralytics en clase

Ultralytics, a pesar de su popularidad, presenta varias características que lo hacen inadecuado para el entorno académico y de investigación que buscamos promover, especialmente porque queremos fomentar el uso de código verdaderamente open source, transparente y con licencias que no limiten la investigación ni el desarrollo industrial.

Licencia problemática

La combinación de licencias AGPL y comercial fuerza a los usuarios a liberar su código o a pagar, lo que contradice el espíritu de un verdadero open-source y limita su uso comercial.

Apropiación de marca YOLO

La designación de sus versiones (YOLOv5-YOLOv11) como «YOLO oficial» es engañosa, ya que no provienen de la familia original de Joseph Redmon ni de la línea académica, causando desinformación generalizada.

Código no modular

El código está fuertemente orientado a su interfaz de línea de comandos (CLI), lo que dificulta su estudio, comprensión interna y adaptación para fines de investigación o proyectos más personalizados.

Limitaciones académicas

Su enfoque lo hace poco apto para la enseñanza y la investigación seria. La falta de modularidad y la escasa reproducibilidad científica lo alejan de los estándares de los papers académicos.

Las condiciones de licencia se revisan por versión y por artefacto: el código y los pesos pueden tener términos distintos y cambiar entre versiones. Conviene comprobar la licencia vigente del repositorio y del checkpoint concreto antes de decidir.

Por qué SÍ usamos RF-DETR en clase

RF-DETR es la base de nuestro currículo de detección de objetos por su diseño avanzado, su alineación con los principios de la investigación y la enseñanza moderna en visión artificial, y porque es un proyecto completamente open source bajo licencia Apache 2.0.

Licencia Apache 2.0

Libre para industria, investigación y enseñanza. Sin obligaciones ocultas que limiten la innovación o el despliegue de proyectos.

Arquitectura moderna y clara

Basado en la familia DETR, ofrece una comprensión profunda de cómo funciona un detector de objetos de vanguardia, sin la complejidad de sistemas heredados.

Código limpio y modular

Diseñado para ser fácil de leer, modificar y explicar a alumnos, facilitando la experimentación y el aprendizaje de la implementación práctica.

Muy buen rendimiento

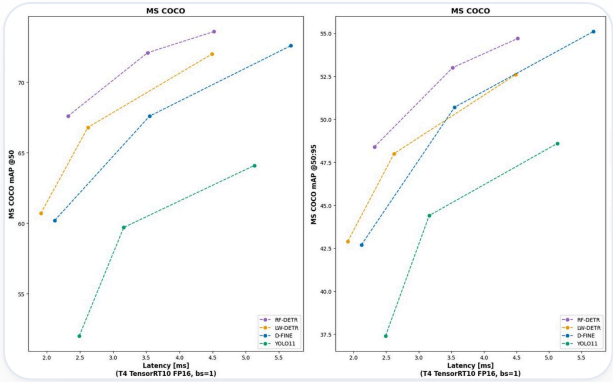
Ofrece una velocidad de inferencia y entrenamiento superior a muchos modelos YOLO modernos, optimizando los tiempos de desarrollo y prueba en el aula.

RF-DETR vs YOLO en producción

La elección entre RF-DETR y YOLO depende de tus **prioridades específicas de producción**.

Consideraciones prácticas:

- **Licencia:** YOLO suele usar AGPL (obliga a publicar código modificado); RF-DETR usa Apache 2.0 (más permisiva).
- **Mantenibilidad:** RF-DETR evita el ajuste manual de anchors y thresholds de NMS.
- **Adopción:** YOLO tiene ecosistema más maduro y extenso.



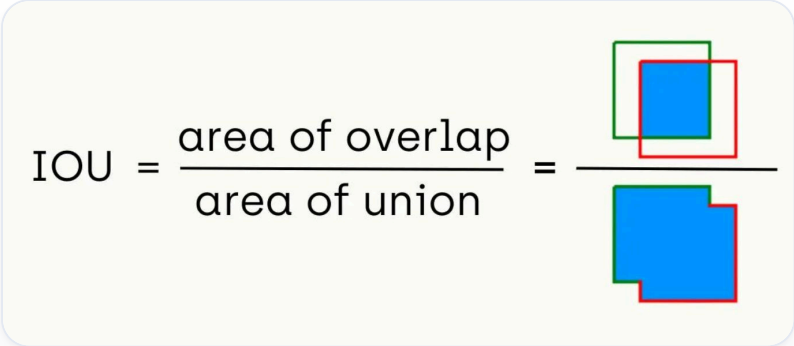
Latencia frente a mAP en MS COCO, medida en T4 con TensorRT10 y FP16.

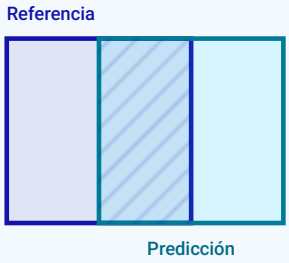
Las curvas proceden del repositorio de RF-DETR y se miden en un hardware concreto. Antes de decidir conviene reproducir la medida en el dispositivo de destino, con el tamaño de entrada, la precisión numérica y el tamaño de lote que se vayan a usar realmente.

IoU (Intersection over Union)

La **Intersection over Union (IoU)** es una métrica fundamental que mide el solapamiento entre dos cajas delimitadoras: una caja predicha por nuestro modelo y una caja de verdad (ground truth). Esta métrica es crucial para determinar si una predicción es correcta (True Positive) o no (False Positive), y su valor se encuentra en un rango de 0 (sin solapamiento) a 1 (solapamiento perfecto).

Para visualizarlo, considera una caja delimitadora predicha por el modelo (en rojo) y la caja de verdad que marca la ubicación real de un objeto (en verde). La IoU calcula el área donde ambas cajas se superponen, dividida por el área total que ocupan ambas cajas combinadas.





IoU

0.33

Solapamiento parcial.

Umbrales típicos de IoU

- **IoU > 0.5:** es el umbral común para considerar una detección como True Positive (TP), especialmente utilizado en el conjunto de datos COCO.
- **IoU > 0.75:** un umbral más estricto que se usa para evaluar la alta precisión de las detecciones.
- **IoU < umbral:** cualquier detección con un IoU inferior al umbral establecido se clasifica como False Positive (FP).

¿Por qué es importante?

- Sin IoU, no podríamos distinguir eficazmente entre True Positives (detecciones correctas) y False Positives (falsas alarmas).
- Es la base para el cálculo de métricas más complejas como la precisión, el recall, la Average Precision (AP) y la mean Average Precision (mAP).

- Determina directamente la calidad y la exactitud espacial de las detecciones de objetos de nuestro modelo.

IoU vale 1 cuando ambas cajas coinciden y 0 cuando no comparten área. Para decidir TP, FP y FN también se tiene en cuenta la clase, la correspondencia uno a uno y un protocolo de evaluación.

Fundamentos: TP, FP, FN (y por qué no hay TN)

TP (True positives)

Detecciones correctas: cajas predichas que coinciden con objetos reales (IoU > umbral).

FP (False positives)

Falsas alarmas: cajas predichas donde no hay objeto real o con IoU muy bajo.

FN (False negatives)

Objetos perdidos: objetos reales que el modelo no detectó.

Por qué NO existen True Negatives (TN) en object detection

La métrica de True Negatives, común en problemas de clasificación binaria, no aplica de la misma manera en la detección de objetos:

- **En clasificación binaria:** TN = «predice que no hay gato y efectivamente no hay gato».
- **En detección de objetos:** el modelo predice cajas delimitadoras en ubicaciones específicas donde cree que hay un objeto.
- No existe el concepto explícito de «predecir ausencia de objeto en una ubicación».
- Las ubicaciones donde no se predice nada son infinitas e impracticables de medir.

Por esta razón, las métricas en detección de objetos se enfocan únicamente en **TP, FP y FN**, ya que son los resultados relevantes de las predicciones del modelo.

Métricas de evaluación: precision y recall

En la detección de objetos, entender la calidad de las predicciones va más allá de solo saber si se detectó algo. Precision y recall nos dan una visión completa de la fiabilidad y exhaustividad del modelo.

La importancia de la precisión

Mide la **confiabilidad de las detecciones positivas**. Un detector con alta precisión minimiza los falsos positivos (detectar algo que no está ahí o identificarlo incorrectamente). Es crucial en aplicaciones donde un error de detección tiene costes elevados, como en inspección de calidad o sistemas de seguridad.

$$\text{Precision} = \frac{TP}{TP + FP}$$

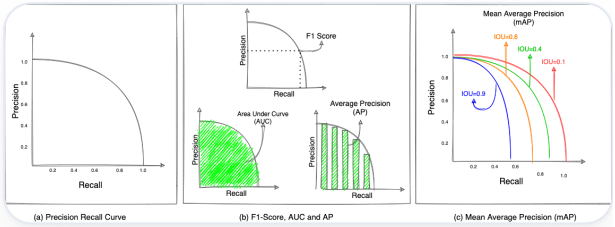
Esto responde a la pregunta: «De todas mis predicciones, ¿cuántas son correctas?».

La importancia del recall

Mide la **capacidad del modelo para encontrar todos los objetos relevantes** en una imagen. Un alto recall significa pocos falsos negativos (no detectar un objeto que sí está presente). Es vital en escenarios donde es crítico no omitir ningún objeto, como la detección de personas en vigilancia o tumores en imágenes médicas.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Esto responde a la pregunta: «De todos los objetos reales, ¿cuántos detecté?».



Ejemplos prácticos de interpretación:

- Si un modelo tiene una precision alta (por ejemplo, 0.9), significa que el 90 % de las detecciones que realiza son correctas. Es muy fiable en lo que afirma haber encontrado.
- Si un modelo tiene un recall alto (por ejemplo, 0.8), significa que detectó el 80 % de todos los objetos que realmente estaban presentes en las imágenes. Es bueno para no perder objetos.

Es importante entender que precision y recall a menudo presentan un **trade-off**. Mejorar una métrica a menudo puede llevar a la disminución de la otra. Por ejemplo, si un modelo es muy agresivo en sus detecciones para asegurar que no se pierda nada (alto recall), podría también generar más falsas alarmas (baja precision). La elección del equilibrio adecuado depende de los requisitos específicos de la aplicación.

Calcula precisión y recall

Aplica las dos fórmulas a un recuento concreto.

Hay 3 TP, 1 FP y 2 FN. ¿Cuáles son precisión y recall?

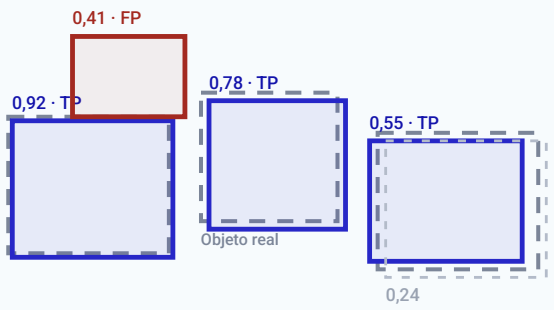
- A Precisión 0,60; recall 0,75.
- B Precisión 0,75; recall 0,75.
- C Precisión 0,75; recall 0,60.

Respuesta C. Exacto: 3/(3+1) y 3/(3+2).

Con 3 TP, 1 FP y 2 FN: precisión = 3 / (3 + 1) = 0,75 y recall = 3 / (3 + 2) = 0,60. La precisión mira el denominador de las predicciones; el recall, el de los objetos reales.

Subir el umbral cambia el equilibrio

Precisión y recall se mueven en direcciones opuestas cuando filtras por puntuación.



Precisión

0,75

Recall

1,00

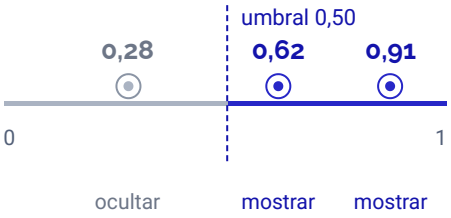
TP 3 FP 1 FN 0

Se ven todos los objetos, pero también sobran predicciones.

Esta página recoge el estado con umbral 0,30. El emparejamiento sigue el protocolo habitual: las predicciones se ordenan por puntuación y cada una reclama el objeto libre con el que comparte más área, siempre que el IoU llegue a 0,50. La predicción de 0,24 solapa con el tercer objeto, pero llega tarde: ese objeto ya está emparejado, así que cuenta como falso positivo duplicado. Al subir el umbral desaparecen antes las predicciones dudosas: sube la precisión y baja el recall. La curva precisión-recall recorre ese intercambio umbral a umbral.

Filtrar no cambia las cajas

El umbral de confianza decide qué predicciones se muestran; IoU evalúa su solapamiento.



El filtro cambia qué predicciones ves, no sus coordenadas.

Subes el umbral de confianza y mantienes fijas las predicciones candidatas.

- A Se conservan las mismas o menos cajas.
- B Aparecen categorías nuevas.
- C Aumenta necesariamente la IoU de todas las cajas restantes.

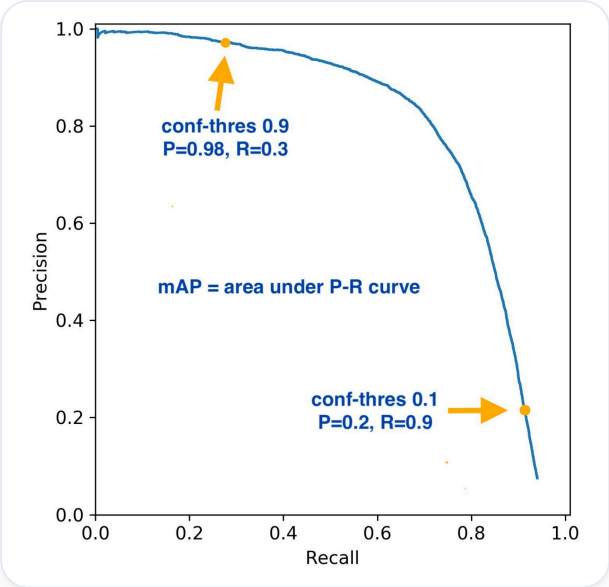
Respuesta A. Exacto: el filtro elimina las puntuaciones que quedan por debajo del nuevo umbral.

Al subir el umbral suelen desaparecer predicciones de baja puntuación: pueden bajar los falsos positivos y también aumentar los falsos negativos. La curva precisión-recall estudia ese intercambio recorriendo muchos umbrales.

Métricas de evaluación: AP y mAP

AP (Average Precision): representa el área bajo la curva Precision-Recall. Mide la habilidad del modelo para detectar todos los objetos relevantes (recall alto) mientras mantiene un número bajo de falsos positivos (precisión alta). Un valor de AP más alto indica un mejor rendimiento.

mAP (mean Average Precision): es la métrica estándar para evaluar detectores de objetos. Se calcula promediando los valores de AP para todas las clases de objetos detectadas, y a menudo, en diferentes umbrales de IoU (ej., mAP@0.5 o mAP@[0.5:0.95]).



Recorrer el umbral de confianza traza la curva: con umbral 0,9 la precisión es alta y el recall bajo; con 0,1 ocurre lo contrario. AP50 y AP promediada entre varios umbrales de IoU responden a protocolos distintos y no son intercambiables.

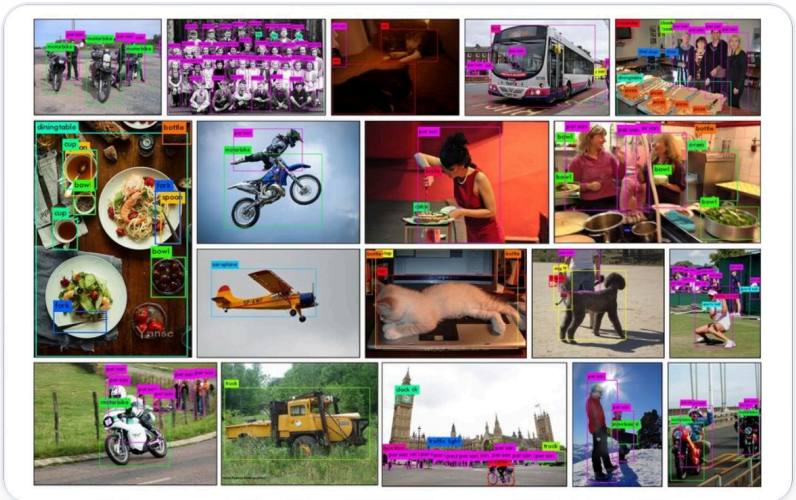
Límites prácticos: COCO y el mundo real

El dataset COCO, con sus 80 clases estándar (persona, coche, perro, silla…), es excelente para benchmarking pero normalmente **no cubre las necesidades reales de industria o dominios especializados**.

Problema del dominio: una fábrica necesita detectar «tornillo oxidado», «conector suelto», «clip roto» o «soldadura defectuosa». Estos objetos específicos no existen en COCO.

Coste de adaptación: re-anotar miles de imágenes para nuevas categorías es caro y lento. Entrenar desde cero o fine-tunear requiere tiempo de GPU, experimentación y validación.

Esta limitación motiva la necesidad de **métodos más flexibles**: detectores guiados por texto (zero-shot), modelos fundacionales multimodales, y técnicas que puedan adaptarse rápidamente a nuevos dominios sin anotación exhaustiva.



El vocabulario depende de los datos

Un checkpoint aprende las clases de sus datos. Tu aplicación puede necesitar otras.

Tu checkpoint solo conoce clases COCO y necesitas detectar un defecto industrial nuevo.

- A Bajar mucho el umbral para crear la categoría.
- B Evaluar adaptación con datos etiquetados o un modelo adecuado al nuevo vocabulario.
- C Aumentar el tamaño de las cajas existentes.

Respuesta B. Exacto: la nueva categoría exige una estrategia y una evaluación en el dominio.

Un detector de vocabulario cerrado solo puede nombrar las categorías con las que fue entrenado. Ampliarlo exige anotar y reentrenar, o cambiar a un enfoque guiado por texto.

Buenas prácticas al evaluar detectores

Evaluar detectores va más allá de mirar mAP en COCO. Para deployment real, debes:

1. Medir en tu dominio específico: crea un conjunto de validación representativo de tus datos reales. Un modelo con 60 % mAP en COCO puede tener 40 % o 80 % en tu caso de uso particular.

2. Balancear velocidad vs precisión: un detector con 2 % más de AP pero 50 % más lento puede no valer la pena si necesitas procesar 30 FPS. Mide latencia end-to-end, no solo teóricos.

3. Analizar errores cualitativamente: inspecciona visualmente los casos de:

- **Falsos Positivos (FP):** detecciones donde no hay objeto.
- **Falsos Negativos (FN):** objetos que el modelo no detectó.
- **Casos límite:** objetos parcialmente ocluidos, muy pequeños, o en condiciones de iluminación extrema.

Los patrones de error te dirán si necesitas más datos, data augmentation específico, o cambiar de arquitectura.

Mide donde vas a usarlo

Un benchmark no sustituye datos representativos, latencia medida y análisis de errores.

Un detector logra mejor AP en un benchmark publicado. ¿Basta para elegirlo?

- A Sí, cualquier mejora de AP garantiza mejor producción.
- B Sí, si es el modelo más reciente.
- C No: hay que medir calidad y latencia con datos y condiciones representativos.

Respuesta C. Exacto: la elección depende del problema, del entorno y de los errores que aceptas.

Un AP mejor en un benchmark público indica rendimiento sobre ese conjunto y ese protocolo. La decisión necesita además datos de tu dominio, latencia medida en el hardware de destino y revisión de los errores que importan.

El siguiente paso

La evolución de la Visión Artificial nos ayuda a superar limitaciones clave de estos modelos, abriendo paso a nuevas arquitecturas más flexibles y potentes.

Limitaciones actuales

- Solo 80 clases COCO
- No entienden lenguaje
- Requieren reentrenamiento

Modelos multimodales

- Entienden texto + imagen
- Zero-shot con lenguaje natural
- Sin reentrenamiento constante

Detectar objetos en seis líneas

Lo mínimo que necesitas para tener cajas sobre una imagen tuya.

```
from rfdetr import RFDETRMedium
from PIL import Image

model = RFDETRMedium()
image = Image.open("mi_imagen.jpg").convert("RGB")

detections = model.predict(image, threshold=0.5)
print(detections.class_id, detections.confidence, detections.xyxy)
```

- **threshold** es el umbral de confianza: súbelo si ves falsas alarmas, bájalo si se pierden objetos.
- **xyxy** son las coordenadas de cada caja en píxeles, en el orden x mínimo, y mínimo, x máximo, y máximo.
- La primera ejecución descarga los pesos. En Colab conviene llamar antes a `model.optimize_for_inference()`.

RF-DETR se publica bajo Apache 2.0 y devuelve las clases de COCO. Para un vocabulario propio hace falta fine-tuning con un dataset anotado, que es lo que mide la pantalla siguiente.

¿De dónde han salido esos pesos?

La receta anterior baja un modelo ya entrenado antes de detectar nada. Ese paso silencioso es el tema siguiente.

Por el paquete de Roboflow

Es lo que acabas de ejecutar. `rfdetr` trae el modelo, la descarga y el `predict` en una sola pieza, y es también por donde se reentrena con tus propias imágenes.

Por el Hub de Hugging Face

El mismo RF-DETR está publicado como `Roboflow/rf-detr-medium` con licencia Apache 2.0, así que `transformers` lo carga igual que a cualquier otro modelo del curso.

```
from transformers import AutoImageProcessor, RfDetrForObjectDetection

processor = AutoImageProcessor.from_pretrained("Roboflow/rf-detr-medium")
model = RfDetrForObjectDetection.from_pretrained("Roboflow/rf-detr-medium")
```

Ese identificador de dos partes, **quién publica** y **qué publica**, es la unidad con la que se trabaja a partir de aquí. El tema 2 va de leer lo que hay detrás: qué licencia trae, cuánta memoria pide y cómo elegir entre los cientos que dicen hacer lo mismo.

Las dos vías cargan los mismos pesos y se diferencian en el envoltorio. El paquete `rfdetr` expone la API de Roboflow, con `predict` y el entrenamiento propio; `transformers` expone la interfaz común de la librería, que es la que comparten todos los modelos de los temas siguientes y la que permite intercambiar un detector por otro sin reescribir el resto. El identificador y el nombre de clase están tomados de la ficha del modelo, consultada el 17 de septiembre de 2026. Hasta este curso RF-DETR solo estaba disponible por el paquete, así que esta equivalencia es nueva.

La misma familia hace tres cosas

El paquete que acabas de usar trae ahora segmentación y puntos clave. Cambia la clase y cambia lo que recibes, con la misma foto y las mismas tres líneas.

```
from rfdetr import RFDETRMedium

modelo = RFDETRMedium()
salida = modelo.predict("foto.jpg", threshold=0.5)

print(salida.xyxy)        # una caja por objeto
print(salida.class_id)    # su clase de COCO
```

Una caja por objeto, en píxeles. Sirve para contar, para disparar un aviso y para recortar la zona antes de mirarla con más detalle.

El ejercicio

Haz una foto a tu mesa, con una persona dentro si puedes, y pasa las tres. Luego responde con cada salida a la misma pregunta: ¿cuánto ocupa ese objeto? La caja da un rectángulo, la máscara da los píxeles de verdad y el esqueleto no responde, porque mide otra cosa.

Para pintarlo

Las tres salidas son objetos de `supervision`, así que se dibujan con `sv.BoxAnnotator`, `sv.MaskAnnotator` y `sv.EdgeAnnotator` sin tocar nada más.

Que las tres tareas salgan de la misma familia no es una comodidad de la API: comparten el mismo backbone DINOv2 y solo cambia la cabeza. Es el argumento de todo el curso visto en tres líneas de código.

Segmentación y puntos clave se publican como «preview», es decir, acceso temprano pensado sobre todo para ajustar el modelo con un dataset propio; las clases son `RFDETRSegPreview` y `RFDETRKeypointPreview`, y la licencia del paquete abierto es Apache 2.0. Los puntos clave son los 17 de COCO, los mismos que se evalúan con OKS en el tema 6. Las variantes de detección aparecen en el tema 1, la segmentación es lo que se estudia en el tema 5 con SAM y la pose, en el tema 6: esta pantalla adelanta que el mismo paquete cubre las tres, con la diferencia de que aquí las clases son las de COCO y con SAM se segmenta cualquier cosa sin lista de clases. Consultado el 17 de septiembre de 2026 en la documentación de Roboflow.

¿Cuánto cuesta anotar tu dataset?

La cuenta que decide si fine-tuneas o te apoyas primero en un detector guiado por texto.

Clases que necesitas **4**

Imágenes por clase **300**

Segundos por imagen **45**

Imágenes que anotar

1200

Horas, con revisión incluida

20

Jornadas de una persona

3,3

Coste asumible para un conjunto propio. Aun así, empieza validando con zero-shot para saber si hace falta.

Estimación de orden de magnitud. Suma un 30 % de revisión sobre el tiempo de anotación y cuenta 6 horas productivas por jornada.

La página recoge el caso de 4 clases, 300 imágenes por clase y 45 segundos por imagen. Las cifras que más se disparan son las clases y los segundos por imagen: anotar segmentación o puntos clave multiplica el tiempo respecto a poner cajas. Antes de asumir ese coste conviene comprobar cuánto resuelve un detector guiado por texto sin anotar nada, que es el tema siguiente.

¿Qué error te sale más caro?

El umbral no se elige en abstracto: depende de qué duele más en tu aplicación.

¿Para qué usas el detector?

- (●) Seguridad: detectar a una persona en zona restringida
- ( ) Inspección: encontrar piezas defectuosas antes de enviarlas
- ( ) Inventario: contar producto en una estantería
- ( ) Analítica: estimar flujo de gente en una calle

Prioriza el recall

No detectar a alguien que sí está es el fallo grave. Una falsa alarma cuesta una comprobación; un objeto perdido cuesta el incidente.

Qué hacer: baja el umbral de confianza y asume más falsos positivos. Añade una segunda comprobación, humana o automática, para filtrarlos.

Prioriza el recall, con revisión humana

Dejar pasar una pieza defectuosa llega al cliente; marcar una buena solo cuesta una revisión en planta.

Qué hacer: umbral bajo y una estación de revisión. Mide cuántas piezas buenas estás parando, porque ese es el coste que notará producción.

Prioriza la precisión

Contar de más es peor que contar de menos: dispara reposiciones y pedidos equivocados que nadie revisa.

Qué hacer: sube el umbral hasta que apenas haya duplicados y vigila el recuento frente a un conteo manual de control.

Busca el equilibrio y mide la tendencia

Aquí no importa cada persona concreta sino que el sesgo sea estable. Un error constante se corrige; uno que cambia con la hora o el clima, no.

Qué hacer: deja el umbral intermedio y valida sobre grabaciones de momentos distintos, no sobre una sola.

La regla general: si el coste de perder un objeto supera al de revisar una falsa alarma, baja el umbral; si es al revés, súbelo. Y en todos los casos, mide el efecto sobre tus propios datos, no sobre el benchmark.

Antes de poner un detector en producción

Ocho comprobaciones que evitan casi todos los sustos.

- [ ] **Conjunto de validación propio**
  Imágenes de tu dominio, tus cámaras y tus condiciones de luz. No basta el mAP del benchmark.
- [ ] **Umbral elegido con criterio**
  Decidido según qué error cuesta más en tu caso, no dejado en el valor por defecto.
- [ ] **Latencia medida en el dispositivo real**
  End to end, incluyendo carga, preprocesado y posprocesado. No el número del paper.
- [ ] **Revisión visual de los errores**
  Mira falsos positivos, falsos negativos y casos límite: ocluidos, pequeños, contraluz.
- [ ] **Licencia comprobada**
  Del repositorio y de los pesos, en la versión concreta que vas a desplegar.
- [ ] **Versión y revisión fijadas**
  Anota el identificador del modelo y su revisión para poder reproducir el resultado.
- [ ] **Comportamiento ante entradas raras**
  Imagen vacía, muy oscura, resolución distinta, formato inesperado.
- [ ] **Plan de seguimiento**
  Cómo vas a detectar que el rendimiento cae cuando cambien las condiciones.

Ahora, con tu propia imagen

Prueba RF-DETR en tu copia del cuaderno.

1. Guarda una copia en Drive.
2. Ejecuta el detector con una imagen.
3. Cambia el umbral de confianza: **0,25 · 0,50 · 0,75**.

Un detector cerrado reconoce sus clases; el siguiente bloque añadirá lenguaje y zero-shot.

Abrir en Colab ↗

En Colab: Archivo → Guardar una copia en Drive. Empieza por inferencia. La disponibilidad de GPU varía; el entrenamiento necesita además el dataset indicado en el notebook. La comparación entre 0,25, 0,50 y 0,75 permite observar qué predicciones desaparecen sin confundir confianza con IoU. El siguiente tema amplía el vocabulario cerrado mediante modelos que relacionan visión y lenguaje.