



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

 etsinf

Escola Tècnica
Superior d'Enginyeria
Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Detección de defectos en objetos en movimiento mediante Redes Neuronales Convolucionales con optimizaciones específicas para hardware NVIDIA

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Haro Armero, Abel

Tutor: Flich Cardo, José
López Rodríguez, Pedro Juan

Curso 2024-2025

Resum

????

Paraules clau: ????, ????????, ????, ??????????????????

Resumen

????

Palabras clave: ????, ???, ??????????????????

Abstract

????

Key words: ????, ????, ????, ??????????????

Índice general

Índice general	v
Índice de figuras	vii
Índice de tablas	vii
<hr/>	
1 Introducción	1
1.1 Motivación	3
1.2 Objetivos	3
1.3 Estructura de la memoria	4
2 Conceptos Previos	5
2.1 Fundamentos y avances en redes neuronales para visión artificial	5
2.1.1 Fundamentos de la inteligencia artificial	5
2.1.2 Tareas fundamentales en visión por computador	6
2.1.3 Arquitectura y funcionamiento de las CNN	7
2.1.4 Detectores de dos etapas	12
2.1.5 Detectores de una etapa	13
2.1.6 Métricas de evaluación	15
2.2 Aceleradores de procesamiento gráfico	16
2.2.1 Limitaciones del hardware tradicional	16
2.2.2 Arquitectura y funcionamiento de las GPUs	18
2.2.3 Serie Jetson: Dispositivos de IA de bajo consumo	19
2.2.4 TensorRT	20
2.3 Seguimiento de objetos en tiempo real	21
2.3.1 Introducción al seguimiento de objetos	21
2.3.2 BYTETrack	23
2.4 Slicing Aided Hyper Inference	25
3 Análisis del problema	27
4 Diseño e implementación de la solución	29
4.1 Descripción del sistema	29
4.2 Diseño de las etapas del sistema	29
4.3 Segmentación de las etapas del sistema	29
5 Análisis de la solución	31
5.1 Variación de los parámetros	31
5.2 Tipo de segmentación	31
5.3 Talla del modelo	31
5.4 Precisión del modelo	31
5.5 Modo de energía y cores de la CPU	32
5.6 Tamaño de la imagen	32
6 Prueba de concepto	33
6.1 Construcción del entorno	33
6.2 Instalación del entorno	33
7 Conclusiones	35

Bibliografía	37
<hr/>	
Apéndices	
A Configuración del sistema	39
A.1 Fase de inicialización	39
A.2 Identificación de dispositivos	39
B ??? ?????????????? ????	41

Índice de figuras

1.1	Evolución del interés público en inteligencia artificial según datos de Google Trends (2020-2025)	1
1.2	Proyección del consumo eléctrico de los centros de datos en el mundo	2
2.1	Estructura de un perceptrón multicapa (MLP).	6
2.2	Tareas fundamentales en visión por computador.	7
2.3	Relación entre Machine Learning, Deep Learning, CNN, Computer Vision y Human Vision.	7
2.4	Operación de convolución sobre los pixeles de una imagen.	8
2.5	Proceso de convolución aplicado a una imagen de un autobús.	8
2.6	Ejemplo de operación de max-pooling con una ventana de 2×2 y un paso de 2. Se selecciona el valor máximo de cada región de color.	9
2.7	Arquitectura de LeNet-5.	12
2.8	Proceso de búsqueda selectiva aplicado a una imagen.	12
2.9	Arquitectura de R-CNN.	13
2.10	Ejemplo de detección de objetos utilizando YOLO.	14
2.11	Arquitectura de YOLO	14
2.12	Evolución histórica de las características de los microprocesadores (1970-2020).	17
2.13	Comparativa de arquitecturas CPU y GPU.	18
2.14	Módulos Jetson de NVIDIA.	20
2.15	Ejemplo de flujo de trabajo de optimización con TensorRT.	21
2.16	Diagrama de flujo del filtro de Kalman.	22
2.17	Comparativa de rendimiento de BYTETrack con otros algoritmos de seguimiento.	23
2.18	Ejemplo de detección y seguimiento de objetos utilizando BYTETrack.	24

Índice de tablas

2.1	Análisis comparativo de las variantes de YOLO11 considerando precisión, velocidad y complejidad computacional.	15
2.2	Comparativa técnica entre diferentes modelos NVIDIA Jetson.	20
5.1	Comparación de modelos en términos de inferencia, consumo de energía y potencia.	31

CAPÍTULO 1

Introducción

Durante los últimos años, la inteligencia artificial ha experimentado un crecimiento en popularidad sin precedentes, transformando nuestra capacidad tecnológica con herramientas revolucionarias. Este avance ha sido impulsado por la disponibilidad de grandes volúmenes de datos, el desarrollo de algoritmos avanzados y las mejoras significativas en el hardware de procesamiento, que han permitido a las máquinas aprender y adaptarse a situaciones complejas. Algunos campos destacados de aplicación incluyen el procesamiento del lenguaje natural, la visión por computador y la robótica. En particular, la visión por computador ha visto un auge significativo, con aplicaciones en áreas como la seguridad, la medicina y la automoción. Esta creciente popularidad por el mundo de la inteligencia artificial se refleja en la evolución del interés público en ella, como muestra la Figura 1.1.

Esto es un texto de prueba Artificial Intelligence (AI) y luego se vuelve a repetir AI.

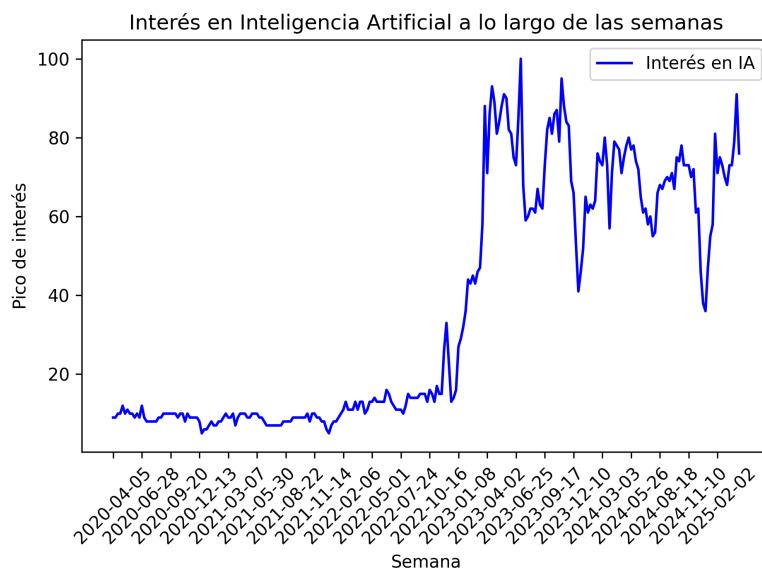


Figura 1.1: Evolución del interés público en inteligencia artificial según datos de Google Trends (2020-2025)

El progreso en visión por computador ha sido posible gracias a los avances en redes neuronales convolucionales, que han revolucionado la capacidad de los sistemas para detectar y clasificar objetos en imágenes y vídeos con una gran precisión y velocidad.

Estos algoritmos de visión artificial requieren una potencia computacional significativa tanto para su entrenamiento como para su ejecución. Las CPUs (Unidades Centrales de Procesamiento) tradicionales resultan insuficientes para estas tareas, por lo que la industria ha desarrollado arquitecturas específicas como las GPUs (Unidades de Procesamiento Gráfico), TPUs (Unidades de Procesamiento Tensorial) y DLAs (Aceleradores de Aprendizaje Profundo). Estos componentes están optimizados para ejecutar operaciones de entrenamiento e inferencia de manera eficiente, permitiendo implementar sistemas de visión artificial capaces de procesar información visual en tiempo real. Sin embargo, estos aceleradores suelen presentar un consumo energético elevado, lo que plantea importantes retos de eficiencia y sostenibilidad.

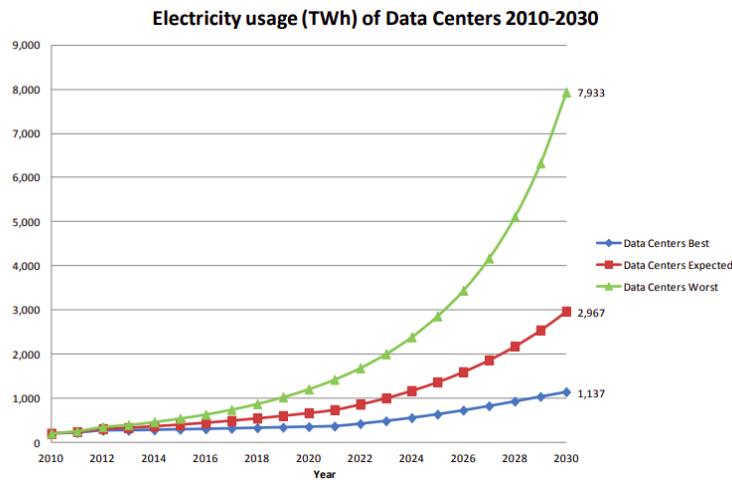


Figura 1.2: Proyección del consumo eléctrico de los centros de datos en el mundo

Como se observa en la Figura 1.2, el consumo eléctrico de los centros de datos en el mundo ha ido aumentando de forma exponencial, lo que plantea un desafío significativo para la sostenibilidad del crecimiento tecnológico [1]. En el peor escenario, esta tendencia podría llevar a un incremento insostenible en la huella de carbono del sector tecnológico, mientras que en el mejor de los casos, la adopción de tecnologías eficientes podría moderar el crecimiento. Este aumento del consumo energético no solo afecta a los centros de datos, sino también a los dispositivos embebidos y móviles, donde la eficiencia energética es crucial para prolongar la vida útil de las baterías y reducir el impacto ambiental.

Para enfrentar estos desafíos, se han desarrollado diversas técnicas de optimización y compresión que reducen el tamaño y la complejidad de los modelos neuronales manteniendo su rendimiento. Paralelamente, han surgido arquitecturas hardware específicamente diseñadas para la inferencia de modelos de aprendizaje profundo en entornos con restricciones energéticas. En este contexto, los dispositivos de la serie Jetson de NVIDIA destacan por ofrecer un equilibrio entre alto rendimiento en tareas de inteligencia artificial y un consumo energético contenido, ideal para aplicaciones embebidas de visión artificial.

La combinación de redes neuronales convolucionales y aceleradores hardware ha permitido la creación de sistemas de visión artificial que pueden detectar y clasificar objetos en movimiento, lo que es esencial en aplicaciones como la vigilancia, la conducción autónoma y la robótica.

1.1 Motivación

Los humanos somos capaces de ver y entender el mundo que nos rodea. Dada una imagen, podemos identificar objetos, reconocer patrones y tomar decisiones basadas en la información visual. Sin embargo, esta capacidad no es innata en las máquinas. La visión por computador es la ciencia que busca dotar a las máquinas de la capacidad de interpretar y comprender imágenes y videos, emulando la forma en que los humanos percibimos el entorno.

Como se mencionó anteriormente, la inteligencia artificial ha revolucionado la forma en que interactuamos con la tecnología. Se ha convertido en una herramienta esencial para aplicar soluciones innovadoras en una amplia gama de campos. En particular, la visión por computador ha demostrado ser un área de gran potencial. También la existencia de dispositivos de bajo consumo, como los de la serie Jetson de NVIDIA, ha permitido llevar la inteligencia artificial a entornos de edge computing (cómputo en el borde), donde se acerca el procesamiento de datos a la fuente de información. Esto reduce la latencia y el consumo energético. Con todo esto, se abre un abanico de posibilidades para la implementación de sistemas de visión artificial en aplicaciones industriales.

Centrándose en el ámbito industrial, la detección y clasificación de objetos en movimiento es crucial para optimizar procesos, mejorar la seguridad y aumentar la eficiencia. En la mayoría de entornos productivos, la detección de defectos se realiza de forma manual, lo que puede ser ineficiente y propenso a errores. La automatización de este proceso mediante sistemas de visión artificial puede reducir costos, aumentar la precisión y mejorar la calidad del producto final.

La motivación de este trabajo radica en la necesidad de desarrollar un sistema de visión artificial capaz de detectar y clasificar objetos en movimiento en un entorno industrial, específicamente en una cinta transportadora.

1.2 Objetivos

El objetivo principal de este trabajo es desarrollar un sistema de visión artificial capaz de detectar y clasificar objetos en movimiento en una cinta transportadora utilizando redes neuronales convolucionales y aceleradores hardware de bajo consumo. Para lograr este objetivo, se plantean los siguientes objetivos específicos:

- Realizar un estudio del estado del arte en redes neuronales convolucionales, aceleradores hardware de bajo consumo y técnicas avanzadas de optimización para visión artificial.
- Desarrollar un conjunto de datos para el entrenamiento y evaluación del sistema, mediante la captura y etiquetado de imágenes de objetos en movimiento.
- Diseñar, entrenar y validar un modelo de red neuronal convolucional optimizado para la detección y clasificación en tiempo real de defectos en objetos en movimiento.
- Implementar un sistema completo de visión artificial que integre el modelo entrenado con los aceleradores hardware NVIDIA, enfocado en maximizar la eficiencia y minimizar la latencia.
- Analizar los cuellos de botella del sistema, y aplicar técnicas específicas de optimización para mejorar el rendimiento y la eficiencia energética.

- Cuantificar de manera exhaustiva el rendimiento del sistema mediante métricas precisas de exactitud (mAP, precisión, recall), latencia (FPS) y consumo energético (W, J/inferencia).
- Realizar un análisis comparativo sistemático entre diferentes configuraciones de hardware, software y parámetros de optimización para identificar la combinación que ofrezca el mejor equilibrio entre precisión, velocidad y eficiencia energética.

1.3 Estructura de la memoria

????? ?????????????? ?????????????? ?????????????? ?????????????? ??????????????

CAPÍTULO 2

Conceptos Previos

En este capítulo se explicará los conceptos previos que constituyen la base teórica y técnica de este trabajo. Primero, se examinarán los fundamentos en Inteligencia Artificial centrándose en las redes neuronales convolucionales, desde sus bases hasta los modelos más recientes en detección de objetos. A continuación, se analizarán los aceleradores hardware de bajo consumo, con especial énfasis en la arquitectura y capacidades de los dispositivos NVIDIA Jetson. Posteriormente, se estudiarán los algoritmos de seguimiento de objetos en tiempo real, fundamentales para aplicaciones con elementos en movimiento. Finalmente, se explorará la técnica de Slicing Aided Hyper Inference (SAHI), una metodología avanzada para mejorar la detección de objetos pequeños o densamente agrupados. Este marco teórico permitirá contextualizar adecuadamente la solución propuesta para la detección de defectos en objetos en movimiento.

2.1 Fundamentos y avances en redes neuronales para visión artificial

En esta sección se realizará un estudio de las redes neuronales profundas hasta las redes neuronales convolucionales, desde sus fundamentos hasta los modelos más recientes en detección de objetos. Se explicarán los conceptos básicos de las redes neuronales y la evolución de las arquitecturas.

2.1.1. Fundamentos de la inteligencia artificial

La *Inteligencia Artificial* es un campo de estudio que busca desarrollar sistemas capaces de realizar tareas que normalmente requieren inteligencia humana, como el reconocimiento de voz, la toma de decisiones y la comprensión del lenguaje natural. Dentro de este campo, existen diversas subdisciplinas, entre las cuales destacan el *Machine Learning* y el *Deep Learning*.

El *Machine Learning* o aprendizaje automático es una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender de los datos y realizar predicciones o tomar decisiones sin ser programadas explícitamente. Este enfoque se basa en la idea de que las máquinas pueden identificar patrones y relaciones en grandes conjuntos de datos, lo que les permite generalizar y adaptarse a nuevas situaciones.

El *Deep Learning* o aprendizaje profundo es una rama del aprendizaje automático que utiliza redes neuronales artificiales con múltiples capas para modelar y resolver problemas complejos. Este enfoque permite aprender representaciones jerárquicas de los datos,

donde cada capa extrae características cada vez más abstractas. Una de las arquitecturas fundamentales es el *Multilayer Perceptron* (MLP) o perceptrón multicapa, que consiste en una red de neuronas artificiales organizadas en al menos tres capas: una de entrada, una o más capas ocultas y una capa de salida, como se muestra en la Figura 2.1 [6]. En un MLP, cada neurona recibe un conjunto de entradas ponderadas por pesos, aplica una función de activación no lineal a la suma de estas entradas ponderadas, y produce una salida que se transmite a la siguiente capa. La capacidad de aprendizaje de estas redes se basa en el algoritmo de retropropagación (backpropagation), que ajusta iterativamente los pesos para minimizar el error entre las predicciones de la red y los valores reales. Este proceso de aprendizaje sigue el paradigma de aprendizaje supervisado, donde el MLP se entrena utilizando datos previamente etiquetados que definen el *ground truth*, y que típicamente se dividen en conjuntos de entrenamiento, validación y prueba para ajustar y evaluar su rendimiento. Esta estructura permite al Deep Learning abordar tareas complejas en visión por computador, procesamiento del lenguaje natural y otros dominios con un alto grado de precisión.

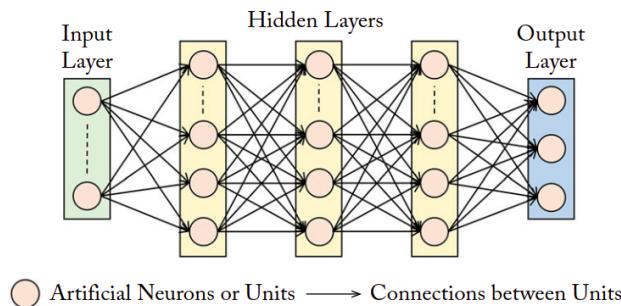


Figura 2.1: Estructura de un perceptrón multicapa (MLP).

2.1.2. Tareas fundamentales en visión por computador

En el ámbito del procesamiento de imágenes mediante técnicas de deep learning, existen diversas tareas con diferentes niveles de complejidad:

1. **Clasificación de imágenes:** Es la tarea más básica, donde la red neuronal asigna una etiqueta a toda la imagen. Por ejemplo, determinar si una imagen contiene un perro, gato o coche. El modelo genera un vector de probabilidades para cada clase posible.
2. **Clasificación con localización:** Además de clasificar el objeto principal, la red también proporciona un cuadro delimitador (bounding box) que indica dónde se encuentra ese objeto en la imagen. Es útil cuando existe un único objeto de interés.
3. **Detección de objetos:** Extiende la tarea anterior para identificar y localizar múltiples objetos en una imagen. Los algoritmos de detección se dividen principalmente en:
 - *Detectores de dos etapas:* Como R-CNN, Fast R-CNN y Faster R-CNN, primero generan propuestas de regiones que podrían contener objetos, y luego clasifican estas regiones. Son más precisos pero computacionalmente más costosos.
 - *Detectores de una etapa:* Como YOLO (You Only Look Once) y SSD (Single Shot MultiBox Detector), que predicen las cajas delimitadoras y las clases directamente en una sola pasada. Son más rápidos aunque tradicionalmente menos precisos.

Ambos enfoques proporcionan para cada objeto detectado su clasificación y cuadro delimitador.

4. **Segmentación:** Es la tarea más compleja, donde la red no solo identifica y localiza objetos, sino que también asigna una etiqueta a cada píxel de la imagen. Esto permite distinguir entre diferentes objetos y sus contornos, facilitando una comprensión más detallada de la escena.

La Figura 2.2 ilustra estas tareas fundamentales en visión por computador. Para este trabajo, nos centraremos en la tarea de detección de objetos, que es esencial para identificar y clasificar varios objetos en movimiento en un vídeo o imagen.

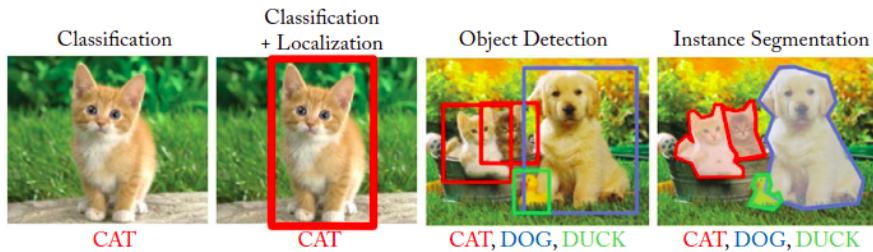


Figura 2.2: Tareas fundamentales en visión por computador.

2.1.3. Arquitectura y funcionamiento de las CNN

Las *Convolutional Neural Networks* (CNN) o redes neuronales convolucionales son un tipo específico de red neuronal profunda. Estas redes están diseñadas para procesar imágenes y extraer características relevantes de manera eficiente, lo que las hace especialmente adecuadas para tareas de visión por computador.

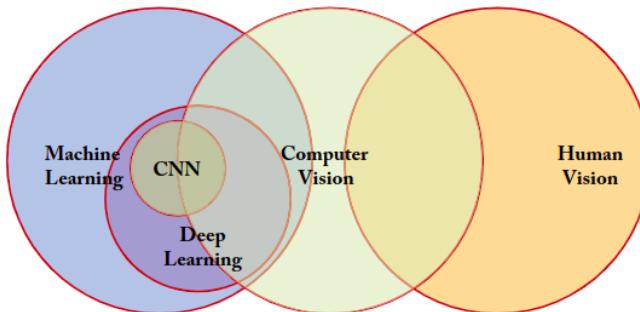


Figura 2.3: Relación entre Machine Learning, Deep Learning, CNN, Computer Vision y Human Vision.

La Figura 2.3 ilustra la relación entre estos conceptos [6]. Las CNN son una subcategoría del Deep Learning, que a su vez es una subcategoría del Machine Learning. Además, las CNN están estrechamente relacionadas con la visión por computador, que busca emular la capacidad de los humanos para interpretar imágenes y vídeos.

Las CNN se inspiran en la forma en que los humanos percibimos el mundo visual. Al igual que nuestro sistema visual, que procesa la información de manera jerárquica, las CNN utilizan capas convolucionales para extraer características de bajo nivel (como bordes y texturas) y capas más profundas para identificar patrones y objetos más complejos. Esta jerarquía de características permite a las CNN aprender representaciones ricas y abstractas de los datos visuales.

Estas redes se componen de varias capas, cada una de las cuales realiza operaciones específicas en los datos de entrada. Las capas más comunes en una CNN son las capas convolucionales, las capas de pooling y las capas completamente conectadas, que se explicarán con más detalle a continuación.

Las capas convolucionales utilizan la operación de convolución para extraer características. La operación de convolución es fundamental en las CNN. Esta operación consiste en aplicar un filtro (o kernel) a una imagen para extraer características locales. El filtro se desliza sobre la imagen, multiplicando sus valores por los valores de la imagen en cada posición y sumando los resultados. Este proceso genera un mapa de activación que resalta las características relevantes de la imagen.

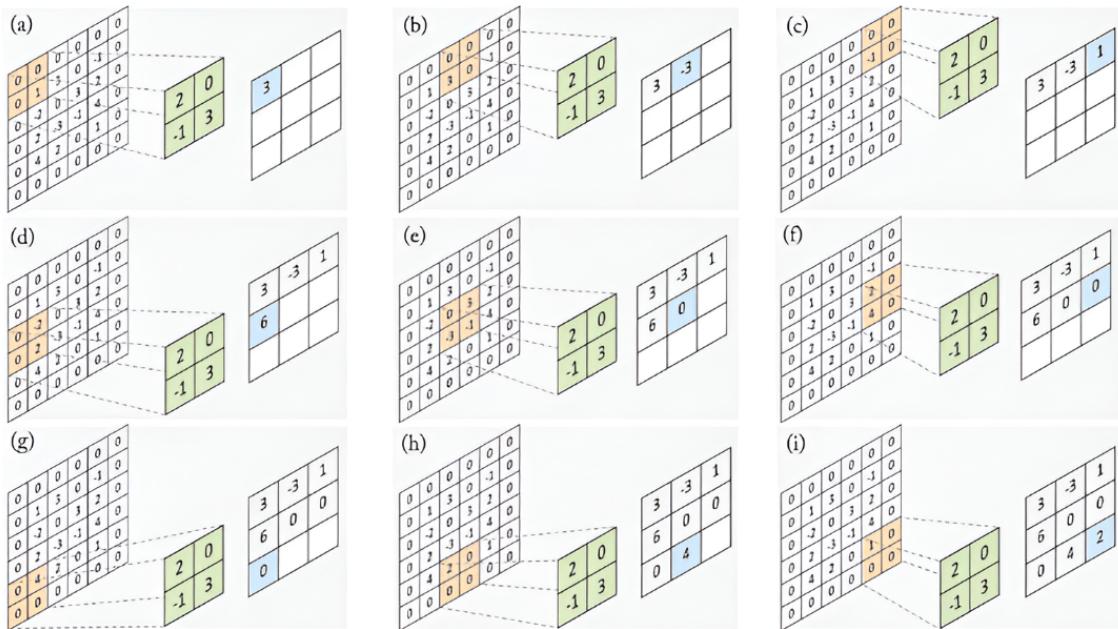
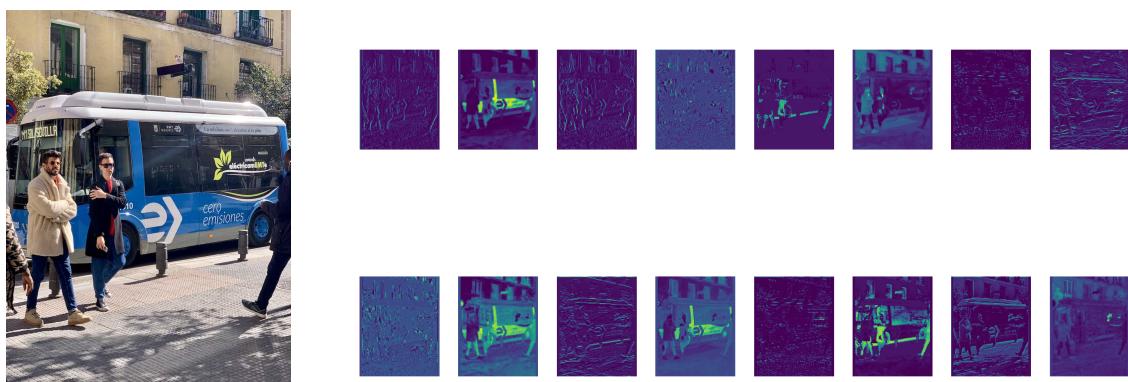


Figura 2.4: Operación de convolución sobre los píxeles de una imagen.

La Figura 2.4[6] ilustra la operación de una capa de convolución. En este ejemplo, se aplica un filtro de 2×2 (mostrado en verde) a un mapa de características de entrada de 6×6 (incluyendo un relleno de ceros de 1) con un paso (stride) de 2. El filtro se desliza sobre la entrada, y en cada paso, se realiza una multiplicación elemento a elemento entre el filtro y la región correspondiente de la entrada. La suma de estos productos genera un valor en el mapa de características de salida (mostrado en azul).



(a) Imagen de un autobús.

(b) Resultado de la operación de convolución.

Figura 2.5: Proceso de convolución aplicado a una imagen de un autobús.

La Figura 2.5 ilustra el proceso de la primera convolución del modelo YOLO11n [4]. En la parte izquierda se muestra la imagen original de un autobús, mientras que en la parte derecha se presenta el resultado de aplicar la operación de convolución. En este caso, los 16 filtros de la primera capa convolucional han detectado diferentes características de la imagen, como bordes y texturas. Este proceso se repite en múltiples capas, lo que permite a la red aprender representaciones cada vez más complejas de la imagen.

Además de las capas convolucionales, las CNN incorporan capas de *pooling* (o submuestreo). Estas capas desempeñan un papel crucial en la reducción progresiva de la dimensión espacial (ancho y alto) de los mapas de características, lo que conlleva varios beneficios importantes. En primer lugar, disminuyen la cantidad de parámetros y la carga computacional de la red, haciendo el modelo más eficiente. En segundo lugar, ayudan a controlar el sobreajuste (*overfitting*) al reducir la complejidad del modelo. El sobreajuste ocurre cuando un modelo aprende los datos de entrenamiento demasiado bien, incluyendo el ruido y los detalles específicos, lo que resulta en un bajo rendimiento con datos nuevos o no vistos. Finalmente, proporcionan un cierto grado de invarianza a pequeñas traslaciones y distorsiones en la imagen de entrada, ya que resumen la información de una región local.

El funcionamiento del *pooling* implica deslizar una ventana (generalmente de tamaño 2×2 o 3×3) sobre el mapa de características de entrada con un cierto paso (*stride*, a menudo igual al tamaño de la ventana para evitar solapamientos). En cada posición de la ventana, se aplica una operación de agregación para obtener un único valor de salida. Las operaciones de pooling más comunes son:

- **Max-Pooling:** Selecciona el valor máximo dentro de la ventana. Es eficaz para capturar las características más prominentes. La Figura 2.6 ilustra esta operación.
- **Average-Pooling:** Calcula el valor promedio de los elementos dentro de la ventana. Tiende a suavizar las características.

El resultado es un mapa de características de menor tamaño pero que conserva la información esencial extraída por las capas convolucionales anteriores.

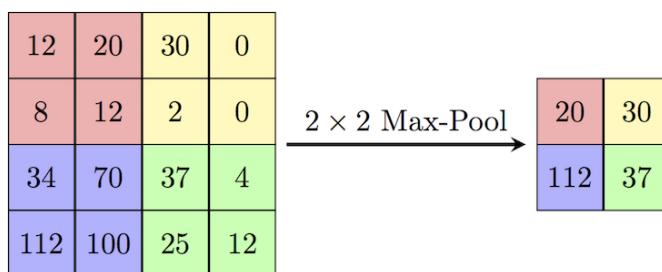


Figura 2.6: Ejemplo de operación de max-pooling con una ventana de 2×2 y un paso de 2. Se selecciona el valor máximo de cada región de color.

Por último, las CNN incluyen capas completamente conectadas (*fully connected*) al final de la red, que permiten realizar la clasificación final de los objetos detectados. Estas capas toman las características extraídas por las capas convolucionales y las combinan para generar una salida que representa la probabilidad de que un objeto pertenezca a una clase específica.

Además de estas capas principales, las CNN modernas incorporan otros tipos de capas que mejoran significativamente su rendimiento. Las capas de activación, especialmente la ReLU (Rectified Linear Unit), son fundamentales para introducir no-linealidad

en el modelo. La función ReLU transforma cada valor negativo en cero mientras mantiene los valores positivos sin cambios, lo que ayuda a mitigar el problema del desvanecimiento del gradiente y acelera el proceso de entrenamiento.

La función ReLU se define como:

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \quad (2.1)$$

Estas capas se aplican típicamente después de cada capa convolucional y completamente conectada.

La **Normalización por Lotes**, comúnmente conocida como *Batch Normalization* (BN), constituye una técnica fundamental en el entrenamiento de redes neuronales profundas, diseñada primordialmente para mitigar el problema del *cambio interno de covariables (internal covariate shift)*. Este fenómeno se refiere a la alteración continua de la distribución estadística de las activaciones de las capas intermedias a medida que los parámetros de las capas precedentes se actualizan durante el proceso de entrenamiento. Dicha inestabilidad en las distribuciones de entrada obliga a las capas subsiguientes a readaptarse constantemente, lo que puede ralentizar la convergencia del modelo y aumentar la dificultad del entrenamiento.

El entrenamiento de redes neuronales profundas se realiza habitualmente procesando los datos en subconjuntos denominados **mini-lotes** (*mini-batches*), en lugar de procesar el conjunto de datos completo o ejemplos individuales. Esta estrategia equilibra la eficiencia computacional con la introducción de cierta estocasticidad en la estimación del gradiente. Batch Normalization opera precisamente a nivel de estos mini-lotes. Para un mini-lote dado $\mathcal{B} = \{x_1, \dots, x_m\}$, BN calcula inicialmente la media empírica ($\mu_{\mathcal{B}}$) y la varianza empírica ($\sigma_{\mathcal{B}}^2$) de las activaciones dentro de dicho mini-lote:

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i \quad (2.2)$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad (2.3)$$

Posteriormente, cada activación x_i del mini-lote es estandarizada utilizando estas estadísticas:

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad (2.4)$$

donde ϵ es una pequeña constante positiva (p. ej., 10^{-5}) añadida para garantizar la estabilidad numérica, previniendo la división por cero en caso de varianza nula. Esta normalización asegura que las activaciones transformadas \hat{x}_i posean, dentro del mini-lote, una media aproximadamente nula y una varianza unitaria.

No obstante, imponer rígidamente esta distribución normalizada podría restringir la capacidad expresiva de la red. Para contrarrestar esta limitación, BN introduce dos parámetros afines, γ (escala) y β (desplazamiento), que son aprendidos durante el entrenamiento junto con los pesos de la red. Estos parámetros permiten a la red realizar una transformación afín sobre las activaciones normalizadas:

$$y_i = \gamma \hat{x}_i + \beta \quad (2.5)$$

Mediante γ y β , la red puede aprender la escala y el desplazamiento óptimos para las activaciones de cada capa, recuperando potencialmente la representación original si fuera

necesario y preservando así la capacidad representacional del modelo. La salida y_i es entonces la que se propaga a la siguiente operación, usualmente una función de activación no lineal.

La incorporación de Batch Normalization conlleva beneficios sustanciales para el proceso de entrenamiento. Al estabilizar la distribución de las entradas a las capas, reduce efectivamente el cambio interno de covariables, lo que facilita un entrenamiento más rápido y estable. Esta estabilidad permite, a su vez, el uso de tasas de aprendizaje más elevadas sin riesgo de divergencia, acelerando la convergencia del modelo. Adicionalmente, BN ejerce un efecto regularizador. El cálculo de la media y la varianza sobre mini-lotes introduce una forma de ruido estocástico en el entrenamiento, ya que estas estadísticas varían entre mini-lotes. Este ruido inherente ayuda a prevenir el sobreajuste (*overfitting*). Convencionalmente, las capas de Batch Normalization se insertan después de las capas convolucionales o totalmente conectadas, pero antes de la aplicación de la función de activación no lineal (como ReLU).

El mecanismo central de Dropout consiste en desactivar (poner a cero) aleatoriamente un subconjunto de neuronas (unidades) en una capa durante cada iteración de entrenamiento (paso hacia adelante y hacia atrás). Cada neurona tiene una probabilidad p (el *dropout rate*, un hiperparámetro típicamente entre 0.2 y 0.5) de ser temporalmente eliminada de la red, junto con todas sus conexiones entrantes y salientes. Esta desactivación se realiza de forma independiente para cada neurona y para cada ejemplo de entrenamiento o mini-lote.

Al desactivar neuronas aleatoriamente, Dropout impide que las neuronas co-adapten excesivamente sus características, es decir, que dependan demasiado de la presencia de otras neuronas específicas para funcionar correctamente. En cambio, obliga a cada neurona a aprender características que sean útiles por sí mismas o en combinación con diferentes subconjuntos aleatorios de otras neuronas. Esto fomenta el aprendizaje de representaciones más robustas y distribuidas, mejorando la capacidad de generalización del modelo.

Dropout se aplica solo durante la fase de entrenamiento. Durante la inferencia (evaluación o predicción), todas las neuronas están activas, pero sus salidas se escalan multiplicando por $1 - p$ para compensar el hecho de que, durante el entrenamiento, solo una fracción de ellas estaba activa. Esto asegura que la salida esperada del modelo durante la inferencia sea comparable a la salida promedio durante el entrenamiento.

En la figura 2.7 se muestra la arquitectura de LeNet-5 [9], una de las primeras CNN desarrolladas. Esta red consta de varias capas convolucionales y de pooling, seguidas de capas completamente conectadas. LeNet-5 fue diseñada para la clasificación de dígitos manuscritos y sentó las bases para el desarrollo de arquitecturas más complejas y eficientes en la actualidad.

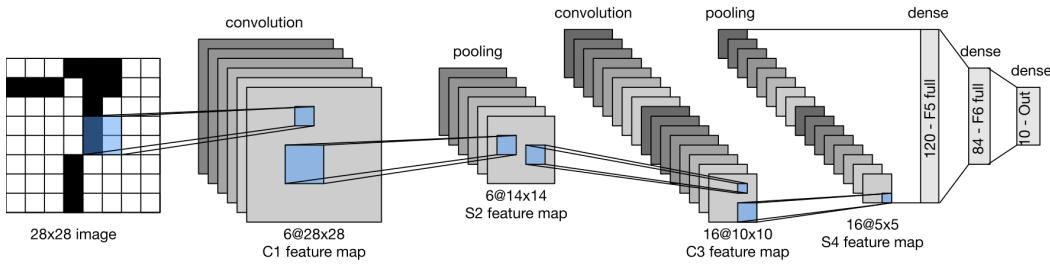


Figura 2.7: Arquitectura de LeNet-5.

2.1.4. Detectores de dos etapas

Los detectores de dos etapas, funcionan mediante un proceso secuencial: primero generan propuestas de regiones (region proposals) que podrían contener objetos y posteriormente clasifican estas regiones. Este enfoque favorece la precisión, aunque generalmente a costa de un mayor tiempo de procesamiento.

La primera arquitectura exitosa de detección de objetos basada en deep learning fue R-CNN (Regions with CNN features) [3]. Este modelo introdujo un enfoque de dos etapas que revolucionó el campo. En su primera fase, R-CNN utiliza un algoritmo de búsqueda selectiva (Selective Search) para generar aproximadamente 2,000 propuestas de regiones que podrían contener objetos. Este algoritmo de búsqueda selectiva divide la imagen en nodos y aristas, e iterativamente agrupa estas regiones en función del color, textura, tamaño y forma hasta que se obtienen las propuestas finales. En la figura 2.8[8] se muestra un ejemplo del resultado del algoritmo de búsqueda selectiva.



(a) Imagen original.



(b) Resultado de búsqueda selectiva.

Figura 2.8: Proceso de búsqueda selectiva aplicado a una imagen.

En la segunda fase, cada región propuesta es redimensionada y procesada individualmente por una CNN para extraer características de alto nivel. Estas características alimentan posteriormente a un clasificador SVM (Support Vector Machine) para determinar la categoría del objeto y a un regresor lineal para mejorar la localización del cuadro delimitador. Como se ilustra en la Figura 2.9, este enfoque fue innovador pero computacionalmente costoso, ya que requiere procesar cada propuesta de región de manera independiente.

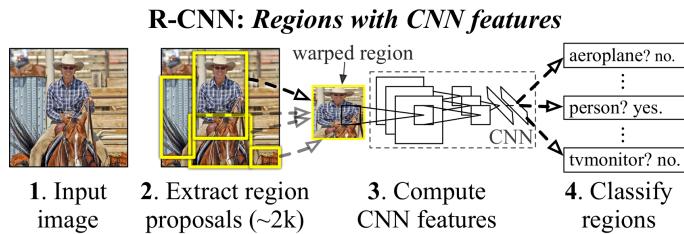


Figura 2.9: Arquitectura de R-CNN.

2.1.5. Detectores de una etapa

En contraste con los detectores de dos etapas, los detectores de una etapa (one-stage detectors) adoptan un enfoque más directo y eficiente. Estos detectores realizan la localización y clasificación de objetos simultáneamente en una sola pasada a través de la red, sin necesidad de un paso intermedio de generación de propuestas.

La arquitectura de los detectores de una etapa procesa la imagen completa una única vez, típicamente mediante una red troncal o *backbone* (generalmente una CNN) para la extracción de características. Estas características son posteriormente procesadas por componentes intermedios (*neck*) y alimentadas a una cabeza de detección (*detection head*) que predice simultáneamente las coordenadas de los cuadros delimitadores y las probabilidades de clase. Esta arquitectura de una etapa prioriza la velocidad de inferencia, resultando idónea para aplicaciones en tiempo real donde la latencia es un factor crítico, aunque pueda suponer una ligera concesión en la precisión máxima. Modelos representativos de este enfoque incluyen SSD (Single Shot MultiBox Detector)[11] y YOLO (You Only Look Once)[13]. Estos han demostrado un equilibrio eficaz entre rapidez y exactitud, permitiendo la detección en tiempo real incluso en dispositivos con recursos computacionales limitados.

YOLO (You Only Look Once) se destaca como una de las arquitecturas de detección de objetos más populares y efectivas. Concebida específicamente para la detección en tiempo real, YOLO introdujo un enfoque unificado que procesa la imagen completa en una sola pasada, realizando la localización y clasificación de forma simultánea. Esta metodología ha sido fundamental para su adopción en aplicaciones que requieren alta velocidad de procesamiento.

YOLO en su primera versión procesa la imagen completa de una vez. Divide la imagen en una cuadrícula de $S \times S$ celdas. Cada celda es responsable de detectar los objetos cuyo centro se encuentre dentro de ella. Para cada celda, YOLO predice B cuadros delimitadores (bounding boxes) y puntuaciones de confianza para esos cuadros. La puntuación de confianza indica la probabilidad de que haya un objeto en el cuadro y la precisión de la predicción del cuadro. Al mismo tiempo, predice las probabilidades de clase para cada objeto detectado en la celda.

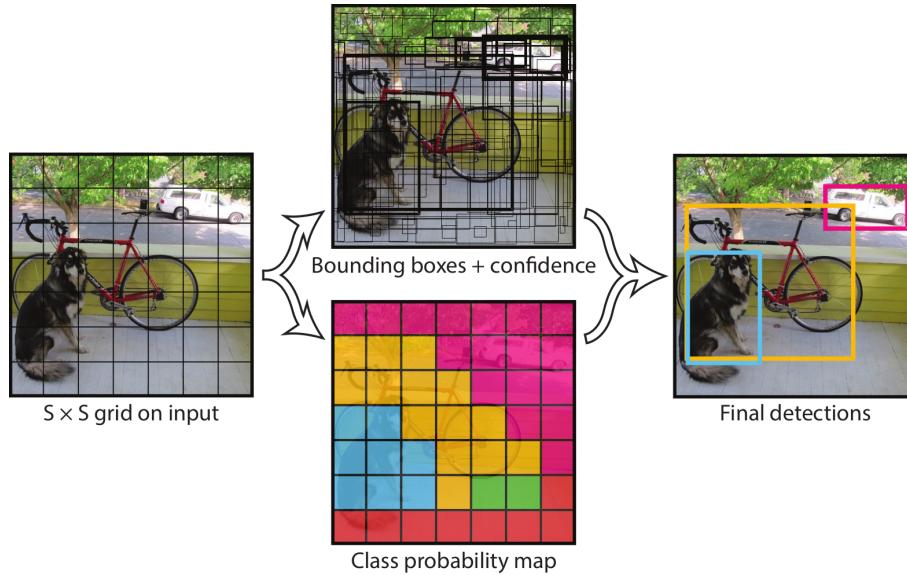


Figura 2.10: Ejemplo de detección de objetos utilizando YOLO.

Como se ilustra en la Figura 2.10[13], YOLO modela la detección como un problema de regresión. Las predicciones del modelo se codifican como una matriz de 3 dimensiones de $S \times S \times (B * 5 + C)$, donde el factor 5 corresponde a las coordenadas x, y , ancho, alto y la puntuación de confianza para cada cuadro delimitador, mientras que C representa el número de clases posibles. Por ejemplo, si se utilizan $B = 2$ cuadros delimitadores (por cada celda, es decir, cada celda realiza 2 predicciones) y $C = 20$ clases, el tensor de salida tendrá dimensiones $S \times S \times (2 * 5 + 20)$.

Una vez generadas todas las predicciones, YOLO implementa un post-procesamiento mediante supresión de no máximos (NMS) para eliminar detecciones redundantes y conservar únicamente las más precisas. Este enfoque unificado permite que YOLO procese imágenes a velocidades significativamente mayores que los detectores de dos etapas, mientras mantiene una precisión competitiva, lo que lo hace ideal para aplicaciones en tiempo real como las que se abordan en este trabajo.

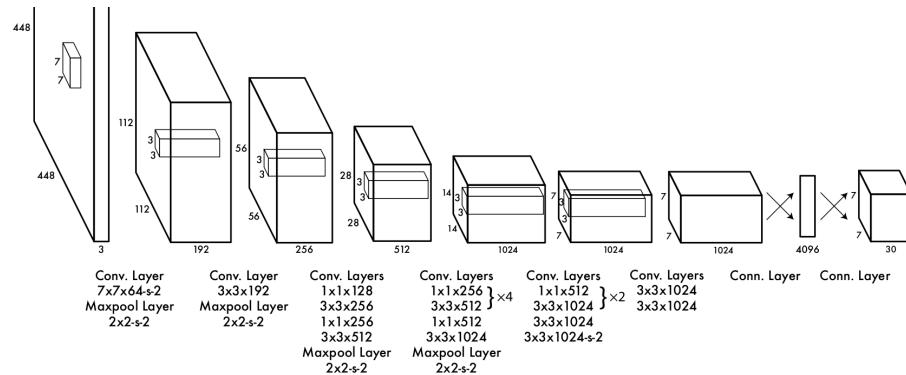


Figura 2.11: Arquitectura de YOLO

En la Figura 2.11 se presenta la arquitectura del modelo primigenio de YOLO [13]. Esta arquitectura se basa en una red neuronal convolucional que extrae características de la imagen de entrada y las procesa a través de varias capas para generar las predicciones finales. A lo largo de los años, se han desarrollado múltiples versiones y mejoras de YOLO, cada una optimizando aspectos como la precisión, la velocidad y la capacidad de detección de objetos pequeños o densamente agrupados.

En este proyecto, se ha seleccionado YOLO11 [4], que representa una mejora y optimización sobre las arquitecturas YOLO precedentes. Dicho modelo forma parte de la librería Ultralytics [5], la cual ofrece una implementación eficaz y sencilla de diversas variantes de YOLO. Las aportaciones de Ultralytics a la comunidad de visión artificial son notables, ya que sus herramientas y recursos agilizan el ciclo de vida (entrenamiento, evaluación, implementación) de los modelos YOLO en múltiples contextos. YOLO11 se ofrece en diversas configuraciones (11n, 11s, 11m, 11l y 11x).

Modelo	Tamaño (px)	Latencia CPU ONNX (ms)	Latencia T4 TRT10 (ms)	mAPval (50-95)	Parámetros / FLOPs (M / B)
YOLO11n	640	56.1 ± 0.8	1.5 ± 0.0	39.5	2.6 / 6.5
YOLO11s	640	90.0 ± 1.2	2.5 ± 0.0	47.0	9.4 / 21.5
YOLO11m	640	183.2 ± 2.0	4.7 ± 0.1	51.5	20.1 / 68.0
YOLO11l	640	238.6 ± 1.4	6.2 ± 0.1	53.4	25.3 / 86.9
YOLO11x	640	462.8 ± 6.7	11.3 ± 0.2	54.7	56.9 / 194.9

Tabla 2.1: Análisis comparativo de las variantes de YOLO11 considerando precisión, velocidad y complejidad computacional.

La Tabla 2.1 presenta un análisis comparativo de las distintas variantes de YOLO11 evaluadas sobre el dataset COCO [10]. Los datos revelan una clara correlación: los modelos más grandes ofrecen mayor precisión (mAP) a costa de mayor complejidad computacional y menor velocidad de procesamiento. Esta escalabilidad permite adaptar la implementación según los requisitos específicos de cada aplicación, balanceando efectivamente el rendimiento con las restricciones computacionales del sistema objetivo.

2.1.6. Métricas de evaluación

Para evaluar el rendimiento de los modelos de detección de objetos, se utilizan métricas específicas que permiten cuantificar la precisión y efectividad del sistema. Entre las métricas más comunes se encuentran:

- **Precisión (Precision):** Mide la proporción de verdaderos positivos (TP) respecto al total de predicciones positivas realizadas por el modelo. Se calcula como:

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

donde FP representa los falsos positivos. Una alta precisión indica que el modelo realiza pocas predicciones incorrectas.

- **Recall:** Mide la proporción de verdaderos positivos respecto al total de objetos relevantes en la imagen. Se calcula como:

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

donde FN representa los falsos negativos. Un alto recall indica que el modelo es capaz de detectar la mayoría de los objetos relevantes, aunque pueda incluir algunas predicciones incorrectas.

- **IoU (Intersection over Union):** Mide la superposición entre el cuadro delimitador predicho y el cuadro delimitador real. Se calcula como:

$$\text{IoU} = \frac{\text{Área de intersección}}{\text{Área de unión}}$$

Una IoU alta indica que el modelo ha localizado correctamente el objeto. Generalmente, se considera que una IoU superior a 0.5 indica una detección correcta.

- **mAP (mean Average Precision):** Es una métrica que combina precisión y recall, promediando la precisión a diferentes niveles de recall. Se utiliza para evaluar el rendimiento general del modelo en múltiples clases.

El cálculo del mAP se realiza en varios pasos: primero, para cada clase se calcula la curva PR (Precision-Recall) y se obtiene el Average Precision (AP), que representa el área bajo esta curva. Luego, el mAP se obtiene como la media de todos los AP de las diferentes clases. En detección de objetos, el mAP suele incorporar diferentes umbrales de IoU (Intersection over Union), expresándose como mAP@IoU=0.5 o mAP@IoU=0.5:0.95. Esta métrica es especialmente útil cuando las clases están desequilibradas, ya que da igual importancia a clases minoritarias y mayoritarias. Un valor de mAP cercano a 1 indica un modelo con alta precisión y recall en todas las clases.

- **Latencia:** Mide el tiempo que tarda el modelo en procesar una imagen y generar una predicción se mide en milisegundos (ms) y un valor bajo indica que el modelo es capaz de realizar inferencias rápidamente.
- **FPS (Frames Per Second):** Mide la velocidad de procesamiento del modelo, indicando cuántas imágenes puede procesar por segundo un alto valor de FPS indica que el modelo es capaz de realizar inferencias rápidamente.
- **F1 Score:** Es la media armónica entre precisión y recall, proporcionando un balance entre ambos. Se utiliza para evaluar el rendimiento del modelo en situaciones donde hay un desbalance entre clases. Se calcula como:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Un F1 Score alto indica que el modelo tiene un buen equilibrio entre precisión y recall, lo que es especialmente importante en aplicaciones donde tanto la detección correcta como la minimización de falsos positivos son críticas.

2.2 Aceleradores de procesamiento gráfico

Un aspecto fundamental para el despliegue eficiente de modelos de inteligencia artificial es el hardware utilizado para su ejecución. A continuación, se analizan los principales aspectos de los aceleradores de procesamiento gráfico y su importancia en aplicaciones de visión artificial.

2.2.1. Limitaciones del hardware tradicional

La Dennard Scaling, formulada por Robert Dennard en 1974, establecía que a medida que los transistores se reducían de tamaño, su consumo de energía por unidad de área se mantenía constante. Esto significaba que, al reducir el tamaño de los transistores a la mitad, su área se reducía a un cuarto, pero su consumo de energía por unidad de área permanecía igual. Como resultado, el consumo total de energía se reducía a la mitad, permitiendo aumentar la frecuencia de reloj y el número de transistores sin incrementar significativamente el consumo total de energía. Este principio fue fundamental para el avance exponencial en el rendimiento de los procesadores durante décadas. Sin embargo, a partir de 2005, la Dennard Scaling dejó de cumplirse debido a varios factores físicos fundamentales. A escalas nanométricas, los efectos cuánticos y las fugas de corriente se

volvieron significativos, impidiendo que el consumo de energía por unidad de área se mantuviera constante.

Por otro lado, la ley de Moore, formulada por Gordon Moore en 1965, predecía que el número de transistores en un chip se duplicaría aproximadamente cada año, predicción que posteriormente se ajustó a cada dos años. Durante décadas, esta ley se cumplió con notable precisión, permitiendo un crecimiento exponencial en la capacidad de procesamiento. Sin embargo, a medida que los transistores se acercan a escalas atómicas (actualmente en torno a los 3-5 nanómetros), los límites físicos y los desafíos de fabricación han ralentizado significativamente este ritmo de avance. Los costes de investigación y desarrollo para mantener esta tendencia se han disparado, y los beneficios en términos de rendimiento por transistor se han reducido.

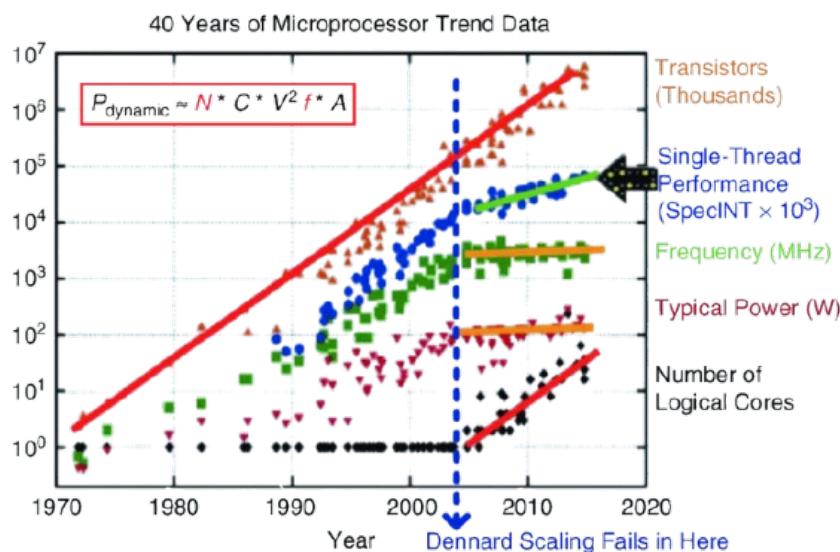


Figura 2.12: Evolución histórica de las características de los microprocesadores (1970-2020).

En la Figura 2.12[14] se ilustra claramente el impacto combinado del fin de la Dennard Scaling y el estancamiento de la ley de Moore. La gráfica muestra cinco métricas fundamentales en escala logarítmica: el número de transistores (triángulos naranja) que sigue la ley de Moore, el rendimiento de un solo hilo (círculos azules), la frecuencia de reloj (cuadrados verdes), el consumo de potencia (triángulos invertidos morados) y el número de núcleos lógicos (rombos negros). La ecuación de potencia dinámica ($P_{dynamic} \approx N * C * V^2 * f * A$) explica la relación entre el número de transistores (N), la capacitancia (C), el voltaje (V), la frecuencia (f) y el factor de actividad (A).

El punto de inflexión en 2005, marcado como *Dennard Scaling Fails in Here*, marca el momento en que la industria tuvo que cambiar radicalmente su estrategia. La frecuencia de reloj se estancó en torno a los 3-4 GHz, el rendimiento por núcleo comenzó a crecer más lentamente, y como respuesta, se adoptaron dos estrategias principales: el aumento del número de núcleos y la estabilización del consumo de potencia alrededor de los 100W. Este fenómeno ha llevado a la industria a buscar alternativas como las arquitecturas de dominio específico para continuar mejorando el rendimiento de los sistemas computacionales.

2.2.2. Arquitectura y funcionamiento de las GPUs

Las GPUs (Unidades de Procesamiento Gráfico) representan un claro ejemplo de arquitecturas de dominio específico, diseñadas para maximizar el rendimiento en tareas altamente paralelizables. El diseño de las GPUs sigue el paradigma *many-core*, lo que implica la integración de miles de núcleos de procesamiento relativamente simples en un solo chip. Esta arquitectura permite que las GPUs ejecuten múltiples hilos de manera concurrente, lo que resulta especialmente beneficioso para cargas de trabajo que pueden dividirse en tareas independientes.

El modelo de programación de las GPUs está basado en la ejecución masiva de hilos, organizados en bloques y rejillas (*blocks* y *grids*), según la terminología de CUDA, el modelo de programación desarrollado por NVIDIA. Cada hilo ejecuta la misma función, conocida como *kernel*, pero opera sobre diferentes fragmentos de datos. Este enfoque, conocido como SIMT (Single Instruction, Multiple Threads), permite aprovechar al máximo el paralelismo inherente a muchas aplicaciones de inteligencia artificial, como el entrenamiento y la inferencia de redes neuronales.

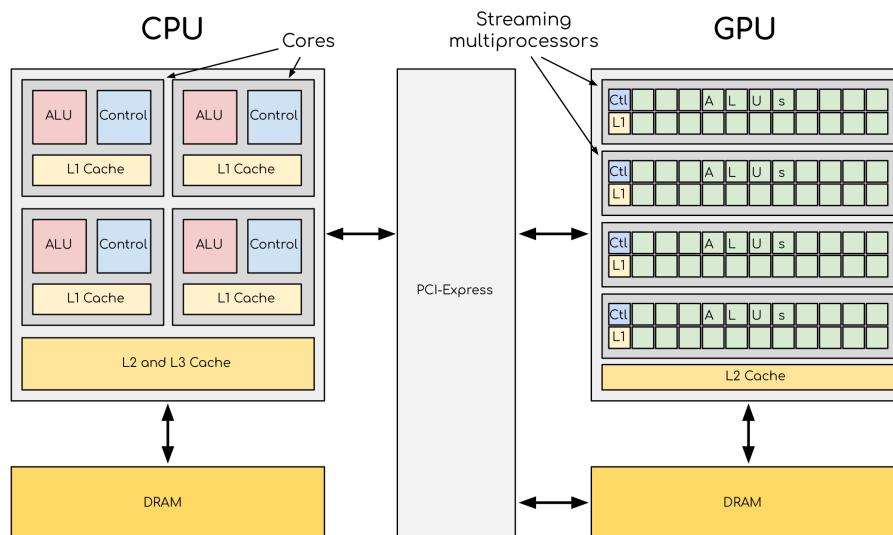


Figura 2.13: Comparativa de arquitecturas CPU y GPU.

La figura 2.13[2] ilustra las diferencias arquitectónicas entre CPU y GPU. Las CPUs constan de un número reducido de núcleos (generalmente entre 4 y 64) optimizados para un alto rendimiento secuencial, donde cada núcleo dispone típicamente de caché y unidad de control privadas. En cambio, las GPUs se basan en una arquitectura *many-core*, integrando miles de núcleos más simples diseñados para el paralelismo masivo, aunque con menor rendimiento individual por núcleo. Estos núcleos de GPU suelen compartir recursos como la caché y las unidades de control, lo que posibilita empaquetar una mayor cantidad de unidades de procesamiento en el mismo chip y, consecuentemente, alcanzar una mayor densidad de cómputo.

NVIDIA ha desempeñado un papel fundamental en el desarrollo de aceleradores para inteligencia artificial, estableciendo un ecosistema robusto que abarca tanto hardware como software. Su enfoque ha permitido optimizar el rendimiento de los modelos de IA, facilitando su implementación en una amplia variedad de aplicaciones.

La programación de GPUs se realiza mediante lenguajes y APIs especializadas como CUDA y OpenCL, que permiten al desarrollador controlar explícitamente la distribución de datos y tareas entre los distintos núcleos. El éxito en este ámbito depende crucialmente

de la capacidad para identificar y explotar el paralelismo inherente a los algoritmos, así como de una gestión eficiente de la memoria, dado que el acceso a la memoria global de la GPU es significativamente más lento que el acceso a la memoria compartida entre hilos de un mismo bloque.

NVIDIA ha sido una figura central en la evolución de la aceleración de IA, realizando contribuciones clave que han modelado el campo. Fue pionera en la computación de propósito general en GPUs (GPGPU) con la introducción de CUDA en 2006, una plataforma que permitió utilizar la masiva capacidad de procesamiento paralelo de las GPUs para tareas computacionales generales, extendiendo su uso más allá de los gráficos tradicionales. Complementando esto, NVIDIA ha desarrollado un ecosistema de software robusto y optimizado, que incluye bibliotecas fundamentales como cuDNN, específica para redes neuronales profundas, y cuBLAS, para operaciones de álgebra lineal básica, ambas esenciales para el desarrollo eficiente en inteligencia artificial. En el frente del hardware, la compañía ha impulsado la innovación con el desarrollo de componentes especializados como los Tensor Cores, diseñados específicamente para acelerar las operaciones matriciales intensivas que son comunes en el entrenamiento e inferencia de modelos de IA. Como resultado de estas continuas innovaciones y la creación de un ecosistema integral, NVIDIA ha logrado establecer estándares de facto para el hardware y software utilizados en la aceleración de IA, consolidándose como la plataforma preferida en una amplia gama de entornos, desde grandes centros de datos hasta sistemas embebidos con recursos limitados.

2.2.3. Serie Jetson: Dispositivos de IA de bajo consumo

La serie Jetson de NVIDIA constituye una familia de módulos computacionales diseñados específicamente para habilitar la inteligencia artificial y el aprendizaje profundo en dispositivos de borde (edge devices). Estos sistemas compactos y de bajo consumo son fundamentales para aplicaciones que requieren procesamiento local de datos con alta capacidad de cómputo.

El enfoque principal de la serie Jetson es la computación en el borde (edge computing), un paradigma que acerca el procesamiento de datos y la inteligencia artificial a la fuente donde se generan. Esto resulta crucial en aplicaciones donde la latencia, el ancho de banda limitado o la privacidad son factores críticos, ya que evita la necesidad de enviar grandes volúmenes de datos a la nube para su análisis. Los dispositivos Jetson están optimizados para operar bajo restricciones significativas de energía, tamaño y coste, características típicas de los entornos embebidos y de borde.

Cada módulo Jetson se basa en una arquitectura System-on-Chip (SoC), que integra múltiples componentes de procesamiento en un único circuito integrado. Esto incluye núcleos de CPU basados en la arquitectura ARM, potentes núcleos de GPU NVIDIA con arquitecturas modernas, y en algunos modelos, aceleradores de hardware dedicados como los Deep Learning Accelerators (DLAs). Esta integración permite una alta eficiencia computacional y energética, reduce la huella física del sistema y simplifica el diseño de la placa portadora, resultando en soluciones más compactas y eficientes.

Un pilar fundamental del diseño de la serie Jetson es la optimización de la eficiencia energética. Estos dispositivos están diseñados para ofrecer un alto rendimiento computacional por cada vatío de energía consumido (TOPS/W), esencial para aplicaciones embebidas con fuentes de alimentación limitadas o restricciones térmicas. La capacidad de configurar diferentes perfiles de energía permite ajustar dinámicamente el equilibrio entre rendimiento y consumo.

Más allá del hardware, el valor de la plataforma Jetson reside en su ecosistema de software integral, proporcionado a través del NVIDIA JetPack SDK. Este paquete incluye el sistema operativo Linux optimizado (L4T), controladores, bibliotecas aceleradas por GPU como CUDA, cuDNN y TensorRT, además de herramientas de desarrollo y documentación. Esta plataforma unificada simplifica el ciclo de desarrollo, desde la creación y entrenamiento de modelos hasta su optimización y despliegue en los dispositivos Jetson, acelerando la innovación.

Modelo	AI Performance	GPU	GPU Max Freq.	CPU	Memoria
Jetson AGX Orin	275 TOPS	2048-core Ampere, 64 Tensor Cores	1.3 GHz	12-core Cortex-A78AE, 3MB L2 + 6MB L3, 2.2 GHz	64GB LPDDR5, 204.8GB/s
Jetson Orin Nano	67 TOPS	1024-core Ampere, 32 Tensor Cores	1020 MHz	6-core Cortex-A78AE, 1.5MB L2 + 4MB L3, 1.7 GHz	8GB LPDDR5, 102 GB/s
Jetson AGX Xavier	32 TOPS	512-core Volta, 64 Tensor Cores	1377 MHz	8-core Carmel v8.2, 8MB L2 + 4MB L3, 2.2 GHz	32GB LPDDR4x, 136.5GB/s

Tabla 2.2: Comparativa técnica entre diferentes modelos NVIDIA Jetson.

La tabla 2.2[12] presenta una comparativa técnica entre diferentes modelos de la serie Jetson. Cada modelo está diseñado para satisfacer diferentes necesidades y requisitos de rendimiento, lo que permite a los desarrolladores seleccionar el módulo más adecuado para su aplicación específica. Para este trabajo, se han utilizado los tres modelos de la tabla y se han comparado sus resultados.

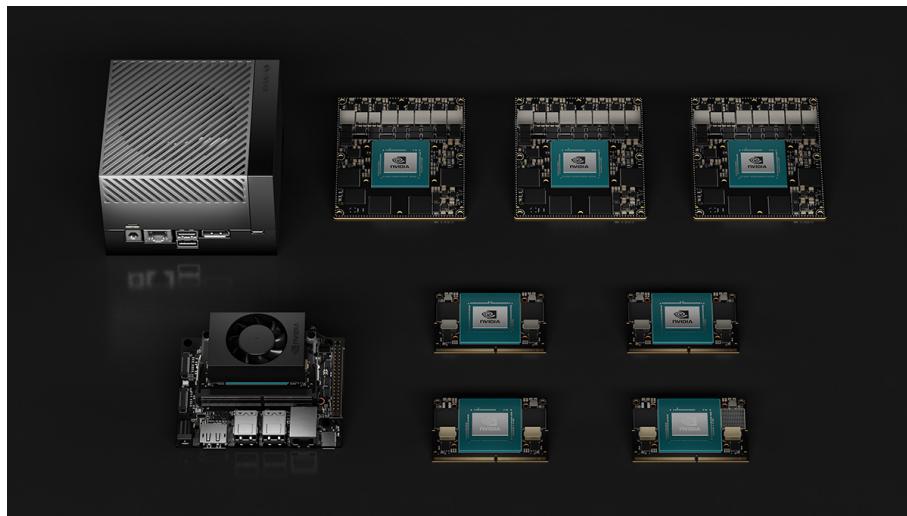


Figura 2.14: Módulos Jetson de NVIDIA.

2.2.4. TensorRT

NVIDIA TensorRT es un ecosistema diseñado específicamente para optimizar modelos de aprendizaje profundo y lograr una inferencia de alto rendimiento en hardware NVIDIA. Funciona como un compilador y motor de ejecución que transforma modelos entrenados en versiones altamente eficientes para despliegue.

TensorRT aplica optimizaciones avanzadas a nivel de grafo, como la fusión de capas computacionales y la selección de kernels optimizados para la plataforma específica. Además, ofrece un soporte robusto para la cuantización, permitiendo que los modelos operen con precisión numérica reducida (como FP16, FP8, FP4 o enteros) con mínima pérdida de exactitud. Esta reducción de precisión, aplicable tanto post-entrenamiento como mediante entrenamiento consciente de la cuantización, disminuye significativamente la latencia y los requisitos de memoria, lo cual es crucial para aplicaciones en tiempo real y sistemas embebidos.

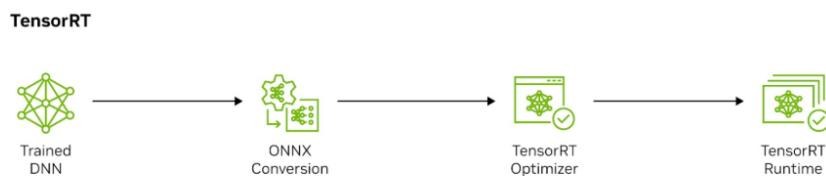


Figura 2.15: Ejemplo de flujo de trabajo de optimización con TensorRT.

El proceso de optimización con TensorRT, como se ilustra en la Figura 2.15, genera un motor de inferencia ajustado para el hardware NVIDIA subyacente, seleccionando las rutas de ejecución más eficientes. TensorRT se integra con los principales frameworks de aprendizaje profundo, como TensorFlow, PyTorch y ONNX, facilitando la importación de modelos para su optimización.

2.3 Seguimiento de objetos en tiempo real

En esta sección se presentará los conceptos básicos del seguimiento de objetos en tiempo real, se explicará el problema del seguimiento de objetos múltiples (MOT) y se presentarán los algoritmos más relevantes en este campo.

2.3.1. Introducción al seguimiento de objetos

El seguimiento de objetos es un proceso que complementa la salida de los modelos de detección. Mientras que un modelo de detección opera sobre cada fotograma de forma independiente, identificando y localizando objetos como si fueran imágenes estáticas, el Seguimiento de Objetos Múltiples (MOT) actúa sobre estas detecciones para establecer una correspondencia temporal. La función esencial del MOT es asignar un identificador único a cada objeto detectado en un fotograma y mantener dicho identificador de manera consistente a lo largo de la secuencia de vídeo. Esto permite reconstruir las trayectorias de los objetos y analizar su comportamiento dinámico en la escena.

Para realizar este seguimiento, el MOT se basa en la información temporal y espacial de las detecciones. Utiliza técnicas de predicción y asociación para determinar la continuidad de los objetos a lo largo del tiempo, teniendo en cuenta factores como la posición, velocidad y apariencia de los objetos. El algoritmo utilizado para el seguimiento puede variar en complejidad, desde enfoques simples que utilizan filtros de Kalman para predecir la posición futura de un objeto, hasta métodos más avanzados que incorporan redes neuronales profundas para aprender características de apariencia y mejorar la robustez del seguimiento.

El filtro de Kalman es un algoritmo de estimación recursivo fundamental en el seguimiento de objetos. Funciona como un estimador óptimo para sistemas dinámicos lineales, permitiendo predecir el estado futuro de un objeto (como su posición y velocidad) a partir de una serie de mediciones ruidosas o incompletas a lo largo del tiempo. Su proceso se basa en dos etapas cíclicas:

- **Predicción:** Utiliza un modelo dinámico del movimiento esperado del objeto para estimar su estado en el siguiente instante de tiempo.
- **Actualización:** Incorpora la nueva medición (detección) obtenida en ese instante para corregir la predicción inicial, ponderando la información del modelo y la medición según su incertidumbre asociada.

Este ciclo permite al filtro refinar continuamente la estimación del estado del objeto, suavizar las trayectorias y manejar eficazmente el ruido inherente a las mediciones del detector. Es una herramienta clave para mantener la identidad de los objetos entre fotogramas, especialmente cuando las detecciones son intermitentes o imprecisas.

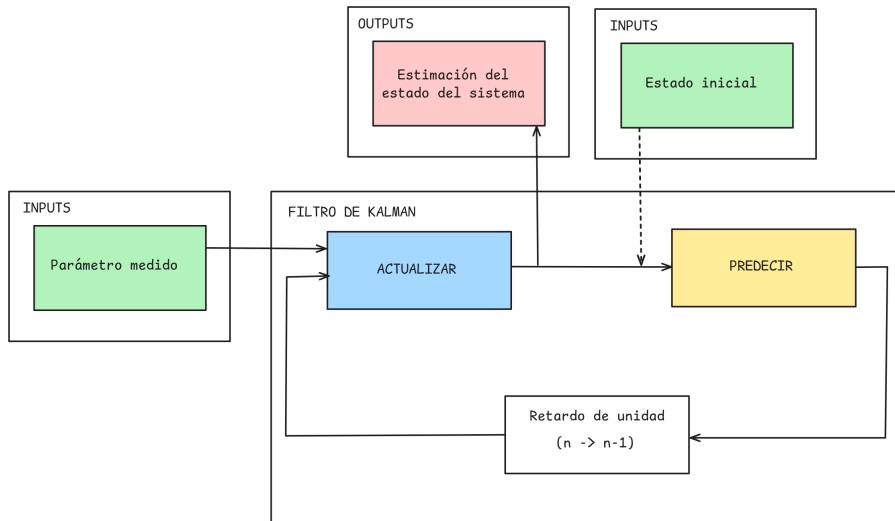


Figura 2.16: Diagrama de flujo del filtro de Kalman.

La Figura 2.16 representa el funcionamiento de un filtro de Kalman, un algoritmo muy utilizado para estimar el estado de un sistema dinámico en presencia de ruido e incertidumbre. A continuación se explica cada bloque:

- **INPUTS: Parámetro medido** Este bloque representa las mediciones que se obtienen del sistema. Estas mediciones contienen errores (ruido), por lo que no se usan directamente, sino que se pasan al filtro para su procesamiento.
- **INPUTS: Estado inicial** Proporciona una estimación inicial del estado del sistema y su incertidumbre asociada. Esta información se utiliza para arrancar el filtro de Kalman.
- **ACTUALIZAR** Este es uno de los dos pasos principales del filtro de Kalman. Combina la predicción previa con la medición actual para actualizar la estimación del estado del sistema y ajusta la incertidumbre de esa estimación.
- **PREDICIR** El otro paso clave del filtro utiliza el modelo del sistema para predecir el siguiente estado y su incertidumbre, basándose en la estimación anterior y considerando el retardo de una unidad.
- **Retardo de unidad ($n \rightarrow n-1$)** Este bloque representa el almacenamiento del estado estimado en el instante anterior para ser utilizado en la siguiente predicción.
- **OUTPUTS: Estimación del estado del sistema** Este es el resultado final del filtro: una estimación refinada del estado actual del sistema, que es más precisa que la simple medición directa.

En conjunto, el filtro de Kalman realiza un ciclo continuo de predicción y corrección, usando tanto el modelo del sistema como las mediciones reales, para obtener una estimación óptima del estado.

2.3.2. BYTETrack

BYTETrack [15] es un algoritmo avanzado de Seguimiento de Objetos Múltiples (MOT) que se inscribe dentro del paradigma de seguimiento por detección (*tracking-by-detection*). Este paradigma consiste en detectar objetos en cada fotograma de forma independiente y luego asociar estas detecciones a lo largo del tiempo para construir trayectorias coherentes. La innovación fundamental de BYTETrack reside en su novedoso método de asociación de datos, denominado BYTE, que aborda explícitamente un problema común en MOT: el manejo de detecciones con baja puntuación de confianza.

Mientras que la mayoría de los algoritmos de seguimiento descartan las detecciones por debajo de un cierto umbral de confianza para evitar la introducción de falsos positivos, BYTETrack reconoce que estas detecciones de baja confianza a menudo corresponden a objetos reales que están parcialmente ocluidos o cuya apariencia ha cambiado temporalmente. Descartarlas puede llevar a la pérdida de trayectorias y a una menor precisión general del seguimiento. Esta estrategia de asociación en dos pasos permite a BYTETrack recuperar objetos reales incluso cuando la confianza del detector disminuye debido a occlusiones o desenfoque, manteniendo la continuidad de las trayectorias. Al separar claramente las detecciones de alta y baja confianza y utilizarlas de manera diferenciada en el proceso de asociación, BYTETrack logra una notable mejora en la robustez del seguimiento, reduce significativamente la fragmentación de las trayectorias (medida por métricas como IDF1) y maneja eficazmente las variaciones en la calidad de las detecciones, todo ello manteniendo una alta eficiencia computacional.

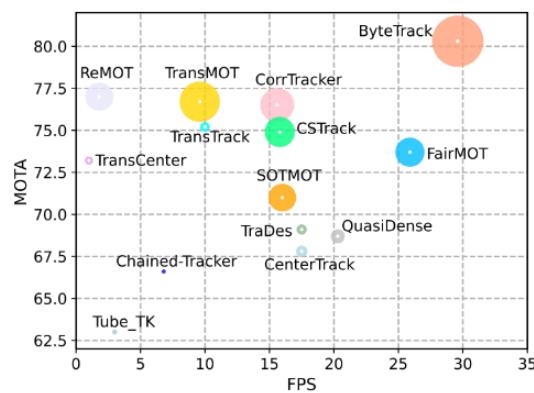


Figura 2.17: Comparativa de rendimiento de BYTETrack con otros algoritmos de seguimiento.

La Figura 2.17 presenta una comparativa de rendimiento que evidencia la superioridad de BYTETrack frente a otros algoritmos de seguimiento, según los resultados publicados en [15]. Como se observa, BYTETrack no solo alcanza una mayor precisión, medida por la métrica MOTA (Multiple Object Tracking Accuracy), sino que también demuestra una velocidad de procesamiento superior. Estas características lo posicionan como una solución particularmente eficaz y atractiva para aplicaciones que demandan seguimiento de objetos en tiempo real.

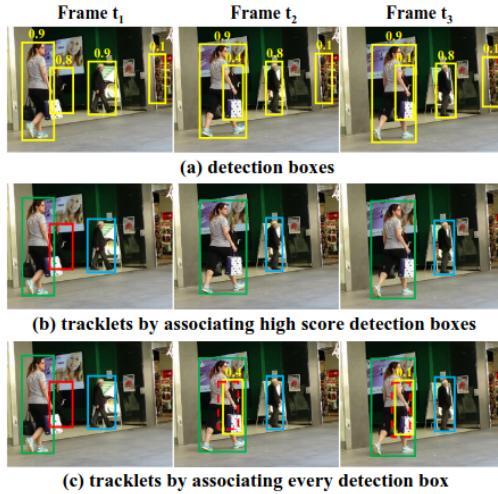


Figura 2.18: Ejemplo de detección y seguimiento de objetos utilizando BYTETrack.

Fuente: [15]

La Figura 2.18 ilustra el proceso de BYTETrack a través de tres fotogramas consecutivos (τ_1, τ_2, τ_3) de una secuencia de vídeo. En (a), se observan las detecciones iniciales que superan un umbral de confianza (p. ej., 0.5). La sección (b) muestra las trayectorias generadas al asociar exclusivamente las detecciones de alta confianza. Por el contrario, (c) presenta el resultado final de BYTETrack, que integra también las detecciones de baja confianza en el proceso de asociación. Esta comparación evidencia cómo la estrategia de BYTETrack permite mantener la continuidad de las trayectorias frente a desafíos como occlusiones parciales y variaciones en la confianza de las detecciones, logrando así una representación más precisa y robusta del movimiento de los objetos en el tiempo.

El funcionamiento del algoritmo BYTETrack es el siguiente:

Input: Una secuencia de vídeo V , un detector de objetos Det , y un umbral de confianza de detección τ .

Output: Las trayectorias \mathcal{T} de los objetos detectados en el vídeo.

1. **Inicialización:** Se inicializa el conjunto de trayectorias \mathcal{T} como vacío.
2. **Procesamiento por fotograma:** Para cada fotograma f_k en la secuencia de vídeo V :
 - 2.1. **Detección:** Se utiliza el detector Det para obtener las cajas delimitadoras y sus puntuaciones de confianza para el fotograma f_k , resultando en un conjunto de detecciones \mathcal{D}_k .
 - 2.2. **Separación de detecciones:** Se inicializan dos conjuntos vacíos: \mathcal{D}_{high} para detecciones de alta confianza y \mathcal{D}_{low} para detecciones de baja confianza. Se itera sobre cada detección d en \mathcal{D}_k :
 - Si la puntuación $d.score$ es mayor que el umbral τ , la detección d se añade a \mathcal{D}_{high} .
 - En caso contrario, la detección d se añade a \mathcal{D}_{low} .
 - 2.3. **Predicción de trayectorias:** Para cada trayectoria existente t en \mathcal{T} , se predice su nueva ubicación utilizando un Filtro de Kalman.
 - 2.4. **Primera asociación:** Se asocian las trayectorias \mathcal{T} con las detecciones de alta confianza \mathcal{D}_{high} utilizando el *Hungarian algorithm*[7] con la métrica de similitud IoU (Intersection over Union). Las detecciones no asociadas se guardan en \mathcal{D}_{remain} y las trayectorias no asociadas en \mathcal{T}_{remain} .

- 2.5. **Segunda asociación:** Se asocian las trayectorias restantes \mathcal{T}_{remain} con las detecciones de baja confianza \mathcal{D}_{low} utilizando otra métrica de similitud (Similarity#2, usualmente IoU). Las trayectorias que siguen sin asociarse se guardan en $\mathcal{T}_{re-remain}$. Solo se asocian detecciones de baja confianza a trayectorias que no pudieron ser asociadas con detecciones de alta confianza.
 - 2.6. **Eliminación de trayectorias no asociadas:** Se eliminan de \mathcal{T} las trayectorias que quedaron en $\mathcal{T}_{re-remain}$ (aquellas que no se pudieron asociar ni en la primera ni en la segunda etapa) si han permanecido sin asociar durante un número determinado de fotogramas (definido por el parámetro track_buffer).
 - 2.7. **Inicialización de nuevas trayectorias:** Se itera sobre las detecciones de alta confianza que no fueron asociadas (\mathcal{D}_{remain}). Cada una de estas detecciones se considera el inicio de una nueva trayectoria y se añade al conjunto \mathcal{T} .
3. **Retorno:** Una vez procesados todos los fotogramas, se devuelve el conjunto final de trayectorias \mathcal{T} .

2.4 Slicing Aided Hyper Inference

Explicación de la técnica de Slicing Aided Hyper Inference, como se utiliza para mejorar la precisión de los modelos de detección de objetos y como se aplica en este trabajo.

CAPÍTULO 3

Análisis del problema

????? ?????????????? ?????????????? ?????????????? ?????????????? ??????????????

CAPÍTULO 4

Diseño e implementación de la solución

En este capítulo se explicará la solución propuesta, como se ha diseñado y como se ha implementado.

4.1 Descripción del sistema

Descripción del sistema de visión artificial propuesto, como se ha diseñado y como se ha implementado.

4.2 Diseño de las etapas del sistema

Descripción de las etapas del sistema, como se han diseñado y como se han implementado.

Etapas del sistema:

- **Captura de imágenes:** Descripción de la etapa de captura de imágenes, como se ha diseñado y como se ha implementado.
- **Inferencia:** Descripción de la etapa de inferencia, como se ha diseñado y como se ha implementado.
- **Seguimiento:** Descripción de la etapa de seguimiento, como se ha diseñado y como se ha implementado.
- **Escritura de resultados:** Descripción de la etapa de escritura de resultados, como se ha diseñado y como se ha implementado.

4.3 Segmentación de las etapas del sistema

Tipos de segmentación de las etapas del sistema:

- **No segmentada:** Secuencial
- **Segmentación basada en hilos:** Cada etapa del sistema se ejecuta en un hilo diferente.

- **Segmentación basada en procesos:** Cada etapa del sistema se ejecuta en un proceso diferente.
- **Segmentación basada en hardware:** La etapa de inferencia se ejecuta en GPU, DLA0 y DLA1.
- **Segmentación basada en procesos con memoria compartida:** Cada etapa del sistema se ejecuta en un proceso diferente, pero comparten la memoria.

CAPÍTULO 5

Análisis de la solución

En este capítulo se analizará la solución propuesta variando los parámetros posibles

5.1 Variación de los parámetros

Explicación de los parámetros que se pueden variar en la solución propuesta y su efecto en el rendimiento del sistema.

—PRUEBA—

Model	IoU	CPU_Inference	GPU_Inference	DLA_Inference	CPU_Power	GPU_Power	DLA_Power	CPU_Energy	GPU_Energy	DLA_Energy
YOLOv11-N	0,85	45,2	12,3	15,8	8,2	12,5	6,8	369,64	153,75	107,44
YOLOv11-S	0,87	52,1	14,8	18,2	8,5	13,2	7,1	442,85	195,36	129,22
YOLOv11-M	0,89	68,4	18,2	22,5	9,1	14,8	7,8	622,44	269,36	175,5
YOLOv11-L	0,91	85,6	24,6	28,9	9,8	16,2	8,4	838,88	398,52	242,76

Tabla 5.1: Comparación de modelos en términos de inferencia, consumo de energía y potencia.

—PRUEBA—

5.2 Tipo de segmentación

En esta sección se analizará el rendimiento de la solución propuesta variando el tipo de segmentación de las etapas del sistema con gráficas y tablas.

5.3 Talla del modelo

En esta sección se analizará el rendimiento de la solución propuesta variando la talla del modelo de detección de objetos con gráficas y tablas.

5.4 Precisión del modelo

En esta sección se analizará el rendimiento de la solución propuesta variando la precisión del modelo de detección de objetos con gráficas y tablas.

5.5 Modo de energía y cores de la CPU

En esta sección se analizará el rendimiento de la solución propuesta variando el modo de energía del dispositivo y el número de cores de la CPU con gráficas y tablas.

5.6 Tamaño de la imagen

En esta sección se analizará el rendimiento de la solución propuesta variando el tamaño de la imagen de entrada del modelo con la técnica de Slicing Aided Hyper Inference (SAHI) con gráficas y tablas.

CAPÍTULO 6

Prueba de concepto

Aquí se explicará la implementación de la solución propuesta en el entorno de producción con la cinta transportadora.

6.1 Construcción del entorno

6.2 Instalación del entorno

????? ?????????????? ?????????????? ?????????????? ?????????????? ??????????????

CAPÍTULO 7

Conclusiones

????? ?????????????? ?????????????? ?????????????? ?????????????? ??????????????

Bibliografía

- [1] Anders S. G. Andrae and Tomas Edler. On global electricity usage of communication technology: Trends to 2030. *Challenges*, 6(1):117–157, 2015.
- [2] ENCCS. The gpu hardware and software ecosystem. <https://enccs.github.io/gpu-programming/2-gpu-ecosystem/>, 2025. Parte del curso "GPU programming: why, when and how?".
- [3] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.
- [4] Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024.
- [5] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2023.
- [6] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun. *A Guide to Convolutional Neural Networks for Computer Vision*. Synthesis Lectures on Computer Vision. Springer Cham, 1 edition, 2018.
- [7] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.
- [8] Tushar Kumar. R-cnn explained, 2024. Accedido: 14 de abril de 2025.
- [9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [10] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.
- [11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. *SSD: Single Shot MultiBox Detector*, page 21–37. Springer International Publishing, 2016.
- [12] NVIDIA Corporation. Jetson modules, support, ecosystem, and lineup. <https://developer.nvidia.com/embedded/jetson-modules>, 2025. Accedido el 24 de abril de 2025.
- [13] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.
- [14] Liming Xiu. Time moore: Exploiting moore's law from the perspective of time. *IEEE Solid-State Circuits Magazine*, 11:39–55, 01 2019.

- [15] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box, 2022.

APÉNDICE A

Configuración del sistema

????? ?????????????? ?????????????? ?????????????? ?????????????? ??????????????

A.1 Fase de inicialización

????? ?????????????? ?????????????? ?????????????? ?????????????? ??????????????

A.2 Identificación de dispositivos

????? ?????????????? ?????????????? ?????????????? ?????????????? ??????????????

APÉNDICE B

??? ?????????????? ????

????? ?????????????? ?????????????? ?????????????? ?????????????? ??????????????