Universidades de Burgos, León y Valladolid

Máster universitario

Inteligencia de Negocio y Big Data en Entornos Seguros

TFM del Máster en Inteligencia de Negocio y Big Data en Entornos Seguros

Estudio de librerías de detección de posturas sobre dispositivos móviles

Presentado por Francisco Javier Romero Carrillo
en Universidad de Valladolid

01 de septiembre de 2025

Tutor: Bruno Baruque Zanón

RESUMEN

Este trabajo presenta un estudio comparativo de diversos modelos de visión artificial preentrenados en estimación de posturas humanas integrables en dispositivos móviles. Existen numerosos modelos de estimación de posturas con características muy heterogéneas y documentaciones dispares según sus desarrolladores, por lo que la aportación principal de este trabajo radica en ofrecer una evaluación homogénea de su funcionamiento. El objetivo es facilitar la elección del más adecuado de ellos para su posterior incorporación en el desarrollo de una futura aplicación para la asistencia a ejercicios de tele-rehabilitación en domicilio que permita guiar al usuario en estos ejercicios y registrar información relevante para el seguimiento clínico, sin necesidad de sensores adicionales ni conexión a internet para el proceso de realización de los ejercicios.

Para garantizar un análisis homogéneo, se eligió un subconjunto de imágenes filtrado del conjunto de datos COCO, compuesto por 316 imágenes que contienen una única persona con al menos 15 keypoints (puntos clave) anotados sobre el cual se han evaluado diferentes versiones de tres familias de modelos: MoveNet, BlazePose y YOLOv8-Pose, desde dos perspectivas:

- Precisión en la detección de puntos clave para identificación de posturas: exactitud de los modelos al predecir las posturas, medida mediante la métrica AP (Average Precision). Se ha utilizado con la finalidad de poder validar resultados del modelo así como para evaluar la idoneidad de las imágenes seleccionadas.

- Rendimiento: tiempo medio de inferencia por imagen sobre un dispositivo móvil Android en condiciones reales. Se ha utilizado para medir el rendimiento de cada modelo (tiempo de ejecución de cada inferencia de cada imagen) en diferentes dispositivos con el fin de poder evaluar la velocidad con la que cada modelo efectúa la estimación.

Los resultados muestran diferencias significativas entre modelos en cuanto a la relación precisión-tiempo, destacando las variantes de la subfamilia Thunder de MoveNet y la versión Nano de YOLOv8-Pose por su equilibrio entre rendimiento y exactitud.

Este estudio aporta una visión clara y práctica sobre la aplicabilidad de distintos enfoques de estimación de postura en entornos móviles, sirviendo como referencia para desarrolladores e investigadores interesados en sistemas embebidos de visión por computador.

ABSTRACT

This paper presents a comparative study of various pre-trained computer vision models for human pose estimation that can be integrated into mobile devices. There are numerous pose estimation models with highly heterogeneous characteristics and uneven documentation provided by their developers, so the main contribution of this work lies in offering a homogeneous evaluation of their performance. The aim is to facilitate the selection of the most suitable model for subsequent incorporation into the development of a future application for assisting with home telerehabilitation exercises that will guide the user through these exercises and record relevant information for clinical follow-up, without the need for additional sensors or an internet connection during the exercise process.

To ensure a homogeneous analysis, a filtered subset of images from the COCO dataset was selected. This subset consists of 316 images containing a single person with at least 15 annotated keypoints. Different versions of three model families, MoveNet, BlazePose, and YOLOv8-Pose, were evaluated from two perspectives:

- Keypoint detection accuracy for pose identification: the accuracy of the models in predicting poses, measured using the Average Precision (AP) metric. It was used to validate model results and to evaluate the suitability of the selected images.

- Performance: average inference time per image on an Android mobile device under real-world conditions. It was used to measure the performance of each model (inference execution time for each image) on different devices to evaluate the speed with which each model performs the estimation.

The results show significant differences between models in terms of accuracy-time ratio, with the Thunder subfamily of MoveNet and the Nano version of YOLOv8-Pose standing out for their balance between performance and accuracy.

This study provides a clear and practical insight into the applicability of different pose estimation approaches in mobile environments, serving as a reference for developers and researchers interested in embedded computer vision systems.

Índice general

Índice de figuras

Índice de tablas

Índice de fórmulas

PARTE I: DESCRIPCIÓN DEL PROYECTO

1. INTRODUCCION

1.1. Contexto

La estimación de posturas humanas (HPE, *Human Pose Estimation*) es una disciplina dentro de la visión por computador que busca identificar la posición y orientación del cuerpo humano en imágenes o vídeos mediante la localización de keypoints o puntos clave (como ojos, manos, codos, rodillas, hombros, tobillos, etc.). El interés por la estimación de posturas humanas ha crecido significativamente en los últimos años debido a su utilidad en ámbitos como el deporte, la salud, la interacción hombre-máquina, la realidad aumentada o la vigilancia automática. Esta tecnología permite identificar con precisión la posición de las articulaciones del cuerpo humano a partir de imágenes o vídeo, dando lugar a aplicaciones inteligentes capaces de interpretar el comportamiento físico de una persona en tiempo real. Este proceso se describe detalladamente en la sección "2.2. Estimación de posturas humanas".

Tradicionalmente, los sistemas de estimación de posturas requerían hardware de alto rendimiento y ejecución en servidores remotos. Sin embargo, los avances recientes en modelos ligeros y técnicas de optimización han permitido llevar estas capacidades a dispositivos móviles, como smartphones y tablets, abriendo la puerta a soluciones descentralizadas, eficientes y respetuosas con la privacidad del usuario.

En los últimos años, los avances en arquitecturas de redes neuronales profundas, junto con la disponibilidad de datasets anotados como COCO (1) o MPII (2) que vemos más adelante, han permitido desarrollar modelos de estimación de posturas cada vez más precisos. Sin embargo, muchos de estos modelos han sido diseñados para ejecutarse en entornos con alto poder computacional, como servidores con GPU, lo que dificulta su despliegue directo en dispositivos móviles con recursos limitados.

1.2. Motivación del estudio en dispositivos móviles

Los teléfonos móviles y dispositivos embebidos han evolucionado significativamente y cada día cuentan con capacidades de computación cada vez mayores. Esto hace que frente al **cloud computing** (computación en la "nube") donde los datos son procesados en centros de datos remotos y centralizados, surja un nuevo paradigma llamado **edge computing** (computación en el "borde") en el cual a diferencia del cloud computing los datos se procesan cerca de la fuente donde se generan en lugar de enviarse a centros remotos.

A pesar de ello, en estos dispositivos aún siguen existiendo restricciones importantes de memoria, potencia de cálculo y consumo energético que requieren la elección cuidadosa de modelos optimizados para estos entornos.

Evaluar el comportamiento de distintos modelos de estimación de posturas directamente sobre dispositivos móviles resulta interesante para determinar su viabilidad en aplicaciones del mundo real, especialmente cuando se busca un equilibrio entre precisión y velocidad. Este tipo de análisis cobra aún más importancia cuando los modelos deben integrarse en aplicaciones

móviles de salud, deporte o monitoreo, donde la fiabilidad y la latencia son factores determinantes.

En este contexto, existe una amplia variedad de modelos preentrenados disponibles públicamente, desarrollados por distintas organizaciones (Google o Ultralytics, entre otras). La motivación de este estudio radica, por tanto, en la necesidad de analizar de forma objetiva y reproducible qué modelos de estimación de posturas resultan más adecuados para su uso en entornos móviles, evaluando tanto su precisión en tareas reales como su rendimiento en tiempo de ejecución.

Este trabajo no solo permitirá identificar los modelos más equilibrados en términos de eficiencia y calidad, sino que también servirá como referencia técnica para desarrolladores, investigadores o empresas interesadas en implementar visión por computador avanzada en entornos móviles.

1.3. Objetivos

Este Trabajo Fin de Máster tiene como objetivo principal estudiar y comparar el comportamiento de distintas librerías y modelos de estimación de postura humana ejecutados sobre dispositivos móviles con sistema operativo Android.

1.3.1 Objetivo general

El objetivo general de este Trabajo Fin de Máster es realizar **un estudio del *estado del arte* en estimación de posturas humanas**, que incluye la **selección de distintas familias de modelos de referencia y su posterior evaluación** en términos de rendimiento y precisión al ejecutarse sobre dispositivos móviles.

1.3.2 Objetivos específicos

Los objetivos específicos de este Trabajo Fin de Master son:

- **Búsqueda y selección** de modelos existentes, revisión de documentación y selección de los modelos para la realización del estudio.
- Implementar un **sistema de pruebas para la ejecución de modelos** con formato TensorFlow Lite (TFLite) en un entorno Android.
- Medir y comparar la **precisión** de los modelos mediante métricas como Average Precision (AP@[0.50:0.95]).
- Medir y comparar el **tiempo medio de inferencia** por imagen en cada modelo.
- Establecer una **relación entre la calidad de las predicciones y los tiempos de ejecución**.

- **Identificar los modelos más adecuados** para su uso en aplicaciones móviles reales, considerando el compromiso entre precisión y eficiencia.

1.4. Estructura del documento

El contenido de esta memoria se estructura de la siguiente manera:

- Apartado 2: **Marco teórico y *estado del arte***, se explican los fundamentos de la estimación de posturas, los diferentes modelos preentrenados existentes y los entornos de ejecución en Android.

- Apartado 3: **Metodología**, se detallan los modelos seleccionados para el estudio, la descripción del dataset de imágenes comunes que utilizaremos y las métricas, herramientas y dispositivos utilizados para la realización de la evaluación.

- Apartado 4: **Planificación**, se definen las fases y subfases de la que consta el proyecto y la planificación temporal prevista.

- Apartado 5: **Fase 1: Preparación del dataset de testeo y obtención de modelos**, se describe el proceso de selección de las imágenes del dataset de testeo así como el de la obtención de los modelos para el estudio.

- Apartado 6: **Fase 2: Desarrollo de la aplicación**, se describe el proceso de desarrollo de la aplicación para Android donde se integraron y probaron tanto modelos como imágenes de testeo, así como la obtención de los datos de salida para su evaluación.

- Apartado 7: **Fase 3: Evaluación y análisis de resultados**, se presentan las métricas obtenidas, tanto de precisión como de rendimiento de todos los modelos examinados y las comparativas entre ambas variables.

- Apartado 8: **Discusión**, se interpreta los resultados, se discuten las fortalezas y debilidades de los modelos y se plantean las implicaciones prácticas.

- Apartado 9: **Conclusiones** y trabajo futuro, se resumen los hallazgos más relevantes y se proponen futuras líneas de investigación.

- Apartado 10: **Referencias bibliográficas**.

- Apartado 11: **Anexos**.

2. MARCO TEORICO Y ESTADO DEL ARTE

En este apartado se introducen los fundamentos necesarios para comprender el entorno donde se desarrolla el estudio, el proceso de estimación de posturas humanas, se describen las familias de modelos preentrenados existentes y cuales utilizaremos para ser evaluados y se presentan los aspectos técnicos relacionados con su integración y ejecución en dispositivos móviles. El objetivo es proporcionar un contexto conceptual y tecnológico que justifique la elección de los modelos y la metodología empleada en este estudio.

2.1. Edge computing

El *edge computing* (o computación en el borde) es un paradigma de computación distribuida que acerca la computación y el almacenamiento de datos a los dispositivos donde se generan, esto es en el "borde" de la red cerca de la fuente de datos (Imagen 1). Esto contrasta con la computación en la nube tradicional, donde los datos se procesan en centros de datos centralizados y a menudo ubicados lejos de la fuente. Al procesar datos en el borde se puede reducir la latencia, mejorar los tiempos de respuesta y minimizar el uso del ancho de banda pero los dispositivos utilizados tienen generalmente capacidades de procesamiento y almacenamiento muy limitadas en comparación con los servidores en la nube (3).

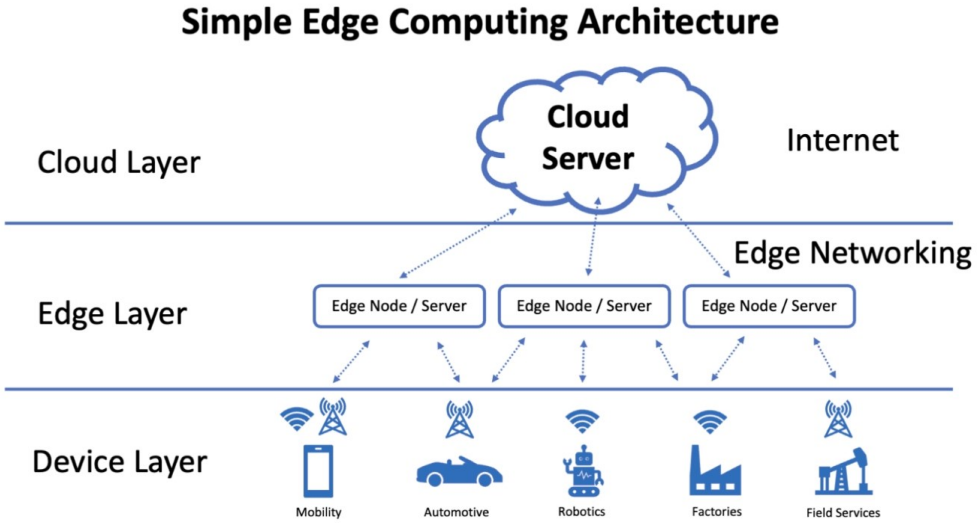


Imagen 1. Arquitectura simple de edge computing (3)

Este trabajo evalúa los resultados de llevar a dispositivos de borde (como pueden ser teléfonos móviles) una tarea como la de estimación de posturas humanas, que hasta hace relativamente poco tiempo requería mayores capacidades de procesamiento que hacían que no pudieran ser ejecutadas de forma distribuida en dispositivos con menor capacidad de proceso al alcance de cualquier persona.

2.2. Estimación de posturas humanas

2.2.1. Conceptos básicos

La estimación de postura humana es una tarea de visión por computador que consiste en **predecir la ubicación espacial de puntos clave (keypoints) del cuerpo humano en imágenes o secuencias de vídeo**. Los keypoints suelen incluir articulaciones como **codos, rodillas, tobillos, caderas, hombros**, entre otros, y se pueden conectar para formar un esqueleto digital. Existen múltiples enfoques para abordar este problema, y su clasificación puede realizarse atendiendo a diferentes criterios que reflejan tanto la estrategia de detección como la forma de representar el cuerpo humano en los modelos computacionales (4).

2.2.2. Clasificación de los tipos de detección de posturas

Desde el punto de vista operativo, los métodos de detección pueden clasificarse desde diferentes enfoques, dependiendo de si atendemos a su clasificación por tipo espacial (2D o 3D), por cantidad de objetivos (mono-persona o multi-persona), al método de detección (*top-down* o *bottom-up*) o al tipo de modelado (modelos cinemáticos, planares o volumétricos). Estas clasificaciones generales permiten entender la diversidad de enfoques existentes, valorar sus ventajas y limitaciones en función del contexto, y facilitar la selección de la solución más adecuada según el caso de uso y los recursos disponibles.

Clasificación por tipo de detección espacial

- Estimación de posturas 2D. Localiza coordenadas en dos dimensiones (x, y) de una o varias personas sobre el plano de una imagen o video, tiene como ventajas mayor velocidad y menor coste computacional sobre las estimaciones de posturas en 3D y como limitaciones que no se obtiene percepción de profundidad.

- Estimación de posturas 3D. Añade la dimensión *Z* proporcionando información sobre la profundidad y la orientación del cuerpo en el entorno físico. Ofrece un mayor realismo y precisión en tareas biomecánicas, robótica o animación aunque requiere un coste computacional mayor.

Clasificación por cantidad de objetivos

- Mono-persona (*single-person*). El modelo asume que la imagen contiene una única persona (Imagen 2), generalmente centrada y completamente visible. Este supuesto permite que la red se enfoque exclusivamente en detectar los keypoints corporales sin necesidad de mecanismos adicionales de segmentación o agrupamiento. Por lo general, estos modelos utilizan una única pasada de inferencia sobre toda la imagen.

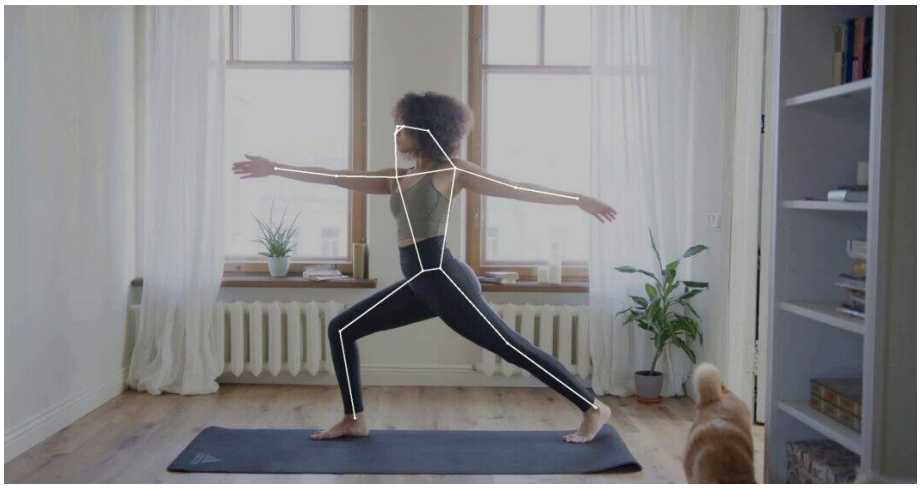


Imagen 2. Ejemplo de detección single-person (4)

- Multi-persona (*multi-person*). El enfoque *multi-person* está diseñado para identificar y estimar la postura de varias personas simultáneamente (Imagen 3) en una misma imagen o secuencia. Este tipo de modelos requiere, además de detectar los keypoints, asignarlos correctamente a cada instancia individual por lo que es más costosa computacionalmente.

Imagen 3. Ejemplo de detección multi-person (4)

Clasificación por método de detección

- *Top-down*. Realizan primero una detección de personas (caja contenedora) y luego aplican un modelo de estimación de postura a cada una de ellas por separado estimando los keypoints dentro de ella.

- *Bottom-up*. Detectan primero todos los keypoints en la imagen y posteriormente los agrupan en función de su pertenencia a cada individuo.

Clasificación por tipo de modelado del cuerpo

- Modelos cinemáticos. Representan el cuerpo humano como un sistema articulado compuesto por un conjunto de puntos clave (keypoints) conectadas por segmentos que representan el cuerpo como un esqueleto de articulaciones. Esta estructura permite modelar los grados de libertad y el movimiento relativo entre las partes del cuerpo. Son ampliamente utilizados en tareas de análisis de movimiento, biomecánica y aplicaciones de realidad aumentada.

- Modelos planares. Se centran en la representación del contorno externo del cuerpo o de sus partes visibles en una imagen 2D. Utilizan técnicas de segmentación para extraer las siluetas y contornos, proporcionando una aproximación más visual y basada en la forma.

- Modelos volumétricos. Los modelos volumétricos buscan reconstruir la forma completa y tridimensional del cuerpo humano, incluyendo su volumen y superficies internas.

2.3. Tecnologías para la estimación de posturas humanas en edge computing

La estimación de posturas humanas mediante visión por computador se apoya principalmente en el uso de modelos de aprendizaje profundo, concretamente en redes neuronales convolucionales (CNN por sus siglas en ingles de *Convolutional Neural Networks*), por su capacidad para extraer representaciones espaciales jerárquicas a partir de imágenes. Este apartado presenta los fundamentos arquitectónicos de las CNNs como base de la mayoría de modelos utilizados en este estudio, así como las técnicas de optimización necesarias para su ejecución eficiente en dispositivos móviles, entre las que destaca la cuantización. Estas estrategias permiten reducir el tamaño y la latencia de los modelos sin comprometer significativamente su precisión, facilitando su integración en entornos de computación embebida o de recursos limitados como dispositivos móviles. La comprensión de estos elementos es clave para contextualizar la selección, implementación y evaluación de los modelos estudiados en este trabajo. La naturaleza de este estudio nos lleva a utilizar redes con tipo de detección espacial en 2D, para la identificación de una única persona o *single-person*, con un enfoque *top-down* (más adecuadas para escenarios *single-person*) y con tipo de modelado cinemático (conjunto de puntos clave o *keypoints*).

2.3.1. Redes neuronales convolucionales (CNNs)

Una **red neuronal artificial** es un **modelo computacional** compuesto por capas de nodos (neuronas) conectados entre sí mediante pesos ajustables. Cada nodo aplica una función de activación a la suma ponderada de sus entradas, y la red aprende a realizar tareas (como clasificación o regresión) ajustando estos pesos mediante un proceso iterativo de entrenamiento con retropropagación.

Una **convolución** es una **operación matemática** que se aplica a una entrada (como una imagen) utilizando un pequeño conjunto de pesos llamado filtro, con el objetivo de extraer

características locales relevantes como bordes, texturas o formas (Imagen 4). Durante esta operación, el filtro se desliza (o convoluciona) sobre la entrada, multiplicando sus valores por los de la región correspondiente de la imagen y sumando los resultados en cada posición. El resultado es un mapa de activación o mapa de características, que conserva la información espacial y refleja la presencia de patrones específicos aprendidos por el filtro. Esta operación permite a las CNNs detectar características jerárquicas en las primeras capas, detectan elementos simples (líneas, esquinas), mientras que en capas más profundas reconocen estructuras más complejas (formas, objetos). La convolución reduce la dimensionalidad y preserva la relación espacial, haciendo a las CNNs especialmente eficaces en tareas de visión por computador.

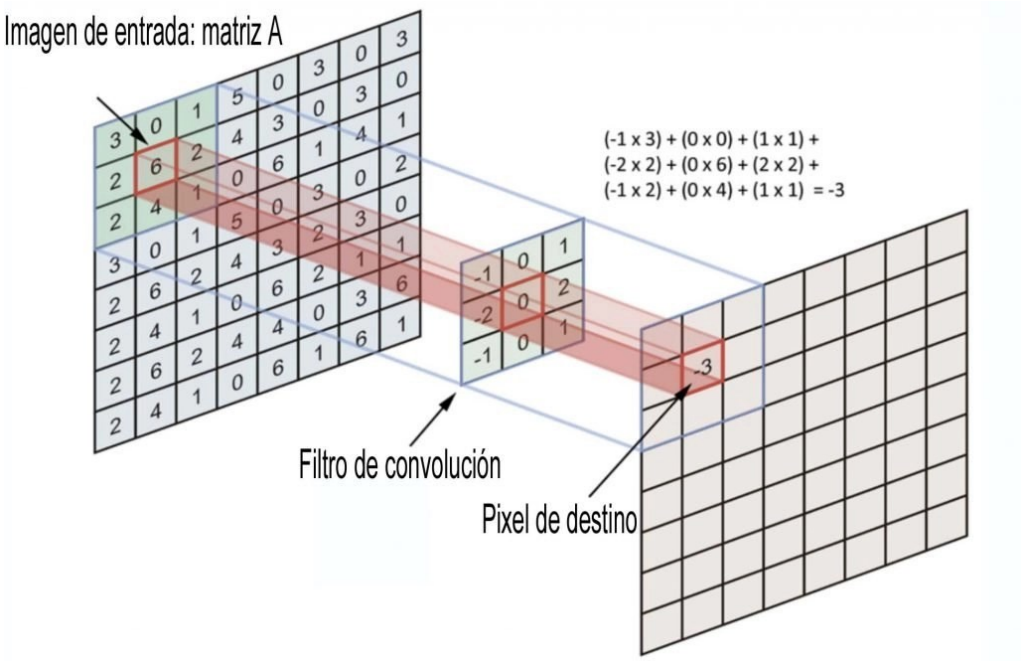


Imagen 4. Esquema operación básica de convolución (5)

Las **redes neuronales convolucionales** son un tipo especializado de red neuronal diseñada para procesar datos con estructura de rejilla (como imágenes) mediante el uso de capas convolucionales que aprenden representaciones espaciales jerárquicas. Estas redes extraen automáticamente características relevantes aplicando filtros locales y compartiendo pesos, lo que las hace muy eficientes y efectivas para tareas de visión por computador (5) (6).

Una CNN se construye como una arquitectura jerárquica compuesta por una secuencia organizada de **capas**, cada una con un propósito específico en el procesamiento y la abstracción progresiva de los datos de entrada (generalmente imágenes). Estas capas se agrupan funcionalmente en **bloques**, cuya disposición refleja el flujo de información desde los niveles bajos de detección de patrones simples hasta los niveles superiores de interpretación semántica.

En el diseño de CNNs por tanto distinguimos entre bloques funcionales y capas individuales, ya que ambos niveles estructurales cumplen roles diferenciados pero interrelacionados en el flujo de procesamiento de la información. Mientras que los bloques agrupan funciones específicas del modelo en etapas coherentes del pipeline de aprendizaje, las capas son las unidades básicas de operación que materializan dichas funciones.

A nivel de **bloques funcionales** nos encontramos varios bloques diferenciados por su función dentro de la arquitectura de la CNN:

- **Capa de entrada**. Recibe los datos en formato tensorial, por ejemplo, una imagen RGB representada como un tensor tridimensional (altura × anchura × canales) y normaliza sus valores si es necesario.

- ***Feature extraction*** (extracción de características) (7). Este bloque se corresponde con las primeras capas de la red y está compuesto principalmente por **capas convolucionales**, para detectar patrones espaciales locales como bordes, texturas y formas, y **capas de activación** no lineales (como ReLU [1]), que permiten a la red modelar relaciones complejas y no lineales. Frecuentemente, se añaden **capas de *pooling*** (como *max pooling* o *average pooling*[2]) que permiten la reducción de la dimensionalidad espacial manteniendo información relevante, y **capas de normalización** que estabilizan el proceso de entrenamiento diseñadas para transformar una imagen cruda en un conjunto de representaciones útiles. Este bloque es esencial porque extrae la información visual jerárquica que luego será interpretada por las *prediction heads*.

- ***Prediction heads*** (bloques de predicción) (8), este bloque está compuesto por un conjunto de capas cuya función principal es transformar las representaciones intermedias extraídas por el modelo en predicciones específicas para una tarea determinada. Estas *prediction heads* operan sobre los mapas de características generados por el bloque de extracción de características (*feature extraction*) y adaptan la salida del modelo a diferentes tipos de problemas, como clasificación, regresión, detección de objetos, segmentación o estimación de posturas humanas. Desde el punto de vista estructural, las *prediction heads* pueden estar integradas por capas completamente conectadas (*fully connected layers*) también llamadas capas densas, capas convolucionales adicionales, o incluso subredes específicas diseñadas para tareas particulares. La elección de su arquitectura depende directamente del tipo de información que se requiere predecir y del grado de detalle espacial que se debe preservar.

- **Capa de salida**. Es la responsable de generar la predicción final del modelo, ya sea en forma de probabilidades, coordenadas, etiquetas o mapas espaciales, dependiendo de la tarea específica para la cual ha sido diseñada la red. La

[1] ReLU (*Rectified Linear Unit)*: función de activación que introduce no linealidad en la red, ReLU(x)=max(0,x)
[2] *Max pooling* selecciona el valor máximo dentro de una ventana de agrupación, mientras que *average pooling* calcula el promedio de todos los valores en la ventana.

configuración de la capa de salida varía según el tipo de problema (clasificación, detección, segmentación, etc.), pero siempre está diseñada para ofrecer una representación final interpretable y directamente utilizable.

En el diseño estructural de una red neuronal convolucional además de las capas especializadas en el procesamiento espacial de datos, como las capas convolucionales y de *pooling*, también se incorporan otros tipos de capas que desempeñan un papel crucial en la integración y decisión final del modelo. Estas capas se dividen fundamentalmente en *connected layers* y *fully connected layers*, y su inclusión varía según el tipo de tarea a resolver:

- *Connected layers* o capas conectadas (9). Hacen referencia a aquellas capas en las que cada nodo de entrada está conectado a uno o varios nodos de la siguiente capa, aunque no necesariamente a todos. Este tipo de conexión se utiliza a menudo en arquitecturas que buscan una transición progresiva entre la representación espacial y la salida vectorial o categórica, permitiendo una reducción gradual de la dimensionalidad sin perder información estructural relevante.

- *Fully connected layers* o capas densas (10). Representan un caso particular de *connected layers*, donde cada neurona de una capa está conectada a todas las neuronas de la capa siguiente. Estas capas son especialmente útiles para tareas de clasificación y regresión, ya que permiten combinar todas las características aprendidas previamente en una representación densa que puede ser fácilmente interpretada por una función de activación final (como softmax [3]o sigmoid[4]). Aunque son potentes en términos de capacidad de representación, también implican un elevado coste computacional y una mayor cantidad de parámetros, lo que puede conllevar un riesgo de sobreajuste si no se utilizan técnicas de regularización adecuadas.

En conjunto, tanto las *connected layers* como las *fully connected layers* permiten a la CNN realizar inferencias de alto nivel, integrando la información extraída por las capas anteriores y generando salidas estructuradas que se ajustan a los requerimientos específicos de la tarea de aprendizaje supervisado (Imagen 5).

[3] La función de activación softmax transforma un vector entero de números en una distribución de probabilidad

[4] Función de activación que toma la suma ponderada de las entradas de la capa anterior y la transformar en un valor de salida entre 0 y 1

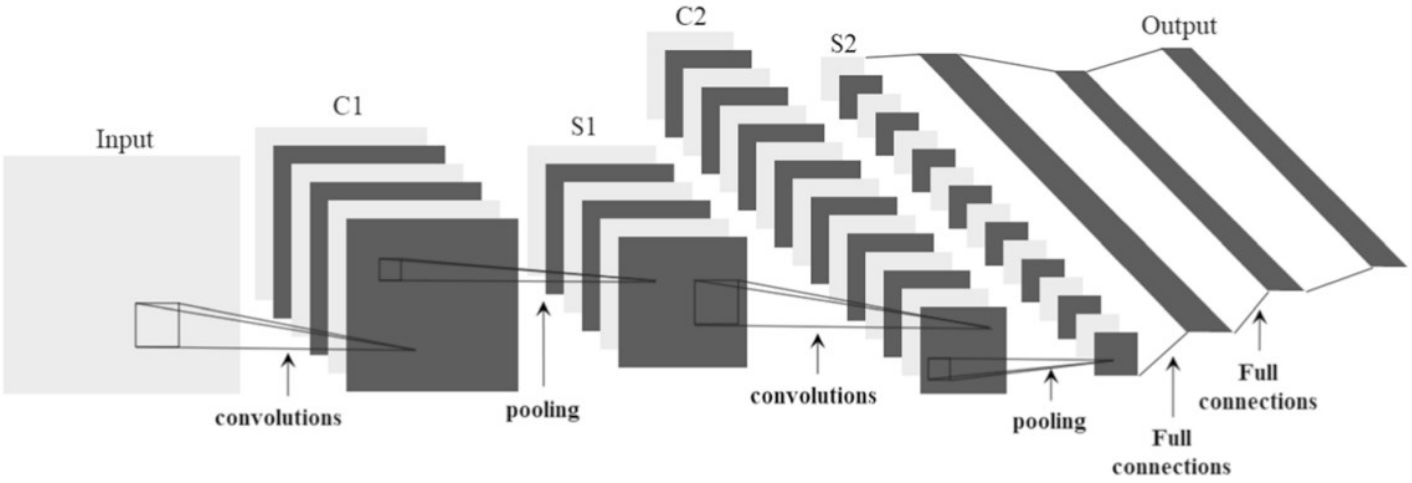


Imagen 5. Esquema general de capas de las CNNs (11)

Arquitecturas de redes neuronales convolucionales

A lo largo de la evolución de esta disciplina, se han propuesto numerosas arquitecturas, desde modelos clásicos hasta redes ligeras optimizadas para dispositivos móviles. Estas arquitecturas responden a las crecientes demandas de precisión, eficiencia computacional y adaptabilidad a diferentes entornos de ejecución. Algunas están específicamente diseñadas para preservar la información espacial a lo largo de las distintas capas de la red, lo cual es esencial para una localización precisa de los puntos clave del cuerpo humano, como en el caso de HRNet o Lite-HRNet. Otras, como MobileNet o GhostNet, han sido optimizadas por grandes compañías tecnológicas como Google (12) y Huawei, respectivamente, con el objetivo de ofrecer inferencias rápidas en tiempo real, incluso en dispositivos con capacidades limitadas (13). Estas optimizaciones suelen involucrar técnicas como la cuantización de pesos, la poda de parámetros o el uso de bloques convolucionales eficientes. Además, la incorporación de arquitecturas basadas en transformadores y modelos híbridos ha permitido una mejor modelización contextual y mejoras en tareas más complejas como la estimación 3D o la multi-persona. A continuación, se presenta una clasificación de algunas de las arquitecturas de CNNs más representativas en el ámbito de la estimación de posturas humanas, destacando sus principales características y cómo han evolucionado para adaptarse a las distintas necesidades de esta área de investigación.

Arquitecturas clásicas (convencionales)

Estas arquitecturas constituyen la base histórica del aprendizaje profundo en visión por computador. Se caracterizan por tener arquitecturas secuenciales y relativamente simples, en las que las capas convolucionales se intercalan con capas de activación (ReLU), *pooling* y capas densas finales (Tabla 1). Aunque hoy en día se consideran menos eficientes, fueron clave en el avance inicial de la disciplina (11).

- LeNet (1998). Utilizado originalmente para reconocimiento de dígitos manuscritos. Fue pionero en el uso de convoluciones y pooling en redes neuronales.

- AlexNet (2012). Revolucionó el campo al ganar el desafío ImageNet[5] con una gran mejora de precisión. Introdujo el uso de GPU para entrenamiento, activación ReLU y regularización con *dropout*[6] (Imagen 6).

- ZFNet (2013). Introdujo técnicas para visualizar filtros y comprender el funcionamiento interno de las redes, mejorando la arquitectura de AlexNet.

Arquitectura	Año	Características
LeNet	1998	Primer uso práctico de CNN; reconocimiento de dígitos
AlexNet	2012	ReLU, dropout, GPU training; ganador ImageNet 2012
ZFNet	2013	Mejora de AlexNet, visualización de filtros

Tabla 1. Arquitecturas clásicas de CNNs

Aunque ya no son las más utilizadas, siguen empleándose como puntos de comparación, en *benchmarks* estandarizados o como punto de partida para el aprendizaje transferido. Su papel histórico como base del desarrollo de redes más modernas les otorga un valor de referencia, especialmente útil para evaluar mejoras en precisión, eficiencia y capacidad generalizadora. Además, su simplicidad relativa permite entender conceptos fundamentales del diseño de redes profundas y facilita su implementación en entornos educativos y de investigación.

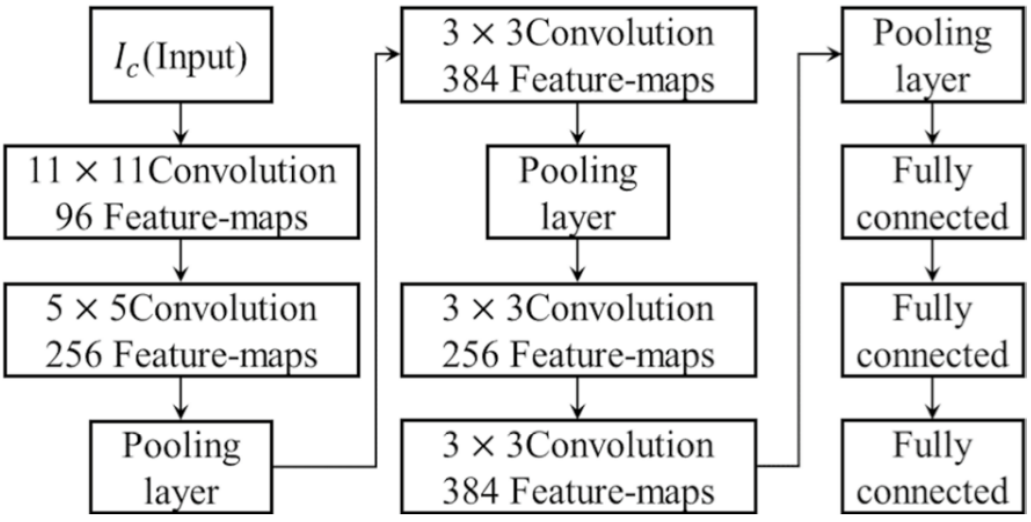


Imagen 6. Esquema de arquitectura de la red AlexNet (11)

[5] ImageNet, desafío anual de reconocimiento visual a gran escala (ILSVRC por sus siglas en inglés)
[6] Técnica de regularización que se basa en la eliminación de neuronas en las capas de la red neuronal que es aplicada en base a la probabilidad dada por la distribución de Bernoulli

Arquitecturas optimizadas con módulos o bloques avanzados

Introducen componentes estructurales innovadores que mejoran la capacidad de aprendizaje y reducen los problemas de entrenamiento en redes profundas, como la desaparición del gradiente (Tabla 2) (11).

- VGG16/VGG19 (2014). Modelos profundos con 16 o 19 capas, conocidos por utilizar exclusivamente convoluciones 3×3 y *pooling* 2×2, lo que los hace conceptualmente simples pero computacionalmente pesados (Imagen 7).

- GoogLeNet / Inception (2014-2016). Introducen módulos Inception, que combinan convoluciones de distintos tamaños (1×1, 3×3, 5×5) en paralelo dentro de un mismo bloque. Aportan eficiencia y profundidad sin un incremento excesivo en parámetros.

- ResNet (2015). Introduce conexiones residuales (*skip connections*[7]) que permiten entrenar redes de más de 100 capas sin degradación del rendimiento. Se convirtió en el nuevo estándar para tareas de clasificación y segmentación.

- DenseNet (2017). Cada capa recibe como entrada todas las salidas anteriores del bloque. Mejora la reutilización de características y permite entrenar modelos muy profundos con menos parámetros.

- ResNeXt. Variante de ResNet que introduce el concepto de cardinalidad (uso de múltiples caminos en paralelo) para mejorar la expresividad sin aumentar demasiado los parámetros.

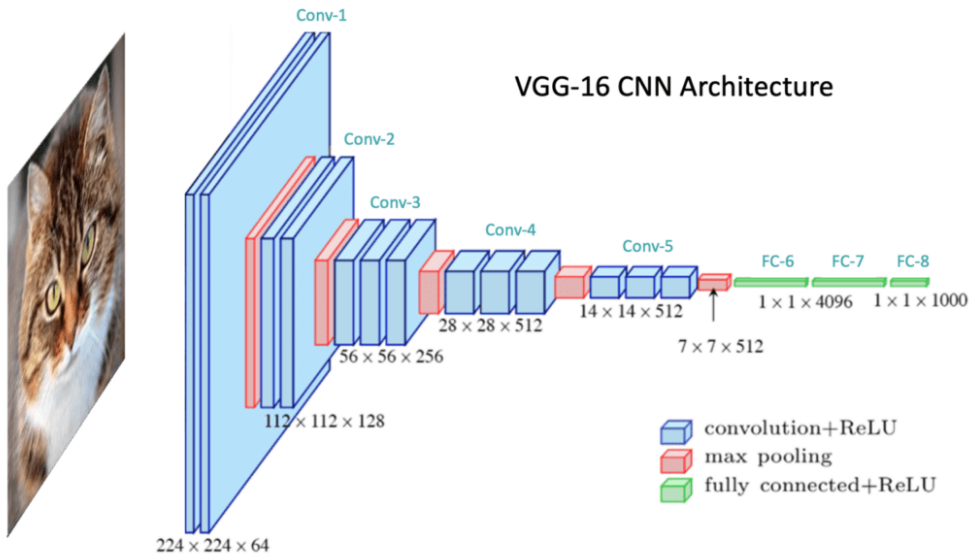


Imagen 7. Representación arquitectura de red neuronal convolucional VGG-16 (14)

[7] Técnica de diseño de redes neuronales que permite que los gradientes fluyan de manera más efectiva durante la retropropagación, lo que ayuda a entrenar modelos más profundos.

Actualmente, estas arquitecturas son ampliamente utilizadas como *backbones*[8] en modelos más complejos dentro del campo de la visión por computador, desempeñando un papel central en tareas avanzadas como la estimación de poses humanas, la segmentación semántica y la detección de objetos. Su éxito radica en un diseño modular y altamente optimizado que permite la extracción jerárquica de características, es decir, la progresiva representación de patrones visuales desde descriptores de bajo nivel (bordes, texturas y colores locales) hasta representaciones de alto nivel (formas, articulaciones o estructuras completas).

Gracias a esta capacidad, dichas arquitecturas se integran con facilidad en sistemas más sofisticados, actuando como bloques fundamentales de procesamiento y sirviendo como base para tareas que requieren una representación espacial rica y profunda.

Arquitectura	Año	Características
VGG16	2014	16 capas; solo convoluciones 3x3 + *max pooling*
VGG19	2014	Igual que VGG16 pero con 3 capas más (19 capas)
GoogLeNet	2014	Módulos Inception; uso de convoluciones 1x1
Inception-v3/v4	2015-16	Profundidad optimizada; batch norm, factorized convs
ResNet	2015	Residual connections (ResNet-18/34/50/101/152)
DenseNet	2017	Conexiones densas entre capas; mejora gradientes
ResNeXt	2017	Bloques en paralelo (cardinalidad)

Tabla 2. Arquitecturas optimizadas de CNNs

Arquitecturas eficientes para móviles y edge computing

Arquitecturas diseñadas para su uso en dispositivos con recursos limitados (*smartphones*, IoT, drones). Priorizan el bajo consumo, la velocidad de inferencia y la compacidad del modelo, a menudo mediante técnicas como cuantización, *pruning*[9] o *depthwise separable convolutions*[10] (Tabla 3).

- MobileNet (v1, v2, v3, de 2017 a 2019). Utiliza convoluciones separables en profundidad para reducir el número de parámetros y operaciones. MobileNetV2 introduce *linear bottlenecks* y conexiones residuales. La v3 combina AutoML para una arquitectura más optimizada.

- ShuffleNet. Usa convoluciones agrupadas y un mecanismo de *channel shuffle*[11] para mezclar información entre canales y mantener precisión con menor coste computacional.

[8] Columna vertebral, en ocasiones referido al bloque de extracción de características (*feature extraction*) de la arquitectura

[9] Técnica que simplifica o reduce el tamaño de un modelo, generalmente eliminando componentes sin importancia como pesos en redes neuronales o secciones de árboles de decisión.

[10] Técnica que descompone la convolución estándar en dos pasos: convolución en profundidad y convolución puntual lo que reduce el número de parámetros y cálculos.

[11] Técnica para mejorar el flujo de información entre grupos de canales en redes neuronales convolucionales.

- EfficientNet. Utiliza una técnica de búsqueda automatizada de arquitectura (NAS) y un enfoque de escalamiento compuesto para obtener redes más pequeñas y precisas.

- GhostNet. Es una arquitectura eficiente que genera mapas de características utilizando pocas convoluciones estándar y múltiples operaciones lineales simples, lo que reduce significativamente el coste computacional.

Actualmente son ideales como *backbone* en tareas en tiempo real, como estimación de poses en móviles, detección en *edge devices* o visión en robots. Su arquitectura ligera y eficiencia computacional permiten desplegar modelos con baja latencia y alto rendimiento en entornos con recursos limitados.

Arquitectura	Año	Características
MobileNet v1	2017	Depthwise separable convolutions
MobileNet v2	2018	Linear bottlenecks + skip connections
MobileNet v3	2019	AutoML + eficiencia optimizada (por Google)
ShuffleNet	2018	Grouped convs + channel shuffle
EfficientNet	2019	Escalamiento compuesto (depth, width, resolution)
GhostNet	2020	Reducción de computación mediante convoluciones fantasma

Tabla 3. Arquitecturas eficientes para edge computing

Arquitecturas híbridas o de transición

Combinan la eficacia de las CNNs para captar patrones locales con la capacidad de *Transformers*[12] para modelar relaciones globales, introduciendo una nueva generación de arquitecturas en visión artificial (Tabla 4).

- FBNet. Otro enfoque basado en NAS (*Neural Architecture Search*) optimizado para dispositivos móviles.

- RegNet. Familia de arquitecturas generadas automáticamente mediante búsqueda en espacios de diseño. Ofrece una buena relación entre precisión y eficiencia.

- ConvNeXt (2022). Arquitectura tipo CNN rediseñada desde cero siguiendo principios de Transformers, como el uso de normalización *LayerNorm* y *kernels* grandes. Mejora el rendimiento en *benchmarks* sin dejar de ser completamente convolucional.

Actualmente son utilizadas en tareas complejas de visión por computador, como clasificación avanzada, segmentación semántica y estimación de poses en entornos exigentes. Estas arquitecturas combinan elementos tradicionales de las CNNs con mecanismos más

[12] Un tipo de arquitectura de red neuronal diseñada para procesar secuencias de datos

recientes, como bloques de atención, conexiones residuales, o módulos de transformación espacial, permitiendo una representación más rica y adaptativa de las características visuales. Su diseño busca un equilibrio entre eficiencia computacional y capacidad expresiva, lo que las hace especialmente adecuadas para aplicaciones que requieren alta precisión en tiempo real, como vehículos autónomos, realidad aumentada o análisis biométrico en condiciones no controladas.

Arquitectura	Año	Características
FBNet	2019	Optimizado por búsqueda de arquitectura (NAS)
RegNet	2020	Arquitecturas generadas automáticamente
ConvNeXt	2022	CNN moderna inspirada en Vision Transformers

Tabla 4. Arquitecturas híbridas

La mayoría de los modelos de estimación de posturas se basan en alguna de estas arquitecturas de CNNs para extraer características espaciales de las imágenes ya que permiten identificar patrones visuales complejos que facilitan la localización de los keypoints que definen las posturas de las personas en cada imagen.

2.3.2. Cuantización de modelos

La cuantización de modelos es una técnica de optimización que convierte los valores de precisión flotante (float32) usados en los modelos de redes neuronales a representaciones más compactas como int8, uint8 o float16 (15). Esta transformación reduce el tamaño del modelo y mejora su eficiencia computacional, especialmente en dispositivos con recursos limitados como móviles o dispositivos de *edge computing*. Los principales objetivos de la cuantización son:

- Reducir el tamaño del modelo.
- Disminuir el tiempo de inferencia.
- Reducir el consumo energético.
- Facilitar el despliegue en hardware especializado.

Los modelos de estimación de poses generan coordenadas para los puntos clave (keypoints) a partir de mapas de calor o regresiones directas. Estos modelos suelen tener arquitecturas de CNNs pesadas o redes híbridas (CNNs + *Transformer*), lo que los hace candidatos ideales para cuantización en escenarios móviles o en tiempo real ya que permite ejecutar inferencias más rápidas mientras que mantiene una precisión aceptable en coordenadas si se calibra correctamente.

Existen dos formas de realizar la cuantización de un modelo:

- Cuantización posterior al entrenamiento (PTQ, *Post-training Quantization*). Se

aplica a un modelo ya entrenado, convirtiendo sus parámetros a una representación de menor precisión sin necesidad de volver a entrenarlo.

- Cuantización consciente del entrenamiento (QAT, *Quantization-Aware Training*). Se incorpora la conversión de los parámetros durante el proceso de entrenamiento o ajuste fino del modelo, lo que puede mejorar el rendimiento.

2.4. Frameworks para estimación de posturas

Un framework es un entorno de desarrollo que permite construir, entrenar y desplegar, en nuestro caso, modelos de estimación de posturas humanas. Estos frameworks proporcionan las herramientas necesarias para entrenar, evaluar y desplegar modelos de *deep learning* que detectan posiciones articulares del cuerpo humano en imágenes o secuencias de video y ofrecen bibliotecas optimizadas, interfaces modulares y soporte para múltiples formatos de despliegue, facilitando tanto la investigación como la aplicación en tiempo real (Tabla 5) (16):

- **TensorFlow / Keras**, es un framework de código abierto desarrollado por Google que permite implementar y entrenar modelos de aprendizaje profundo. Su integración con TensorFlow Lite lo convierte en una opción ideal para el despliegue en dispositivos móviles, como ocurre con modelos como MoveNet y BlazePose.

- **PyTorch**, desarrollado por Meta AI, este framework es ampliamente usado en investigación debido a su ejecución dinámica (*define-by-run*) y facilidad de depuración. Modelos de alta precisión como HRNet, AlphaPose y RTMPose se entrenan habitualmente en PyTorch.

- **MediaPipe**, es una librería de Google que ofrece soluciones listas para usar en visión por computadora en tiempo real. Integra modelos optimizados en flujos de procesamiento altamente eficientes para tareas como estimación de pose corporal, facial y de manos.

- **OpenCV + DNN**, biblioteca de visión por computadora que incluye un módulo de redes neuronales profundas capaz de cargar modelos en formatos como ONNX y Caffe. Es útil para la inferencia ligera en entornos con restricciones de hardware.

- **MMPose** (OpenMMLab), framework especializado en estimación de posturas basado en PyTorch.

- **Detectron2**, plataforma de visión por computadora de Facebook AI Research, centrada en tareas como segmentación y detección, incluyendo variantes de estimación de postura como DensePose.

Framework	Descripción	Lenguaje principal	Utilización	Fecha introducción (aproximada)
TensorFlow / Keras	Framework de código abierto ampliamente utilizado para deep learning.	Python	PoseNet, MoveNet, BlazePose	2015 (TensorFlow Lite 2017)
PyTorch	Framework muy popular en investigación; permite desarrollo dinámico.	Python	HRNet, RTMPose, AlphaPose	2017
MediaPipe	Librería de Google para visión por computadora en tiempo real.	C++, Python, JS	BlazePose, Holistic	2019
OpenCV + DNN	Librería con soporte para modelos preentrenados.	C++, Python	PoseNet, OpenPose (ONNX/Caffe)	OpenCV: 2000 DNN: 2017
MMDetection / MMPose	Frameworks modulares de OpenMMLab para tareas de visión, incluyendo pose.	Python	RTMPose, HRNet, ViTPose	2020
Detectron2	Plataforma de Facebook AI Research (FAIR)	Python	DensePose	2019

Tabla 5. Frameworks para estimación de posturas humanas

2.4.1. TensorFlow

TensorFlow es una plataforma de código abierto para aprendizaje automático, desarrollada por **Google**, que permite crear y entrenar modelos de redes neuronales y otras aplicaciones de aprendizaje automático. Es utilizado para una amplia variedad de tareas, desde reconocimiento de imágenes y procesamiento de lenguaje natural hasta predicción y modelado estadístico (17). Sus características principales son:

- Código abierto. TensorFlow es gratuito y de código abierto, lo que significa que cualquier persona puede usarlo, modificarlo y distribuirlo.

- Aprendizaje automático. Se enfoca en el desarrollo y entrenamiento de modelos de aprendizaje automático, incluyendo redes neuronales.

- Gráficos de flujo de datos. Utiliza gráficos de flujo de datos para representar las operaciones computacionales, lo que permite una ejecución eficiente y escalable.

- Amplia gama de aplicaciones. Puede ser aplicado en diversos campos, como visión por computadora, procesamiento de lenguaje natural, reconocimiento de voz, y otros.

- Flexibilidad y escalabilidad. Ofrece flexibilidad para construir modelos complejos

y escalarlos para diferentes plataformas, desde dispositivos móviles hasta grandes servidores en la nube.

- Comunidad activa. Cuenta con una gran comunidad de usuarios y desarrolladores que contribuyen con recursos, herramientas y soporte.

Componentes de la plataforma TensorFlow

- TensorFlow Lite, una versión optimizada para dispositivos móviles y sistemas embebidos.

- Keras, una API de alto nivel que facilita la construcción de modelos de redes neuronales.

- TensorFlow.js, permite el desarrollo y despliegue de modelos en navegadores web y entornos Node.js[13].

- TensorFlow Hub, un repositorio de modelos de aprendizaje automático preentrenados que pueden ser reutilizados para acelerar el desarrollo.

Ventajas de TensorFlow

Entre las ventajas de uso de TensorFlow se encuentran su facilidad de uso ya que ofrece APIs intuitivas en diferentes lenguajes, como Python, C++, Java, y Go. También podemos destacar la facilidad y flexibilidad de su despliegue ya que permite desplegar modelos entrenados en diversas plataformas y dispositivos. Además cuenta con una amplia comunidad que brinda soporte y recursos para el desarrollo que lo hace una herramienta poderosa y versátil para el aprendizaje automático con una amplia gama de aplicaciones.

TensorFlow Lite (TFLite)

TensorFlow Lite (TFLite) es una **versión optimizada de TensorFlow** diseñada para ejecutar modelos de aprendizaje automático en dispositivos con recursos limitados. Permite realizar inferencias rápidas y eficientes mediante técnicas como la **cuantización**, reduciendo el tamaño del modelo y el consumo de energía sin comprometer significativamente la precisión. El formato de modelo TFLite se distingue por su diseño compacto, portabilidad y compatibilidad con distintas arquitecturas de hardware.

En Imagen 8 podemos ver la arquitectura de un modelo TensorFlow Lite, con la capa de entrada en amarillo, en azul las capas de los bloques *Feature extraction* y *Prediction heads* (con tres *Fully connected layers*) y por último en verde la capa de salida.

[13] Entorno en tiempo de ejecución multiplataforma, de código abierto, para crear aplicaciones web rápidas y escalables basadas en el lenguaje JavaScript.

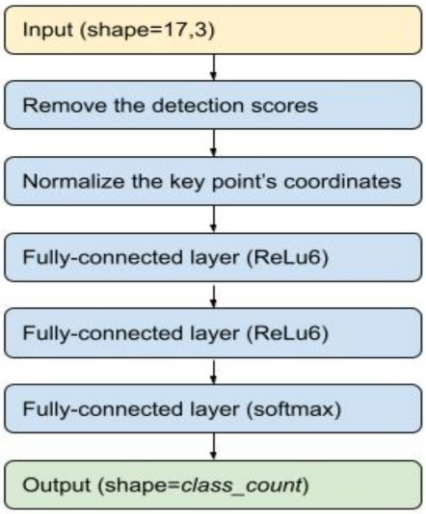


Imagen 8. Esquema del modelo TensorFlow Lite (18)

2.4.2. PyTorch

PyTorch es un marco de código abierto para aprendizaje automático, especialmente enfocado en aprendizaje profundo, desarrollado originalmente por Meta (anteriormente Facebook). Se basa en Python y se utiliza para construir modelos de redes neuronales y realizar cálculos numéricos, incluyendo la ejecución en GPU para acelerar el proceso. PyTorch es popular tanto en investigación como en aplicaciones de producción, incluyendo empresas como Tesla, Microsoft y OpenAI (19). Sus principales características son:

- Framework de código abierto, PyTorch es gratuito y de código abierto, lo que significa que cualquiera puede usarlo, modificarlo y distribuirlo.

- Lenguaje Python, se basa en el lenguaje Python, conocido por su facilidad de uso y amplia adopción en ciencia de datos.

- Cálculo con tensores, PyTorch utiliza tensores para representar datos y realizar cálculos matemáticos, lo que permite operaciones eficientes, especialmente en GPUs.

- Aprendizaje profundo, es una herramienta fundamental en el desarrollo de redes neuronales profundas (un tipo de algoritmo de aprendizaje automático).

- Flexibilidad y rapidez, PyTorch destaca por su flexibilidad para crear prototipos rápidamente y su capacidad de adaptarse a diferentes necesidades de investigación y desarrollo.

- Auto-diferenciación, PyTorch facilita la implementación de gráficos computacionales y cálculos con gradientes, esencial para el entrenamiento de redes

neuronales.

- Desarrollo por Meta AI, aunque fue desarrollado originalmente por Meta, ahora es administrado por la Fundación PyTorch, que asegura su desarrollo continuo y colaboración en la comunidad, donde es una de las herramientas más populares para investigación en aprendizaje profundo y se utiliza en muchos proyectos de IA de producción.

PyTorch Mobile

PyTorch Mobile es la extensión del framework PyTorch diseñada para permitir la ejecución de modelos de aprendizaje profundo en dispositivos móviles y sistemas embebidos. Surge como una respuesta a la creciente necesidad de desplegar modelos de deep learning en entornos con recursos limitados, tales como smartphones, tablets o dispositivos IoT, donde la inferencia debe ser eficiente en cuanto a tiempo de ejecución, consumo energético y memoria.

A nivel arquitectónico, PyTorch Mobile se basa en el uso de TorchScript, un formato intermedio que combina las ventajas de la representación estática de grafos con la flexibilidad del entorno dinámico de PyTorch. TorchScript permite transformar un modelo entrenado en PyTorch estándar a un formato optimizado (.pt) que puede ejecutarse de manera independiente, reduciendo la dependencia de librerías pesadas y facilitando la portabilidad.

El proceso general de despliegue en PyTorch Mobile sigue tres fases técnicas:

1. Conversión del modelo. Se utiliza *tracing* o *scripting* para transformar el modelo PyTorch original en un objeto TorchScript.

2. Optimización. El modelo puede someterse a técnicas de cuantización como PTQ (cuantización posterior al entrenamiento) o QAT (cuantización consciente del entrenamiento), lo que reduce su tamaño y acelera la inferencia, con pérdidas controladas de precisión.

3. Ejecución en dispositivo. El modelo TorchScript se integra en una aplicación Android (Java/Kotlin con JNI) o iOS (Swift/Objective-C) mediante las librerías de PyTorch Mobile, posibilitando la ejecución en CPU, NNAPI (Android) o Metal (iOS).

2.4.3. MediaPipe

MediaPipe es un framework de código abierto y multiplataforma desarrollado por Google para construir y desplegar pipelines de procesamiento multimedia, incluyendo la estimación de la postura humana. Ofrece modelos preentrenados y soporte para múltiples plataformas, lo que lo convierte en una herramienta versátil y potente para aplicaciones en tiempo real (20). Sus principales características son:

- Detección de rostros, manos y poses, MediaPipe proporciona modelos preentrenados para la detección de estos elementos en imágenes y videos.

- Aprendizaje automático en dispositivos de *edge computing*, permite ejecutar

modelos en dispositivos móviles, lo que reduce la latencia y la dependencia de la nube.

- Seguimiento de objetos y reconocimiento de gestos, permite identificar y rastrear objetos en tiempo real y detecta y reconoce diferentes gestos de manos.

- Personalización, permite a los desarrolladores ajustar los modelos predeterminados con sus propios datos utilizando MediaPipe Model Maker.

- Integración con otras herramientas, se puede combinar con otras herramientas como OpenCV para proyectos de visión artificial.

2.4.4. OpenCV + DNN

OpenCV (*Open Source Computer Vision Library*) es una librería de visión por computadora ampliamente utilizada en aplicaciones en tiempo real. Su módulo DNN (*Deep Neural Network*) permite la ejecución de redes neuronales preentrenadas sin necesidad de frameworks externos como TensorFlow o PyTorch. Soporta modelos en formatos como ONNX, Caffe, TensorFlow y Torch, permitiendo ejecutar tareas de estimación de posturas y gracias a su bajo nivel de dependencia y eficiencia computacional, es una opción adecuada para implementaciones ligeras en dispositivos embebidos o en aplicaciones donde se requiere rapidez de inferencia sin entrenamiento (21).

OpenCV + DNN en sí no implementa una arquitectura propia de red neuronal, sino que funciona como un motor de inferencia que carga y ejecuta modelos preentrenados desarrollados en otros frameworks (como TensorFlow o PyTorch)

En estimación de postura, OpenCV + DNN se ha utilizado para desplegar modelos como PoseNet y versiones convertidas de OpenPose, lo que permite detectar keypoints corporales a partir de imágenes o video en tiempo real.

2.4.5. MMPose (OpenMMLab)

MMPose es un framework de código abierto desarrollado por el grupo OpenMMLab para la estimación de posturas humanas 2D y 3D. Está basado en PyTorch y proporciona una infraestructura modular y altamente extensible que facilita el entrenamiento, evaluación y comparación de múltiples arquitecturas. Ofrece soporte a gran variedad de *backbones* como HRNet, ViTPose, ResNet, MobileNet, etc., y cubre tareas *single-person* y *multi-person*.

2.4.6. Detectron2

Detectron2 es un framework de visión por computadora desarrollado por Facebook AI Research (FAIR), diseñado para tareas avanzadas como detección de objetos, segmentación de instancias, segmentación semántica, y estimación de poses humanas. Está implementado en

PyTorch y es la segunda generación del sistema original Detectron, basado en Caffe2[14].

2.4.7. Formatos de modelos de estimación de posturas

En el campo del aprendizaje profundo, la representación y almacenamiento de modelos entrenados varía en función del framework utilizado y del objetivo final del modelo, ya sea continuar el entrenamiento, realizar inferencia eficiente o garantizar interoperabilidad entre plataformas. Esta diversidad ha dado lugar a múltiples formatos de fichero (como .pth, .pt, .pb, .onnx o .tflite), cada uno con características técnicas que responden a distintos requerimientos de uso, portabilidad y rendimiento.

Los formatos propietarios de frameworks, como .pth (PyTorch) son ideales para el entrenamiento y reutilización dentro del mismo entorno, pero presentan limitaciones para el despliegue multiplataforma mientras que los formatos interoperables como ONNX (.onnx) permiten exportar modelos entrenados en distintos frameworks para su ejecución en múltiples entornos de inferencia. ONNX es ampliamente utilizado en producción debido a su eficiencia y portabilidad.

En el contexto de dispositivos móviles o embebidos, formatos como TensorFlow Lite (.tflite) son comunes. Estos ficheros están optimizados para tamaños reducidos y bajo consumo computacional, y suelen incorporar técnicas de cuantización (por ejemplo int8) para acelerar la inferencia sin comprometer significativamente la precisión.

Los formatos más recientes como TorchScript (.pt) ofrecen un equilibrio entre rendimiento y compatibilidad en el ecosistema PyTorch, facilitando tanto el despliegue como la serialización eficiente.

TensorFlow SavedModel (.pb)

TensorFlow SavedModel es el formato estándar de serialización y exportación de modelos en TensorFlow, diseñado para almacenar tanto la arquitectura del modelo como sus pesos y metadatos de forma estructurada y portable. Su principal componente es el archivo **.pb** (*Protocol Buffer*), que representa el grafo computacional del modelo, incluyendo las operaciones, variables y conexiones necesarias para realizar inferencias.

Este formato permite guardar un modelo completo en un único directorio, facilitando su reutilización, despliegue y compatibilidad entre diferentes entornos y versiones de TensorFlow. Junto al archivo .pb, el directorio SavedModel puede contener subdirectorios como variables (para los pesos del modelo) y assets (para recursos auxiliares), lo que garantiza una separación clara entre los distintos elementos del modelo (22).

El formato SavedModel es ampliamente utilizado en aplicaciones de producción, ya que admite inferencias eficientes en servidores, exportación a otras plataformas como TensorFlow Lite (TFLite) o TensorFlow.js, y compatibilidad con APIs de TensorFlow Serving para entornos de despliegue escalables. Gracias a su diseño modular y extensible, el formato .pb también

[14] Caffe2 actualmente está deprecado habiendo sido integrado en PyTorch

facilita tareas como la congelación del grafo, la optimización para hardware específico (TPUs, GPUs), y la integración en flujos de trabajo de machine learning *end-to-end*.

TensorFlow Lite (.tflite)

Los modelos de TensorFlow Lite se almacenan en un archivo binario con extensión **.tflite**, el cual representa una versión serializada y optimizada de un modelo de TensorFlow convencional. Esta serialización utiliza el formato FlatBuffers, una biblioteca de serialización binaria de alto rendimiento que permite la lectura directa de datos sin necesidad de descompresión o análisis complejo, lo que reduce significativamente la latencia en el inicio de la inferencia (23). El archivo .tflite encapsula varios elementos clave:

- Metadatos del modelo. Incluye información básica como nombres de entrada y salida, formas (*shapes*), tipos de datos (por ejemplo, float32, int8, etc.), y posibles etiquetas semánticas para facilitar la integración con bibliotecas de procesamiento de datos o interfaces de usuario.

- Red neuronal codificada. Contiene una representación compacta del grafo computacional, incluyendo las operaciones (kernels) soportadas por TFLite. Estas operaciones han sido previamente convertidas desde el grafo original de TensorFlow mediante el TFLite Converter.

- Pesos y parámetros preentrenados. Los valores numéricos entrenados durante la fase de aprendizaje son empaquetados en el modelo, con posibles técnicas de cuantización para reducir el tamaño del archivo y acelerar su ejecución.

- Soporte para delegados: Aunque el modelo es independiente de la plataforma, TFLite puede emplear "delegados" en tiempo de ejecución para redirigir la ejecución a aceleradores de hardware específicos, como GPU, DSP o unidades de inferencia (TPU).

Una característica fundamental del formato TFLite es su compatibilidad con técnicas de optimización como la cuantización post-entrenamiento y la cuantización durante el entrenamiento, que permiten reducir el tamaño del modelo y el uso de memoria, además de incrementar la velocidad de inferencia. Estas optimizaciones transforman los parámetros y activaciones del modelo de precisión flotante a tipos enteros, como int8 o uint8, manteniendo un impacto mínimo en la precisión del modelo.

El formato .tflite es independiente de la plataforma y puede ejecutarse en diversos entornos mediante el uso del **TensorFlow Lite Interpreter**. Este intérprete está disponible para múltiples sistemas operativos y arquitecturas, incluidos Android, iOS, Linux embebido y microcontroladores (a través de TFLite Micro).

ONNX (.omnx)

ONNX (Open Neural Network Exchange) es un **formato** de especificación abierta diseñado para representar modelos de aprendizaje automático de manera interoperable entre

diferentes frameworks. Fue desarrollado inicialmente por Facebook y Microsoft, y actualmente es mantenido por la comunidad en colaboración con la Linux Foundation y la iniciativa AI Infra. Su propósito principal es facilitar el intercambio y despliegue de modelos en diversos entornos, incluyendo servidores, dispositivos embebidos y plataformas en la nube. El formato ONNX representa un estándar abierto, eficiente y extensible para la representación de modelos de aprendizaje automático (24).

Los modelos ONNX se almacenan en archivos binarios con la extensión **.onnx**, estructurados utilizando el formato de serialización Protocol Buffers (Protobuf), desarrollado por Google. Este formato permite representar estructuras de datos complejas de manera eficiente, lo cual es esencial para modelos de redes neuronales con múltiples capas, pesos y configuraciones. Un archivo de modelo ONNX incluye los siguientes componentes principales:

- Grafo computacional. Representa el flujo de datos a través de la red neuronal. Este grafo está compuesto por nodos, donde cada nodo corresponde a una operación (por ejemplo, convolución, activación, normalización). Cada nodo incluye información sobre sus entradas, salidas y atributos específicos.

- Operadores estándar. ONNX define un conjunto estandarizado de operadores que son independientes del framework original. Esto garantiza que un modelo exportado desde PyTorch, TensorFlow, MXNet u otro entorno, pueda ser interpretado correctamente en cualquier motor de inferencia compatible con ONNX.

- Inicializadores. Contienen los parámetros entrenados del modelo, como pesos y sesgos, empaquetados como tensores dentro del archivo. Estos datos están almacenados directamente en el archivo .onnx, lo que garantiza que el modelo es autosuficiente y portable.

- Metadatos. Incluyen información adicional como el nombre del modelo, la versión del operador, la versión de la especificación ONNX utilizada, y los nombres y formas de las entradas y salidas. Esta información es esencial para la integración en sistemas de producción y para la depuración del modelo.

PyTorch (.pt/.pth)

El formato PyTorch (**.pt o .pth**) es el estándar utilizado por la biblioteca PyTorch para almacenar modelos de aprendizaje profundo entrenados. Este formato permite guardar tanto los pesos del modelo como, opcionalmente, la estructura del modelo (si se utiliza el enfoque de serialización completa). Los archivos .pt y .pth no difieren funcionalmente; su elección suele responder a convenciones del desarrollador (25).

Desde una perspectiva académica, este formato se basa en el módulo `torch.save()`, que emplea el sistema de serialización de Python (*pickle*) para codificar los objetos del modelo. Esto permite conservar de forma eficiente el estado interno de la red neuronal, que incluye los tensores de pesos, sesgos y parámetros de entrenamiento.

Existen dos formas principales de guardar modelos en PyTorch:

- Solo el `state_dict`: es la forma recomendada y más robusta, ya que separa la definición

del modelo del almacenamiento de los pesos. Esto facilita portabilidad y reutilización.

- Serialización completa del modelo: guarda tanto la arquitectura como los pesos, pero puede generar problemas de compatibilidad entre versiones o entornos.

El formato .pt/.pth es ampliamente utilizado en investigación y producción debido a su flexibilidad, compatibilidad con GPU/CPU, y facilidad de integración en flujos de trabajo de inferencia o transferencia de aprendizaje.

2.5. Modelos de estimación de posturas

Este apartado presenta una revisión de los preentrenados más relevantes desarrollados para la estimación de posturas humanas, abarcando tanto enfoques clásicos como modernos. Se incluyen todos los modelos diseñados para estimación 2D y 3D, así como aquellos orientados a dispositivos móviles y entornos de alta complejidad, como escenas con múltiples personas o una única persona. La descripción de cada modelo contiene sus arquitecturas, características técnicas, ventajas, limitaciones, número de keypoints que estiman, etc., con el objetivo de ofrecer un panorama claro y actualizado sobre el estado del arte en esta área de investigación.

2.5.1. OpenPose (2017)

OpenPose es uno de los modelos pioneros y más influyentes en la estimación de posturas humanas. Desarrollado por el Carnegie Mellon Perceptual Computing Lab, introduce una arquitectura *bottom-up* que detecta de manera simultánea los keypoints de múltiples personas en una imagen sin necesidad de una etapa previa de detección individual. Su innovación central son los Part Affinity Fields (PAFs), campos vectoriales que permiten asociar puntos clave entre sí para reconstruir estructuras corporales completas, incluso en entornos con múltiples individuos y oclusiones (26).

OpenPose puede estimar diferentes configuraciones de puntos: 18 puntos (COCO), 25 (BODY-25) y configuraciones extendidas incluyendo manos (21 puntos por mano) y rostro (70 puntos), superando los 135 keypoints en total. Aunque es altamente preciso, OpenPose es computacionalmente intensivo, lo que limita su uso en dispositivos móviles o en tiempo real sin hardware especializado (GPU).

El sistema está implementado principalmente en C++ y es de código abierto, lo que ha facilitado su adopción en investigación, salud, deportes, animación y robótica. Su estructura modular también ha inspirado el desarrollo de variantes más ligeras y eficientes.

2.5.2. AlphaPose (2018)

AlphaPose es un modelo destacado en la estimación de posturas humanas, reconocido por su enfoque *top-down* que primero detecta individuos en la imagen y luego estima sus poses de manera independiente. Propuesto inicialmente en 2018, AlphaPose se caracteriza por su alta precisión y capacidad para manejar múltiples personas en escenarios complejos. Su arquitectura

combina detectores de objetos eficientes con redes neuronales convolucionales para predecir keypoints de manera precisa y robusta. AlphaPose utiliza postprocesamiento para refinar las estimaciones y mejorar la coherencia espacial de las articulaciones (27).

El modelo típicamente estima 17 keypoints principales según el estándar COCO, abarcando las articulaciones principales del cuerpo humano. Aunque su enfoque *top-down* ofrece una precisión superior comparado con métodos *bottom-up*, su costo computacional es mayor, lo que puede limitar su aplicación en tiempo real o dispositivos con recursos limitados. AlphaPose ha sido ampliamente adoptado en aplicaciones de análisis de movimiento, vigilancia, y realidad aumentada, y ha inspirado versiones optimizadas para entornos móviles y de baja latencia.

2.5.3. PoseNet (2018)

PoseNet es un modelo ligero y eficiente para la estimación de posturas humanas en imágenes, diseñado especialmente para su uso en dispositivos móviles y navegadores web. Introducido en 2018 por Google, PoseNet utiliza arquitecturas basadas en MobileNet para realizar la predicción de 17 keypoints en tiempo real con un consumo de recursos reducido. Su enfoque está orientado a la estimación de poses individuales (*single-person*) o múltiples personas (*multi-person*) mediante un diseño flexible y modular (28).

PoseNet destaca por su capacidad de funcionar en tiempo real con hardware limitado, gracias a su compatibilidad con TensorFlow Lite, lo que facilita su integración en aplicaciones móviles y web. Sin embargo, su precisión es inferior comparada con modelos más complejos y pesados, lo que limita su uso en escenarios que requieren alta fidelidad. A pesar de estas limitaciones, PoseNet ha sido fundamental para democratizar el acceso a tecnologías de estimación de postura, facilitando su aplicación en ámbitos de fitness, juegos interactivos y accesibilidad.

2.5.4. DensePose (2018)

DensePose es un modelo avanzado desarrollado por Facebook AI Research en 2018 que va más allá de la estimación clásica de posturas humanas 2D, mapeando cada píxel del cuerpo humano visible en una imagen a una superficie 3D paramétrica del cuerpo. A diferencia de otros modelos que estiman únicamente un conjunto discreto de keypoints, DensePose realiza una segmentación densa y una correspondencia directa con un modelo 3D anatómico, permitiendo reconstrucciones detalladas de la forma y la postura humana (29).

Esta capacidad ofrece un nivel de detalle muy superior, ideal para aplicaciones en realidad aumentada, animación digital y análisis biomédico. Sin embargo, DensePose requiere una gran potencia computacional y no es adecuado para ejecución en dispositivos móviles o en tiempo real. Además, su entrenamiento y despliegue son más complejos debido a la necesidad de datos anotados en 3D. DensePose representa un importante avance en la representación morfológica del cuerpo humano en visión por computador, ampliando las fronteras entre visión 2D y reconstrucción 3D.

2.5.5. HRNet (2019)

HRNet (High-Resolution Network) es un modelo diseñado para tareas de visión por computadora que requieren una preservación precisa de detalles espaciales, como la estimación de postura humana. A diferencia de muchas arquitecturas convencionales que reducen progresivamente la resolución de las características a lo largo de la red, HRNet mantiene representaciones de alta resolución durante todo el proceso de inferencia. Para lograr esto, introduce un enfoque de procesamiento paralelo mediante múltiples ramas que operan a diferentes resoluciones y se comunican continuamente entre sí mediante fusión de información, permitiendo una integración efectiva de contextos locales y globales. Esta arquitectura mejora significativamente la precisión en la localización de puntos clave del cuerpo humano, incluso en condiciones de oclusión o poses complejas. HRNet ha demostrado resultados de vanguardia en *benchmarks* como COCO y MPII, siendo ampliamente adoptado en aplicaciones de análisis de movimiento, interfaces hombre-máquina y medicina deportiva. Su diseño innovador establece un nuevo paradigma en el equilibrio entre precisión espacial y capacidad semántica en redes profundas.

2.5.6. EfficientPose (2020)

Basado en la arquitectura EfficientNet, este modelo optimiza el balance entre velocidad y rendimiento, permitiendo una detección robusta de las articulaciones humanas en tiempo real. EfficientPose incorpora una técnica de aprendizaje multitarea que mejora la precisión en la detección de múltiples personas y reduce errores en escenarios con oclusiones, lo que lo vuelve más robusto frente a condiciones del mundo real comparado con modelos tradicionales. Su diseño modular facilita la integración en sistemas de visión por computadora, aplicaciones de realidad aumentada y análisis de movimientos deportivos. También destaca por su capacidad para funcionar en dispositivos con recursos limitados, sin sacrificar la calidad de la estimación. Este enfoque representa un avance significativo en el campo de la visión artificial aplicada a la interacción humano-computadora y el análisis biomecánico.

2.5.7. MoveNet (2020)

MoveNet es un modelo de estimación de posturas *bottom-up* que utiliza mapas de calor (*heatmaps*) para localizar con precisión los puntos clave del cuerpo humano. Su arquitectura se compone de dos partes principales: un *feature extractor* (un componente de una red neuronal que extrae características de la imagen) y un conjunto de *prediction heads* (un componente de una red neuronal que transforma características aprendidas en predicciones específicas normalmente situado al final de la red). Todos los modelos se entrenan utilizando la API de detección de objetos de TensorFlow (30).

La arquitectura de la CNN que utiliza MoveNet es MobileNetV2 junto con una red de pirámide de características (FPN), lo que permite generar mapas de características de alta resolución con gran riqueza semántica. El extractor se conecta con cuatro prediction heads, cada una encargada de estimar lo siguiente (Imagen 9):

- Mapa de calor del centro de la persona, predice el centro geométrico de cada instancia de persona.

- Campo de regresión de puntos clave, predice el conjunto completo de puntos clave por persona, útil para agruparlos en instancias individuales.

- Mapa de calor de puntos clave, predice la ubicación de todos los puntos clave, independientemente de a qué persona pertenecen.

- Campo de desplazamiento 2D por punto clave, predice el desplazamiento local desde cada píxel del mapa de características hasta la ubicación precisa (subpíxel) de cada punto clave.

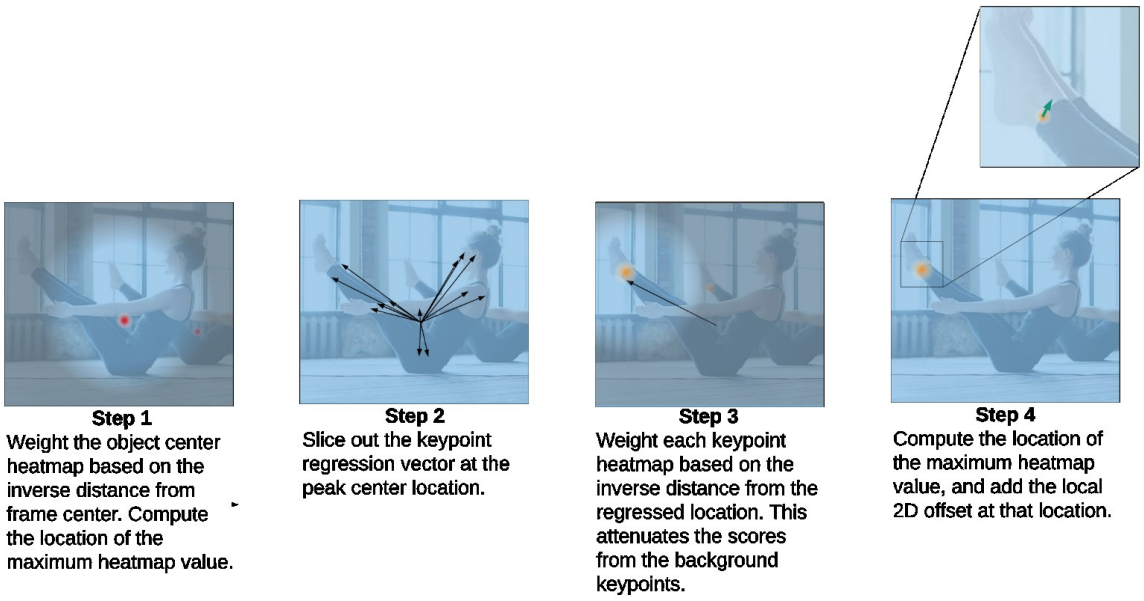


Imagen 9. Pasos de la inferencia del modelo MoveNet (30)

MoveNet tiene dos variantes: MoveNet Lightning y MoveNet Thunder.

MoveNet Lightning es un modelo de estimación de posturas humanas desarrollado por Google en 2020, diseñado para ofrecer una solución extremadamente rápida y eficiente en dispositivos con recursos limitados, como teléfonos móviles y sistemas embebidos. Forma parte de la familia MoveNet y está optimizado para lograr baja latencia manteniendo una precisión competitiva en la predicción de 17 keypoints principales, siguiendo el estándar COCO.

MoveNet Lightning emplea una arquitectura ligera basada en convoluciones eficientes y técnicas de optimización (Imagen 10) que permiten su ejecución en tiempo real incluso en CPUs de gama baja. Aunque sacrifica algo de precisión en comparación con su contraparte más robusta, MoveNet Thunder, su velocidad y tamaño reducido lo hacen ideal para aplicaciones en tiempo real que requieren un balance entre rendimiento y eficiencia, como fitness, juegos interactivos y realidad aumentada. Además, su compatibilidad con TensorFlow Lite facilita su integración en aplicaciones móviles y de *edge computing*.

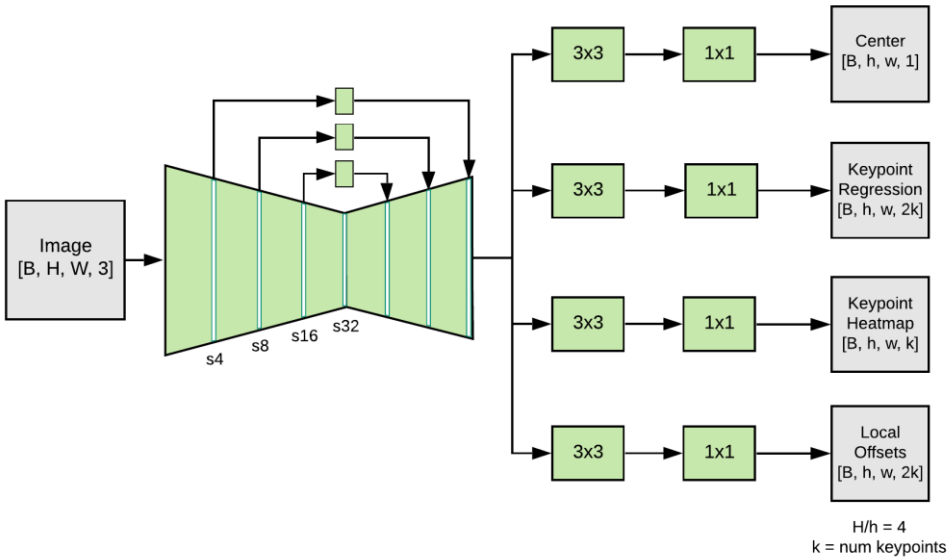


Imagen 10. Esquema de arquitectura modelo MoveNet (30)

MoveNet Thunder es la otra variante del modelo MoveNet lanzada por Google en 2020, diseñada para ofrecer una mayor precisión en la estimación de posturas humanas a costa de un mayor consumo computacional en comparación con MoveNet Lightning. Este modelo predice 17 keypoints clave siguiendo el estándar COCO, y está optimizado para ejecutarse en tiempo real en dispositivos con mayor capacidad de procesamiento, como GPUs móviles o CPUs de alto rendimiento.

MoveNet Thunder emplea una arquitectura más profunda y compleja que incorpora convoluciones eficientes y técnicas avanzadas de aprendizaje profundo para mejorar la exactitud y robustez frente a variaciones de pose, oclusiones y condiciones de iluminación adversas. Su diseño equilibra la necesidad de precisión con la latencia, siendo adecuado para aplicaciones que requieren un análisis detallado del movimiento humano, como rehabilitación, deportes y análisis biomecánico. Asimismo, es compatible con TensorFlow Lite, facilitando su despliegue en entornos móviles y *edge computing* con hardware más potente.

2.5.8. BlazePose (2021)

BlazePose es un modelo de estimación de posturas humanas desarrollado por Google (31), diseñado para ofrecer un seguimiento preciso y en tiempo real de la postura corporal en dispositivos móviles. Su arquitectura ligera permite la inferencia en dispositivos con recursos limitados, alcanzando más de 30 fotogramas por segundo en un Google Pixel 2.

BlazePose estima 33 puntos clave del cuerpo humano, incluyendo cabeza, tronco, extremidades y manos, proporcionando una representación detallada de la postura humana. Su diseño modular consta de dos componentes principales: un detector que identifica la región del cuerpo en la imagen y un estimador que predice las coordenadas de los puntos clave. El estimador utiliza una combinación de mapas de calor y regresión directa para mejorar la

precisión y eficiencia.

Este modelo ha sido ampliamente utilizado en aplicaciones de seguimiento de actividad física, control gestual y realidad aumentada, gracias a su capacidad para operar en tiempo real sin la necesidad de hardware especializado. Además, su implementación en MediaPipe facilita su integración en diversas plataformas y dispositivos.

2.5.9. PoseWarper (2021)

PoseWarper es un modelo avanzado de estimación de postura humana en video que introduce un enfoque novedoso para explotar la información temporal mediante el alineamiento espacial entre fotogramas consecutivos. A diferencia de métodos tradicionales que procesan cada frame de manera independiente, PoseWarper utiliza una arquitectura basada en "*warping*" o deformación de características, lo que permite transferir información clave desde frames anteriores al frame actual. Esta técnica mejora la coherencia temporal y la precisión en la detección de articulaciones, especialmente en casos de oclusiones, movimientos rápidos o poses poco convencionales. El modelo emplea una red de extracción de características que aprende a alinear mapas de calor de keypoints a través del tiempo, reduciendo errores comunes en secuencias de video. PoseWarper se basa en una arquitectura eficiente y modular, lo que facilita su integración en sistemas de análisis de movimiento en tiempo real. Su enfoque temporal representa un avance significativo frente a modelos estáticos, logrando mejoras sustanciales en *benchmarks* como PoseTrack y subrayando la importancia de la dinámica del movimiento en la estimación de postura humana.

2.5.10. YOLO-Pose (2021)

YOLO-Pose es un modelo de estimación de posturas humanas basado en la arquitectura YOLO (You Only Look Once), conocido por su capacidad para realizar detección de objetos en tiempo real con alta precisión. Adaptado para la tarea de estimación de posturas, YOLO-Pose integra una *prediction head* especializada para predecir keypoints corporales directamente junto con la detección de personas en una sola pasada, optimizando la eficiencia y velocidad de procesamiento (32).

Este enfoque *single-shot* permite la estimación simultánea de múltiples poses en escenas con varias personas, generalmente prediciendo 17 keypoints conforme al estándar COCO. YOLO-Pose destaca por su equilibrio entre precisión y rendimiento, haciéndolo adecuado para aplicaciones en tiempo real como vigilancia, análisis deportivo y realidad aumentada. Además, su diseño modular facilita la implementación en dispositivos con recursos limitados y su compatibilidad con frameworks como PyTorch y TensorFlow amplía su aplicabilidad en entornos móviles y *edge computing*.

YOLO-pose tiene diferentes versiones publicadas hasta la fecha como se puede ver en la Tabla 6.

Versión YOLO	Variantes	Tamaño aprox.	Características
YOLOv5-Pose	YOLOv5s-Pose	~14 MB	Versión pequeña, rápida
	YOLOv5m-Pose	~41 MB	Balance entre tamaño y precisión
	YOLOv5l-Pose	~88 MB	Mayor precisión
	YOLOv5x-Pose	~168 MB	Versión extra grande
YOLOv7-Pose	YOLOv7-tiny-Pose	~14 MB	Versión ligera para edge
	YOLOv7-Pose (full)	~70-90 MB	Versión completa
YOLOv8-Pose	YOLOv8n-Pose (nano)	~6 MB	Muy ligero
	YOLOv8s-Pose (small)	~22 MB	Ligero
	YOLOv8m-Pose (medium)	~50 MB	Balance
	YOLOv8l-Pose (large)	~87 MB	Alta precisión
	YOLOv8x-Pose (x-large)	~136 MB	Muy alta precisión
YOLOv11-Pose	YOLOv11n-Pose (nano)	~6 MB	Muy ligero
	YOLOv11s-Pose (small)	~19 MB	Ligero
	YOLOv11m-Pose (medium)	~40 MB	Balance
	YOLOv11l-Pose (large)	~60 MB	Alta precisión
	YOLOv11x-Pose (x-large)	~120 MB	Muy alta precisión

Tabla 6. Versiones de modelos YOLO de estimación de posturas humanas

2.5.11. RTMPose (2023)

RTMPose (*Real-Time Multi-Person Pose Estimation*) es un modelo de estimación de postura humana diseñado específicamente para lograr alto rendimiento en tareas en tiempo real, sin comprometer la precisión. Desarrollado con un enfoque modular y eficiente, RTMPose utiliza técnicas modernas como el *backbone* RTMDet basado en la arquitectura ConvNeXt y estrategias de optimización ligeras para acelerar la inferencia, siendo especialmente adecuado para aplicaciones en dispositivos con recursos limitados. A diferencia de modelos tradicionales, RTMPose emplea una representación directa de keypoints y una arquitectura centrada en la eficiencia computacional, eliminando componentes costosos como el procesamiento de mapas de calor. Además, introduce un esquema de entrenamiento robusto, basado en técnicas como SimDR (*Simple Disentangled Representation*), que mejora la estabilidad y la precisión de la predicción de coordenadas. Este modelo es altamente competitivo en *benchmarks* como COCO y CrowdPose, destacando por su capacidad para manejar múltiples personas, oclusiones y variabilidad en las poses. Su diseño versátil lo hace ideal para aplicaciones en visión artificial, realidad aumentada, deportes y vigilancia inteligente.

2.5.12. Resumen

Además de las características descritas existen otros parámetros importantes a la hora de seleccionar un modelo muy dependiente el entorno donde se vaya a utilizar y son su tamaño y su formato (ver Tabla 7 para tamaños aproximados y formatos).

El **tamaño** de los modelos es tan variable como versiones y/o familias del modelo se hayan desarrollado, como ya vimos en el apartado "*2.3.2 Cuantización de modelos*" existen técnicas para reducir el tamaño de un modelo sin que la precisión del mismo se vea

excesivamente comprometida. Las versiones sin "cuantizar" suelen usar precisión de 32 bits en punto flotante (float32), lo cual produce modelos grandes pero precisos. La cuantización a menor precisión (como int8 o float16), utilizada en modelos como MoveNet o BlazePose, reduce significativamente el tamaño del modelo, además de mejorar la velocidad de inferencia y disminuir el consumo de memoria.

La arquitectura base (*backbone*) empleada en un modelo tiene un impacto sustancial en su tamaño. Modelos como HRNet, OpenPose o AlphaPose (basados en la arquitectura ResNet) utilizan backbones pesados y profundos (por ejemplo, ResNet-101 o HRNet-W48), diseñados para preservar información espacial a lo largo de toda la red, lo cual incrementa tanto el número de parámetros como el tamaño total del modelo. En cambio, arquitecturas ligeras como MobileNet, utilizadas por modelos como PoseNet o MoveNet Lightning, están específicamente optimizadas para reducir la complejidad computacional, lo que resulta en tamaños significativamente menores.

Además de la arquitectura, los modelos suelen ofrecer múltiples variantes (por ejemplo versiones tiny, small, medium, large, nano, etc.), cada una con diferentes profundidades y anchos de red. Estas variantes permiten al usuario seleccionar un punto de equilibrio entre precisión, latencia y tamaño de almacenamiento, lo cual es crucial en aplicaciones para dispositivos de *edge computing* (Tabla 8).

Modelo	**Fecha**	**Formato de Fichero**	**Tamaño aproximado**
OpenPose	2017	.caffemodel / .onnx	~100 MB
AlphaPose	2018	.pth (PyTorch)	~92 MB
PoseNet	2018	.tflite / .json / .pb	~5 MB
DensePose	2018	.pkl / .pth	~13.8 MB
HRNet	2019	.pth	~112 MB
EfficientPose	2020	.h5 / .onnx	<10 MB
MoveNet	2020	.tflite	5-20 MB
BlazePose	2021	.tflite / .pb	3-26 MB
PoseWarper	2021	.pth	N/D
YOLO-Pose	2021	.pt (→.onnx, →.tflite)	6-200 MB
RTMPose	2023	.onnx / .pth	18-65 MB

Tabla 7. Resumen características modelos preentrenados

Modelo	**Entrada**	***Feature Extraction***	***Prediction Heads***	**Salida (estimación poses)**
OpenPose	Imagen RGB	VGG-19 o variantes personalizadas	Heatmaps de keypoints + mapas de afinidad (PAFs)	18-135 keypoints (cuerpo/rostro/manos)
AlphaPose	Imagen RGB	ResNet (usualmente ResNet-50 o ResNet-101)	Heatmaps de keypoints + regresión de offsets	17-136 keypoints
PoseNet	Imagen RGB	MobileNet o ResNet	Heatmaps de keypoints	17 keypoints
DensePose	Imagen RGB	ResNet-101	Mapas de UV (superficie corporal) + segmentación	Cuerpo completo en malla UV (no keypoints estándar)
HRNet	Imagen RGB	High-Resolution Network (múltiples ramas paralelas)	Heatmaps de keypoints	17 (COCO) / 16 (MPII) keypoints
EfficientPose	Imagen RGB	EfficientNet (B0-B4)	Heatmaps de keypoints + offsets (opcional)	17 keypoints
MoveNet	Imagen RGB	CNN propietaria optimizada para móvil	Regresión directa de coordenadas keypoints	17 keypoints
BlazePose	Imagen RGB (ROI del cuerpo)	MobileNetV2 ligera o CNN personalizada	Regresión directa de keypoints 3D	33 keypoints
PoseWarper	Secuencia de imágenes	Hourglass o ResNet	Heatmaps de keypoints + módulo de warping	17 keypoints
YOLO-Pose	Imagen RGB	CSPDarknet (YOLO backbone)	Regresión directa de bounding boxes + keypoints	17 keypoints
RTMPose	Imagen RGB	MobileNetV3 o HRNet-lite	Regresión directa o heatmaps simplificados	17 keypoints

Tabla 8. Resumen arquitecturas modelos preentrenados

2.6. Datasets de estimación de posturas

El desarrollo de algoritmos robustos de estimación de posturas ha dependido críticamente de la disponibilidad de **datasets públicos bien anotados**, que sirven como base tanto para el entrenamiento como para la evaluación de modelos supervisados.

Los datasets de estimación de postura pueden clasificarse según varias dimensiones técnicas: tipo de anotación (2D o 3D), número de personas por imagen (*single-person* o *multi-person*), tipo de sensor (RGB, RGB-D, multivista), y contexto (interior, exterior, sintético o realista). Entre los más influyentes se encuentran COCO Keypoints y MPII Human Pose, cada uno con diferentes coberturas de poses, diversidad de sujetos, condiciones de iluminación, y esquemas de anotación.

Técnicamente, un dataset de postura humana incluye no solo las imágenes, sino también las **coordenadas** (en píxeles o en 3D) de los puntos anatómicos relevantes (como hombros, codos, rodillas, tobillos, etc.), frecuentemente junto con etiquetas de visibilidad o confiabilidad. La calidad, cantidad y diversidad de estos datos tienen un impacto directo sobre la capacidad de generalización de los modelos entrenados, especialmente en escenarios desafiantes como la oclusión, las poses poco frecuentes o las variaciones culturales.

Para la realización de este estudio se han analizado los dos datasets más utilizados y relevantes en el ámbito de la visión por computador, COCO (1) y MPII (2).

2.6.1. Dataset COCO (Common Objects in COntext)

El dataset COCO (Common Objects in COntext) es un conjunto de datos ampliamente utilizado en la investigación y desarrollo de modelos de visión por computador. Fue introducido por Microsoft en 2014 con el objetivo de proporcionar un recurso estandarizado para el entrenamiento y la evaluación de algoritmos en tareas complejas como detección de objetos, segmentación semántica, segmentación de instancias, detección de poses humanas y *captioning* de imágenes. Su diseño se centra en ofrecer imágenes realistas con objetos en contextos naturales, lo que lo diferencia de conjuntos anteriores con escenarios más simplificados o sintéticos.

Es uno de los datasets más reconocidos desde su aparición, y, desde 2015, la COCO Challenge[15] ha sido un catalizador permanente de nuevos *state-of-the-art* en tareas clave como detección, segmentación y estimación de posturas. La primera edición se celebró en 2015 y el ganador fue el modelo Faster R-CNN (33) que es considerado un avance crucial en detección de objetos porque introduce el *Region Proposal Network* (RPN), un sub-módulo entrenable dentro de la red que aprende a generar propuestas de regiones directamente desde los *feature maps*[16] de la CNN que hace que el tiempo de inferencia pase de segundos por imagen a solo 0.2 segundos aproximadamente, haciendo posible la detección casi en tiempo real.

---

[15] COCO Challenge es una competencia anual que mide los algoritmos más avanzados en detección, segmentación, poses y captioning, usando el dataset COCO como referencia.

[16] Son las estructuras de datos específicas dentro de las capas de extracción de características (*Feature extraction*) que contienen estas características aprendidas de una manera espacialmente organizada

También ha sido designado como un estándar para la comunidad de detección de objetos en papers como "*Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks*" (34) donde se cita textualmente:

> "*Like ImageNet in its time, MS-COCO has become the de facto standard for the object detection community and any method winning the state-of-the-art on it is assured to gain much traction and visibility.*"

COCO está compuesto por más de 330.000 imágenes, de las cuales más de 200.000 cuentan con anotaciones detalladas que abarcan más de 1,5 millones de instancias de objetos. Estos objetos pertenecen a 80 categorías comunes que incluyen personas, animales, vehículos, muebles, utensilios cotidianos, entre otros, lo que permite abordar tareas de detección y segmentación de objetos en contextos muy variados y realistas.

En el ámbito de la estimación de poses humanas, COCO ofrece anotaciones precisas de keypoints corporales para más de 250.000 personas. Estas anotaciones incluyen posiciones de articulaciones clave como hombros, codos, muñecas, caderas, rodillas y tobillos, proporcionando una base sólida para entrenar y evaluar modelos de estimación de postura en 2D bajo condiciones complejas, con variaciones de iluminación, oclusión, perspectiva y diversidad de posturas.

El dataset se organiza en varias particiones para facilitar el desarrollo y la evaluación de modelos:

- **train2017**: conjunto de entrenamiento con aproximadamente 123.000 imágenes anotadas, utilizadas para ajustar los parámetros de los modelos.

- **val2017**: conjunto de validación con unas 5.000 imágenes, destinado a ajustar hiperparámetros y realizar pruebas preliminares de desempeño.

- **test-dev2017** y **test-challenge2017**: conjuntos de prueba sin etiquetas visibles públicamente, diseñados para evaluaciones de *benchmark* oficiales, donde los resultados se comparan de manera objetiva entre diferentes algoritmos.

Puntos clave o keypoints

Son coordenadas específicas que en estimación de poses humanas se corresponden con una articulación o región anatómica relevante como hombros, codos, rodillas o tobillos (Imagen 11) y su detección precisa permite reconstruir la estructura y postura del cuerpo. Estos puntos se utilizan como entidades de referencia para tareas de análisis de movimiento, biometría, interacción hombre-máquina y seguimiento visual, y suelen ir acompañados de indicadores de visibilidad o confianza que cuantifican la certeza del modelo en su localización.

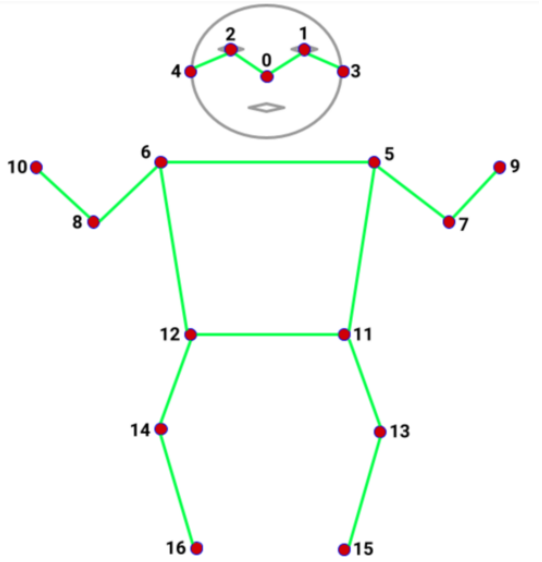


Imagen 11. Representación de las anotaciones de dataset COCO por persona

El dataset COCO incluye un subconjunto específicamente diseñado para la tarea de estimación de posturas humanas en 2D, que constituye uno de los estándares de referencia más utilizados en visión por computador. En este subconjunto, cada instancia de persona está anotada mediante un conjunto fijo de **17 keypoints corporales** (Tabla 9), definidos para capturar la estructura esquelética humana de manera coherente, reproducible y adecuada para diferentes escenarios de aplicación.

Estos keypoints corresponden a las principales articulaciones y regiones anatómicas del cuerpo humano: nariz, ojos, orejas, hombros, codos, muñecas, caderas, rodillas y tobillos. La disposición de estas anotaciones permite representar de forma aproximada la cinemática del cuerpo y posibilita la construcción de esqueletos simplificados que pueden ser empleados en tareas de análisis de movimiento, interacción humano-computadora, biometría o deportes.

Un aspecto clave de COCO es que estas **anotaciones** están recogidas en **condiciones no controladas**, es decir, en escenas naturales y cotidianas con variaciones significativas en iluminación, poses, oclusiones parciales, ángulos de visión y escalas de representación. Esta diversidad dota al dataset de un alto nivel de complejidad y realismo, lo que lo convierte en un recurso fundamental para evaluar la robustez de los modelos de estimación de postura en contextos desafiantes.

Cada uno de los keypoints se anotan con coordenadas (x, y) y con una etiqueta de visibilidad con los valores 0 (no visible), 1 (marcado pero no visible), 2 (visible).

Estas anotaciones permiten no solo la localización precisa de cada keypoint, sino también la evaluación estructurada de modelos bajo métricas ampliamente adoptadas que veremos más adelante como *Average Precision* (AP), *Object Keypoint Similarity* (OKS) y *Average Recall* (AR) por lo que además de su uso como *benchmark*, COCO se ha consolidado como estándar de facto para la comparación entre arquitecturas de visión.

Nº de keypoint	Nombre del keypoint	Descripción anatómica
1	Nose	Punta de la nariz
2	Left Eye	Centro del ojo izquierdo
3	Right Eye	Centro del ojo derecho
4	Left Ear	Parte visible de la oreja izquierda
5	Right Ear	Parte visible de la oreja derecha
6	Left Shoulder	Articulación del hombro izquierdo
7	Right Shoulder	Articulación del hombro derecho
8	Left Elbow	Articulación del codo izquierdo
9	Right Elbow	Articulación del codo derecho
10	Left Wrist	Articulación de la muñeca izquierda
11	Right Wrist	Articulación de la muñeca derecha
12	Left Hip	Articulación de la cadera izquierda
13	Right Hip	Articulación de la cadera derecha
14	Left Knee	Articulación de la rodilla izquierda
15	Right Knee	Articulación de la rodilla derecha
16	Left Ankle	Articulación del tobillo izquierdo
17	Right Ankle	Articulación del tobillo derecho

Tabla 9. Listado de keypoints de dataset COCO

Anotaciones de COCO para la estimación de posturas humanas

Para las anotaciones el dataset utiliza un esquema jerárquico en formato JSON siguiendo una especificación propia conocida como COCO JSON Format que permite no solo entrenar modelos supervisados de estimación de pose 2D sino también evaluar modelos bajo métricas como AP, OKS y AR, generar esqueletos y visualizar poses humanas en entornos reales y complejos, etc.. Cada anotación de persona contiene los siguientes campos:

- `image_id`: ID de la imagen donde se encuentra la persona.
- `category_id`: Siempre 1 para personas.
- `keypoints`: Lista de 51 valores (17 keypoints × 3 valores por keypoint).
  Cada keypoint contiene: (x, y, v):
    - x, y: coordenadas del punto en píxeles.
    - v: visibilidad (0=no etiquetado, 1=etiquetado pero no visible, 2=etiquetado y visible).
- `num_keypoints`: número de puntos anotados con v > 0.
- `bbox`: coordenadas [x, y, width, height] de la caja que rodea a la persona.
- `area`: área de la caja (útil para normalizar el error en métricas como OKS).
- `iscrowd`: si la anotación pertenece a un grupo denso de personas (0 o 1).

- segmentation: polígonos que segmentan la silueta del cuerpo (opcional).

Validación de resultados con COCO

El cálculo de resultados en la API de COCO para estimación de poses humanas se basa en una evaluación que mide la precisión y exhaustividad de la localización de puntos clave (keypoints) en imágenes. Técnicamente, el proceso sigue estas etapas fundamentales:

1. Entrada de predicciones. El modelo genera un conjunto de predicciones para cada persona detectada en la imagen, donde cada predicción contiene coordenadas (x, y) para 17 keypoints predefinidos (hombros, codos, etc.) y una puntuación de confianza.

2. Correspondencia con anotaciones (*ground truth*). Cada predicción se asocia con una anotación real mediante la métrica de similitud de puntos clave (OKS). El OKS evalúa la proximidad espacial entre los keypoints predichos y anotados, normalizada por la escala del objeto y ponderada por la visibilidad de cada punto.

3. Asignación de verdaderos positivos y falsos positivos:

    - Para distintos umbrales de OKS (desde 0.50 a 0.95 en incrementos de 0.05), la API asigna cada predicción a una anotación única si el OKS excede el umbral, clasificándola como verdadero positivo (TP).

    - Predicciones sin correspondencia o con OKS bajo el umbral se consideran falsos positivos (FP).

    - Las anotaciones no detectadas cuentan como falsos negativos (FN).

4. Construcción de la curva *Precision-Recall*. Para cada umbral, se calcula la precisión y el *recall* acumulados ordenando las predicciones según su puntuación de confianza. Esto permite trazar la curva de precisión en función del *recall*.

5. Cálculo del Average Precision (AP). La métrica AP se obtiene integrando el área bajo la curva *Precision-Recall* interpolada en 101 puntos de *recall*, proporcionando una medida robusta y estable del rendimiento del modelo.

6. Agregación multi-umbral y global. Finalmente, la API calcula el *mean Average Precision (mAP)* promediando los AP obtenidos en los diferentes umbrales de OKS, reflejando la capacidad del modelo para localizar con precisión keypoints en diversos grados de tolerancia espacial. **En los resultados de la API COCO se considera como la métrica principal (*mAP)* al valor de AP [IoU=0.50:0.95]**.

Este método garantiza una evaluación precisa, que tiene en cuenta la variabilidad en la visibilidad y escala de las personas, y promueve la comparación justa y estandarizada entre diferentes modelos de estimación de poses humanas.

2.6.2. MPII (Max Planck Institute for Informatics)

El dataset MPII Human Pose es otra referencia fundamental en el campo de la estimación de poses humanas en imágenes. Desarrollado por el Max Planck Institute for Informatics (MPII) en Alemania (centro de investigación líder en visión por computador y aprendizaje automático) para capturar la diversidad y complejidad de posturas humanas en contextos cotidianos. MPII contiene aproximadamente 25.000 imágenes extraídas de videos reales, abarcando una amplia variedad de actividades y situaciones. Cada persona en estas imágenes está anotada con 16 puntos clave (keypoints) que representan las principales articulaciones y partes del cuerpo, como cabeza, hombros, codos y rodillas, proporcionando una representación detallada de la configuración corporal en 2D (2).

Además de las coordenadas anatómicas, MPII incluye información contextual sobre la actividad realizada por la persona, lo que enriquece su utilidad para tareas que combinan estimación de pose y reconocimiento de acciones. Este dataset se ha convertido en un estándar para la evaluación de algoritmos de estimación de pose gracias a la calidad y precisión de sus anotaciones, así como a la diversidad de su contenido.

Utilizado en numerosos estudios y trabajos importantes como por ejemplo "*Compositional Human Pose Regression*" (35) que introduce un enfoque de regresión estructurada para estimación de postura que permite modelar dependencias espaciales entre articulaciones, base para muchos métodos posteriores, o "*P-CNN: Pose-based CNN Features for Action Recognition*" (36) que introduce la combinación de pose + CNN para reconocimiento de acciones y donde MPII se usa como *benchmark* para evaluar la precisión de keypoints humanos contribuyendo a popularizar el uso de poses humanas como característica para tareas adicionales de visión.

Puntos clave o keypoints

Los 16 keypoints de MPII (Imagen 12) incluyen posiciones en 2D correspondientes a cabeza, cuello, hombros, codos, muñecas, caderas, rodillas y tobillos, proporcionando una cobertura detallada de las principales articulaciones para la reconstrucción precisa de la postura humana. Las anotaciones son realizadas manualmente sobre imágenes provenientes de videos cotidianos con gran diversidad de posturas, actividades y condiciones visuales, lo que permite modelar un amplio rango de configuraciones corporales.

A diferencia de otros datasets como COCO, MPII pone énfasis en **posturas complejas y dinámicas en actividades específicas**, ofreciendo además metadatos con información contextual sobre la actividad realizada, lo que posibilita un análisis más rico y aplicaciones avanzadas en reconocimiento de acciones.

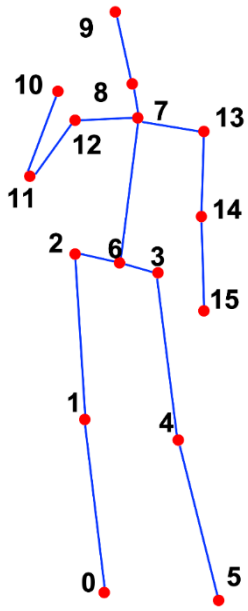


Imagen 12. Representación de las anotaciones de dataset MPII por persona

MPII al igual que COCO incluye información especializada para la tarea de estimación de posturas humanas en 2D, donde cada instancia de persona está anotada con un conjunto fijo en este caso de **16 keypoints** corporales por los 17 que tiene COCO. Estas anotaciones son similares a las de COCO, aunque no iguales, pero también están diseñadas para capturar la estructura esquelética humana y se encuentran distribuidas en las principales articulaciones y regiones del cuerpo humano: cabeza, cuello, pelvis, torso, hombros, codos, muñecas, caderas, rodillas y tobillos.

Nº de keypoint	**Nombre del keypoint**	**Descripción anatómica**
0	Right Ankle	Articulación del tobillo derecho
1	Right Knee	Articulación de la rodilla derecha
2	Right Hip	Articulación de la cadera derecha
3	Left Hip	Articulación de la cadera izquierda
4	Left Knee	Articulación de la rodilla izquierda
5	Left Ankle	Articulación del tobillo izquierdo
6	Pelvis	Centro de la pelvis / región lumbar
7	Thorax	Centro superior del torso
8	Neck	Base del cuello
9	Head	Parte superior de la cabeza
10	Right Wrist	Articulación de la muñeca derecha
11	Right Elbow	Articulación del codo derecho
12	Right Shoulder	Articulación del hombro derecho
13	Left Shoulder	Articulación del hombro izquierdo
14	Left Elbow	Articulación del codo izquierdo
15	Left Wrist	Articulación de la muñeca izquierda

Tabla 10. Listado de keypoints de dataset MPII

Anotaciones de MPII para la estimación de posturas humanas

Las anotaciones de keypoints en el dataset MPII Human Pose se almacenan principalmente en archivos MATLAB, que contienen estructuras de datos detalladas para cada imagen y persona anotada. Cada entrada incluye coordenadas 2D (x, y) de 16 keypoints anatómicos específicos, numerados y definidos dentro de cada anotación de cada persona, que contiene los siguientes campos:

- .annolist(imgidx): anotaciones para la imagen *imgidx*
    - .image.name: nombre del fichero de la imagen
    - .annorect(ridx): anotaciones corporales de la persona *ridx*
        - .x1, .y1, .x2, .y2: coordenadas del rectángulo de la cabeza
        - .scale: escale de la persona
        - .objpos: posición humana en la imagen
        - .annopoints.point: anotaciones de los keypoint
            - .x, .y: coordenadas del punto
            - id: identificador del punto (Tabla 10)
            - is_visible: visibilidad del punto
    - .vidx: índice en el video *video_list*
    - .frame_sec: posición de la imagen en el video en segundos
- img_train(imgidx): asignación de la imagen a training/testing
- single_person(imgidx): rectángulo con identificador *ridx*
- act(imgidx): etiqueta de actividad/categoría para la imagen *imgidx*
    - act_name: nombre de actividad
    - cat_name: nombre de categoría
    - act_id: identificador de la actividad
- video_list(videoidx): identificador del video de YouTube. Para visualizarlo ir a https://www.youtube.com/watch?v=video_list(videoidx)

Medidas de precisión de dataset MPII

Las medidas de precisión empleadas en el dataset MPII Human Pose para la evaluación de modelos de estimación de postura humana están diseñadas para cuantificar la exactitud en la localización de los keypoints en imágenes 2D. La métrica principal es el PCKh (Percentage of Correct Keypoints, head-normalized), que calcula el porcentaje de keypoints correctamente detectados dentro de un umbral de distancia relativo al tamaño de la cabeza del sujeto.

Matemáticamente, un keypoint se considera correctamente estimado si la distancia euclidiana entre la predicción y la anotación *ground truth* es menor que un umbral $\boldsymbol{\alpha}$ × **head size**, donde el parámetro $\boldsymbol{\alpha}$ suele establecerse en **0.5**. Este criterio de normalización mediante el tamaño de la cabeza permite adaptar la evaluación a diferentes escalas y tamaños corporales, ofreciendo una comparación justa y robusta entre individuos y escenarios variados.

Además del PCKh, se utiliza el **PCK** (*Percentage of Correct Keypoints*) en otras variantes, que emplea umbrales absolutos o relativos a otras dimensiones corporales para casos específicos o comparaciones con otros datasets. La métrica PCKh se reporta tanto para cada articulación individual como en forma agregada, proporcionando un análisis detallado de las fortalezas y limitaciones del modelo en distintas regiones corporales.

El dataset MPII también incluye evaluaciones con curvas PCK, que representan la precisión en función del umbral de distancia, y métricas complementarias como el error medio euclidiano, para una comprensión más fina del desempeño.

Validación de resultados con MPII

El proceso de validación de resultados con el dataset MPII Human Pose se realiza mediante la evaluación cuantitativa de las predicciones del modelo sobre un conjunto de imágenes de prueba etiquetadas con anotaciones *ground truth* de keypoints. Este procedimiento sigue los siguientes pasos técnicos y académicos:

1. Preparación de datos. Se utilizan las imágenes de test con sus correspondientes anotaciones de 16 keypoints y sus estados de visibilidad. Estas anotaciones actúan como referencia para comparar las predicciones del modelo.

2. Predicción de keypoints. El modelo genera estimaciones de las posiciones 2D de los keypoints para cada persona en las imágenes de test. Las predicciones deben estar en el mismo sistema de coordenadas y escala que las anotaciones *ground truth*.

3. Normalización y umbral. Para evaluar la precisión, se normalizan las distancias entre los keypoints predichos y los anotados usando la dimensión de la cabeza (*head size*), que es un indicador del tamaño relativo del sujeto. Se define un umbral, comúnmente el 50% del tamaño de la cabeza (PCKh@0.5), para determinar si un keypoint está correctamente localizado.

4. Cálculo de métricas. Se calcula el porcentaje de keypoints detectados correctamente (PCKh) y se reporta para cada articulación y globalmente. También se pueden analizar curvas PCK que muestran la precisión en función de distintos umbrales, así como errores promedio.

5. Tratamiento de visibilidad. Los keypoints marcados como no visibles o fuera de imagen en las anotaciones *ground truth* se excluyen de la evaluación para evitar penalizar al modelo por detectar puntos imposibles de observar.

2.7. Métricas de precisión

Las métricas de precisión constituyen herramientas cuantitativas diseñadas para evaluar de manera objetiva el rendimiento de modelos en tareas como detección de objetos, clasificación y estimación de poses. Su función es medir, bajo diferentes perspectivas, el grado

de concordancia entre las predicciones del modelo y el *ground truth*[17], permitiendo comparaciones entre arquitecturas, configuraciones y datasets.

Técnicamente, estas métricas se apoyan en la teoría de detección de señales y en el análisis de TP (True Positives), FP (False Positives), TN (True Negatives) y FN (False Negatives), adaptando su formulación a la naturaleza de cada tarea:

- En clasificación, la métrica base es la precisión (*accuracy*), que mide la proporción de predicciones correctas sobre el total, complementada con métricas como *recall* y matrices de confusión para abordar problemas de clases desbalanceadas.

- En detección de objetos, se utilizan métricas basadas en el solapamiento geométrico, como el IoU (*Intersection over Union*), y medidas agregadas como el AP (*Average Precision*) y mAP (*mean Average Precision*), evaluando el rendimiento bajo múltiples umbrales de coincidencia para capturar tanto la capacidad de localizar como de clasificar correctamente.

- En estimación de poses, las métricas deben adaptarse a datos estructurados de puntos clave. Aquí, el OKS (*Object Keypoint Similarity*) sustituye al IoU, ya que considera distancias euclidianas normalizadas, escala del objeto y visibilidad de keypoints. Sobre esta base se calculan métricas como AP/AR de COCO keypoints, que miden simultáneamente exhaustividad y precisión a distintos niveles de tolerancia.

La selección y análisis de estas métricas no solo determina la interpretación del rendimiento de un modelo, sino que también condiciona el desarrollo de arquitecturas y técnicas de entrenamiento, ya que optimizar para una métrica específica puede producir sesgos hacia ciertos aspectos de la tarea (por ejemplo alta precisión pero bajo *recall*, o viceversa).

2.7.1. Recall

Recall (exhaustividad o sensibilidad) es la métrica que mide la capacidad de un modelo para encontrar todas las instancias relevantes de la clase objetivo dentro de un conjunto de datos. Un *recall* alto indica que el modelo detecta la mayoría de los objetos reales (o keypoints), aunque no necesariamente con alta precisión (37).

Desde un punto de vista matemático, *recall* se define como vemos en la Ecuación 1 cantidad de casos correctamente acertados dividido entre casos correctamente acertados + casos relevantes no acertados.

[17] Conjunto de datos o anotaciones de referencia verificadas manualmente contra la cual se comparan y evalúan las predicciones de un modelo.

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

donde:

- **TP** (*True Positives*): casos relevantes correctamente detectados.
- **FN** (*False Negatives*): casos relevantes que el modelo no detectó.

Ecuación 1. Cálculo de Recall (37)

2.7.2. Intersection over Union (IoU)

La IoU es una métrica ampliamente utilizada en visión por computador para cuantificar la superposición entre dos regiones, una región predicha y su correspondiente *ground truth* (valores de referencia) (Imagen 13). Es un indicador clave en tareas de detección de objetos, segmentación semántica, instancia y estimación de poses (cuando se evalúan *bounding boxes*[18]) (38).

El cálculo de IoU produce un valor normalizado entre 0 y 1. Un valor cercano a 1 indica una predicción con una alta coincidencia espacial respecto a la anotación de referencia, mientras que valores bajos reflejan discrepancias significativas en localización, escala o forma. En la práctica, la IoU se utiliza con umbrales definidos (por ejemplo, IoU ≥ 0.5) para determinar si una predicción se considera un acierto o un fallo. La variación de dichos umbrales da lugar a métricas más expresivas, como la AP en intervalos múltiples (AP@[0.5:0.95]), ampliamente adoptada en *benchmarks* como COCO.

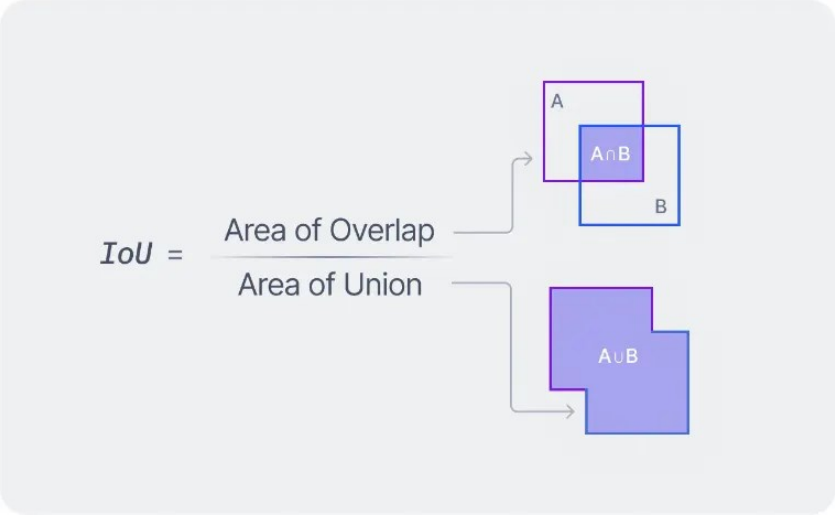


Imagen 13. Representación IoU (38)

[18] Una caja delimitadora (*bounding box* en inglés) es un rectángulo que se utiliza para delimitar la posición y el tamaño de un objeto dentro de una imagen o un fotograma de vídeo.

Desde un punto de vista matemático, el IoU se define como el cociente entre el área de la intersección de las dos regiones y el área de su unión como se puede ver en la Ecuación 2.

$$\text{IoU} = \frac{|B_p \cap B_{gt}|}{|B_p \cup B_{gt}|}$$

donde:

- B_p es el bounding box (o máscara) predicho por el modelo.
- B_{gt} es el bounding box (o máscara) de referencia.
- $|\cdot|$ denota el área.

Ecuación 2. Cálculo de IoU (Intersection Over Union) (38)

2.7.3. Estimación de puntos clave: Object Keypoint Similarity (OKS)

IoU es la métrica de referencia para medir la precisión de la detección de objetos pero al calcularse utilizando las áreas de las regiones predichas y real no puede ser aplicada cuanto estamos trabajando con detección de puntos. En la estimación de puntos clave la métrica homóloga a IoU es el OKS (*Object Keypoint Similarity*) (39), que es una **métrica de evaluación** utilizada para cuantificar la similitud entre los keypoints predichos por un modelo de estimación de pose humana y los keypoints de referencia (*ground truth*) en un contexto de objetos con estructura articulada como personas. Esta métrica fue introducida por el equipo de COCO como una generalización del IoU, adaptada a la naturaleza puntual y estructural de los esqueletos humanos.

OKS se define como una función de penalización basada en la distancia euclidiana entre cada par de keypoints (el predicho y el real), normalizada por la escala del objeto y ponderada por un factor de visibilidad anatómica. Su expresión matemática se puede ver en la Ecuación 3.

$$\text{OKS} = \frac{\sum_i \left[\exp\left(-\frac{d_i^2}{2s^2k_i^2}\right) \cdot \delta(v_i > 0)\right]}{\sum_i \delta(v_i > 0)}$$

donde:

- d_i: distancia euclidiana entre el keypoint predicho y el real para la articulación i.
- s: escala del objeto (normalmente el área de la persona anotada).
- k_i: constante que controla la tolerancia al error para el keypoint i; define la sensibilidad anatómica (algunos puntos permiten mayor error).
- v_i: visibilidad del keypoint i (0 = no etiquetado, 1 = etiquetado no visible, 2 = visible).
- $\delta(v_i > 0)$: indica si ese punto fue etiquetado.

Ecuación 3. Cálculo de OKS de COCO (39)

El OKS toma valores en el rango [0,1], donde 1 indica una coincidencia perfecta entre la predicción y la anotación. A diferencia del IoU, el OKS es robusto a errores de localización relativa gracias a la normalización por escala y sensibilidad. Es la métrica oficial para los *benchmarks* de COCO Keypoints y se utiliza para calcular métricas como:

- **mAP (media de AP en múltiplos de OKS de 0.50 a 0.95).**
- AP@OKS=0.75 (estricto).
- AP@OKS=0.50 (tolerante).

2.7.4. Medidas de precisión de dataset COCO

Para evaluar la precisión de los modelos de estimación de postura humana en el dataset COCO se utilizan las métricas definidas por el COCO Keypoint Evaluation API, que siguen los criterios de detección de objetos adaptados al contexto de keypoints. Las métricas principales son:

- **AP. Average Precision (precisión promedio)**. Es una métrica integral que cuantifica el rendimiento de un modelo promediando la precisión a lo largo de distintos niveles de exhaustividad (*recall*) (40). En lugar de evaluar la precisión en un único punto, COCO integra el área bajo la curva *Precision-Recall*, lo que proporciona una medida más estable y representativa del comportamiento global del sistema (Ecuación 4).

  COCO define AP como la media de la precisión calculada para un conjunto discreto de niveles de *recall*, típicamente 101 puntos equidistantes en el intervalo [0,1]. En la tarea de detección de objetos y estimación de poses, la API no calcula AP para un único umbral de coincidencia, sino que la promedia sobre múltiples umbrales de IoU (en detección) u OKS (en pose), con pasos de 0.05, cubriendo desde 0.50 hasta 0.95.

$$AP_{c,t} = \frac{1}{N_r} \sum_{r \in R} P_{interp}(r)$$

donde:

- R es el conjunto de niveles de *recall* evaluados (101 en COCO),
- $P_{interp}(r)$ es la precisión interpolada para el *recall* r,
- N_r es el número total de niveles de *recall* considerados.

Ecuación 4. Cálculo Average Precision en COCO (40)

- **AR. Average Recall (recuperación media)**. La recuperación media AR evalúa la capacidad del modelo para detectar todas las personas y sus keypoints relevantes, es decir, cuántos casos relevantes logra capturar correctamente, sin importar tanto la confianza del score. En contextos de estimación de postura, es importante para medir

si un modelo no deja sin detectar personas o keypoints, especialmente en escenas complejas o con múltiples sujetos. Una AR elevado indica que el modelo es capaz de recuperar la mayoría de las poses humanas relevantes, aunque algunas estimaciones no sean perfectas en todos los puntos.

2.8. Utilización en dispositivos móviles y consideraciones técnicas

La integración de modelos de estimación de posturas humanas en dispositivos móviles constituye un área de investigación y desarrollo de creciente relevancia debido a la necesidad de soluciones portables, en tiempo real y con bajo consumo de recursos computacionales. Estos modelos permiten identificar y localizar puntos clave en el cuerpo humano directamente desde la cámara del dispositivo, habilitando aplicaciones en ámbitos como la salud digital, el deporte, la rehabilitación, la interacción hombre-máquina, el entretenimiento y la realidad aumentada.

2.8.1. Hardware

La ejecución de modelos de estimación de posturas en dispositivos móviles puede abordarse desde diferentes estrategias teóricas, las cuales dependen de la disponibilidad de recursos de hardware, del ecosistema de software del dispositivo y de los requisitos de precisión y latencia de la aplicación. Estas estrategias pueden implementarse mediante frameworks optimizados para entornos móviles como TensorFlow Lite, PyTorch Mobile, CoreML, NNAPI (Android Neural Networks API) o OpenVINO para dispositivos *edge*, que proporcionan las abstracciones necesarias para ejecutar el mismo modelo en diferentes backends (CPU, GPU, NPU). La selección del método depende de los criterios de diseño del sistema, el tipo de aplicación y las limitaciones impuestas por el hardware del dispositivo. De manera general, se pueden distinguir cuatro enfoques principales.

Ejecución directa en CPU

Los modelos pueden ejecutarse en la unidad central de procesamiento (CPU) del dispositivo, sin requerir hardware especializado. Este enfoque **maximiza la portabilidad y compatibilidad**, ya que prácticamente todos los dispositivos móviles disponen de CPU. Sin embargo, su principal limitación radica en la baja velocidad de inferencia en comparación con otros métodos, lo que restringe su uso a modelos altamente optimizados o aplicaciones con requisitos de latencia poco estrictos.

Aceleración mediante GPU móvil

La unidad de procesamiento gráfico (GPU) integrada en los dispositivos móviles puede aprovecharse para la ejecución de estos modelos, especialmente aquellos basados en operaciones de convolución intensiva. Frameworks como TensorFlow Lite GPU Delegate o Metal Performance Shaders (en iOS) permiten explotar el paralelismo masivo de la GPU, incrementando la velocidad de inferencia de manera significativa.

En la práctica, este enfoque es posible gracias a las GPUs integradas en los SoCs modernos que equipan a la mayoría de teléfonos y tablets actuales. Entre ellas se incluyen

arquitecturas Mali (ARM), Adreno (Qualcomm), PowerVR (Imagination Technologies) y las GPUs diseñadas por Apple, todas con soporte para APIs gráficas como OpenGL ES 3.x, Vulkan o Metal, según la plataforma. Estas capacidades permiten ejecutar cargas de trabajo de inferencia en paralelo, aunque con diferencias en rendimiento y eficiencia energética según la generación y el nivel del hardware. No obstante, el consumo energético y la variabilidad de soporte entre diferentes dispositivos constituyen limitaciones importantes.

Uso de aceleradores especializados (NPUs, DSPs, TPUs móviles)

Una tendencia creciente en la computación móvil es la incorporación de unidades de procesamiento neuronal (NPUs) o *Digital Signal Processors* (DSPs) especializados en la ejecución de cargas de trabajo de inteligencia artificial. Fabricantes como Qualcomm (Hexagon DSP), Huawei (Ascend NPU) o Google (Edge TPU en dispositivos Pixel) integran este tipo de hardware. El uso de estas unidades permite una ejecución altamente eficiente en términos energéticos, con latencias muy bajas y optimización para inferencia en tiempo real, lo que los convierte en la opción más adecuada para aplicaciones de estimación de posturas en entornos móviles.

Ejecución híbrida con soporte en la nube

Una alternativa teórica consiste en combinar la inferencia en el dispositivo con el procesamiento en la nube. En este escenario, el dispositivo móvil ejecuta una primera etapa de procesamiento (ej. detección de personas en la escena) y delega la parte más costosa del modelo a servidores remotos. Esto reduce los requisitos de hardware en el móvil y posibilita el uso de modelos de gran escala, aunque introduce problemas de latencia, dependencia de conectividad y privacidad de los datos.

2.8.2. Android

La ejecución de modelos de estimación de posturas humanas en dispositivos móviles depende en gran medida del framework de inferencia utilizado (TensorFlow Lite, PyTorch Mobile, entre otros), así como del soporte de hardware (CPU, GPU, NPU/TPU) disponible en el dispositivo. No obstante es posible establecer un rango de requisitos mínimos en cuanto a versiones de Android que aseguren la compatibilidad.

- TensorFlow Lite (TFLite) requiere como mínimo Android 4.1 (API 16, Jelly Bean), ya que está implementado sobre el Android NDK y puede ejecutarse en arquitecturas ARMv7 y superiores. Sin embargo, a partir de Android 8.0 (API 26, Oreo) se introdujeron mejoras en la API de aceleración de hardware (NNAPI, Neural Networks API), lo que permite una ejecución mucho más eficiente en procesadores modernos.

- PyTorch Mobile requiere Android 5.0 (API 21, Lollipop) como versión mínima para la ejecución básica en CPU. Sin embargo, el soporte para aceleración mediante GPU (Vulkan, OpenGL) y optimizaciones recientes está pensado para dispositivos con Android 8.1 (API 27) o superior.

- ONNX Runtime Mobile en su configuración estándar admite dispositivos con Android 5.0 (API 21) en adelante, aunque al igual que PyTorch y TensorFlow Lite, el rendimiento real depende del acceso a bibliotecas de cómputo optimizado como NNAPI o Core ML (en iOS).

Por tanto aunque la ejecución básica de estos frameworks es posible en Android 5.0 (API 21) en adelante, se considera que la versión mínima recomendada para la ejecución de modelos de estimación de posturas humanas en condiciones prácticas y eficientes es Android 8.0 (API 26). A partir de esta versión, los dispositivos incluyen soporte maduro para NNAPI, drivers más optimizados para GPU y librerías de aceleración de hardware que resultan esenciales en modelos de visión por computadora de alta carga computacional, como los de estimación de posturas.

Según el análisis de Wikipedia de datos de Statcounter Global Stats[19] la cuota de mercado de las versiones de Android más utilizadas hasta abril del 2025 son:

1. Android 14.0 - 33.44 %
2. Android 13.0 - 16.94 %
3. Android 12.0 - 12.11 %
4. Android 11.0 - 10.41 %
5. Android 15.0 - 10.06 %
6. Android 10.0 - 5.57 %
7. Android 9.0 Pie - 3.18 %
8. Android 8.0 Oreo - 2.18 %
9. Android 5.0 Lollipop - 1.74 %
10. Otros - 4.37 %

Por lo que sumando los porcentajes se estima que a hasta abril de 2025 un **93,89%** de los dispositivos Android en el mercado tienen una versión igual o superior a la 8.0 y podrían ejecutar este estudio.

[19] https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

3. METODOLOGÍA

El objetivo general de este apartado es definir una metodología que permita seleccionar un conjunto de modelos y un dataset de testeo para evaluar y comparar su precisión y rendimiento computacional en la detección de posturas humanas utilizando varios dispositivos móviles, proporcionando criterios objetivos para la selección de los modelos adecuados para el estudio.

En este contexto **se ha elegido Android** como plataforma base para la implementación y validación por criterios tanto técnicos como de aplicabilidad práctica, Android constituye el sistema operativo móvil más ampliamente utilizado a nivel global, con una cuota de mercado superior al 70 %, lo que garantiza la relevancia y transferibilidad de los resultados a un gran espectro de dispositivos y escenarios de uso.

Por otro lado entre los diversos frameworks disponibles, **TensorFlow Lite** se presenta como una opción preferente para implementaciones en dispositivos Android, por varias razones, primero porque mantiene la compatibilidad con modelos previamente entrenados en TensorFlow permitiendo la conversión a un formato optimizado para móviles sin perder la arquitectura ni la precisión del modelo original, y también porque la infraestructura de TensorFlow Lite incluye soporte nativo para Android mediante APIs estables y documentadas, lo que simplifica la integración de modelos de visión por computadora en aplicaciones móviles. Esto reduce la complejidad del desarrollo y permite focalizar los recursos en el diseño experimental y la evaluación de métricas

3.1. Selección de modelos

Como hemos visto en el apartado “2.5. Modelos de estimación de posturas” se dispone de un amplio espectro de familias de modelos desarrolladas por la comunidad académica e industrial, muchas de las cuales presentan similitudes estructurales (por ejemplo, en la arquitectura de red empleada, el número y disposición de keypoints que predicen o la naturaleza del backbone de extracción de características), mientras que otras difieren sustancialmente en su enfoque metodológico, el tipo de inferencia (*top-down*, *bottom-up*, híbrido), o el formato de salida.

Un aspecto común a la mayoría de estas familias es la existencia de múltiples **versiones derivadas del modelo base**, diseñadas para satisfacer diferentes compromisos entre precisión, latencia y uso de recursos. Dichas variantes suelen generarse mediante técnicas de optimización y compresión de modelos, como la cuantización (descrita en el apartado “2.3.2. Cuantización de modelos”),

Debido a la gran cantidad de modelos existentes, las diferencias entre ellos y a las versiones o familias derivadas de cada uno resultaría inviable incluirlos todos en un mismo estudio por lo que para este estudio comparativo se pretende seleccionar únicamente **2 o 3 modelos principales** e incluir todas las familias o variantes de los mismos que sean posibles para poder analizar cómo se comportan los diferentes modelos seleccionados no solo en comparación de otros modelos sino también en comparación con otras versiones de su propia familia, lo que puede constituir un estudio bastante extenso.

3.1.1. Criterios de selección y modelos seleccionados

En este apartado trataremos de justificar los criterios en base a los cuales seleccionaremos los modelos para el estudio. Se valoraran diferentes aspectos que consideramos fundamentales:

- **Disponibilidad y accesibilidad** como modelo preentrenado. Este factor resulta determinante para garantizar la reproducibilidad del estudio y la viabilidad técnica de su implementación en la plataforma objetivo. Deberá existir un modelo accesible para su descarga y posterior integración en la plataforma de estudio, en cualquier formato que pueda ser integrado en un framework para dispositivos móviles con sistema operativo Android.

- **Estimación de posturas humanas, *single-person*, 2D y keypoints consistentes y comunes**[20]. Otro criterio clave en la selección de modelos será la capacidad del modelo para predecir una única persona en dos dimensiones y con un conjunto de keypoints consistente y común entre las diferentes arquitecturas evaluadas. Este aspecto es fundamental para garantizar la comparabilidad directa de los resultados, ya que la precisión y las métricas de evaluación solo son válidas si se calculan sobre puntos anatómicos equivalentes en todas las predicciones.

- **Madurez/obsolescencia del modelo**. La madurez temporal de un modelo constituye un indicador clave para valorar su estabilidad tecnológica, grado de adopción y nivel de validación por parte de la comunidad científica e industrial. En el presente estudio, se establece como criterio que los modelos seleccionados tengan **al menos un año de disponibilidad pública y no superen los cinco años desde su lanzamiento oficial**. El requisito de un mínimo de un año responde a la necesidad de garantizar que el modelo ha pasado por un ciclo razonable de uso, validación y retroalimentación por parte de desarrolladores e investigadores y el límite de cinco años desde su lanzamiento busca evitar la selección de modelos que, si bien pudieron ser punteros en su momento, pueden estar tecnológicamente superados por nuevas arquitecturas o por versiones más eficientes y precisas.

- **Tamaño del modelo**. Cuando el despliegue se orienta a dispositivos con recursos limitados como teléfonos móviles, tabletas o sistemas embebidos (*edge devices*) el tamaño del modelo tiene que ser adecuado a este entorno. En este estudio, se establece un **umbral recomendado de aproximadamente 50 MB** para el fichero de pesos del modelo dejando fuera del estudio modelos que aunque presuponen mejor precisión podrían no ser adecuados para dispositivos con pocos recursos por su gran tamaño (tamaño de almacenamiento de la APP en la que se integrara, tamaño en memoria RAM, etc.,).

- **Facilidad de integración** (existencia de APIs, documentación, demos). Si bien este último punto no es determinante es un criterio a tener en cuenta en la selección, ya que determina el esfuerzo técnico y los recursos necesarios para integrar el modelo en un entorno funcional.

---

[20] Recordemos que el objetivo del estudio descrito en el *Abstract* es la recomendación de modelos para una futura APP de tele rehabilitación, lo que reduce el ámbito del estudio a estimación *single-person* y 2D.

Los resultados obtenidos tras aplicar los criterios establecidos en esta investigación indican que, en una primera instancia, los modelos que se perfilan como más adecuados para ser incluidos en el estudio comparativo como se puede ver en la Tabla 11 son **MoveNet**, **BlazePose** y **YOLO-Pose**. Estos tres modelos representan una muestra representativa y diversificada de arquitecturas contemporáneas y robustas, específicamente diseñadas o adaptadas para la estimación de posturas humanas en entornos móviles.

Modelo	Seleccionable según criterios	Razones
OpenPose	No	- Año 2017 - Tamaño < 100MB - Número de keypoints no estandarizado (18-135)
AlphaPose	No	- Año 2018 - Tamaño < 92MB
PoseNet	No	- Año 2018 - Superado por MoveNet[21]
DensePose	No	- Año 2018 - No enfocado en keypoints 2d estándar
HRNet	No	- Año 2019 - Tamaño < 112MB
EfficientPose	No	- Facilidad de integración (dificultad para encontrar documentación técnica)
MoveNet	Sí	- Año2020 - Disponibilidad y accesibilidad - Estimación single-person, 2D y 17 keypoints - Tamaño 3-25 MB
BlazePose	Sí	- Año2021 - Disponibilidad y accesibilidad - Estimación single-person, 2D y 33 keypoints (17 comunes) - Tamaño 3-26 MB
PoseWarper	No	- Disponibilidad y accesibilidad (dificultad para encontrar modelo preentrenado) - Facilidad de integración (dificultad para encontrar documentación técnica)
YOLO-Pose	Sí	- Año2021-2024 (según versión) - Disponibilidad y accesibilidad - Estimación single-person, 2D y 17 keypoints - Tamaño 5-50 MB
RTMPose	No	- Disponibilidad y accesibilidad - Facilidad de integración (dificultad para encontrar documentación técnica) - Basado en arquitectura YOLO

Tabla 11. Resumen aplicación criterios selección de modelos

[21] Ambos modelos desarrollados por Google (aunque por diferentes equipos)

Los tres modelos seleccionados (MoveNet, BlazePose y YOLO-Pose) son representativos y están consolidados para la estimación de posturas humanas. Cada uno ofrece un enfoque arquitectónico distinto y características técnicas que permiten abordar la tarea con diferentes balances entre precisión, velocidad y eficiencia computacional.

3.1.2. MoveNet (Google)

MoveNet se basa en una arquitectura ligera y eficiente de red neuronal convolucional propia diseñada para inferencias rápidas en dispositivos con recursos limitados. Su diseño utiliza convoluciones profundas optimizadas para mantener un alto rendimiento en tiempo real. MoveNet prioriza un procesamiento rápido y un tamaño reducido del modelo, lo que lo hace ideal para aplicaciones móviles y en tiempo real.

Está orientado a la detección de 17 puntos clave estándar del cuerpo humano siguiendo la convención del dataset COCO (nariz, ojos, orejas, hombros, codos, muñecas, caderas, rodillas y tobillos), donde cada keypoint estimado incluye:

- Coordenadas (x, y) normalizadas al tamaño de entrada.
- Score de confianza asociado.

Como ya vimos en la descripción previa en el apartado “2.5.7. MoveNet” cuenta con dos variantes principales cada una de ellas a su vez con varias versiones cuantizadas:

MoveNet Lightning

- Tamaño de la imagen de entrada: 192×192 píxeles.
- Optimizado para latencia mínima y procesamiento en tiempo real en dispositivos móviles.
- Arquitectura reducida y más agresivamente cuantizada.
- Menor precisión que Thunder, pero tiempos de inferencia muy bajos.

MoveNet Thunder

- Tamaño de la imagen de entrada: 256×256 píxeles.
- Optimizado para máxima precisión manteniendo latencia aceptable.
- Arquitectura más profunda con mayor capacidad de extracción de características.
- Mayor tamaño de entrada y mayor coste computacional.

Tanto la variante Lightning como la variante Thunder tienen 3 versiones, la estándar con representación en 32 bits, una versión reducida mediante cuantización a 16 bits y otra aún más reducida de 8 bits. Con el objetivo no solo de comparar precisiones y rendimientos de modelos entre si sino de poder observar los efectos de técnicas como la cuantización sobre los mismos modelos **en este estudio se incluirán todas las versiones de ambas variantes de la familia MoveNet**, con un total de 6 modelos: MoveNet Lightning 8, 16 y 32 y MoveNet Thunder 8, 16 y 32.

3.1.3. BlazePose (Google MediaPipe)

BlazePose, desarrollado por Google MediaPipe, emplea una arquitectura de red convolucional compacta (MobileNetV2). A diferencia de otros modelos, BlazePose estima un conjunto más amplio de 33 keypoints, (incluyendo articulaciones detalladas y puntos faciales) pero a su vez manteniendo la compatibilidad con los puntos más estandarizados para la estimación de posturas humanas (17 de los 33 son comunes) coincidiendo con el resto de modelos seleccionados. Esta capacidad lo hace especialmente adecuado para aplicaciones de *fitness*, realidad aumentada y entornos donde se requiera un seguimiento detallado y preciso.

Aunque los modelos de la familia BlazePose trabajan con salidas de 33 keypoints estimados mantienen una compatibilidad con los 17 puntos estándar que suelen estimar la mayoría de los modelos al **coincidir 17 de los 33** puntos estimados por BlazePose con estos 17 puntos como vemos en la Tabla 12. BlazePose hace una estimación de keypoints extendida (estima más puntos) pero compatible con el resto de modelos a nivel de estudio comparativo.

Nº keypoint estándar	Nº keypoint BlazePose	Descripción anatómica
0	0	Nariz
1	2	Ojo izquierdo
2	5	Ojo derecho
3	7	Oreja izquierda
4	8	Oreja derecha
5	11	Hombro izquierdo
6	12	Hombro derecho
7	13	Codo izquierdo
8	14	Codo derecho
9	15	Muñeca izquierda
10	16	Muñeca derecha
11	23	Cadera izquierda
12	24	Cadera derecha
13	25	Rodilla izquierda
14	26	Rodilla derecha
15	27	Tobillo izquierdo
16	28	Tobillo derecho

Tabla 12. Equivalencia puntos BlazePose

BlazePose presenta tres versiones con distinto grado de cuantización: Lite, Full y Heavy todas ellas con el mismo tamaño de la imagen de entrada (256×256 píxeles). **En este estudio se incluirán las tres versiones de familia BlazePose.**

3.1.4. YOLOv8-Pose (Ultralytics)

Ultralytics es la compañía que desarrolla y mantiene el ecosistema de software del mismo nombre especializado en visión por computadora y aprendizaje profundo, ampliamente reconocida por ser la responsable del desarrollo y mantenimiento de los modelos YOLO (You Only Look Once).

YOLO-Pose adapta la reconocida arquitectura de detección rápida YOLO para la estimación de poses, integrando *prediction heads* especializadas para la regresión de keypoints después de la detección de personas. Esta aproximación de una sola etapa permite realizar detecciones y estimaciones simultáneas, logrando un equilibrio eficiente entre rapidez y precisión. YOLO-Pose destaca por su capacidad para manejar múltiples personas en imágenes con alta velocidad, manteniendo una precisión competitiva, ideal para entornos móviles y aplicaciones en tiempo real.

Como hemos visto en el apartado "2.5.10. YOLO-Pose" este modelo ha ido evolucionando en el tiempo con la aparición de diferentes versiones, habiendo para la estimación de posturas humanas varias versiones disponibles: YOLOv5-Pose, YOLOv7-Pose, YOLOv8-Pose y YOLOv11-Pose. Atendiendo a los criterios fijados para la selección de modelos y tratándose la última versión (YOLOv11-Pose) de una versión relativamente reciente (menos de un año) durante el desarrollo de este estudio, se opta por **incorporar la versión YOLOv8-Pose** en lugar de la última para intentar realizar la comparativa con los otros modelos con versiones similares en el tiempo.

El modelo YOLOv8-Pose al igual que los anteriores tiene diferentes versiones publicadas, siendo algunas de ellas susceptibles de ser aptas para su utilización en dispositivos *edge* (apartado "2.5.10. YOLO-Pose") según los criterios de tamaño de selección de modelos que hemos definido:

- YOLOv8n-Pose. Versión "nano".
- YOLOv8s-Pose. Versión "small".
- YOLOv8m-Pose. Versión "medium".

En el estudio **se incluirán estas tres versiones** que cumplen los criterios de selección mientras que no se consideran adecuadas las versiones YOLOv8l-Pose ("large") y YOLOv8x-Pose ("extra large") por exceder su tamaño el criterio que hemos establecido como máximo para su utilización en dispositivos móviles.

Como vimos anteriormente (Tabla 7, "Resumen características modelos preentrenados") los modelos YOLO se encuentran en un formato PyTorch (.pt) ya que es el formato nativo de

entrenamiento e inferencia usado por Ultralytics para sus modelos por lo que es seguro que para homogeneizar todos los modelos que vamos a incluir en el estudio con un mismo formato haya que realizar una conversión de formatos durante el proceso de implementación.

Nombre del modelo	Puntos clave estimados	Formato	Tamaño (KBytes)
MoveNet Lightining 8	17	TFLite	2.895
MoveNet Lightining 16	17	TFLite	4.759
MoveNet Lightining 32	17	TFLite	9.373
MoveNet Thunder 8	17	TFLite	7.127
MoveNet Thunder 16	17	TFLite	12.584
MoveNet Thunder 32	17	TFLite	25.026
BlazePose Lite	33	MediaPipe / TFLite	2.818
BlazePose Full	33	MediaPipe / TFLite	6.441
BlazePose Heavy	33	MediaPipe / TFLite	27.709
Yolo8-pose Nano	17	ONNX / TFLite	6.771
Yolo8-pose Small	17	ONNX / TFLite	23.422
Yolo8-pose Medium	17	ONNX / TFLite	50.120

Tabla 13. Resumen de características modelos incluidos en el estudio

3.2. Selección de dataset de testeo

La elección del dataset de testeo es un paso fundamental en la evaluación rigurosa y objetiva de modelos de estimación de posturas humanas. Un dataset representativo y bien anotado permite no solo medir con precisión la capacidad del modelo para detectar y localizar puntos clave del cuerpo, sino también evaluar su robustez frente a variaciones en pose, iluminación, entorno y sujetos. La calidad y diversidad de las anotaciones, junto con un protocolo de evaluación estandarizado, facilitan la comparación directa entre diferentes arquitecturas y versiones de modelos. Además, la accesibilidad y documentación del dataset son cruciales para reproducibilidad y validación externa. Por último la compatibilidad con los modelos seleccionados previamente tiene una importancia casi definitiva a la hora de escoger un dataset sobre los que vimos en el apartado “2.6. Datasets de estimación de posturas”.

3.2.1. Criterios de selección

En este apartado trataremos de justificar los criterios en base a los cuales seleccionaremos al dataset que utilizaremos para el estudio de los modelos. Se valoraran diferentes aspectos que consideramos fundamentales, si bien como hemos ido viendo durante el análisis del marco teórico y estado del arte en el apartado 2 y durante la selección de los modelos a estudiar, estos criterios van a tener una menor influencia en la detección del datset en tanto en cuanto la mayoría de modelos observados (incluidos los seleccionados en el punto anterior para el estudio) no solo adoptan el estándar de keypoints del dataset **COCO** sino que además están entrenados o mejorados en cierta manera con datos de este dataset y como vimos en “Tabla 8. Resumen arquitecturas modelos preentrenados” están diseñados para una

estimación de 17 keypoints coincidentes con el número de anotaciones por persona existente en el dataset COCO. Aun así definiremos los siguientes criterios de selección:

- **Cobertura y diversidad de poses**. El dataset debe incluir una amplia variedad de posturas, movimientos y actividades para reflejar escenarios reales. Esto asegura que el modelo evaluado sea robusto a diferentes posiciones y articulaciones.

- **Cantidad y calidad de anotaciones** (keypoints). Número de keypoints anotados por persona, compatibilidad con los modelos seleccionados para facilitar comparativas. Precisión y consistencia en las anotaciones (manuales o automáticas) que minimicen ruido y errores.

- **Diversidad de sujetos y variedad en condiciones de captura**. Incluir diferentes edades, géneros, tipos corporales y condiciones para evitar sesgos y asegurar generalización. Inclusión de diversos fondos, iluminación, ángulos de cámara, y resolución de imagen. Entornos controlados y no controlados (interiores y exteriores).

- **Disponibilidad y accesibilidad**. Dataset público, con documentación clara, formatos estándar y licencia compatible con investigación y desarrollo.

Criterio	**COCO**	**MPII**
Cobertura y diversidad de poses	Alta diversidad con escenas cotidianas y actividades variadas; incluye poses complejas y contextos con múltiples personas.	Enfocado principalmente en actividades humanas diarias, especialmente deportes, con buena variedad pero menos contexto complejo.
Cantidad y calidad de anotaciones (keypoints)	**17 keypoints** bien definidos; anotaciones extensas y precisas, con estándares para métricas como OKS.	**16 keypoints** con anotaciones detalladas en articulaciones principales; calidad alta pero menos cantidad total.
Diversidad de sujetos y condiciones de captura	Gran variedad de sujetos, etnias y entornos; imágenes tomadas en condiciones muy variadas (interior, exterior, iluminación, fondo).	Menor diversidad en sujetos y escenarios; principalmente imágenes enfocadas en personas individuales y deportes, con condiciones más controladas.
Disponibilidad y accesibilidad	Publicado ampliamente con fácil acceso, documentación completa y soporte para evaluación automatizada; estándar de facto en la comunidad.	También público y accesible, con documentación clara, pero menos extendido como *benchmark* comparativo global.

Tabla 14. Resultados criterios selección de dataset

La Tabla 14 sintetiza las principales razones que hacen de COCO un dataset preferido para evaluación general, mientras que MPII puede ser más específico para ciertos dominios como análisis de movimientos deportivos. Por tanto el resultado de la selección de dataset de testeo indica como mejor opción **utilizar el dataset COCO para el estudio**.

Para llevar a cabo una mejor evaluación de los modelos, se seleccionan además dentro del dataset de testeo **dos subconjuntos** específicos de imágenes. Esta segmentación busca analizar el desempeño del modelo bajo condiciones variadas y realistas, diferenciando entre situaciones óptimas de inferencia y situaciones desafiantes.

3.3. Selección de imágenes de testeo

Debido a la naturaleza heterogénea de COCO, se establecerá un proceso de filtrado sistemático con el fin de generar un subconjunto controlado de imágenes específicamente diseñado para pruebas comparativas bajo condiciones constantes. Dicho filtrado se realizará a partir de dos criterios principales:

- **Una única persona visible en la imagen**. Este criterio responde al objetivo inicial del proyecto (estudio de modelos para la realización de una aplicación de tele-rehabilitación) el cual iría dirigido a la utilización por parte de un único usuario. Este criterio además elimina la complejidad derivada de la presencia de múltiples individuos en el mismo cuadro, lo que puede introducir ambigüedad en la asignación de keypoints y en la interpretación de los resultados. Al garantizar un único sujeto, se reduce la variabilidad no deseada y se asegura que los errores de inferencia estén asociados únicamente al modelo, y no a interferencias en la segmentación de múltiples instancias.

- **Al menos 15 de los 17 keypoints correctamente anotados**. Al igual que con el anterior criterio, en el marco del proyecto es razonable asumir que el sujeto que está realizando los ejercicios de tele-rehabilitación tendrá una visibilidad máxima dentro de la aplicación, por lo que, incluyendo una pequeña tolerancia, estableceremos que de los 17 keypoints anotados al menos 15 estén presentes en las imágenes de pruebas. Este requisito asegura la calidad de las anotaciones de referencia (*ground truth*) y permite contar con una representación anatómica casi completa de la persona en la imagen. La disponibilidad de la mayoría de los puntos clave evita que la evaluación se vea afectada por anotaciones incompletas, incrementando la confiabilidad de las métricas de desempeño.

La aplicación de estos criterios generará un subconjunto controlado de imágenes, con condiciones homogéneas de anotación y representación corporal, que constituirá la base experimental para la comparación de los diferentes modelos. Este enfoque metodológico permite minimizar fuentes externas de variabilidad, garantizando que las diferencias observadas en las métricas de precisión puedan atribuirse de manera directa a las capacidades de los modelos, y no a inconsistencias del dataset de testeo.

3.3.1. Subconjuntos del dataset de testeo

La elección de los dos subconjuntos diferenciados dentro del dataset general de testeo se fundamenta en la necesidad de evaluar de manera completa y representativa la precisión de los modelos ya que cada subconjunto cumple una función específica que contribuye a la caracterización del rendimiento del modelo bajo distintas condiciones de captura y que nos permitirá comparar los resultados obtenidos en cuanto a precisión se refiere de cada uno de los modelos en condiciones muy diferentes.

Imágenes con características adecuadas para la estimación de posturas

El primer subconjunto agrupa imágenes clasificadas como adecuadas para la estimación de posturas humanas y estaría compuesto por instancias seleccionadas para representar condiciones óptimas de captura visual. Cada imagen contiene una única persona situada de manera centrada en el encuadre, con una proximidad suficiente a la cámara que permite la observación clara de las articulaciones principales, incluyendo hombros, codos, muñecas, caderas, rodillas y tobillos. Además el tamaño relativo de la persona respecto al tamaño de la imagen asegura que las proporciones de las articulaciones se representen consistentemente, reduciendo la variabilidad introducida por escalas extremas Esta disposición garantiza que los modelos puedan realizar inferencias precisas sobre la geometría corporal y la posición de cada articulación.

Este subconjunto cumple un rol fundamental en la evaluación de modelos ya que permite medir una precisión “máxima” alcanzable en condiciones controladas, sirviendo como referencia para comparar el rendimiento del modelo frente a escenarios más complejos o adversos. Al centrarse en imágenes donde la persona es claramente observable y la postura es discernible, se garantiza que los errores de estimación se deban principalmente a las limitaciones del modelo y no a deficiencias en la calidad de los datos de entrada.

Imágenes con características inadecuadas para la estimación de posturas

El subconjunto de imágenes clasificadas como no adecuadas para la estimación de posturas humanas incluiría instancias que representan condiciones adversas o desafiantes para la inferencia de posturas. Estas imágenes contienen personas que se encuentran descentradas o lejanas a la cámara, lo que dificulta la observación clara de hombros, codos, muñecas, caderas, rodillas y tobillos. La disposición de la persona en el encuadre, así como la variabilidad en el tamaño relativo y la orientación, introduce complejidades que simulan escenarios del mundo real donde la calidad de la captura puede ser subóptima.

El análisis de este subconjunto proporcionará información valiosa sobre las limitaciones del modelo, identificando casos en los que la precisión se ve comprometida debido a factores externos a la arquitectura del algoritmo. De esta manera, se logra una evaluación más completa y realista del desempeño del modelo, complementando los resultados obtenidos con el subconjunto de imágenes adecuadas y permitiendo derivar conclusiones sobre su aplicabilidad en entornos no controlados.

3.4. Métricas de validación y evaluación

La evaluación de modelos de estimación de posturas humanas requiere la adopción de métricas que permitan cuantificar tanto la calidad de las predicciones como la eficiencia en su ejecución. En este contexto, las dos dimensiones fundamentales consideradas son precisión y rendimiento, las cuales ofrecen una visión complementaria del desempeño del modelo en condiciones prácticas.

Ambas métricas deben analizarse de manera conjunta, ya que un modelo extremadamente preciso puede resultar inviable si su latencia es demasiado alta, mientras que un modelo muy rápido pero con baja precisión carece de utilidad práctica. La evaluación equilibrada de precisión y rendimiento permite establecer compromisos óptimos, adaptados a los requerimientos específicos de la aplicación.

3.4.1. Precisión

La precisión se refiere al grado de correspondencia entre los puntos clave (keypoints) predichos por el modelo y las anotaciones de referencia (*ground truth*). En estudios de posturas como ya hemos visto en el apartado "2.7. Métricas de precisión", esta métrica suele calcularse mediante indicadores como OKS (*Object Keypoint Similarity*) o Mean Average Precision (mAP), que permiten evaluar qué tan cercanos están los keypoints estimados a sus posiciones reales en la imagen.

Una alta precisión implica que el modelo es capaz de identificar correctamente las articulaciones incluso en posturas complejas o bajo condiciones variables de iluminación, escala y perspectiva. Esta métrica es esencial para determinar la validez técnica del modelo y su aplicabilidad en contextos donde la exactitud en la identificación de posturas es crítica, como en rehabilitación médica, análisis deportivo o interacción en entornos de realidad aumentada.

3.4.2. Rendimiento

El rendimiento se relaciona con la eficiencia computacional del modelo, es decir, con la velocidad y los recursos necesarios para realizar inferencias en tiempo real. En entornos móviles o embebidos, como dispositivos Android, el rendimiento se mide a través de métricas como la latencia de inferencia (tiempo necesario para procesar una imagen), el número de *frames* por segundo (FPS) alcanzado y el consumo de memoria y energía durante la ejecución. En nuestro estudio tomaremos los tiempos de ejecución de cada inferencia para cada imagen del dataset de testeo.

Un alto rendimiento implica que el modelo es capaz de operar en condiciones de baja capacidad de cómputo, lo cual es fundamental para garantizar una experiencia de usuario fluida y sostenible en aplicaciones en dispositivos *edge*.

3.5. Herramientas y entorno de desarrollo

El presente estudio se desarrolla utilizando como entorno base el sistema operativo **Windows 10** versión 10.0.

Para la obtención de datos de testeo, conversión de modelos y comprobación de resultados se utiliza **Jupyter Notebook 6.4.5**, un entorno basado en **Python** que facilita la exploración de datasets y el trabajo con los resultados obtenidos a través de la integración de bibliotecas científicas y de visión por computadora para la evaluación de precisión de los modelos.

La construcción de aplicaciones móviles para la plataforma Android se lleva a cabo mediante **Android Studio Electric Eel | 2022.1.1 Patch** 2 (Build #AI-221.6008.13.2211.9619390), que proporciona un entorno completo de desarrollo integrado con soporte nativo para el lenguaje Kotlin, Java y herramientas de depuración específicas de Android. Este IDE permite la integración de modelos mediante TensorFlow Lite, así como la generación de ficheros de aplicación (apk) para Android para la realización de pruebas directa de inferencia en dispositivos móviles.

Adicionalmente, se incorpora el plugin **PlantUML** integration (6.0.0-IJ2020.3) para la creación de diagramas UML directamente dentro de Android Studio, que facilita la documentación de la arquitectura del sistema.

PARTE II: PLANIFICACION, IMPLEMENTACIÓN Y RESULTADOS

4. PLANIFICACIÓN

4.1. Workflow general del proyecto y fases del desarrollo

En este apartado se describe el workflow general adoptado en este estudio, organizado en tres fases principales: preparación, desarrollo y evaluación/análisis de resultados. En cada fase se utilizan herramientas diferentes para su ejecución, mientras que las fases de obtención de datos de testeo y análisis de resultados se realizan utilizando Jupyter Notebook, Excel y Word, la fase desarrollo se utiliza Android Studio para implementar la aplicación que ejecuta el proyecto (Imagen 14).

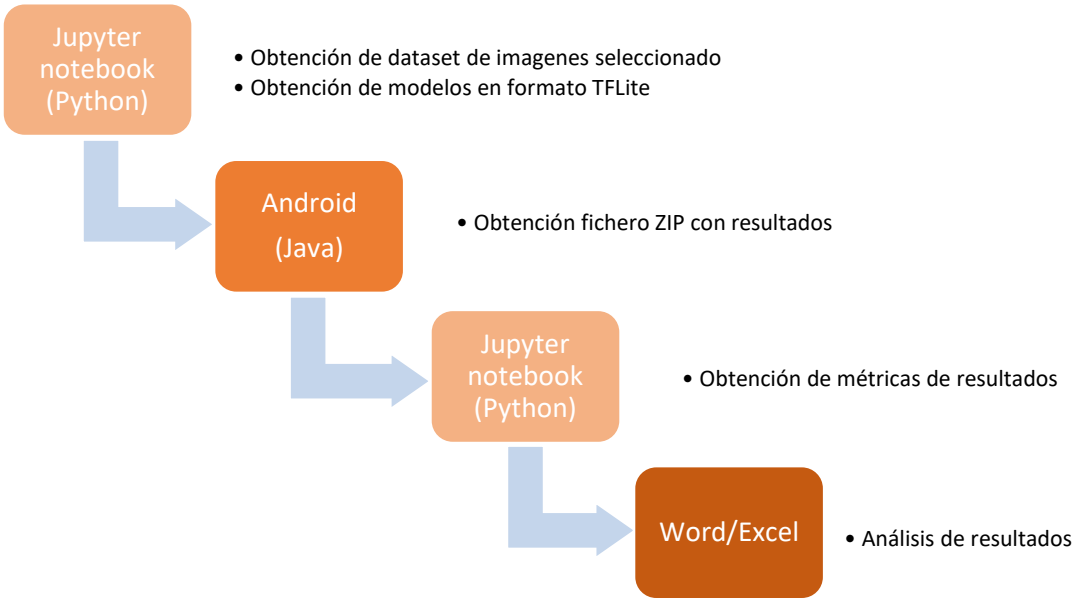


Imagen 14. Workflow general del proyecto

4.1.1. Fase 1: preparación del dataset de testeo y obtención de modelos

En esta primera etapa se lleva a cabo la recopilación y configuración de los recursos fundamentales para el proyecto: modelos de estimación e imágenes de testeo. El ella se utiliza **Jupyter Notebook** para implementar una serie de scripts en lenguaje **Python** que permitan obtener tanto el dataset de las imágenes que cumplan los requisitos descritos previamente como los modelos seleccionados en formato TensorFlow Lite (TFLite).

Obtención del dataset de imágenes seleccionado

Se realiza la definición y filtrado del conjunto de datos de referencia (del dataset COCO), aplicando los criterios definidos previamente de calidad y representatividad (imágenes

con una única persona y keypoints correctamente anotados) con el fin de garantizar un subconjunto adecuado para las pruebas comparativas bajo condiciones constantes. La salida es un subconjunto de imágenes de testeo (dataset filtrado).

Obtención de modelos en formato TFLite.

Incluye la descarga directa (en caso de existir los modelos seleccionados en formato TFLite) y la conversión (de los modelos que lo requieran) a formato TFLite, formato optimizado para dispositivos móviles. Como resultado obtenemos los modelos en un formato homogéneo que constituyen la base para la implementación de la aplicación de pruebas de modelos en Android.

4.1.2. Fase 2: desarrollo de la aplicación

En esta fase de desarrollo se lleva a cabo la implementación de la aplicación **Android**, utilizando como lenguaje de programación **Java**, debido a su compatibilidad nativa con el ecosistema Android y a la disponibilidad de librerías optimizadas para la gestión de recursos y la ejecución de modelos de aprendizaje automático en dispositivos móviles. La aplicación integra los modelos previamente seleccionados en formato TFLite, los cuales han sido seleccionados por su ligereza y eficiencia en entornos con recursos limitados, como smartphones y tablets.

El flujo de trabajo de la aplicación contempla la carga e integración de los modelos TFLite, seguidos de la procesamiento de las imágenes de testeo extraídas del dataset de validación. Cada imagen es sometida a un proceso de inferencia a través del modelo correspondiente, y los resultados son gestionados de forma sistemática. Para garantizar un análisis exhaustivo, la aplicación genera dos tipos de ficheros de salida independientes por cada modelo:

- Un fichero destinado a almacenar las métricas de precisión, donde se registran los valores de exactitud obtenidos en cada predicción de cada imagen, permitiendo así evaluar la capacidad del modelo para reconocer correctamente las instancias del dataset.

- Un fichero orientado a almacenar las métricas de rendimiento computacional, en el que se registra el tiempo de ejecución requerido por cada estimación, con el fin de valorar la eficiencia del modelo en dispositivos móviles.

Posteriormente, todos los ficheros generados se compilan y organizan en un archivo comprimido (ZIP), lo que no solo asegura una gestión más estructurada y compacta de los resultados, sino que también facilita su transferencia, almacenamiento y análisis posterior. Esta estrategia permite disponer de un repositorio unificado de resultados, optimizando tanto la trazabilidad de los resultados como la comparación entre diferentes modelos bajo condiciones homogéneas de evaluación.

4.1.3. Fase 3: evaluación, análisis de resultados y documentación

La etapa final del proceso se centra en la interpretación y análisis del desempeño de los modelos, constituyendo un componente esencial para la validación de la investigación. En este punto no solo se recopilan las métricas obtenidas en fases anteriores, sino que además se someten a un proceso comparativo lo que permite establecer relaciones entre los diferentes modelos bajo condiciones experimentales homogéneas.

La evaluación contempla tanto métricas de precisión como métricas de eficiencia computacional (rendimiento del modelo). Este enfoque posibilita una caracterización más completa de cada modelo, superando las limitaciones de un análisis basado únicamente en la precisión. Para la automatización de este proceso, se desarrollan scripts específicos en **Python**, ejecutados en el entorno **Jupyter Notebook** que permiten procesar los resultados generados por los modelos y calcular métricas de desempeño clave tales como AP (Average Precision). Este enfoque garantiza que la comparación se realice bajo condiciones homogéneas y reproducibles, eliminando posibles sesgos derivados de procedimientos manuales o inconsistentes.

Durante la interpretación se identifican fortalezas y limitaciones de cada modelo. Por ejemplo, un modelo puede presentar un alto nivel de exactitud pero requerir un tiempo de inferencia excesivo, lo que lo haría menos viable en dispositivos con recursos restringidos. En contraste, otro modelo podría mostrar un rendimiento computacional óptimo, aunque con ligeras pérdidas en precisión. La comparación permite establecer un balance entre la calidad de las predicciones y la eficiencia en la ejecución. Los hallazgos de esta fase constituyen la base para la formulación de conclusiones y la propuesta de líneas de investigación futura.

Fase	Nombre	Duración estimada	Objetivo
FASE 1	Preparación del dataset de testeo y obtención de modelos	2-3 semanas	Obtención de un dataset homogéneo de imágenes de testeo con el que poder validar los modelos. Obtención de los ficheros de los modelos a testear.
FASE 2	Desarrollo de la aplicación	8-10 semanas	Desarrollar una aplicación para Android en la que poder integrar los modelos y las imágenes de testeo para obtener resultados de precisión de estimación de posturas de los modelos así como de rendimiento (tiempo que tarda cada modelo en realizar la inferencia).
FASE 3	Evaluación, análisis de resultados y documentación	3-4 semanas	Ejecución en distintos dispositivos, recopilación y análisis de los resultados. Generación de documentación con la descripción del desarrollo del proyecto, documentación técnica asociada, realización de gráficos, tablas e ilustraciones para la descripción de los resultados obtenidos, discusión de posibles aplicaciones y futuras ampliaciones y conclusiones.

Tabla 15. Fases generales del proyecto

4.2. FASE 1: Preparación del dataset de testeo y obtención de modelos

4.2.1. Análisis del dataset COCO

Como ya vimos en el apartado "2.6.1. Dataset COCO (Common Objects in COntext)" el dataset COCO se caracteriza por su diseño jerárquico, en el cual las imágenes se encuentran vinculadas a metadatos en formato JSON que describen instancias de objetos, anotaciones de keypoints, segmentaciones y categorías.

En la primera etapa del proyecto se lleva a cabo un análisis detallado de la estructura del dataset COCO con el propósito de comprender la organización de los datos, las categorías disponibles, todas las anotaciones asociadas a las imágenes y las herramientas disponibles para la evaluación de los modelos.

4.2.2. Selección del dataset de imágenes de testeo

Una vez comprendida la estructura del dataset COCO y los recursos asociados, la siguiente etapa consiste en el diseño y ejecución de scripts automatizados para la selección de imágenes de testeo que utilizaremos en la aplicación Android. Esta fase es clave para la preparación del conjunto de datos de prueba, ya que permite garantizar la reproducibilidad del proceso, la trazabilidad de las imágenes seleccionadas y la consistencia de los criterios aplicados para todos los modelos de forma homogénea.

El propósito de los scripts es seleccionar y posteriormente automatizar la extracción de un subconjunto controlado de imágenes del dataset COCO, en concordancia con los criterios definidos previamente, inclusión de imágenes con una única persona visible y elección de imágenes con al menos 15 de los 17 keypoints anotados visibles, e incluir un fichero auxiliar para el registro y control de las imágenes utilizadas y que sirva además para facilitar la ejecución dentro la aplicación Android.

Los scripts se desarrollan en Python, haciendo uso de la API oficial de COCO para la gestión de anotaciones y metadatos, así como de un conjunto de librerías complementarias orientadas al manejo de datos y operaciones científicas como **pycocotools**, fundamental para la interacción estructurada con las anotaciones del dataset. La instalación y verificación de estas dependencias constituye un paso previo indispensable para la configuración del entorno de trabajo.

La ejecución de estos scripts realiza la descarga física de los archivos de las imágenes a disco, los cuales se almacenan en formatos estándar como JPEG o PNG para su posterior incorporación en la aplicación Android. Además de la descarga se genera un fichero de texto que contiene una lista con los nombres de las imágenes descargadas y que facilita el control de estos así como un acceso más eficiente a las imágenes en etapas posteriores, particularmente durante la ejecución de la aplicación Android para la evaluación de los modelos.

4.2.3. Obtención de modelos para el estudio

Posteriormente se realiza la obtención de los modelos ya preentrenados mediante la

descarga directa de los sitios oficiales cuando están disponibles y mediante descarga directa de los sitios oficiales y conversión de formato en caso de que sea necesario al formato homogéneo para todos los modelos que utilizamos en el estudio (TensorFlow Lite). Los ficheros de los modelos preentrenados obtenidos junto con el dataset seleccionado en el punto anterior constituyen los elementos principales para este estudio de estimación de posturas humanas.

Fase	Nombre	Duración estimada	Objetivo
Fase 1.1	Análisis del dataset COCO.	0,5 semanas	Estudio de la estructura del dataset COCO, categorías, número de imágenes de personas, anotaciones disponibles y API de testeo de resultados.
Fase 1.2	Selección del dataset de imágenes de testeo.	1-2 semanas	Realización de un script para descarga de las imágenes filtradas válidas para la estimación de posturas humanas. Ejecución del script de descarga y verificación del contenido del dataset, ajustes y correcciones del proceso.
Fase 1.3	Obtención de modelos para el estudio.	0,5 semanas	Descarga y conversión de los ficheros de los modelos seleccionados.

Tabla 16. Sub-fases de la fase de preparación del dataset de testeo y obtención de modelos

4.3. FASE 2: Desarrollo de la aplicación para Android

4.3.1. Análisis, diseño y preparación

Esta subfase constituye el punto de partida del ciclo de desarrollo del proyecto de estimación de posturas humanas. En esta etapa se integran diversas actividades fundamentales que permiten establecer las bases para la implementación de las fases posteriores:

- Análisis de requisitos funcionales y no funcionales de la aplicación.
- Diseño de la arquitectura del código. Separación de componentes para facilitar la reutilización y escalabilidad.
- Diseño de la salida. Especificación de estructura para el almacenamiento de predicciones y tiempos de inferencia en ficheros estructurados (JSON /CSV) integrados en un fichero ZIP de salida, asegurando que contenga métricas de predicciones y rendimiento en un formato reproducible.

- Diseño de la interfaz de usuario.

- Análisis de las estructuras de entrada y salida de los modelos TFLite. Identificación de los tensores de entrada (necesidades de preprocesamiento y normalización de imágenes específicas) y salida (vectores de keypoints y scores de confianza).

- Configuración del entorno de trabajo. Instalación y verificación de dependencias necesarias (TensorFlow Lite), integración con Android Studio como entorno de despliegue.

- Implementación de un sistema de versionado (GIT) para garantizar la seguridad y seguimiento del desarrollo.

4.3.2. Implementación del núcleo de la aplicación

Esta subfase consiste en la construcción de una aplicación móvil en Java utilizando Android Studio, cuyo propósito es ejecutar el pipeline de inferencia en dispositivos Android. Se aborda en tres dimensiones principales:

- Implementación de arquitectura de clases Java de la aplicación, clase de control, clases para soporte de la inferencia de los modelos seleccionados y clases base para la optimización de la arquitectura. Esta implementación gestiona el ciclo de inferencia, desde la carga de tensores hasta la recuperación de los vectores de salida de keypoints.

- Integración de los modelos con formato TFLite en la aplicación Android. Se utiliza la API de TensorFlow Lite Java (integrando TensorFlow Lite Interpreter) para cargar y ejecutar los modelos previamente seleccionados.

- Integración de las imágenes del dataset seleccionado dentro de la aplicación para su carga desde el almacenamiento interno del dispositivo. Implementación de las acciones de preprocesamiento necesarias sobre las imágenes de entrada (redimensionamiento, normalización de valores de píxeles y conversión a ByteBuffer compatible con TFLite).

Como vimos en el apartado “2.8.1. Hardware” existe una gran heterogeneidad en cuanto a hardware disponible para la ejecución de aplicaciones dentro del ecosistema Android en cada dispositivo (CPU, GPU, NPU,…). Dicha variabilidad puede generar diferencias significativas en el rendimiento y la precisión de los modelos, dificultando la comparación entre dispositivos por lo que con el fin de realizar una implementación lo más homogénea posible y que pueda ser ejecutada en el mayor número de dispositivos posible en la implementación de este estudio se ha decidido emplear exclusivamente la **Unidad Central de Procesamiento (CPU) como recurso de ejecución**. Al emplear únicamente la CPU, se garantiza un entorno de ejecución uniforme y controlado, independiente de las configuraciones específicas de cada dispositivo de prueba.

4.3.3. Generación y gestión de ficheros de salida

Se generan ficheros de salida por cada modelo evaluado, un fichero de predicciones para el posterior cálculo de precisión en formato JSON y un fichero de tiempos de inferencia por cada imagen en formato CSV para el posterior análisis de rendimiento. Todos los ficheros de salida de todos los modelos se integran al final del proceso en un único fichero ZIP para su mejor exportación y manejo.

Ficheros de predicciones.

Para cada modelo se generan archivos JSON que contienen las predicciones de los keypoints de las posturas humanas sobre cada imagen del conjunto de prueba. Estos ficheros constituyen la base para el cálculo posterior de métricas de precisión, permitiendo evaluar de manera cuantitativa la exactitud del modelo en la estimación de posiciones articulares. La generación de estos ficheros se realiza de manera estandarizada, asegurando que cada entrada corresponda de forma inequívoca a la imagen original y manteniendo la trazabilidad de los datos.

Ficheros de tiempos de inferencia.

Paralelamente, se registra el tiempo de inferencia por imagen de cada modelo en ficheros con formato CSV, documentando la duración de la ejecución en el dispositivo móvil donde se realiza la ejecución. Esta información es esencial para el análisis de rendimiento, permitiendo comparar la eficiencia de los distintos modelos y su viabilidad para aplicaciones en tiempo real.

4.3.4. Desarrollo de la interfaz

En esta subfase se desarrolla una interfaz de usuario especializada con el objetivo de facilitar la ejecución sistemática del proceso de inferencia sobre el conjunto completo de datos de prueba para cada modelo. Esta interfaz cumple el doble propósito de automatizar el procesamiento de grandes volúmenes de imágenes y de proporcionar información en tiempo real sobre el progreso de la ejecución.

La interfaz permite ejecutar de manera consecutiva la inferencia sobre todas las imágenes del dataset en cada uno de los modelos. Este enfoque garantiza que cada modelo se evalúe bajo condiciones homogéneas, eliminando la necesidad de intervención manual repetitiva y minimizando errores operativos.

Durante el proceso de inferencia, la interfaz muestra visualmente una lista de los modelos en ejecución de forma que cada uno de ellos va evolucionando a través de un código de colores según va finalizando su ejecución para monitorizar el estado global de la ejecución.

Una vez finalizado el proceso de inferencia, la interfaz ofrece la posibilidad de exportar (compartir) de manera directa el fichero ZIP que integra todos los resultados generados, incluyendo las predicciones de keypoints y los tiempos de inferencia por imagen.

4.3.5. Pruebas y correcciones

Pruebas funcionales, de estabilidad y de portabilidad. Comprobación de que las imágenes procesadas generan predicciones válidas con keypoints consistentes, medición de la latencia por inferencia (ms/imagen) y ejecución continúa de la aplicación durante intervalos prolongados para detectar posibles fugas de memoria, caídas de la aplicación o degradación de rendimiento. Validación en distintos dispositivos Android para evaluar variaciones de rendimiento debidas a las características del hardware.

Fase	Nombre	Duración estimada	Objetivo
Fase 2.1	Análisis, diseño y preparación.	1 semana	Definición de la arquitectura de la aplicación, estructuras de datos, workflows y ficheros de salida. Preparación del entorno de desarrollo.
Fase 2.2	Implementación.	4-5 semanas	Implementación de la estructura de clases Java de la aplicación, integración de modelos TFLite y dataset de imágenes de testeo, ejecución de inferencias de los modelos sobre el dataset de testeo, obtención de datos de precisiones por modelo y recopilación de tiempos de inferencia por modelo.
Fase 2.3	Generación y gestión de ficheros de salida.	1 semana	Creación de ficheros de salida con los datos obtenidos.
Fase 2.4	Desarrollo de la interfaz.	1 semana	Creación de una interfaz con las operaciones disponibles en la aplicación.
Fase 2.5	Pruebas y correcciones.	1-2 semanas	Pruebas de ejecución en diferentes dispositivos y corrección de bugs observados.

Tabla 17. Sub-fases de la fase de desarrollo de la aplicación para Android

4.4. FASE 3: Evaluación y análisis de resultados

4.4.1. Evaluación de precisión obtenida

Núcleo del análisis orientado a la evaluación de la precisión de los modelos de estimación de posturas humanas. En esta etapa se examina la capacidad de los modelos incluidos en este estudio para predecir la localización de los puntos clave del cuerpo humano, comparando los resultados obtenidos frente a los valores esperados de referencia.

La precisión es evaluada mediante métricas estandarizadas en el campo de la visión por computadora como AP (Average Precision), ampliamente utilizada en *benchmarks* como COCO, que evalúa la precisión media considerando diferentes umbrales de tolerancia, ofreciendo una medida global del desempeño del modelo (ver apartado "2.6.1. Dataset COCO (Common Objects in COntext)").

Con el objetivo de obtener una visión estructurada y comparativa de los resultados se generan tablas de precisión por modelo, en las que se reflejan los valores de AP a nivel global y se elaboran **gráficos de barras comparativos**, permitiendo identificar de forma visual las diferencias de precisión entre modelos evaluados. Adicionalmente se analizan las diferencias entre la precisión en escenarios óptimos (subconjunto de imágenes adecuadas para estimación de posturas humanas) y los resultados en escenarios de condiciones adversas o desafiantes (subconjunto de imágenes menos adecuadas para estimación de posturas humanas).

4.4.2. Evaluación del rendimiento obtenido

Análisis del rendimiento computacional de los modelos de estimación de posturas en distintos dispositivos Android. El objetivo principal es caracterizar la eficiencia de los modelos en condiciones reales de ejecución móvil considerando métricas de velocidad de inferencia. Este análisis complementa la evaluación de precisión realizada en la fase anterior, permitiendo determinar el balance entre exactitud y eficiencia alcanzado por cada modelo.

El rendimiento se mide en base a indicadores clave ampliamente utilizados en entornos de computación móvil como el tiempo de inferencia (ms/img) que es el tiempo promedio requerido por el modelo para procesar una única imagen de entrada.

Con el fin de asegurar la validez de los resultados, se realizan las pruebas ejecutando la aplicación en varios dispositivos Android con características de hardware diferenciadas. Esta variación permite estudiar cómo factores como el tipo de CPU, la memoria o la versión de Android impactan en el rendimiento de los modelos.

Para documentar y comparar de manera clara los resultados obtenidos, se elaboran **gráficos de barras agrupadas comparativas** de rendimiento organizadas por dispositivo y modelo, incluyendo valores de tiempo de inferencia promedio.

Por último se generan **gráficos de dispersión** mostrando la relación entre tiempo de inferencia y precisión comparativamente entre todos los modelos estudiados para tener de una forma visual directa una imagen comparativa del desempeño de los modelos estudiados.

4.4.3. Realización de la memoria del proyecto

Constituye el cierre formal del proceso metodológico y tiene como objetivo la recopilación, sistematización y publicación de la memoria técnica del proyecto. Esta etapa es fundamental desde una perspectiva académica y científica, ya que transforma los resultados experimentales y de implementación en un documento estructurado, verificable y transferible, garantizando tanto la reproducibilidad del estudio como la difusión del conocimiento generado.

Incluye la descripción detallada del marco teórico en el que se desarrolla este estudio,

la metodología aplicada para la selección y preparación de los modelos a estudiar y del dataset empleado (incluyendo el filtrado de imágenes y las condiciones de control experimental), el detalle del workflow de desarrollo, desde la fase de preparación de datos y modelos hasta la implementación en Android y la evaluación de resultados y por último la recopilación de métricas de precisión y rendimiento, en forma de tablas y gráficos, organizadas para permitir un análisis comparativo entre modelos y dispositivos.

Fase	Nombre	Duración estimada	Objetivo
Fase 3.1	Evaluación de precisión obtenida.	0,5 semanas	Generación de gráficos y tablas con la precisión obtenida por cada modelo y comparativas con la precisión esperada.
Fase 3.2	Evaluación del rendimiento obtenido.	0,5 semanas	Generación de gráficos y tablas con los datos de rendimiento obtenidos por cada modelo en cada dispositivo.
Fase 3.3	Realización de comparativas, extracción de conclusiones y realización de la memoria del proyecto.	1-2 semanas	Recopilación y publicación de la memoria relativa al desarrollo del proyecto.

Tabla 18. Sub-fases de la fase de evaluación y análisis de resultados

4.5. Planificación temporal

La planificación temporal representa un componente esencial dentro de la gestión de proyectos de investigación aplicada en informática, al proporcionar un marco estructurado que posibilita una distribución eficiente de los recursos disponibles, el cumplimiento de plazos establecidos y la organización de las actividades. Su importancia radica no solo en la asignación temporal de tareas, sino también en la capacidad de establecer dependencias entre fases y garantizar la coherencia metodológica durante todo el ciclo de vida del proyecto.

En el presente estudio se adopta una metodología secuencial organizada en **fases y subfases**, pensada para proporcionar un desarrollo progresivo, desde la preparación inicial de los datos y modelos hasta la obtención, análisis e interpretación de los resultados finales. Cada fase responde a un objetivo específico claramente delimitado, lo cual facilita el seguimiento del avance, la identificación de hitos críticos y la evaluación continua del grado de cumplimiento de los objetivos planteados.

En **Imagen 15** podemos ver un **diagrama de Gantt** con la planificación temporal de la evolución del proyecto basado en las estimaciones temporales definidas durante la planificación y reflejadas en Tabla 15 (fases generales) y Tabla 16, Tabla 17 y Tabla 18 (subfases) con los códigos de color utilizados para cada una de ellas.

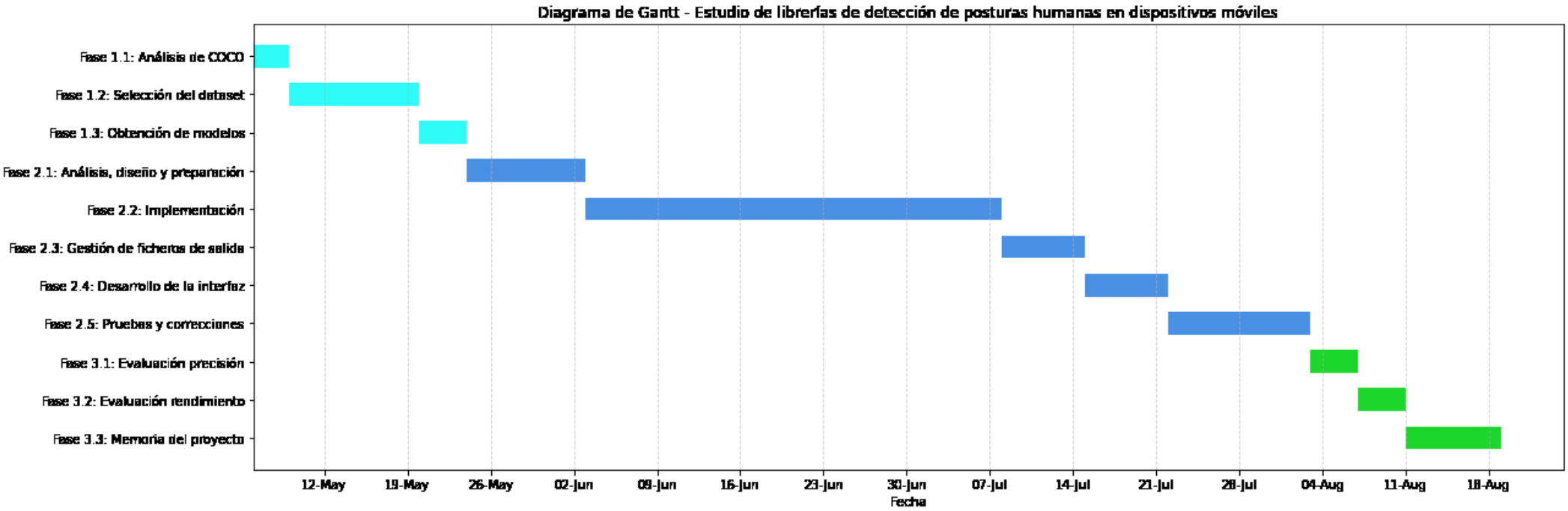


Imagen 15. Diagrama de Gantt del proyecto

4.6. Viabilidad técnica

La viabilidad técnica de un proyecto orientado a la estimación de posturas humanas depende de la disponibilidad, estabilidad y licenciamiento de los modelos de inteligencia artificial que se empleen, así como de los conjuntos de datos utilizados para su entrenamiento y validación. En este sentido, resulta fundamental evaluar también las condiciones legales y técnicas bajo las cuales pueden integrarse en una solución final.

Modelos de Estimación de Posturas Humanas

- MoveNet (Lightning y Thunder). El modelo MoveNet en sus variantes Lightning y Thunder está licenciado bajo Apache 2.0, tal como está especificado en su ficha técnica ("Model Card") de Google (41).

- BlazePose (Lite, Full, Heavy). El modelo GHUM-3D (Lite, Full, Heavy) de BlazePose está licenciado bajo Apache License, Version 2.0. En la documentación oficial de MediaPipe/BlazePose, se indica que el contenido general y ejemplos de código están bajo Creative Commons Attribution 4.0 y Apache 2.0 para los ejemplos (42).

- YOLOv8-pose (Nano, Small, Medium). La licencia del código de YOLOv8, que incluye modelos como los especializados para pose, es AGPL-3.0. Es una licencia de código abierto aprobada por la OSI "*ideal para estudiantes ya que promueve la colaboración abierta y el intercambio de conocimientos*" (32).

Conjuntos de Datos de Pose Estimation

- COCO (Common Objects in COntext). El dataset COCO utiliza varias licencias de Creative Commons (1).

- MPII Human Pose Dataset- Está licenciado bajo una Simplified BSD License, versión 2 cláusulas (BSD-2-Clause) pero aclaran que el uso es libre solo con fines de investigación, y no se permite el uso comercial, debido a que el instituto no posee los derechos de las imágenes (2).

5. FASE 1: PREPARACIÓN DEL DATASET DE TESTEO Y OBTENCIÓN DE MODELOS

5.1. Selección del dataset de imágenes de testeo

Como hemos visto (apartado "2.6.1. Dataset COCO (Common Objects in COntext)") el dataset COCO está compuesto por cientos de miles de imágenes distribuidas en distintas categorías, entre las cuales destaca el conjunto de anotaciones para keypoints humanos lo que lo convierte en una base idónea para el entrenamiento, y, en nuestro caso de estudio, evaluación de modelos de estimación de posturas humanas.

Como vimos también COCO se organiza en diferentes subconjuntos:

- Entrenamiento (train2017, imágenes con anotaciones, utilizadas para el entrenamiento y ajuste de parámetros de los modelos)

- Validación (val2017, aproximadamente 5.000 imágenes, utilizadas para ajustar hiperparámetros y realizar comprobaciones preliminares del desempeño).

- Testeo (test-dev y test-challenge): sin anotaciones visibles, se emplea en competiciones y evaluaciones finales mediante envío a los servidores oficiales de COCO.

En este estudio el foco se encuentra en la fase de evaluación, por lo que adoptamos el **conjunto val2017 como referencia principal**. Su tamaño intermedio y diversidad de contextos lo hacen ideal para pruebas controladas, garantizando una validación robusta de la capacidad de los modelos sin incurrir en sobreajuste.

Librerías necesarias

El proceso de selección y descarga del subconjunto de imágenes se apoya en un conjunto de librerías especializadas de Python:

- **pycocotools.coco.COCO**. La clase COCO, incluida en el paquete pycocotools, constituye la API oficial del dataset COCO. Su finalidad es entre otras gestionar y manipular las anotaciones del dataset, permitiendo cargar y explorar los ficheros de anotaciones en formato JSON, acceder a categorías, imágenes y anotaciones de keypoints humanos, filtrar subconjuntos de datos en función de criterios definidos (por ejemplo, número de personas en una imagen o cantidad de keypoints visibles) y obtener las rutas de descarga de las imágenes asociadas a cada anotación.
  Sintetizando, pycocotools.coco.COCO proporciona la infraestructura básica para interactuar con el dataset COCO, automatizando el acceso a metadatos y facilitando la preparación de subconjuntos para pruebas.

- **pathlib.Path**. El módulo Path de la librería estándar pathlib en Python ofrece una interfaz para el manejo de rutas de archivos y directorios que permite entre otras operaciones construir rutas de forma segura e independiente del sistema operativo (Windows, Linux, macOS), crear y verificar la existencia de directorios para

almacenar imágenes descargadas y resultados de inferencia, gestionar operaciones como concatenar rutas y listar ficheros o mover elementos dentro de la estructura del proyecto.
En este contexto, Path se utiliza principalmente para organizar de manera estructurada y reproducible el almacenamiento local de los subconjuntos de imágenes y ficheros derivados del proceso experimental.

Descarga del fichero de anotaciones

El dataset COCO está organizado en diferentes componentes: imágenes, anotaciones y utilidades asociadas. Dentro de las anotaciones, los archivos JSON contienen información estructurada que describe categorías, instancias, *bounding boxes* y keypoints. El fichero **person_keypoints_val2017.json** constituye la base de referencia para evaluar modelos de estimación de poses en el subconjunto de validación del dataset COCO. Su descarga se realiza a través del paquete oficial **annotations_trainval2017.zip**, ya sea manualmente desde la web de COCO o de forma automatizada mediante código Python.

La ubicación oficial de los ficheros de anotaciones de COCO se distribuyen desde la página oficial del dataset[22] y los ficheros de anotaciones están en el ZIP annotations_trainval2017.zip. Este archivo contiene varios ficheros JSON, entre ellos el de las anotaciones relativas a keypoints que necesitamos (person_keypoints_val2017.json) y que encontramos en la carpeta "annotations" al descomprimir annotations_trainval2017.zip.

Por tanto para el caso del estudio de estimación de posturas humanas el fichero que contiene la información de validación de las imágenes del conjunto de validación se encuentra dentro del ZIP annotations_trainval2017.zip descargado de la página oficial del dataset COCO en la siguiente ruta:

annotations/person_keypoints_val2017.json

Estructura de carpetas del dataset

El dataset de testeo se descarga mediante un script realizado en Python generando una estructura sencilla de carpetas y archivos, formada por una carpeta principal "Dataset" que contendrá una carpeta "Images" en la cual se incluyen las imágenes seleccionadas para testeo (con el mismo nombre que tienen en las anotaciones COCO) y un fichero de texto (**imageFileNames.txt**) con los nombres de la imágenes descargadas para labores de control y para facilitar la localización e identificación de las imágenes que componen el dataset de testeo (Imagen 16).

En el dataset COCO, el campo `file_name` dentro de las anotaciones hace referencia al nombre de cada archivo de imagen tal y como está almacenado en el conjunto de datos (val2017). El formato de los nombres de archivo sigue esta convención:

[identificador_de_imagen_12_digitos].jpg

[22] http://cocodataset.org/#download

Cada nombre de archivo está compuesto por un identificador numérico único de 12 dígitos. Este identificador se corresponde con el campo id de la anotación de imagen dentro del fichero JSON y la extensión utilizada es siempre JPG (por ejemplo 000000000785.jpg).

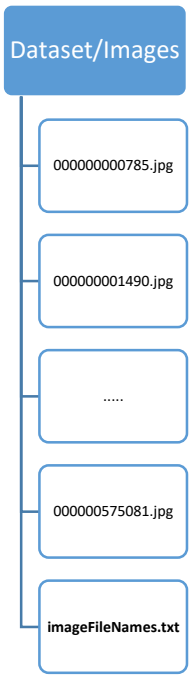


Imagen 16. Estructura carpetas descarga imágenes dataset de testeo

Script de descarga de imágenes del dataset

El algoritmo del script Python tiene como propósito seleccionar, filtrar y descargar un subconjunto de imágenes del dataset COCO que cumpla unos criterios de validez para el estudio anteriormente definido. Su funcionamiento puede dividirse en varias etapas:

- Definición de **criterios de filtrado**. Se establece como condición inicial que una imagen será considerada válida únicamente si contiene al menos un número mínimo de keypoints visibles en las anotaciones de la persona. Este umbral actúa como filtro de calidad, descartando imágenes en las que la anotación carezca de información suficiente para un análisis fiable.

- **Carga de anotaciones** y metadatos. Se cargan las anotaciones asociadas a la categoría "persona", incluyendo los keypoints de las distintas instancias. Para ello:

    - Se obtiene el identificador único de la categoría de personas.
    - Se recupera la lista de identificadores de imágenes que pertenecen a dicha categoría.
    - Se accede a las anotaciones específicas de cada imagen, que contienen los vectores de keypoints y metadatos adicionales (por ejemplo, bounding boxes).

- **Identificación de imágenes candidatas**. Se implementa una estrategia de selección que evita el uso de imágenes con múltiples personas.

    - Se mantiene un array temporal donde se almacenan los IDs de todas las imágenes procesadas.
    - En paralelo, se registra en una lista separada el conjunto de imágenes con más de una anotación, lo que indica que contienen más de una persona.
    - Filtramos el primer array temporal con los IDs de todas las imágenes procesadas con la lista del conjunto de imágenes con más de una anotación para obtener únicamente los IDs de las imágenes que no tienen más de una anotación (lo que supone que solo hay una persona en la imagen).

      De este modo, es posible aislar un **subconjunto de imágenes con únicamente una persona** visible, condición definida previamente.

- **Filtrado por keypoints** y selección final. Entre las imágenes con una sola persona, se realiza un filtrado adicional:

    - Se descartan aquellas en las que la anotación tenga menos del número mínimo de keypoints visibles definido previamente.
    - Se conserva una lista definitiva con los identificadores de imágenes válidas para descarga.

- **Preparación del entorno local**. Antes de iniciar la descarga, el algoritmo prepara la estructura de directorios en el sistema local creando una carpeta principal con una subcarpeta destinada a albergar el dataset filtrado.

- **Descarga y registro de imágenes**. Se procede a la descarga de las imágenes filtradas desde el repositorio oficial de COCO:

    - Cada imagen seleccionada se descarga y se almacena en la carpeta previamente creada.
    - Se mantiene una lista de control con los nombres de las imágenes descargadas, lo que permite verificar la integridad del proceso y garantizar la reproducibilidad.
    - Finalmente, se genera un fichero de control, en el que se documentan los nombres de todas las imágenes descargadas.

- **Seguimiento del proceso**. Durante la ejecución, el algoritmo incluye mecanismos de impresión de datos en consola, que sirven como traza de seguimiento. Esto permite monitorizar el progreso de la descarga y detectar posibles incidencias, tales como la ausencia de imágenes o fallos en la conexión.

En Imagen 17 se puede observar que el conjunto de imágenes de validación val2017 de COCO tiene 2.693 imágenes pertenecientes a la categoría *person* (personas) las cuales están anotadas con 11.004 anotaciones y de las cuales 1.045 imágenes tienen una sola persona y 316 tienen además el número mínimo de keypoints que hemos definido que necesitamos para el estudio.

```
------------------------------------------------
Iniciado procesado del dataset
------------------------------------------------
loading annotations into memory...
Done (t=0.33s)
creating index...
index created!


------------------------------------------------
Id de la categoría "person":  [1]
Número de imágenes pertenecientes a la categoría "person":  2693
Anotaciones totales para las imágenes de la categoría "person" (por número total de ids):  11004
Anotaciones totales para las imágenes de la categoría "person" (cargadas del fichero json):  11004
Número de imágenes con una sola persona:  1045
Número de imágenes con una sola persona filtradas (tienen 15 o más keypoints):  316
------------------------------------------------


Descargando imágenes del dataset...
Descargando imágenes del dataset...Finalizado


------------------------------------------------
Finalizado procesado del dataset
------------------------------------------------
```

Imagen 17. Salida de ejecución de script de obtención de dataset de imágenes de testeo

5.2. Obtención de modelos para el estudio

Se utilizan dos procedimientos principales para la obtención de los ficheros de los modelos preentrenados que utilizamos en este estudio, la descarga directa de modelos publicados en formato TFLite y la descarga con posterior conversión a este formato.

5.2.1. Modelos con opción de descarga directa en formato TFLite

Los modelos diseñados y publicados oficialmente por equipos de investigación o plataformas reconocidas (por ejemplo, TensorFlow Hub, Google Research o el propio repositorio de TensorFlow Lite) representan una fuente fiable y estandarizada. Estas versiones suelen estar optimizadas específicamente para ejecución en entornos móviles y embebidos, y se publican en formatos ya adaptados, como TFLite. Estos modelos están preentrenados y optimizados, son distribuidos en formatos listos para su ejecución en entornos móviles y están disponibles en repositorios oficiales. Los modelos incluidos en este estudio que ofrecen la posibilidad de una descarga directa desde un repositorio oficial son los siguientes:

- **MoveNet**. Los seis modelos (las tres versiones cuantizadas de la versión Lightning más las tres versiones cuantizadas de la familia Thunder) de la familia MoveNet incluidos en el estudio están disponibles para su descarga directa en formato TFLite desde la plataforma Kaggle, plataforma en la que TensorFlow Hub integró sus

modelos. Se pueden descargar de forma directa en este enlace[23], al que a su vez se puede acceder desde el siguiente enlace[24] de TensorFlow Hub (opción "*See TF Hub models*").

- **BlazePose**. Los tres modelos (Lite, Full y Heavy) de la familia BlazePose incluidos en el estudio están disponibles para su descarga directa en formato TFLite desde el repositorio de github del framework MediaPipe[25], desde donde podemos acceder a la descarga de los modelos concretamente en el apartado "*Pose landmark model"* de la página "*MediaPipe Models and Model Cards*" siguiendo este enlace[26].

  Como se puede observar en el aviso de la *front page* para más información acerca del framework MediaPipe debemos redirigirnos a la nueva URL donde ha migrado la documentación:

  "Attention: We have moved to https://developers.google.com/mediapipe as the primary developer documentation site for MediaPipe as of April 3, 2023."

5.2.2. Modelos que requieren conversión de formato

La segunda opción de obtención de modelos se utiliza en la obtención de los modelos YOLO de Ultralytics que como vimos en el apartado "3.1.4. YOLOv8-Pose (Ultralytics)" originalmente se distribuyen en formato PyTorch (.pt) por lo que para su inclusión en nuestro estudio después de la obtención de los modelos es necesario realizar un proceso de conversión de formato (para mantener la homogeneidad con el resto de modelos), generalmente hacia ONNX y posteriormente hacia TensorFlow Lite (TFLite).

Estructura de carpetas de descarga y conversión

Se genera una sencilla estructura formada por una carpeta "Modelos_YOLO8" donde se descargan los modelos en su formato original (PyTorch) y desde la cual se realiza la conversión al formato necesario para su integración en la aplicación de Android desarrollada para el estudio.

Descarga directa de modelos originales en formato PyTorch (.pt)

Los tres modelos que utilizamos de la familia YOLO (Lite, Full y Heavy) están disponibles para su descarga directa en formato PyTorch (.pt) desde este enlace[27] oficial proporcionado por la compañía que los desarrolla y mantiene actualmente (Ultralytics) lo que garantiza la utilización de versiones fiables, verificadas y actualizadas. Pinchando en cada uno de los tres modelos que incluimos en el estudio realizamos la descarga en la carpeta de descarga descrita anteriormente.

---

23 https://www.kaggle.com/models/google/movenet

24 https://www.tensorflow.org/hub/tutorials/movenet

25 https://github.com/google-ai-edge/mediapipe

26 https://github.com/google-ai-edge/mediapipe/blob/master/docs/solutions/models.md#pose

27 https://docs.ultralytics.com/es/models/yolov8/#performance-metrics

Librerías necesarias para la conversión de formato

El proceso de conversión de los modelos de la familia YOLO descargados implica la utilización de las librerías del framework de Ultralytics:

- **ultralytics.YOLO**. Se utiliza como entorno integral para trabajar con modelos YOLO, proporcionando herramientas de entrenamiento, inferencia, evaluación y exportación. Permite convertir modelos a otros formatos como ONNX o TensorFlow Lite.

Conversión de modelos YOLO a formato TensorFlow Lite

Los modelos del framework YOLO pueden exportarse a múltiples formatos (TorchScript, ONNX, TensorRT, CoreML, TensorFlow Lite,…). Durante este proceso de exportación se pueden aplicar ciertos parámetros de configuración que ajustan el comportamiento del modelo y que resultan especialmente importantes para nuestro caso de estudio al exportar modelos para entornos móviles.

La librería Ultralytics provee un método directo para esta conversión mediante la función `export`, que permite ajustar parámetros clave que influyen tanto en la compatibilidad como en el rendimiento de los modelos (42). Para este estudio el script de Python que utiliza la librería ultralytics.YOLO para la descarga de los modelos YOLO realiza varias acciones adicionales sobre el modelo:

- Especificación del formato de exportación (`format="tflite"`). Indica que el modelo debe exportarse en formato TensorFlow Lite. Como ya hemos visto este formato está diseñado para ejecutar inferencias de forma eficiente en dispositivos móviles, reduciendo el tamaño del modelo y optimizando la velocidad de cálculo sin comprometer de manera significativa la precisión.

- Especificación del tamaño de las imágenes (`imgsz=320`). Define el tamaño de entrada de la imagen en 320×320 píxeles (por defecto 640×640 píxeles). Este valor implica un equilibrio entre velocidad y precisión al tener dimensiones menores permiten una inferencia más rápida, lo que resulta ventajoso en dispositivos con recursos limitados mientras asegura un nivel de detalle suficiente para detectar personas y keypoints sin degradar drásticamente la exactitud del modelo. Como vimos en los apartados “3.1.2. MoveNet (Google)” y “3.1.3. BlazePose (Google MediaPipe)” el resto de modelos del estudio utiliza tamaños de entrada de imágenes de 192×192 píxeles y 256×256 píxeles las familias de MoveNet Lightning y Thunder respectivamente y de 256×256 píxeles la familia BlazePose por lo que el tamaño de imagen de entrada seleccionado para los modelos YOLO es el más aproximado al resto de entre los disponibles.

- Especificación número máximo de detecciones por imagen (`max_det=1`). Limita el número máximo de detecciones por imagen a una sola instancia. Dado que el objetivo es trabajar con imágenes de una única persona este valor elimina falsos positivos derivados de múltiples detecciones y simplifica el análisis posterior de precisión y rendimiento.

- Activación del uso de Non-Maximum Suppression (`nms=True`). Elimina predicciones redundantes que se solapan, manteniendo únicamente la detección más confiable lo que contribuye a asegurar que únicamente se retenga la predicción más relevante para cada imagen, evitando duplicidades en la salida.

Como entrada al proceso de exportación utilizamos la ruta de cada uno de los tres modelos descargados en la carpeta de descarga en formato .pt y como resultado de la exportación obtenemos en la misma carpeta de descarga, además de ficheros temporales en formato .omnx de cada modelo, tres subcarpetas, una para cada modelo, que contienen entre otros ficheros dos versiones cuantizadas de cada uno de los modelos, una float32 y otra float16. Como el objetivo de este estudio es analizar el rendimiento en dispositivos *edge* utilizaremos únicamente las versiones cuantizadas float16 de cada uno de ellos ya que son las que se adaptan a las condiciones que habíamos descrito en el apartado "3.1.1. Criterios de selección y modelos seleccionados" ocupando bastante menos espacio que sus homólogas float32.

5.3. Dispositivos de prueba

Para evaluar el rendimiento de los modelos de estimación de poses implementados en dispositivos Android, se ha seleccionado una muestra representativa de terminales con diferentes configuraciones de hardware. La selección incluye tanto teléfonos móviles como tabletas, cubriendo distintas gamas de rendimiento y versiones del sistema operativo, con el fin de analizar la escalabilidad de los modelos y su aplicabilidad en contextos reales.

5.3.1. Listado de dispositivos de prueba

En este apartado se describen las características técnicas de cada uno de los tres dispositivos utilizados en las pruebas de medición de precisión y rendimiento: una Tablet Samsung Galaxy Tab A7 Lite, un móvil Samsung Galaxy M32 y una Tablet Samsung Galaxy Tab A9.

Samsung Galaxy Tab A7 Lite (Tablet)

- Procesador (SoC): MediaTek MT8768T Helio P22T, Octa-core (4x2.3 GHz & 4x1.8 GHz Cortex-A53)
- GPU: PowerVR GE8320
- Memoria RAM: 3 GB
- Almacenamiento: 32 GB
- Sistema operativo: Android 14
- Pantalla: 8.7" TFT, resolución 800 × 1340 píxeles

Esta tablet representa la gama de entrada, con recursos limitados en CPU, GPU y RAM.

Se incluye para evaluar el comportamiento de los modelos en dispositivos con restricciones de capacidad, comunes en entornos educativos, sanitarios o de bajo coste.

Samsung Galaxy M32 (Móvil)

- Procesador (SoC): MediaTek Helio G80, Octa-core (2x2.0 GHz Cortex-A75 & 6x1.8 GHz Cortex-A55)
- GPU: Mali-G52 MC2
- Memoria RAM: 6 GB
- Almacenamiento: 128 GB
- Sistema operativo: Android 13
- Pantalla: 6.4" Súper AMOLED, resolución 1080 × 2400 píxeles
- Otros: Batería de 6000 mAh

El Galaxy M32 representa un dispositivo de gama media con buen rendimiento gráfico y capacidad suficiente para ejecutar modelos ligeros y medios. Es adecuado para evaluar la eficiencia de inferencia en terminales móviles convencionales.

Samsung Galaxy Tab A9 (Tablet)

- Procesador (SoC): Unisoc T618, Octa-core (2x2.0 GHz Cortex-A75 & 6x1.8 GHz Cortex-A55)
- GPU: Mali-G52 MP2
- Memoria RAM: 4 GB
- Almacenamiento: 64 GB
- Sistema operativo: Android 15
- Pantalla: 8.7" TFT LCD, resolución 800 × 1340 píxeles

La tablet Galaxy Tab A9 se sitúa en una gama media actualizada, con mejor capacidad de procesamiento que la Tab A7 Lite, aunque sin alcanzar el nivel de un smartphone moderno. Este dispositivo permite medir la eficiencia de los modelos en un entorno más equilibrado, ideal para aplicaciones industriales, educativas o comerciales.

5.3.2. Justificación de la selección

La elección de estos dispositivos responde a los siguientes criterios:

- Diversidad de capacidades de hardware (procesadores ARM heterogéneos, distintas GPUs).

- Representatividad de escenarios reales de uso (móviles, tablets, gama baja/media).

- Compatibilidad con Android 12 o superior, necesaria para ejecutar modelos con soporte NNAPI y TFLite.

Esta variedad permite analizar el rendimiento cruzado de los modelos y determinar qué configuraciones de hardware resultan más adecuadas para cada tipo de modelo de estimación de posturas, desde los más ligeros hasta los más complejos.

6. FASE 2: DESARROLLO DE LA APLICACIÓN PARA ANDROID

Este apartado describe el **proceso de implementación** técnica de la aplicación Android donde se cargan los modelos seleccionados junto a las imágenes del dataset seleccionadas para extraer resultados de predicciones y tiempos de inferencia de los modelos sobre el dataset. Esta sección por lo tanto constituye el **núcleo técnico** del proyecto y conecta directamente la investigación y análisis previos con la validación práctica del desempeño de los modelos en un **entorno real de ejecución**.

La arquitectura implementada en la aplicación garantiza la ejecución secuencial y automatizada de todos los modelos sobre el conjunto de imágenes definido, lo que facilita la recolección sistemática de datos para su posterior análisis comparativo.

6.1. Análisis, diseño y preparación

6.1.1. Análisis y diseño

Esta fase se concibe con varios objetivos, definir los requisitos funcionales y no funcionales de la aplicación, diseñar una arquitectura que garantice eficiencia, mantenibilidad y escalabilidad, definir las estructuras de datos que se utilizan en la exportación de resultados, diseñar la interfaz que interactuará con el usuario y por último analizar las estructuras de datos tanto de entrada como de salida de cada modelo del estudio.

Análisis de requisitos

Los requisitos funcionales se centran en las capacidades que debe ofrecer la aplicación para cumplir los objetivos del proyecto:

- Carga y gestión de modelos. La aplicación debe poder integrar diferentes modelos de estimación de posturas en formato TFLite.

- Ejecución de inferencias sobre un dataset de imágenes. Debe permitir cargar imágenes desde una carpeta, procesarlas y obtener y registrar los keypoints correspondientes.

- Medición del tiempo de inferencia. El sistema debe registrar el tiempo de procesamiento por imagen y por modelo.

- Almacenamiento y exportación de resultados. Los resultados de inferencia registrados (keypoints y tiempos) deben guardarse en ficheros estructurados y exportarse en un contenedor ZIP.

- Interfaz de usuario básica. Debe proveer un mecanismo simple para iniciar el proceso de inferencia, mostrar el progreso y confirmar la correcta finalización de la tarea.

Además de las funcionalidades principales, la aplicación debe cumplir con una serie de restricciones técnicas o requisitos no funcionales:

- Compatibilidad. el nivel mínimo de SDK debe asegurar ejecución en un amplio rango de dispositivos Android contemporáneos.

- Mantenibilidad, el código debe estar modularizado y documentado para facilitar futuras mejoras o integración de nuevos modelos.

- Escalabilidad, la arquitectura debe permitir incorporar nuevas métricas o datasets sin necesidad de rediseñar el núcleo de la aplicación.

Diseño de la arquitectura

La arquitectura planteada guarda similitudes con el patrón **MVC** (Modelo-Vista-Controlador), ampliamente utilizado en el desarrollo de aplicaciones por su capacidad de separar responsabilidades y promover la escalabilidad. En este contexto, se ha diseñado sobre los principios de herencia y polimorfismo, características intrínsecas de los lenguajes orientados a objetos, lo que permite reutilizar código común y, al mismo tiempo, adaptar el comportamiento específico a cada modelo de estimación de posturas humanas que estamos estudiando.

El Modelo representa la capa encargada de la gestión de datos y lógica de negocio, lo que incluye el manejo de las imágenes de entrada, la carga y ejecución de los modelos de inferencia en formato TensorFlow Lite, y la organización y persistencia de los ficheros de salida generados (predicciones y tiempos de inferencia). Esta capa abstrae la complejidad del preprocesamiento, inferencia y postprocesamiento, de modo que la lógica asociada a cada modelo concreto queda contenida en subclases especializadas que heredan de una clase base común.

La Vista constituye la capa de interacción con el usuario y es responsable de visualizar las operaciones y resultados. En el caso de una aplicación Android, esto se materializa mediante actividades o interfaces gráficas que muestran el estado de ejecución e indicadores de progreso. La Vista es deliberadamente independiente de la lógica del modelo, de manera que su papel se centra en presentar información y recibir interacciones del usuario sin conocer en detalle cómo se ejecutan las operaciones subyacentes.

El Controlador actúa como capa intermedia que coordina la comunicación entre la Vista y el Modelo. Su función es recibir las acciones iniciadas desde la interfaz de usuario (ejecución de las pruebas o exportación de resultados), traducirlas en operaciones sobre el Modelo y devolver a la Vista los resultados o el estado actualizado. En este sentido, el Controlador encapsula la sincronización entre procesos, asegurando que las respuestas se gestionan de manera coherente y en tiempo oportuno.

Este esquema facilita el mantenimiento y la extensión del sistema (nuevos modelos de estimación pueden integrarse mediante la adición de subclases que respeten la interfaz definida en el modelo base) mientras que permite una modularización del código que mejora la legibilidad.

Diseño de estructuras de datos

Se definen las estructuras de datos necesarias para almacenar los datos de salida de la ejecución del test. Esta salida está compuesta por dos ficheros por cada modelo incluido:

- Fichero de **predicciones**. Este fichero en formato **JSON** está formado por un array con las predicciones del modelo para cada imagen del dataset de testeo. El formato de cada una de las predicciones viene determinado por el formato utilizado por el dataset COCO para evaluar los resultados de predicciones de modelos, que es muy similar al descrito en el apartado "2.6.1. Dataset COCO (Common Objects in COntext)" para las anotaciones del dataset y se compone de los siguientes campos para cada imagen:

    - `image_id`: ID de la imagen donde se encuentra la persona.
    - `category_id`: Siempre 1 para personas.
    - `keypoints`: Lista de 51 valores (17 keypoints × 3 valores por keypoint).
      Cada keypoint contiene: (x, y, v) donde x e y son las coordenadas 2D del keypoint en píxeles y v es un código de visibilidad del keypoint (0=no etiquetado, 1=etiquetado pero no visible, 2=etiquetado y visible).
    - `score`: valor numérico (normalmente entre 0 y 1) que indica el nivel de confianza del modelo en la predicción de la posición de ese keypoint.

- Fichero de **tiempos** de inferencia. Fichero de texto en formato **CSV** donde cada línea del fichero incluye información con el nombre de la imagen a modo de identificador y los valores de los tiempos de inferencia del modelo para esa imagen separados por comas.

Diseño de la Interfaz de Usuario

Aunque la aplicación no requiere una interfaz compleja, se definen ciertos elementos que deben estar presentes en la pantalla principal:

- Información de modelos de estimación de posturas incluidos en el test. Listado con todos los modelos que ejecutarán inferencias sobre el dataset de imágenes de prueba.
- Botón de inicio del test en el dispositivo. Botón para iniciar el test en el dispositivo.
- Indicador visual de progreso de la ejecución del test. Avance de la ejecución por modelo.
- Botón de exportación de resultados. Botón para exportar los ficheros con los resultados de inferencia y rendimiento de todos los modelos testados.
- Botón de salir de la aplicación.

Análisis de entradas y salidas de los modelos

La estimación de posturas humanas mediante redes neuronales requiere comprender las estructuras de entrada y salida de los modelos empleados. Estas estructuras determinan tanto la forma en que las imágenes deben preprocesarse como la interpretación de los resultados generados por cada modelo. En términos de entrada, todos los modelos reciben **tensores que representan imágenes RGB** redimensionadas a unas resoluciones fijas y normalizadas en rango de valores. El tamaño de estos tensores difiere según el modelo y la versión:

- MoveNet utiliza entradas compactas de **192×192** píxeles (Lightning) o **256×256** píxeles (Thunder) para sus dos variantes que representan imágenes RGB normalizadas, redimensionadas al tamaño requerido por cada versión.

- BlazePose requiere imágenes de resolución **256×256** píxeles igualmente representando imágenes RGB normalizadas y redimensionadas.

- YOLOv8-Pose emplea entradas por defecto con tensores de 640×640 píxeles, aunque en este estudio y como hemos comentado anteriormente los modelos exportados a formato TFLite para su integración en la aplicación han sido adaptados para soportar una entrada de **320×320** píxeles de un tamaño más parecido al resto de modelos.

Respecto a la salida, cada modelo proporciona tensores estructurados con distinta granularidad de información:

- MoveNet devuelve un **tensor de tamaño [1, 1, 17, 3]**. Cada uno de los 17 keypoints del estándar COCO está representado por con las coordenadas (x, y) normalizadas y el nivel de confianza para los 17 keypoints definidos por el estándar COCO (41).

- BlazePose, más detallado, devuelve un **tensor de tamaño [1, 195]** ya que estima hasta 33 keypoints, incluyendo las coordenadas (x, y) normalizadas más una componente z que representa la profundidad relativa del punto respecto al cuerpo, además de otros dos valores, confianza de visibilidad (0 a 1) y presencia del keypoint (0 a 1) (43).

- YOLOv8-Pose integra detección de personas y estimación de posturas en un único proceso, generando bounding boxes, scores globales y los 17 keypoints de COCO para cada persona detectada en la imagen. Devuelve un **tensor de tamaño [N, 57]** donde N es el número máximo de detecciones que durante la exportación del modelo hemos establecido a 1 con el parámetro `max_det=1` (ver apartado "5.2.2. Modelos que requieren conversión de formato") por lo que en nuestro caso el tensor devuelto por los modelos YOLO es de tamaño **[1, 57]** y está compuesto por varios valores de bounding box (x, y, w, h), confianza (score), clase, y 17×3 valores correspondientes a los keypoints (x, y, score).

6.1.2. Diagrama de clases

El diagrama de clases es una representación fundamental dentro de la documentación

técnica del proyecto, ya que permite visualizar la estructura estática de la aplicación y las relaciones entre sus componentes principales. En este caso, el diagrama refleja la arquitectura orientada a objetos definida, mostrando la jerarquía de herencia, la interacción entre las diferentes clases que conforman el sistema y los componentes públicos, protegidos y privados de cada clase con sus parámetros en el caso de los métodos (Imagen 18).

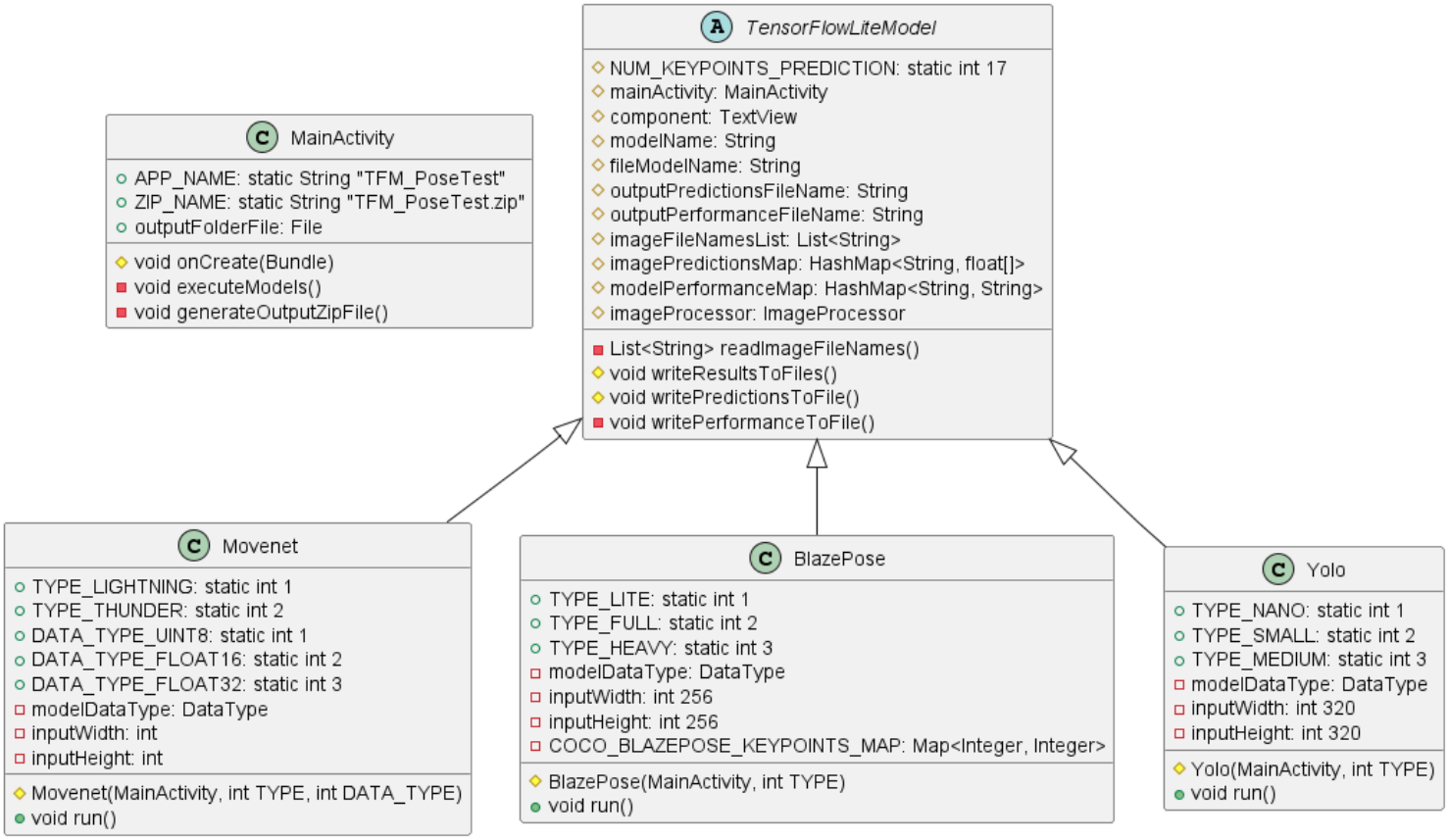


Imagen 18. Diagrama de clases de la aplicación para Android.

6.1.3. Preparación

Preparación entorno de desarrollo

El proceso se inicia mediante la generación de un nuevo proyecto de tipo "Empty Activity", que se selecciona por su carácter básico y altamente personalizable. Esta plantilla ofrece una estructura mínima sobre la cual es posible construir la aplicación de manera progresiva, integrando los componentes específicos requeridos para la ejecución de modelos de estimación de posturas humanas.

- Creación del proyecto. El asistente de creación de proyectos de Android Studio permite definir la configuración inicial del proyecto en varios pasos:
    - Asignación del nombre del proyecto y del paquete de aplicación

(Application ID), que servirán como identificadores únicos en el ecosistema Android.

- Definición de la ubicación de almacenamiento del proyecto en el sistema local.

- Selección del lenguaje de programación (en este caso, Java).

- Configuración del nivel mínimo de SDK (API Level mínimo).

Como resultado de este proceso tenemos una aplicación inicial funcional que contiene un único archivo de actividad principal (MainActivity) y los recursos básicos asociados, como el layout en XML. Sobre esta base mínima se construye la lógica necesaria para la gestión del dataset, la carga y ejecución de los modelos TFLite.

- Incorporación de las dependencias de las librerías TensorFlow. En el fichero **build.gradle** (Module:app) se añaden las dependencias necesarias para el funcionamiento del framework TensorFlow (Imagen 19):

  - **tensorflow-lite:2.12.0**. Es la **librería principal** de TensorFlow Lite, encargada de la ejecución de modelos de aprendizaje automático optimizados en dispositivos móviles y embebidos. Proporciona el motor de inferencia que permite cargar un modelo en formato TFLite y ejecutar predicciones de manera eficiente en la CPU (o en otros delegados cuando se configuran). Se utiliza por tanto para ejecutar los modelos de estimación de posturas en Android con bajo consumo de recursos y alta eficiencia.

  - **tensorflow-lite-metadata:0.1.0**. Librería orientada al manejo de la información descriptiva (metadatos) incluida dentro de algunos modelos TFLite. Estos metadatos contiene detalles como nombres de entradas y salidas, dimensiones de tensores, escalas de normalización o categorías de salida. Permite interpretar de manera más sencilla los resultados de la inferencia.

  - **tensorflow-lite-support:0.4.3**. Conjunto de utilidades complementarias que extiende TensorFlow Lite, proporcionando funciones de preprocesamiento y postprocesamiento de datos (como manipulación de imágenes, conversión de tensores, normalización y transformación de formatos).

```
dependencies {

    ……

    implementation 'org.tensorflow:tensorflow-lite:2.12.0'
    implementation 'org.tensorflow:tensorflow-lite-metadata:0.1.0'
    implementation 'org.tensorflow:tensorflow-lite-support:0.4.3'
}
```

Imagen 19. Dependencias TensorFlow Lite añadidas al fichero build.gradle

- Creación de carpeta de recursos para almacenamiento de las **imágenes del dataset**. En esta carpeta se introducen las imágenes descargadas en el apartado "5.1. Selección del dataset de imágenes de testeo" junto con el fichero de control generado en esa misma fase, imageFileNames.txt.

- Importación uno por uno de todos los **modelos** en formato TFLite obtenidos en el apartado "5.2. Obtención de modelos para el estudio" a la aplicación mediante la opción "New -> Other -> TensorFlow Lite Model" del menú contextual de la carpeta de recursos del proyecto.

Repositorio GIT

La utilización de un sistema de control de versiones es una práctica fundamental en proyectos de investigación aplicada e ingeniería de software. En este caso, la elección de GIT como herramienta central permite garantizar la seguridad, trazabilidad, organización y replicabilidad de todas las fases de desarrollo.

Teniendo en cuenta las fases del proyecto el repositorio está dividido en dos partes:

- **APP**. Contiene todo el proyecto de la aplicación Android (Fase 2), incluidos:
    - **Dataset** de imágenes utilizado (APP/TFM/app/src/main/resources).
    - **Modelos** estudiados (APP/TFM/app/src/main/ml).
    - **Código fuente** de la aplicación (APP/TFM/app/src/main/java).
    - Ficheros de configuración del proyecto Android Studio (build.graddle, AndroidManifest.xml, etc,..).
- **Notebook**. Contiene un notebook de Jupyter Notebook (TFM.ipynb) con los scripts de preparación del dataset de testeo (Fase 1) y los scripts de evaluación de resultados (Fase 3).

La localización del repositorio está disponible en este enlace[28].

6.2. Implementación del núcleo de la aplicación

La fase de implementación constituye el punto en el que se materializa el diseño previamente definido, trasladando los modelos y el dataset a una aplicación funcional para dispositivos Android.

La implementación incluye la programación de las clases Java que definen la arquitectura de la aplicación, la incorporación de los modelos en formato TensorFlow Lite y la adopción de los mecanismos necesarios para la carga, preprocesamiento y ejecución del dataset

[28] https://github.com/jr-gh/TFM

de imágenes. Asimismo, se desarrollan las rutinas específicas para la obtención de predicciones de keypoints y la medición precisa de los tiempos de inferencia por imagen.

6.2.1. Arquitectura de la aplicación

La arquitectura de la aplicación Android se organiza en torno a un diseño orientado a objetos que sigue un esquema inspirado en el patrón **Modelo-Vista-Controlador** (MVC *Model View Controler*). En este caso, las diferentes clases cumplen roles claramente diferenciados y colaboran entre sí para ejecutar el flujo completo de inferencia de modelos de estimación de posturas humanas.

6.2.2. Librerías TensorFlow Lite.

Para la implementación de las clases se utilizan diferentes clases proporcionadas por la API y las **librerías de TensorFlow Lite**. Estas clases permiten gestionar de manera eficiente tanto el preprocesamiento de datos como la ejecución de inferencias y el manejo de resultados. Las funciones principales de cada una de ellas son:

- org.tensorflow.lite.**DataType.** Define los tipos de datos soportados por los tensores de TensorFlow Lite (por ejemplo, FLOAT32, UINT8). Se utiliza para garantizar que las estructuras de entrada y salida del modelo se correspondan con los formatos esperados evitando incompatibilidades.

- org.tensorflow.lite.**InterpreterApi.** Representa la interfaz de alto nivel que permite **cargar un modelo TFLite y ejecutar inferencias** sobre él. A través de esta clase se inicializa el intérprete con un modelo previamente cargado y se proporcionan los tensores de entrada y salida necesarios para la ejecución del modelo.

- org.tensorflow.lite.support.common.**FileUtil**. Proporciona utilidades para gestionar la lectura de ficheros como el propio archivo .tflite del modelo y preparar su integración en el intérprete.

- org.tensorflow.lite.support.common.ops.**CastOp**. Permite realizar conversiones de tipo de dato en tensores, por ejemplo de UINT8 a FLOAT32. Se utiliza en las fases de preprocesamiento y postprocesamiento para garantizar la compatibilidad entre los datos de entrada/salida y el modelo de TensorFlow Lite.

- org.tensorflow.lite.support.common.ops.**NormalizeOp**. Encapsula operaciones de normalización de datos, generalmente aplicadas sobre imágenes antes de ser procesadas por el modelo. Por ejemplo, permite escalar valores de píxeles a rangos específicos como [0,1].

- org.tensorflow.lite.support.image.**ImageProcessor**. Componente fundamental para el preprocesamiento de imágenes. Permite construir pipelines de transformación (como redimensionado, normalización o rotación), asegurando que las imágenes de entrada se ajusten a los requisitos de cada modelo.

- org.tensorflow.lite.support.image.**TensorImage** Representa imágenes en forma de

tensores compatibles con los modelos TFLite. Facilita la conversión desde formatos comunes (Bitmap, JPEG, etc.) a tensores que pueden ser interpretados directamente por el modelo durante la inferencia.

- org.tensorflow.lite.support.image.ops.**ResizeOp.** Permite redimensionar imágenes a un tamaño específico generalmente requerido como paso previo al procesamiento por el modelo. Se integra dentro de ImageProcessor y es esencial para ajustar imágenes a resoluciones como 192x192 o 256x256, dependiendo del modelo.

- org.tensorflow.lite.support.tensorbuffer.**TensorBuffer.** Utilizada para gestionar los tensores de salida generados por el modelo. Permite almacenar, acceder y manipular los resultados de la inferencia en diferentes formatos, facilitando la posterior interpretación de keypoints, coordenadas o métricas derivadas del procesamiento realizado.

6.2.3. Implementación de las clases de la aplicación

Clase MainActivity (controlador principal)

La clase MainActivity actúa como punto de entrada y controlador principal de la aplicación. Su función es coordinar las interacciones entre la vista (interfaz de usuario) y el modelo (clases que gestionan los modelos TensorFlow Lite).

Su implementación incluye acciones como gestionar el ciclo de vida de la aplicación en Android, inicializar y configurar los modelos que se ejecutan en el test, ejecutar la inferencia de cada modelo sobre el conjunto de imágenes del dataset de testeo, mostrar el progreso de la ejecución en la interfaz gráfica y habilitar la exportación de resultados (ficheros de predicciones y tiempos de inferencia).

De esta forma, MainActivity actúa como organizador sin concentrar lógica de procesamiento que queda delegada en las clases de modelos.

Clase TensorFlowLiteModel (clase base abstracta)

La clase TensorFlowLiteModel constituye la superclase abstracta que encapsula las operaciones comunes a todos los modelos.

Entre sus acciones implementadas se incluyen la lectura de la lista de imágenes del dataset para su procesamiento por el modelo, la gestión de las estructuras de almacenamiento de los datos de salida (estimaciones de keypoints y tiempos de inferencia) y el almacenamiento de estos resultados en ficheros.

Esta clase es abstracta y no se instancia directamente, su propósito es proveer una base sólida para la implementación de los modelos concretos.

Clase Movenet (subclase de TensorFlowLiteModel)

Especialización de la clase base para los modelos Movenet (Lightning y Thunder). Su constructor admite todos los parámetros necesarios para definir el tipo de modelo que ejecuta y sus tipos de datos para los seis modelos MoveNet (los tres de la familia Lightning y los tres de la familia Thunder). La ejecución del test del modelo sobre el dataset de imágenes se compone de las siguientes acciones para la inferencia:

1. Inicialización del modelo. Se instancia el modelo **MoveNet** en memoria mediante el intérprete oficial (InterpreterApi).

2. Se crea un objeto TensorImage, contenedor de la imagen de entrada, configurado en el tipo de dato requerido por el modelo (UINT8, FLOAT16 o FLOAT32).

3. Se instancia un TensorBuffer con la forma de salida esperada del modelo **(1, 1, 17, 3)**, que representa una imagen procesada, 17 puntos clave, y tres valores por keypoint (x, y y score).

4. Se recorre la lista de imágenes a procesar y para cada una:

    a. Se obtiene un bitmap de la imagen en formato estándar.

    b. La imagen se carga en el objeto TensorImage.

    c. Se aplica un proceso de **preprocesamiento** a la imagen (**cambio de tamaño**) para adaptarla a la estructura de entrada del modelo (imágenes de **192×192 o 256×256** píxeles según si el modelo es de tipo Lightning o Thunder).

    d. Se ejecuta la inferencia del modelo con la imagen utilizando el intérprete de TensorFlow Lite.

    e. Se registra el tiempo de inferencia nativo de la imagen (métrica de rendimiento) mediante el método `getLastNativeInferenceDurationNanoseconds()` del interprete.

    f. Se recuperan las dimensiones originales de la imagen, con el fin de normalizar las coordenadas estimadas y adaptarlas al tamaño real de la imagen.

    g. Se accede al array de salida del modelo, recorriendo los 17 keypoints estimados y almacenando sus coordenadas normalizadas al tamaño real de la imagen.

    h. Se almacenan los resultados de la inferencia de la imagen en estructuras de datos que contienen las predicciones de keypoints y los tiempos de inferencia.

5. Se procede al cierre del modelo y liberación de los recursos asociados.

6. Se escribe en disco la información de las estructuras de datos con las predicciones de keypoints y los tiempos de inferencia en los ficheros JSON/CSV definidos previamente (apartado “6.1.1. Análisis y diseño”) para el posterior análisis de precisión y rendimiento.

Clase BlazePose (subclase de TensorFlowLiteModel)

Subclase encargada de gestionar los modelos BlazePose (Lite, Full y Heavy). A diferencia de las otras clases *wrapper* de los otros modelos esta clase realiza un **mapeo** de los 17 puntos que queremos estudiar, coincidentes con el resto de modelos y con el estándar del dataset COCO, sobre los 33 que estima el modelo para extraer únicamente estos 17 puntos del resultado de la inferencia (Tabla 12 "Equivalencia puntos BlazePose"). Sus funciones son análogas a las de la clase Movenet:

1. Inicialización del modelo. Se instancia el modelo **BlazePose** en memoria mediante el intérprete oficial (InterpreterApi).

2. Se crea un objeto TensorImage, contenedor de la imagen de entrada, de tipo común para los modelos de la familia (FLOAT32).

3. Se instancia un TensorBuffer con la forma de salida esperada del modelo **(1, 195)**, cuyos primeros 165 valores (33×5) representan los keypoints inferidos en la imagen con 5 atributos (x, y, z, *visibility*, *presence*).

4. Se recorre la lista de imágenes a procesar y para cada una:

    a. Se obtiene un bitmap de la imagen en formato estándar.

    b. La imagen se carga en el objeto TensorImage.

    c. Se aplica un proceso de **preprocesamiento** a la imagen (**cambio de tamaño**) para adaptarla a la estructura de entrada del modelo (imágenes de **256×256** píxeles), una **normalización** y una conversión del tipo de datos a FLOAT32.

    d. Se ejecuta la inferencia del modelo con la imagen utilizando el intérprete de TensorFlow Lite.

    e. Se registra el tiempo de inferencia nativo de la imagen (métrica de rendimiento) mediante el método `getLastNativeInferenceDurationNanoseconds()` del interprete.

    f. Se calcula los ratios de ancho y alto con respecto a la imagen original para la normalización de las coordenadas de los keypoints.

    g. Se accede al array de salida de *landmarks* del modelo que contiene las predicciones de los **33 keypoints** estimados y utilizando la estructura de mapeo de la clase (ver apartado "3.1.3. BlazePose (Google MediaPipe)") se extraen únicamente **las correspondientes a los 17 keypoints que estamos estudiando**, almacenando sus coordenadas normalizadas al tamaño real de la imagen.

    h. Se almacenan los resultados de la inferencia de la imagen en estructuras de datos que contienen las predicciones de keypoints y los tiempos de inferencia.

5. Se procede al cierre del modelo y liberación de los recursos asociados.

6. Se escribe en disco la información de las estructuras de datos con las predicciones de

keypoints y los tiempos de inferencia en los ficheros JSON/CSV definidos previamente (apartado "6.1.1. Análisis y diseño") para el posterior análisis de precisión y rendimiento.

Clase Yolo (subclase de TensorFlowLiteModel)

Subclase orientada a gestión de los modelos YOLOv8-Pose (Nano, Small, Medium) exportados a TensorFlow Lite. Sus funciones son análogas a las del resto de *wrappers* de modelos adaptadas a las características del modelo YOLO:

1. Inicialización del modelo. Se instancia el modelo **Yolo** en memoria mediante el intérprete oficial (InterpreterApi).

2. Se crea un objeto TensorImage, contenedor de la imagen de entrada, de tipo común para los modelos de la familia (FLOAT32).

3. Se instancia un TensorBuffer con la forma de salida esperada del modelo **(1, 1, 57)**, que representa una imagen procesada, con máximo una detección de personas (como habíamos especificado en la exportación del modelo a formato TFLite, ver apartado "5.2.2. Modelos que requieren conversión de formato") y 57 puntos clave que definen cuatro puntos de *bounding boxes*, *objectness* (confianza), clase y los keypoints inferidos en formato (x, y, score).

4. Se recorre la lista de imágenes a procesar y para cada una:

    a. Se obtiene un bitmap de la imagen en formato estándar.

    b. La imagen se carga en el objeto TensorImage.

    c. Se aplica un proceso de **preprocesamiento** a la imagen (**cambio de tamaño**) para adaptarla a la estructura de entrada del modelo (imágenes de **320×320** píxeles) y una **normalización.**

    d. Se ejecuta la inferencia del modelo con la imagen utilizando el intérprete de TensorFlow Lite.

    e. Se registra el tiempo de inferencia nativo de la imagen (métrica de rendimiento) mediante el método `getLastNativeInferenceDurationNanoseconds()` del interprete.

    f. Se recuperan las dimensiones originales de la imagen, con el fin de normalizar las coordenadas estimadas y adaptarlas al tamaño real de la imagen.

    g. Se accede al array de salida del modelo, recorriendo en el array de salida los 17 keypoints estimados y almacenando sus coordenadas normalizadas al tamaño real de la imagen.

    h. Se almacenan los resultados de la inferencia de la imagen en estructuras de datos que contienen las predicciones de keypoints y los tiempos de inferencia.

5. Se procede al cierre del modelo y liberación de los recursos asociados.

6. Se escribe en disco la información de las estructuras de datos con las predicciones de keypoints y los tiempos de inferencia en los ficheros JSON/CSV definidos previamente (apartado "6.1.1. Análisis y diseño") para el posterior análisis de precisión y rendimiento.

Relación entre las clases

MainActivity instancia y gestiona objetos de tipo Movenet, BlazePose y Yolo, los cuales heredan de TensorFlowLiteModel. Utilizando polimorfismo, MainActivity invoca métodos comunes (`run()`) definidos en la clase base TensorFlowLiteModel.

De esta manera, la arquitectura resulta modular, extensible y mantenible, permitiendo incorporar nuevos modelos en el futuro simplemente creando nuevas subclases de TensorFlowLiteModel e implementando las características propias de preprocesamiento y postprocesamiento de cada modelo.

Consideración sobre scoring/visibility/presence

Las predicciones que estima cada modelo se componen básicamente de coordenadas (x, y) más valores de confianza del modelo en la predicción que son heterogéneos por cada modelo como vimos en el apartado "6.1.1. Análisis y diseño" en "Análisis de entradas y salidas de los modelos", unos modelos calculan un score mientras que otros calculan visibility y presence. Como estos valores no tienen influencia posterior en el cálculo de la precisión, durante la implementación se ha incluido un valor fijo para todos ellos.

6.3. Generación y gestión de ficheros de salida

En la fase de análisis (apartado "6.1.1. Análisis y diseño" dentro de "Diseño de estructuras de datos") se definen los formatos de los ficheros de salida con las predicciones y tiempos de inferencia de la aplicación.

La implementación de la gestión de estos ficheros de salida está integrada en la superclase abstracta que encapsula las operaciones comunes a todos los modelos, siendo ésta la que ofrece las labores de almacenamiento temporal de los datos de inferencia y tiempos en estructuras de datos durante la ejecución del test para posteriormente al finalizar la inferencia de todo el dataset por parte del modelo realizar el volcado de todos los datos en los ficheros de salida del modelo. Estas estructuras temporales son:

- **imagePredictionsMap**. Estructura para almacenar las predicciones de keypoints, es un *HashMap* con una cadena como clave (nombre de la imagen de inferencia) y un array de valores float como valor que almacena secuencialmente los keypoints estimados por el modelo para esa imagen.

- **modelPerformanceMap**. Estructura para almacenar los tiempos de inferencia del modelo por cada imagen. Es otro *HashMap* con una cadena como clave (nombre de

la imagen de inferencia) y otra cadena como valor donde se almacena en formato texto los tiempos de inferencia del modelo para la imagen.

Al finalizar la inferencia por parte del modelo de todo el dataset se realiza el proceso de almacenamiento en disco de los resultados almacenados temporalmente en estas estructuras en los ficheros descritos en el apartado “6.1.1. Análisis y diseño”, manteniendo una nomenclatura que permite identificar a que modelo pertenece cada fichero de resultados. En total se generan dos ficheros por modelo en el estudio (el fichero JSON de predicciones y el fichero de texto con formato CSV con los tiempos de inferencia) lo que al tratarse de 12 modelos hace un total de 24 ficheros de salida que al finalizar el proceso global son incluidos en un ZIP para su exportación y posterior análisis.

6.4. Desarrollo de la interfaz

La estructura principal de la interfaz está definida en el archivo **activity_main.xml**, generado y editado desde Android Studio, empleando el editor de diseño integrado. Este archivo XML constituye la descripción declarativa de los elementos gráficos que conforman la pantalla principal de la aplicación, así como de su disposición, estilos y comportamiento básico.

En primer lugar, incorpora una lista estática en la que aparecen todos los modelos incluidos en el estudio, es decir, los distintos algoritmos de estimación de posturas previamente integrados en la aplicación (todas las versiones de MoveNet, BlazePose y YOLO-Pose). Esta lista constituye el núcleo visual de la interfaz (Imagen 20), ya que sobre ella el usuario identifica qué modelos van a ser evaluados y, además, recibe información dinámica sobre el estado de cada uno durante la ejecución del test.

La interacción se organiza en torno a tres botones principales:

- Botón "EJECUTAR TEST". Inicia el proceso de inferencia de cada modelo sobre el conjunto de imágenes seleccionado como dataset de testeo. Durante este proceso, la lista de modelos se actualiza dinámicamente para reflejar el progreso, los modelos en ejecución cambian de color a amarillo, lo que comunica al usuario que el algoritmo está siendo evaluado y una vez completada la ejecución satisfactoriamente, el nombre del modelo pasa a verde, indicando la finalización correcta. En caso de error o fallo durante la inferencia, el modelo se marca en rojo, proporcionando una señal inmediata del problema ocurrido.

- Botón "COMPARTIR RESULTADOS". Permanece inicialmente inactivo (deshabilitado), activándose únicamente al concluir la ejecución completa del test sobre todos los modelos. Su función es facilitar la exportación de resultados de manera intuitiva, generando un fichero comprimido ZIP que contiene tanto los archivos de predicciones (keypoints estimados) como los registros de tiempos de inferencia por imagen. A través de la integración con los mecanismos estándar de Android (*Intents*), este botón permite enviar el fichero ZIP al usuario mediante diversos canales disponibles en el dispositivo, como correo electrónico, aplicaciones de mensajería o almacenamiento en la nube.

- Botón "SALIR": proporciona un mecanismo directo para cerrar la aplicación,

finalizando la sesión de uso de manera controlada y limpia.

La lógica de actualización dinámica de la lista de modelos y el cambio de colores durante la ejecución hace que la interfaz no solo cumpla una función estética, sino también de monitorización en tiempo real de la evaluación de los modelos.

Imagen 20. Interfaz de la aplicación Android (Samsung Galaxy Tab A9)

6.5. Pruebas y correcciones

Se realizan **pruebas** unitarias centradas en la correcta implementación de cada clase de modelo integrada en la aplicación. Cada modelo, ya sea MoveNet, BlazePose o YOLO-Pose, fue evaluado de manera independiente para verificar que la lectura de imágenes desde el dataset seleccionado se realizaba correctamente, el preprocesamiento de las imágenes se ajustaba a los requisitos de entrada del modelo, incluyendo la normalización y el dimensionamiento de los tensores de entrada, la ejecución de la inferencia generaba salidas consistentes con la estructura esperada de keypoints y probabilidades de visibilidad y los tiempos de inferencia por imagen se recogían de manera precisa y se almacenaban correctamente en los ficheros de salida.

Posteriormente, se llevan a cabo pruebas de integración, centradas en la interacción entre los distintos componentes de la aplicación, la clase principal de control (MainActivity), las clases de los modelos y los mecanismos de almacenamiento de resultados. Durante estas pruebas se evalúa que la ejecución secuencial de los modelos sobre el dataset completo se realiza sin interrupciones, la interfaz gráfica refleja correctamente el estado de cada modelo mediante los cambios de color en la lista (amarillo, verde y rojo) durante la ejecución y los botones de la interfaz funcionaran según lo esperado, habilitando la opción de compartir resultados únicamente al finalizar todas las inferencias y permitiendo la salida de la aplicación sin bloqueos o pérdidas de datos.

Durante la fase de pruebas se identificaron diversos **errores** relacionados con la interpretación de los datos de salida de los modelos, un aspecto crítico y complejo debido a la heterogeneidad de las salidas de cada modelo. Cada familia de modelos, ya sea MoveNet, BlazePose o YOLO-Pose, genera resultados en formatos distintos, con estructuras de tensores, dimensiones y significados de cada valor específicos, lo que dificultó la correcta normalización y almacenamiento de los keypoints y sus probabilidades de visibilidad.

Los errores detectados incluyen:

- **Confusión en los índices de los keypoints**, especialmente en BlazePose, donde la correspondencia entre la posición del tensor y la articulación real requiere un mapeo explícito.

- **Interpretación errónea de los valores de visibilidad o confianza**, lo que generaba la inclusión de keypoints poco fiables en los ficheros de salida.

La corrección de estos errores implicó una labor cuidadosa de análisis de la documentación oficial de cada modelo, pruebas unitarias de cada salida. Durante el proceso de prueba además se detectaron y corrigieron los siguientes problemas menores:

- Errores de compatibilidad de tipos de datos en la carga de imágenes y en la ejecución de modelos, solucionados mediante ajustes en las clases de TensorFlow Lite, incluyendo el uso adecuado de TensorImage y TensorBuffer.

- Fallos de sincronización en la interfaz, que impedían la actualización inmediata de los colores de la lista de modelos, solucionadas mediante la ejecución de la acción especificada en el subproceso de la interfaz de usuario que garantiza la actualización en el hilo principal de la interfaz.

- Problemas en la generación de los ficheros de resultados y del fichero ZIP de general, que fueron corregidos configurando la carpeta *Documents* como “external_documents” en fichero file_paths.xml.

Finalmente se realizan pruebas de aceptación, ejecutando la aplicación completa que permiten validar la aplicación, procesar todo el dataset correctamente, ejecutar todos los modelos de manera fiable, generar y almacenar los resultados de forma estructurada y accesible, proporcionar al usuario retroalimentación visual clara sobre el estado de cada modelo y compartir los resultados a través de los canales de Android sin pérdida de información.

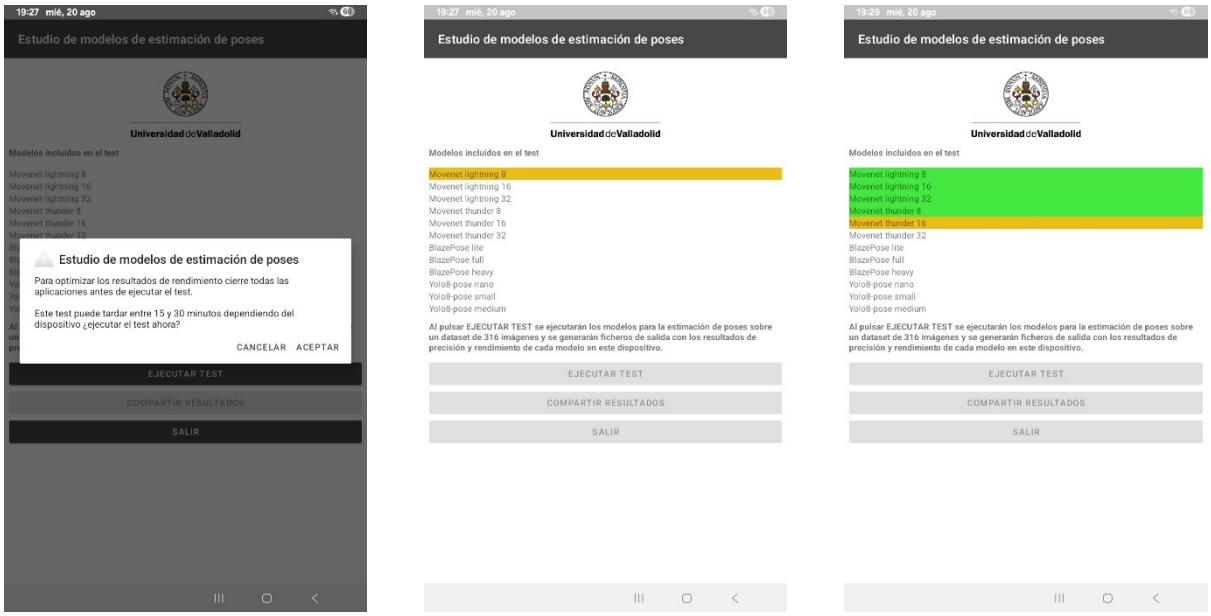


Imagen 21. Imágenes de la interfaz con secuencia de inicio y avance del proceso

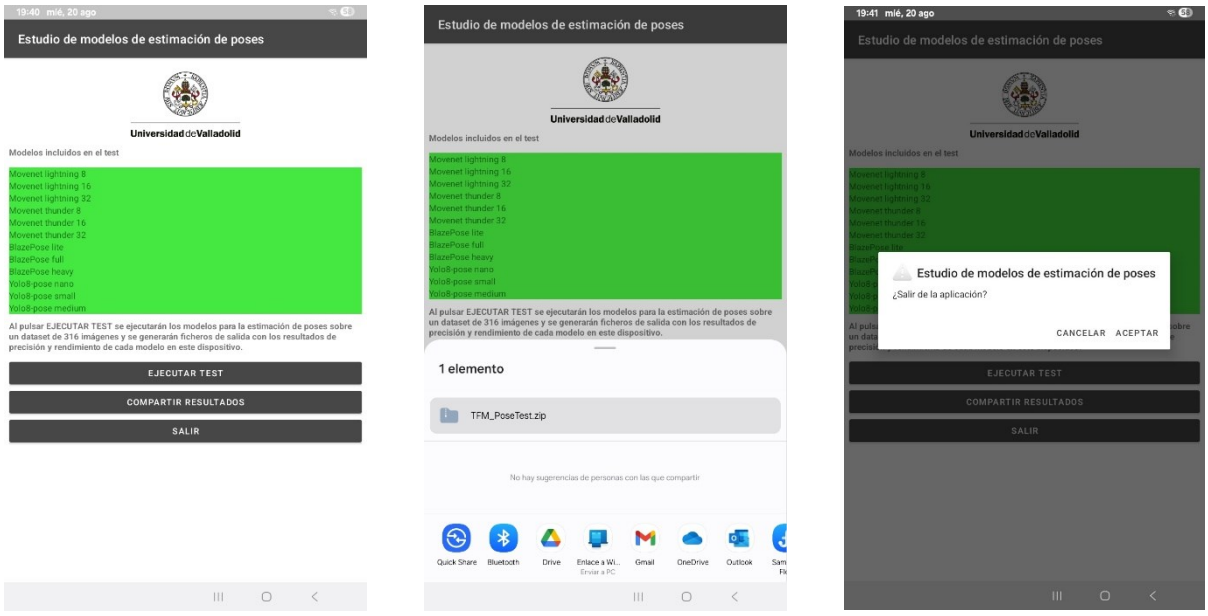


Imagen 22. Imágenes de la interfaz con finalización del proceso, compartir resultados y salida de la aplicación

7. FASE 3: EVALUACION Y ANALISIS DE RESULTADOS

La evaluación de los resultados se realiza de forma dual, primero enfocada en la precisión obtenida por las predicciones de los modelos estudiados con el dataset seleccionado y después analizando los tiempos de inferencia de esas mismas predicciones.

7.1. Resultados obtenidos de precisión

La obtención de las medidas de precisión se realiza mediante el uso de la API oficial de COCO y de scripts desarrollados en Python ejecutados en un entorno Jupyter Notebook. La API de COCO permite evaluar los keypoints estimados por cada modelo **comparándolos con las anotaciones de referencia** (ground truth) disponibles en los ficheros de anotaciones del dataset. Para ello, se utilizan métricas consolidadas en la comunidad de visión por computador, tales como el Average Precision (AP) y el Average Recall (AR) bajo distintos umbrales de coincidencia. Estos cálculos requieren la generación previa de ficheros de predicciones por cada modelo como los obtenidos en este estudio, los cuales incluyen las coordenadas estimadas de los keypoints y sus valores de confianza, que posteriormente son procesados por la API para determinar el grado de concordancia con las anotaciones reales del dataset.

En este apartado se presentan los resultados obtenidos en la evaluación de precisión de los modelos de estimación de poses humanas estudiados, utilizando como métrica principal la Average Precision definida en el protocolo de evaluación de COCO. Para realizar esta evaluación se han tomado de los ficheros ZIP de resultados exportados de la ejecución de los tres dispositivos de prueba **los ficheros relativos a precisiones de cada modelo** (formato JSON).

Como ya hemos visto, se han evaluado un total de 12 modelos ya descritos, MoveNet (Lightning y Thunder, en resoluciones 8, 16 y 32), BlazePose (Lite, Full y Heavy) y YOLOv8-Pose (Nano, Small y Medium), y se ha utilizado un dataset general de 316 imágenes extraídas del dataset COCO (filtradas con las condiciones ya descritas). Sobre este dataset, como se describió en al apartado “3.3.1. Subconjuntos del dataset de testeo” de la metodología, trabajamos con tres subconjuntos de imágenes para poder **evaluar la influencia de las características de las imágenes en las precisiones obtenidas** de los modelos:

- Dataset **general**. Contiene la totalidad de las **316 imágenes** seleccionadas con las que se ha ejecutado el test a los modelos.

- Dataset de **imágenes adecuadas**: formado por un subconjunto de **65 imágenes** del dataset general (316 imágenes) donde la persona aparece centrada y a distancia cercana (condiciones óptimas para estimación de posturas).

- Dataset de **imágenes no adecuadas**: formado por un subconjunto de **61 imágenes** del dataset general (316 imágenes) donde la persona aparece descentrada de la imagen o de un tamaño reducido, lo que implica lejanía (condiciones adversas para estimación de posturas).

Los procesos de filtrado de imágenes adecuadas/no adecuadas, obtención de las medidas

de precisión, la evaluación de los resultados se apoyan en librerías especializadas de Python.

Librerías necesarias

- **pycocotools.coco.COCO**. La clase COCO constituye la API oficial del dataset COCO. Su finalidad es entre otras validar resultados de inferencia mediante la comparación con las anotaciones de referencia, calculando métricas estándar como AP (Average Precision) y AR (Average Recall).

- **pycocotools.cocoeval.COCOeval**. Es la clase principal de evaluación de COCO API. Permite comparar las predicciones generadas por un modelo (detección, segmentación o keypoints) contra las anotaciones reales del dataset COCO, obteniendo métricas como AP (Average Precision) o AR (Average Recall) bajo distintos umbrales de coincidencia.

- **json**. Sirve para leer y escribir ficheros en formato JSON, que es el estándar utilizado por COCO para almacenar anotaciones y también el formato en que suelen exportarse las predicciones de los modelos.

Dataset de imágenes adecuadas

Los criterios seleccionados e implementados en el script de selección de este subconjunto son que las imágenes más adecuadas para estimación de posturas son las que tienen una persona cuya caja (*bounding box*) ocupa un mínimo del 25% del ancho de la imagen o un 75% del alto de la imagen (**implica que la persona puede estar cerca** en el plano) y además el centro de la caja de la persona no está más alejado de un 15% del centro de la imagen (**implica que la persona está centrada en la imagen**).

Dataset de imágenes no adecuadas

A su vez los criterios seleccionados e implementados en el script de selección de este subconjunto son que las imágenes no adecuadas para estimación de posturas son las que tienen una persona cuya caja (*bounding box*) ocupa menos del 25% del ancho o del alto de la imagen (**implica que la persona puede no estar cerca** en el plano) o el centro de la caja de la persona está más alejado de un 25% del centro de la imagen (**implica que la persona no está centrada en la imagen**).

7.1.1. Resultados obtenidos de precisión AP (Average Precision)

A continuación se presentan los resultados obtenidos de AP para cada modelo en el dispositivo **Samsung Galaxy M32** con los tres subconjuntos de datos mediante gráficas de barras comparativas que permiten visualizar la precisión relativa de cada variante y familia con cada dataset, ordenadas por dataset de menor a mayor precisión obtenida.

Gráfica con los resultados de precisión (AP) para el modelo **MoveNet Lightning 8**.

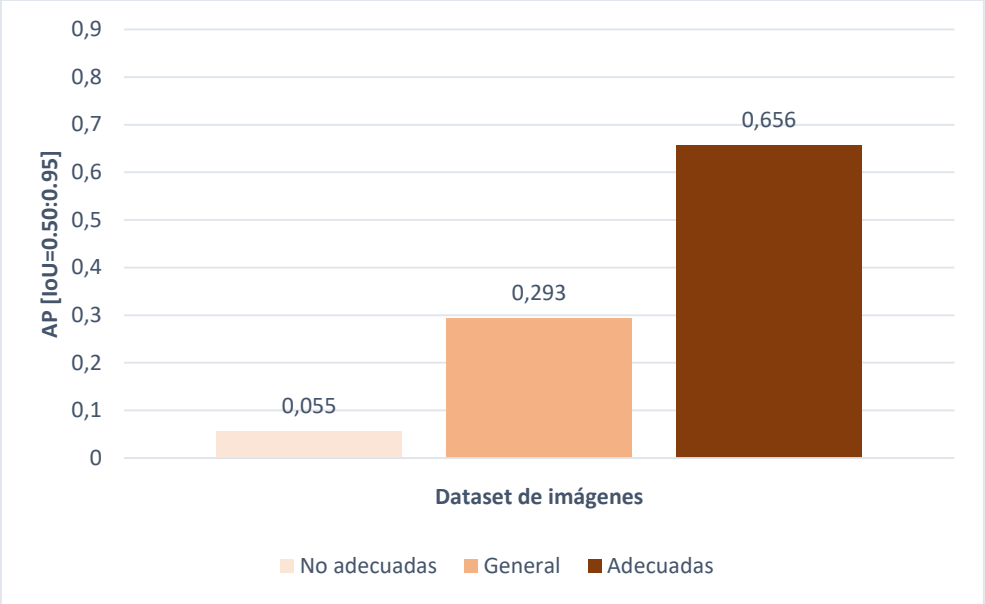


Imagen 23. AP por dataset del modelo MoveNet Lightning 8

Gráfica con los resultados de precisión (AP) para el modelo **MoveNet Lightning 16**.

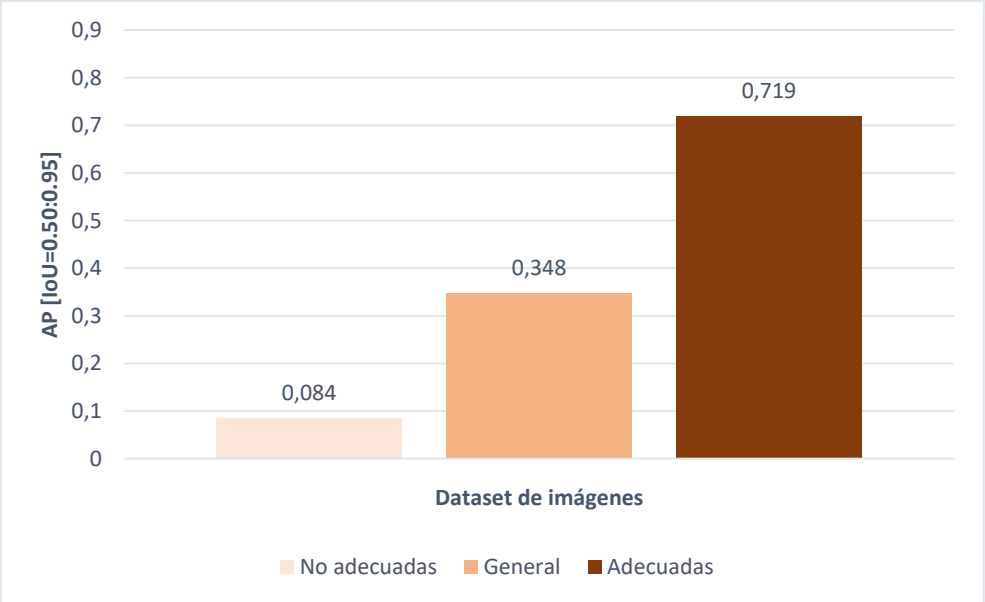


Imagen 24. AP por dataset del modelo MoveNet Lightning 16

Gráfica con los resultados de precisión (AP) para el modelo **MoveNet Lightning 32**.

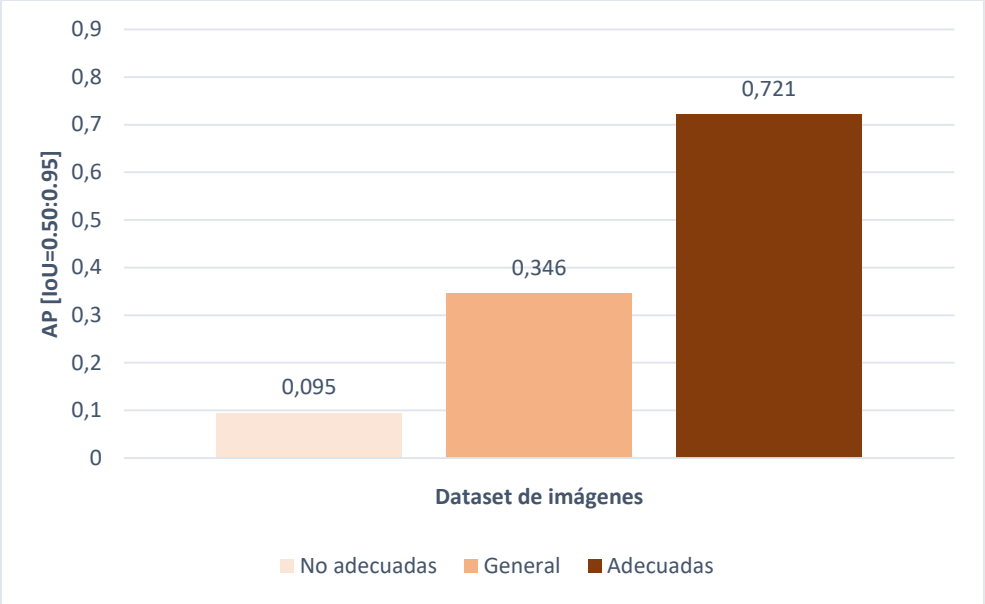


Imagen 25. AP por dataset del modelo MoveNet Lightning 32

Gráfica con los resultados de precisión (AP) para el modelo **MoveNet Thunder 8**.

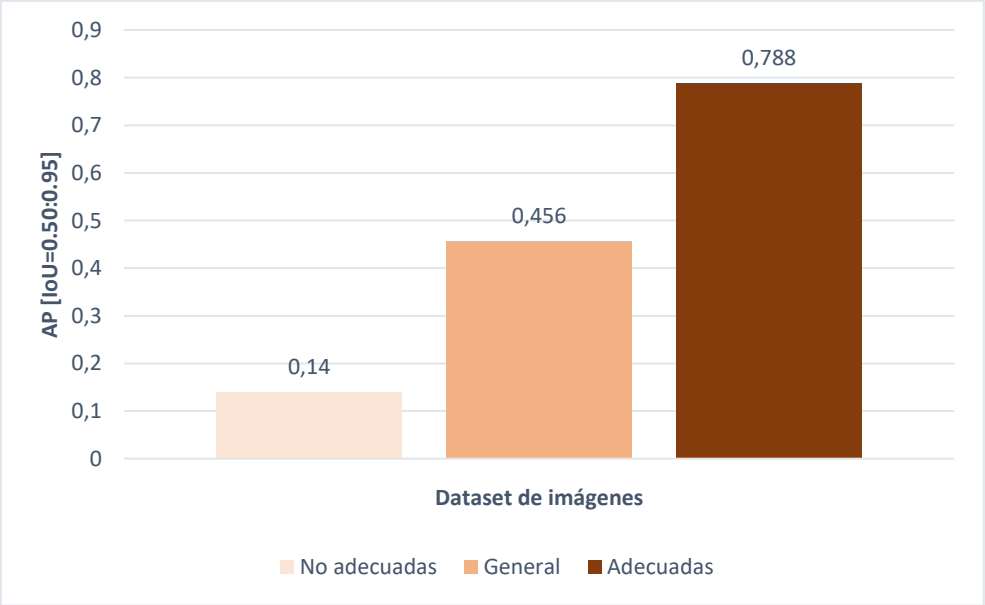


Imagen 26. AP por dataset del modelo MoveNet Thunder 8

Gráfica con los resultados de precisión (AP) para el modelo **MoveNet Thunder 16**.

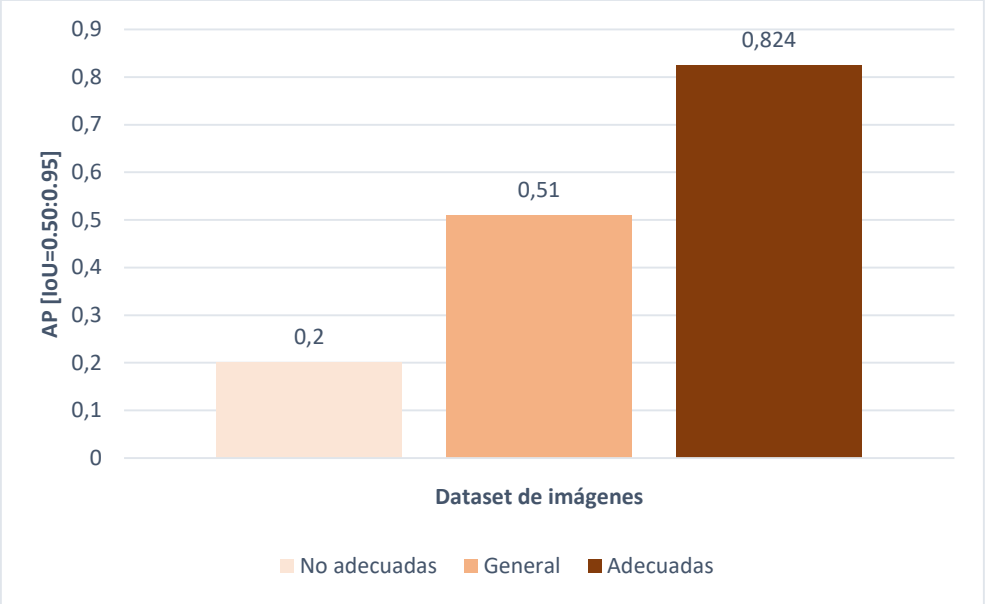


Imagen 27. AP por dataset del modelo MoveNet Thunder 16

Gráfica con los resultados de precisión (AP) para el modelo **MoveNet Thunder 32**.

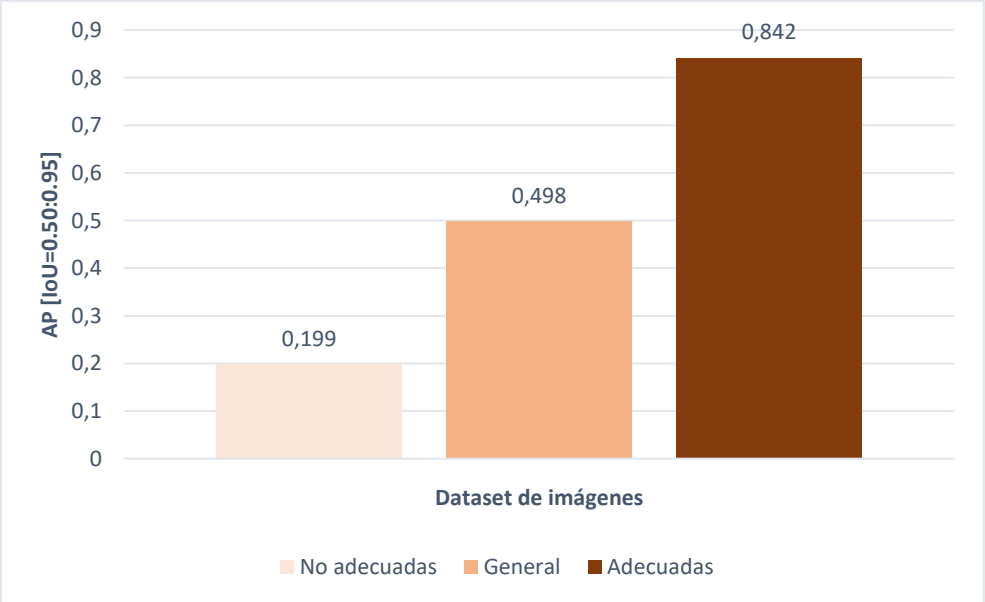


Imagen 28. AP por dataset del modelo MoveNet Thunder 32

Gráfica con los resultados de precisión (AP) para el modelo **BlazePose Lite**.

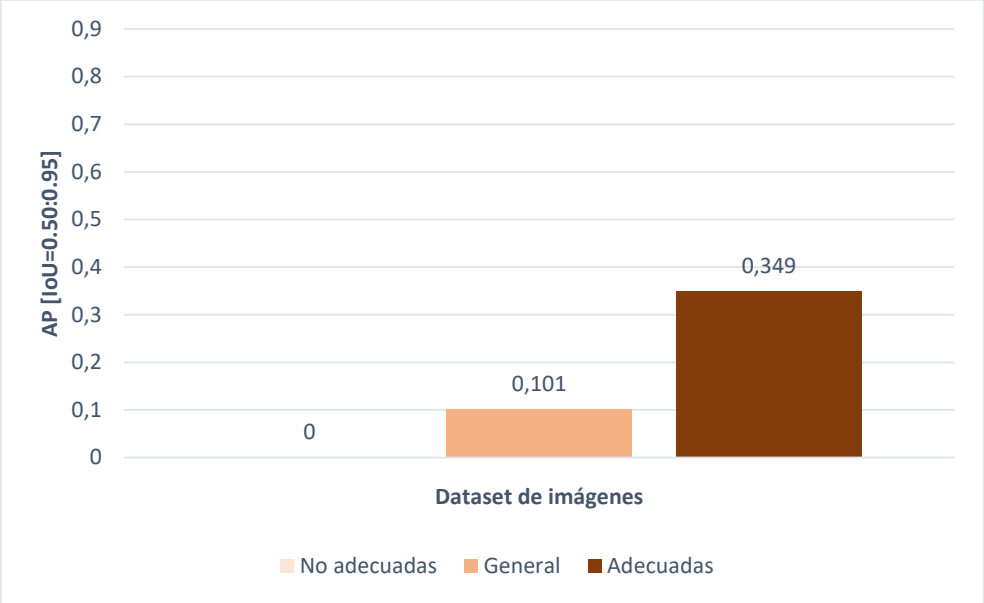


Imagen 29. AP por dataset del modelo BlazePose Lite

Gráfica con los resultados de precisión (AP) para el modelo **BlazePose Full**.

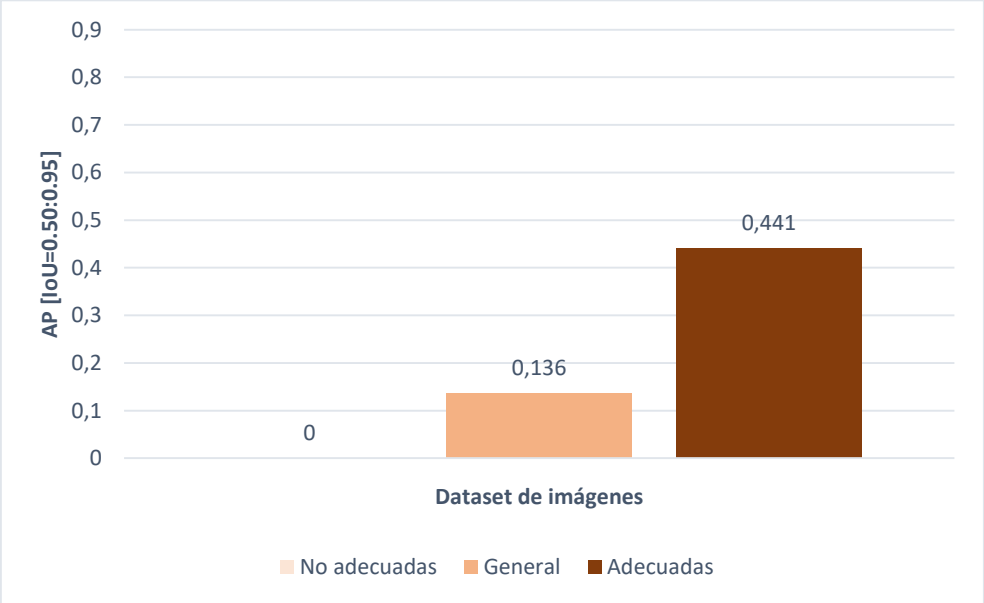


Imagen 30. AP por dataset del modelo BlazePose Full

Gráfica con los resultados de precisión (AP) para el modelo **BlazePose Heavy**.

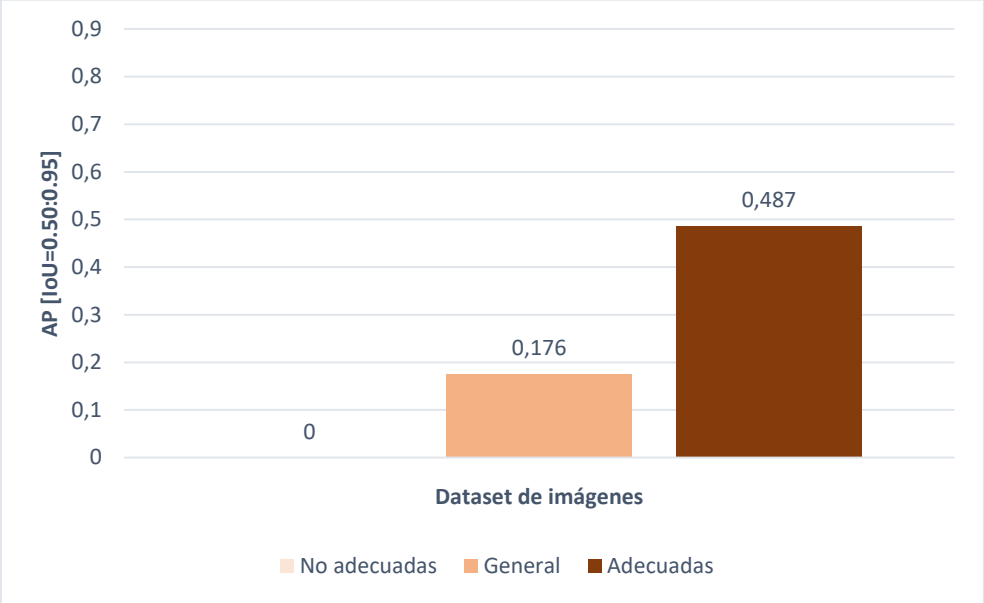


Imagen 31. AP por dataset del modelo BlazePose Heavy

Gráfica con los resultados de precisión (AP) para el modelo **Yolo8-pose Nano**.

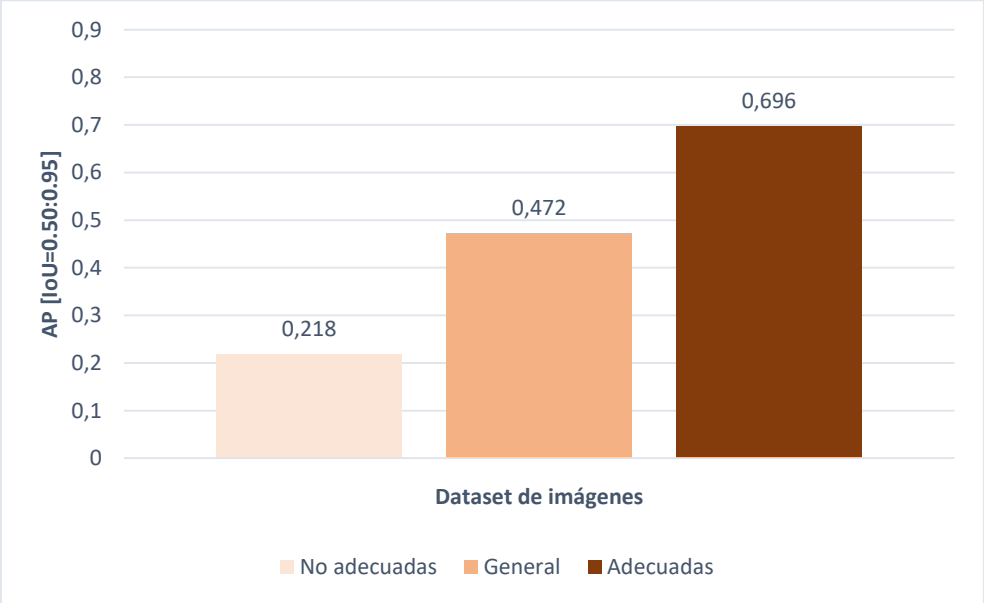


Imagen 32. AP por dataset del modelo Yolo8-pose Nano

Gráfica con los resultados de precisión (AP) para el modelo **Yolo8-pose Small**.

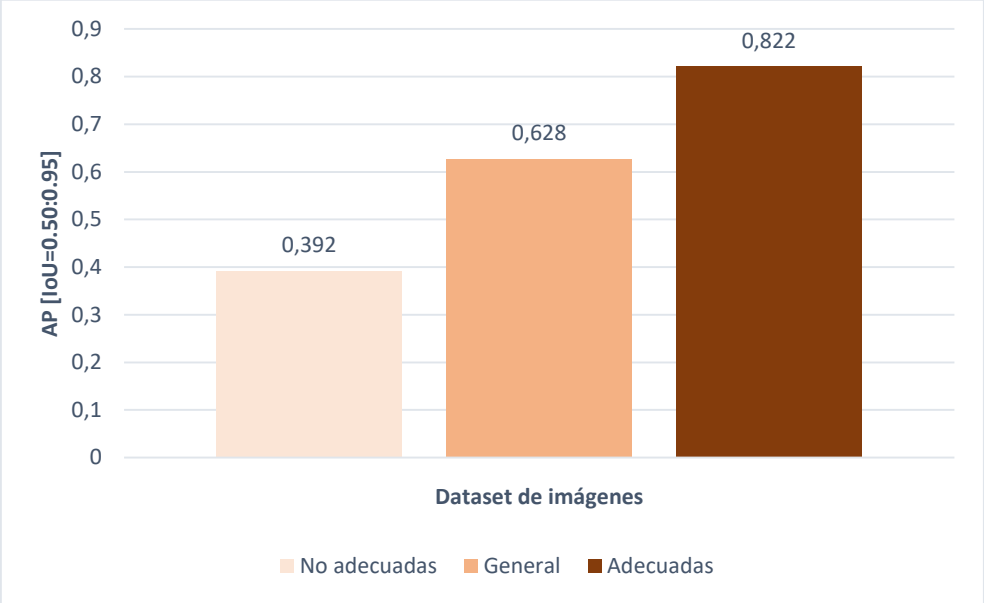


Imagen 33. AP por dataset del modelo Yolo8-pose Small

Gráfica con los resultados de precisión (AP) para el modelo **Yolo8-pose Medium**.

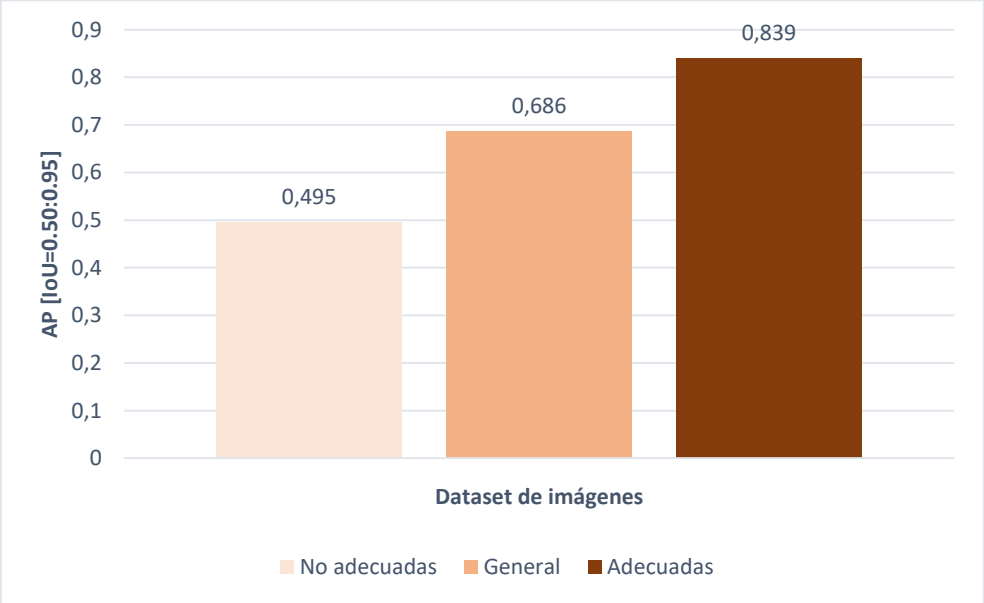
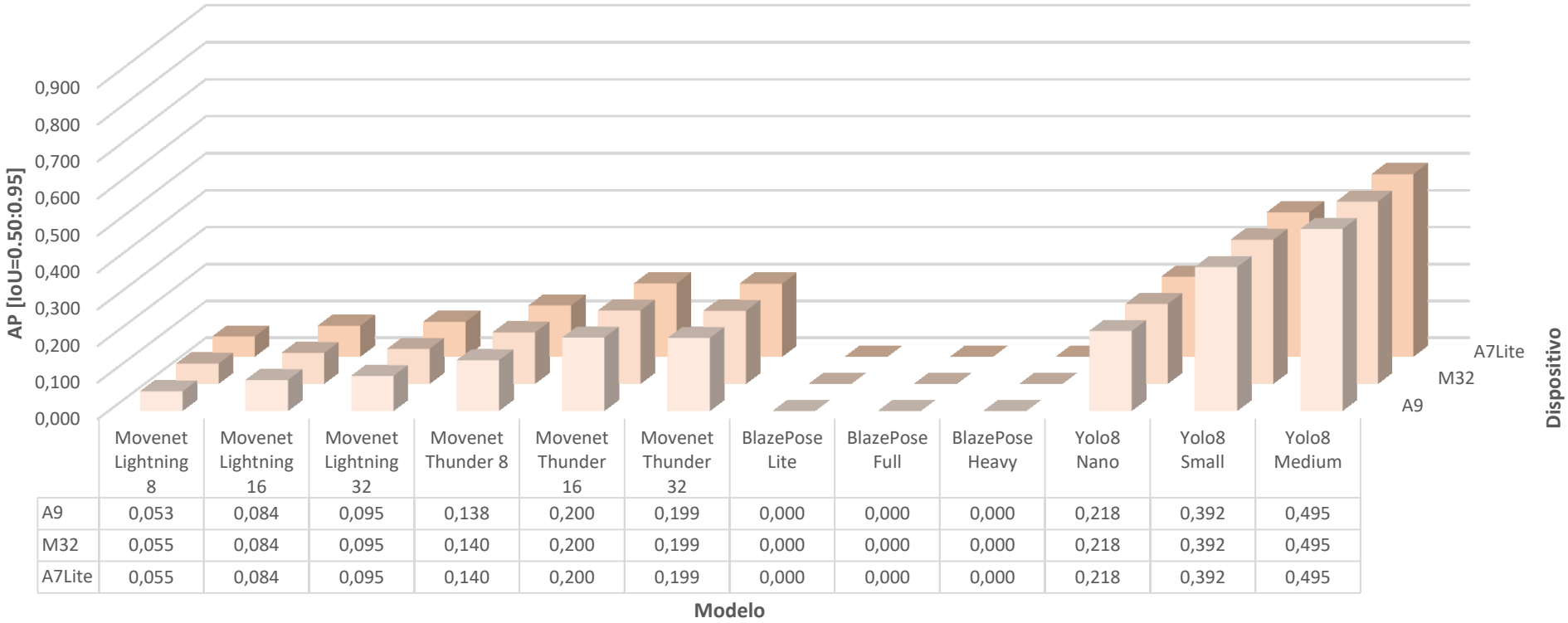


Imagen 34. AP por dataset del modelo Yolo8-pose Medium

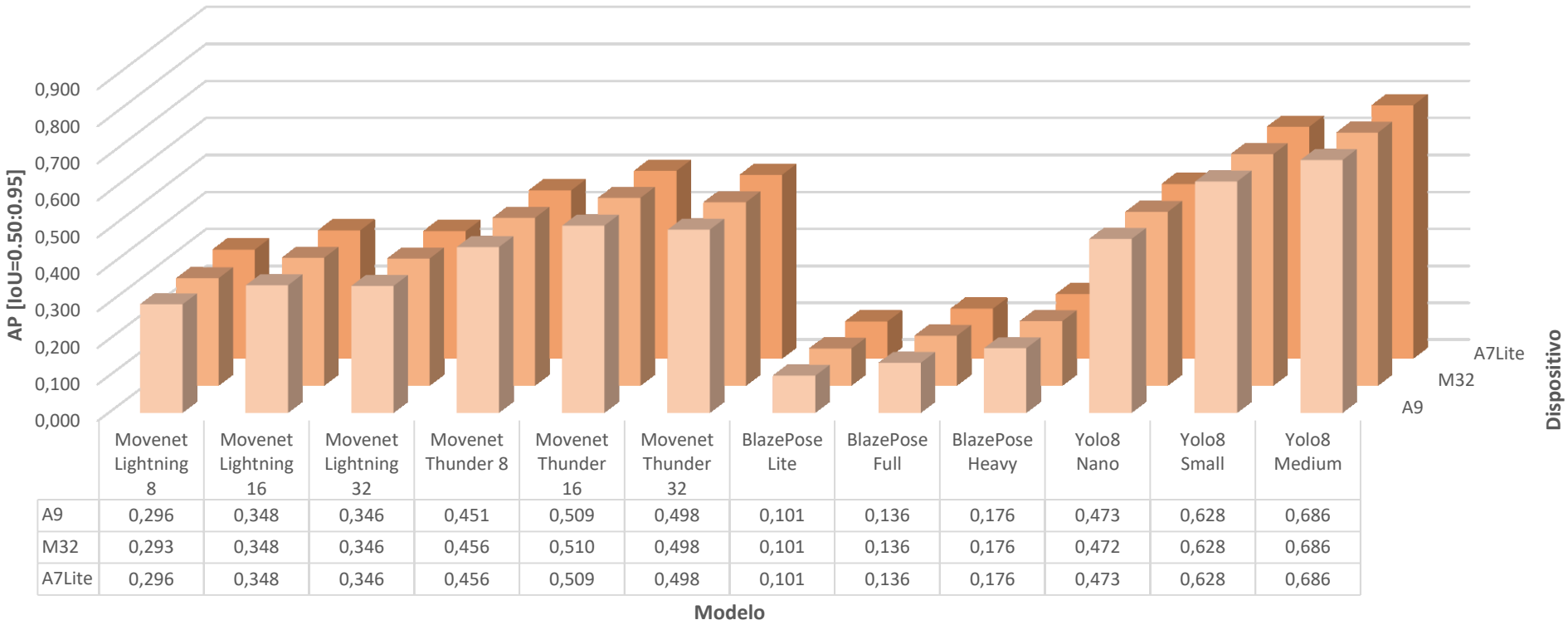
Gráfica comparativa de **precisión** (AP [IoU=0.50:0.95]) **por modelo** en diferentes dispositivos para el dataset de imágenes **inadecuadas**.



	Movenet Lightning 8	Movenet Lightning 16	Movenet Lightning 32	Movenet Thunder 8	Movenet Thunder 16	Movenet Thunder 32	BlazePose Lite	BlazePose Full	BlazePose Heavy	Yolo8 Nano	Yolo8 Small	Yolo8 Medium
A9	0,053	0,084	0,095	0,138	0,200	0,199	0,000	0,000	0,000	0,218	0,392	0,495
M32	0,055	0,084	0,095	0,140	0,200	0,199	0,000	0,000	0,000	0,218	0,392	0,495
A7Lite	0,055	0,084	0,095	0,140	0,200	0,199	0,000	0,000	0,000	0,218	0,392	0,495

Imagen 35. Comparativa precisión/modelos por dispositivo con imágenes inadecuadas

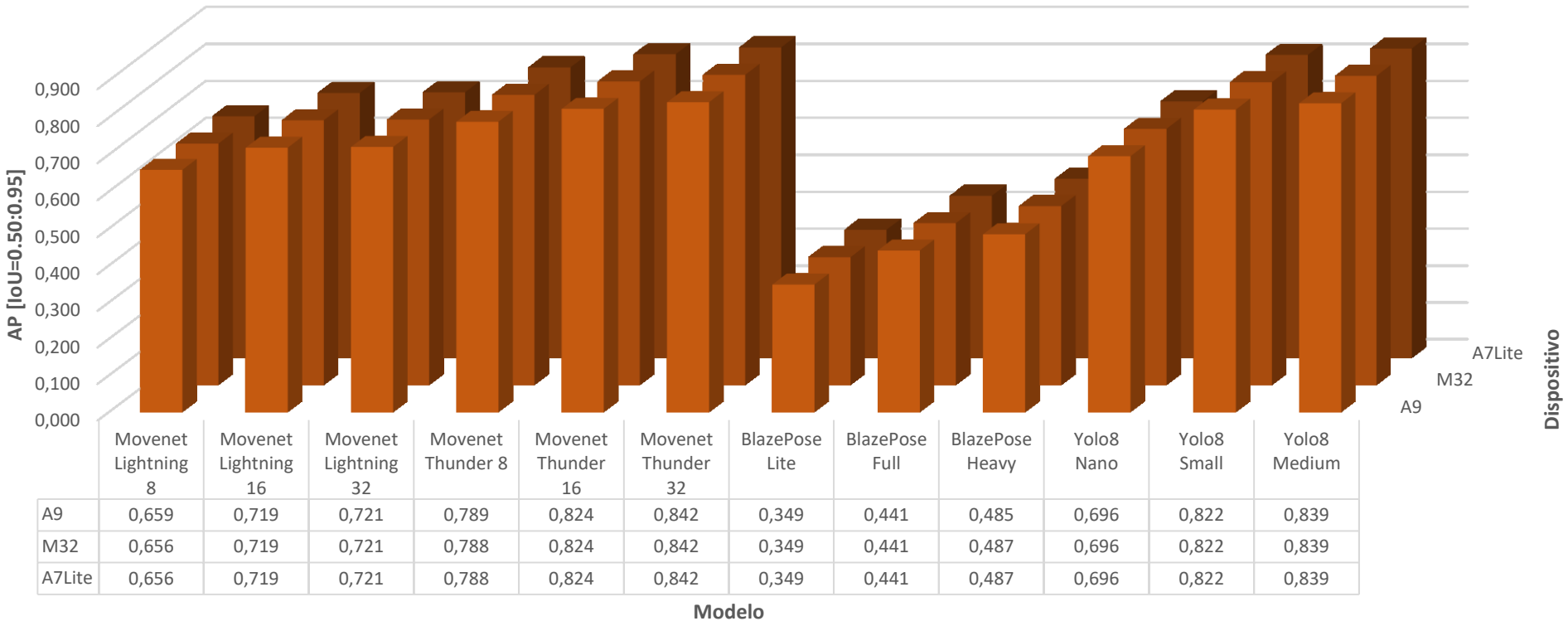
Gráfica comparativa de **precisión** (AP [IoU=0.50:0.95]) **por modelo** en diferentes dispositivos para el dataset **general** de imágenes.



	Movenet Lightning 8	Movenet Lightning 16	Movenet Lightning 32	Movenet Thunder 8	Movenet Thunder 16	Movenet Thunder 32	BlazePose Lite	BlazePose Full	BlazePose Heavy	Yolo8 Nano	Yolo8 Small	Yolo8 Medium
A9	0,296	0,348	0,346	0,451	0,509	0,498	0,101	0,136	0,176	0,473	0,628	0,686
M32	0,293	0,348	0,346	0,456	0,510	0,498	0,101	0,136	0,176	0,472	0,628	0,686
A7Lite	0,296	0,348	0,346	0,456	0,509	0,498	0,101	0,136	0,176	0,473	0,628	0,686

Imagen 36. Comparativa precisión/modelos por dispositivo con dataset general

Gráfica comparativa de **precisión** (AP [IoU=0.50:0.95]) **por modelo** en diferentes dispositivos para el dataset de imágenes **adecuadas**.



	Movenet Lightning 8	Movenet Lightning 16	Movenet Lightning 32	Movenet Thunder 8	Movenet Thunder 16	Movenet Thunder 32	BlazePose Lite	BlazePose Full	BlazePose Heavy	Yolo8 Nano	Yolo8 Small	Yolo8 Medium
A9	0,659	0,719	0,721	0,789	0,824	0,842	0,349	0,441	0,485	0,696	0,822	0,839
M32	0,656	0,719	0,721	0,788	0,824	0,842	0,349	0,441	0,487	0,696	0,822	0,839
A7Lite	0,656	0,719	0,721	0,788	0,824	0,842	0,349	0,441	0,487	0,696	0,822	0,839

Imagen 37. Comparativa precisión/modelos por dispositivo con imágenes adecuadas

7.1.2. Análisis resultados precisión

En cuanto al desempeño comparativo entre familias de modelos, se observa que los modelos de la familia **YOLO son los que alcanzan una mejor precisión**. Estos modelos destacan por su robustez frente a variaciones en la posición de la persona y por la capacidad de mantener niveles de desempeño relativamente altos incluso cuando las condiciones de las imágenes no son óptimas (Imagen 35). En el extremo opuesto, los modelos de la familia **BlazePose presentan el peor comportamiento**, evidenciando mayores dificultades en la identificación de keypoints en escenarios desfavorables y ofreciendo resultados inferiores de manera consistente en todos los subconjuntos evaluados.

El pobre desempeño de los modelos de la familia BlazePose sobre todo con imágenes menos adecuadas (obtienen un 0 de precisión) (Imagen 35) puede ser debido a que estos estén diseñados para ser utilizados junto con otras herramientas de preprocesado previo de las imágenes (identificación de la persona en la imagen, crop de área y centrado, etc,..) ya que como su propia documentación indica quedan fuera de su alcance "Personas demasiado alejadas de la cámara (p. ej., a más de 4 metros)" o imágenes donde "La cabeza no es visible".

Un hallazgo especialmente interesante se observa en los modelos de la familia **MoveNet**. Si bien su desempeño general puede considerarse intermedio, son los modelos que **muestran una mayor sensibilidad a la idoneidad de las imágenes** empleadas. Esto se traduce en una marcada diferencia de precisión entre los subconjuntos adecuados e inadecuados, los modelos MoveNet mejoran significativamente cuando se trabaja con imágenes donde la persona está centrada y correctamente representada, pero degradan su precisión de forma acusada en presencia de condiciones adversas. Este comportamiento pone de manifiesto que, aunque la arquitectura MoveNet está optimizada para dispositivos móviles y entornos de inferencia en tiempo real, su desempeño se ve afectado por la calidad de entrada de los datos visuales.

En relación con el entorno de ejecución, los experimentos confirman algo lógico y esperado, que **la precisión obtenida por los modelos no varía en función del dispositivo** (Imagen 35, Imagen 36 e Imagen 37) en el que se lleva a cabo la inferencia. Este resultado era previsible, dado que la precisión está determinada por la arquitectura del modelo y el algoritmo de inferencia, y no por las características del hardware en el que se ejecuta. El dispositivo podría afectar de forma clara al rendimiento temporal (tiempo de inferencia por imagen), pero no a la exactitud de los keypoints detectados.

Resumen

En las gráficas de resultados se observa a simple vista que si bien como era de esperar la precisión es independiente del dispositivo en que se ejecute, ésta si está altamente influenciada por las condiciones visuales de las imágenes de entrada.

Los modelos de mayor complejidad (YOLOv8-Pose versiones small y medium) ofrecen una mejor generalización, mientras que los modelos ligeros sufren caídas significativas de precisión en escenarios adversos, más acentuadas en los modelos de la familia BlazePose que en los de la familia MoveNet.

7.2. Resultados obtenidos de rendimiento

En este apartado se presentan los **resultados de rendimiento (en tiempos de inferencia)** de cada modelo medido en tres dispositivos representativos de gamas medias y bajas del mercado, todos con sistema operativo Android y arquitectura ARM y con las características que vimos en el apartado "5.3.1. Listado de dispositivos de prueba":

- *Samsung Galaxy Tab A7 Lite (Tablet)*
  Procesador: MediaTek Helio P22T, Memoria: 3GB, Versión de android: 14

- *Samsung Galaxy M32 (Móvil)*
  Procesador: MediaTek Helio G80, Memoria: 6GB, Versión de android: 13

- *Samsung Galaxy Tab A9 (Tablet)*
  Procesador: MediaTek Helio G99, Memoria: 4GB, Versión de android: 15

Para realizar esta evaluación se han tomado de los ficheros ZIP de resultados exportados de la ejecución de los tres dispositivos de prueba **los ficheros relativos a tiempos de inferencia de cada modelo** (ficheros de texto con formato CSV) por lo que se ha utilizado los tiempos de inferencia de cada modelo para el **dataset general de 316 imágenes** extraídas del dataset COCO (filtradas con las condiciones ya descritas). Sobre los tiempos obtenidos por cada modelo para cada imagen se calculan los tiempos medios de inferencia por cada modelo y los tiempos totales del proceso completo por modelo y en base a ellos se elaboran las gráficas de este apartado.

El objetivo es determinar la viabilidad real de cada modelo para ser utilizado en dispositivos móviles Android, teniendo en cuenta las limitaciones de CPU, GPU y memoria de cada terminal.

7.2.1. Gráficas comparativas de tiempos de estimación

En este apartado se presentan las gráficas del rendimiento en tiempo de ejecución, centrado en la latencia de inferencia o **tiempo medio de procesamiento por imagen** (expresado en **milisegundos**). Para obtener estos datos se utiliza la medición de la operación de inferencia que hace el propio interprete de la API de TensorFlow Lite mediante el método `getLastNativeInferenceDurationNanoseconds()` (como vimos en el apartado "6.2.3. Implementación de las clases de la aplicación") y que hemos recopilado en los ficheros de salida relativos a tiempos de inferencia.

En las siguientes gráficas comparativas para una mejor visualización se han querido agrupar por colores las familias a las que pertenece cada modelo:

- MoveNet en azul.

- BlazePose en verde.

- Yolo8-pose en gris.

Gráfica con resultados de rendimiento para la tablet **Samsung Galaxy Tab A7 Lite**.

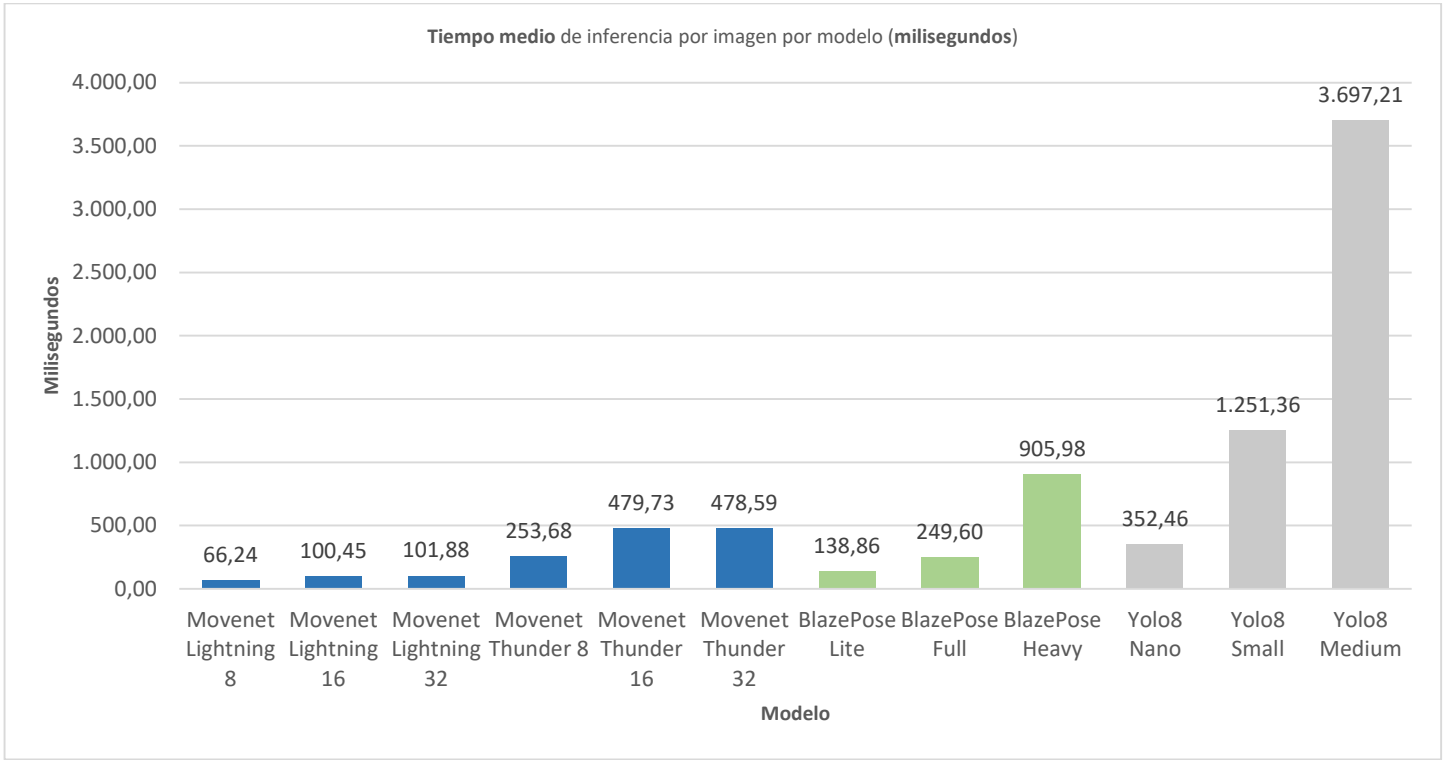


Imagen 38. Tiempo medio inferencia Samsung Galaxy Tab A7 Lite

Gráfica con resultados de rendimiento para el teléfono móvil **Samsung Galaxy M32**.

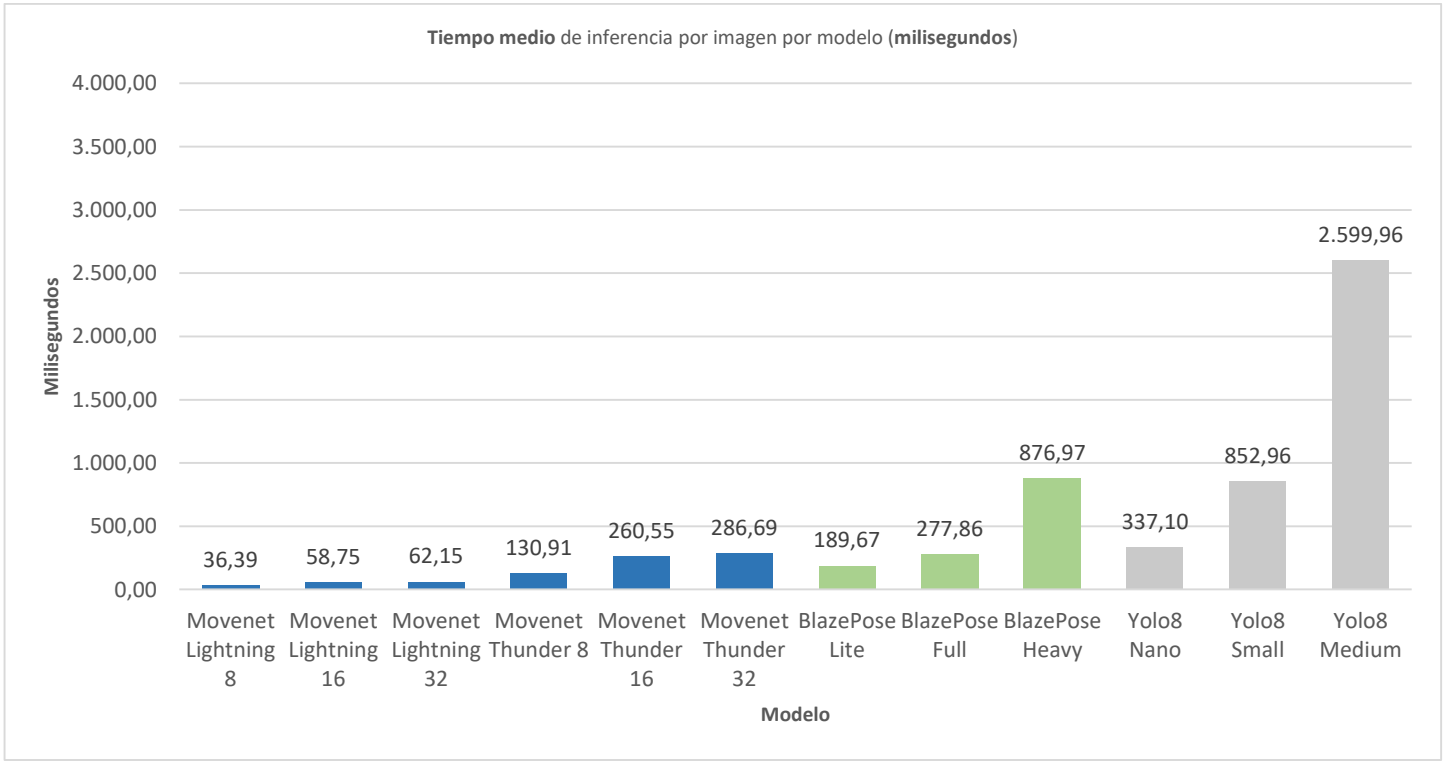


Imagen 39. Tiempo medio inferencia Samsung Galaxy M32

Gráfica con resultados de rendimiento para la tablet **Samsung Galaxy Tab A9**.

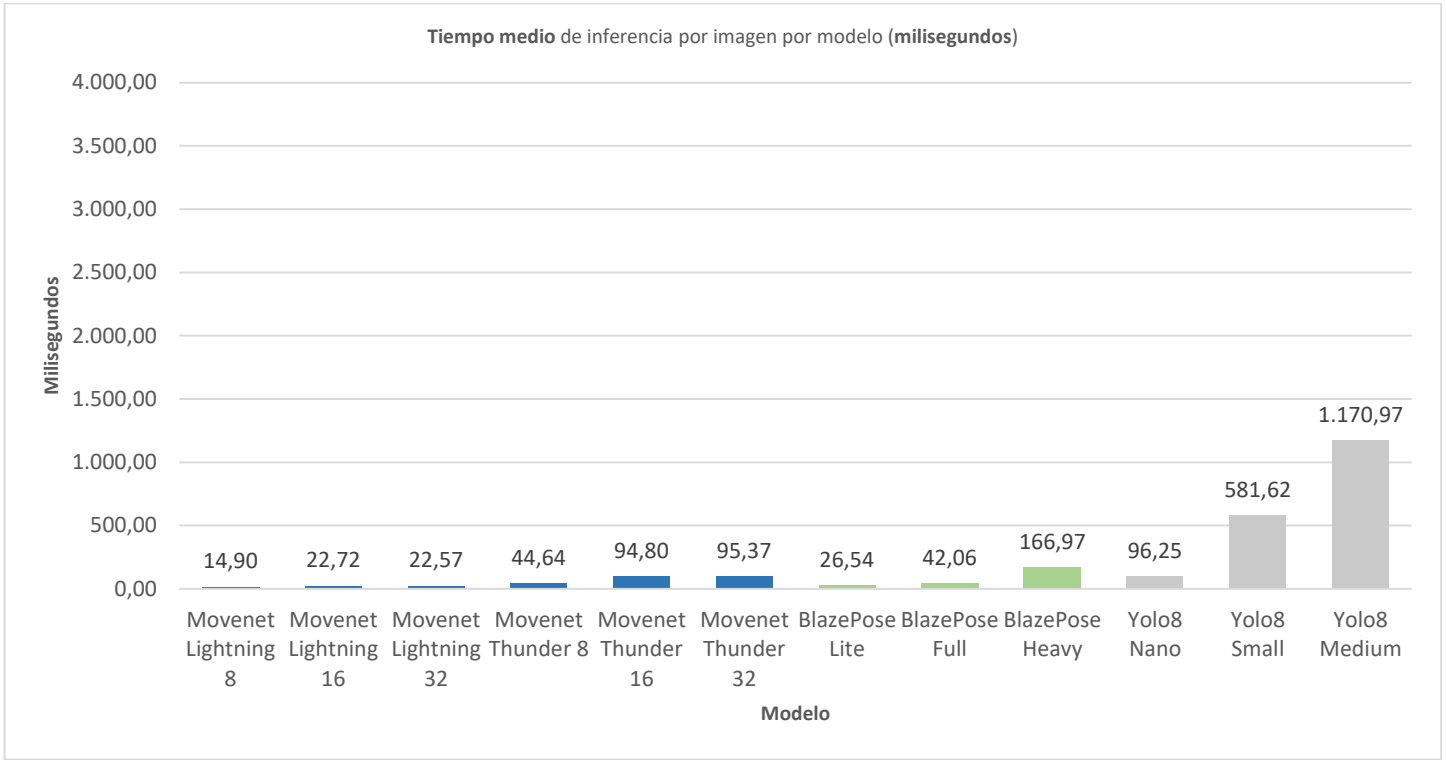
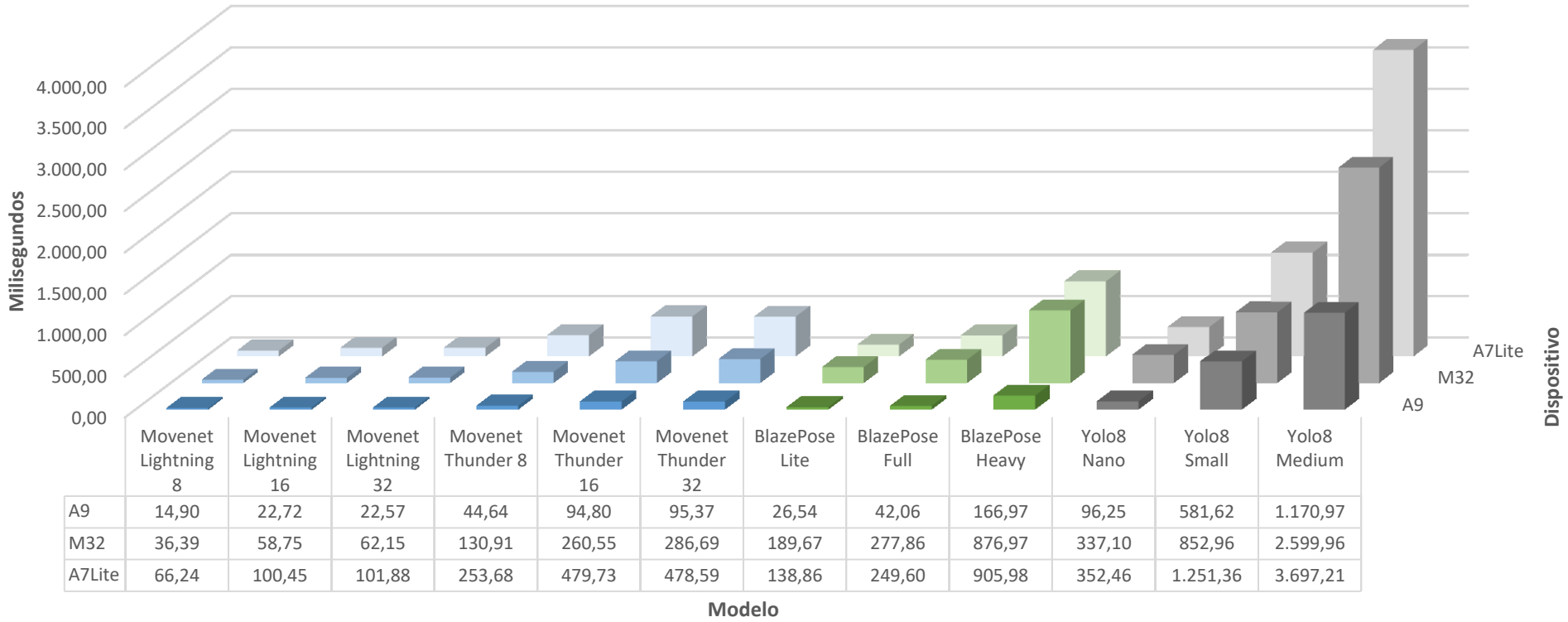


Imagen 40. Tiempo medio inferencia Samsung Galaxy Tab A9

Comparativa de **tiempo MEDIO** de inferencia (en **milisegundos**) **por imagen** del **dataset general** por modelo en diferentes dispositivos.



	Movenet Lightning 8	Movenet Lightning 16	Movenet Lightning 32	Movenet Thunder 8	Movenet Thunder 16	Movenet Thunder 32	BlazePose Lite	BlazePose Full	BlazePose Heavy	Yolo8 Nano	Yolo8 Small	Yolo8 Medium
A9	14,90	22,72	22,57	44,64	94,80	95,37	26,54	42,06	166,97	96,25	581,62	1.170,97
M32	36,39	58,75	62,15	130,91	260,55	286,69	189,67	277,86	876,97	337,10	852,96	2.599,96
A7Lite	66,24	100,45	101,88	253,68	479,73	478,59	138,86	249,60	905,98	352,46	1.251,36	3.697,21

Imagen 41. Comparativa tiempos medio inferencia por modelo por dispositivo

Aunque las siguientes gráficas pudieran considerarse en cierto modo redundantes respecto a los análisis anteriores centrados en los tiempos medios de inferencia por imagen, se ha considerado oportuno incluir también la representación de los **tiempos totales de ejecución** que requirió cada modelo al procesar el conjunto completo de imágenes del dataset general (316 imágenes filtradas del dataset COCO, correspondientes a escenarios de una sola persona y con al menos 15 keypoints visibles).

Esta inclusión permite complementar la perspectiva de los tiempos promedio por imagen con una visión global del coste temporal agregado, lo que resulta especialmente relevante en aplicaciones reales donde no se procesan imágenes de manera aislada, sino lotes completos de datos. Por otro lado, facilita la comparación directa entre dispositivos de prueba, ya que los tiempos totales reflejan con claridad las diferencias de rendimiento cuando la carga de trabajo se mantiene constante para todos los modelos.

En estas gráficas se presentan los tiempos totales consumidos por cada modelo al ejecutar la inferencia sobre el dataset completo de testeo. A diferencia de las gráficas anteriores (donde la métrica principal eran los milisegundos por imagen) en esta ocasión **los valores se expresan en segundos**, dado que se trata de intervalos de tiempo considerablemente más elevados. Este cambio de escala responde a la necesidad de presentar resultados más legibles y comprensibles, evitando una precisión excesiva que no aporta valor analítico en este contexto.

Gráfica con tiempo total de inferencia en la tablet **Samsung Galaxy Tab A7 Lite**.

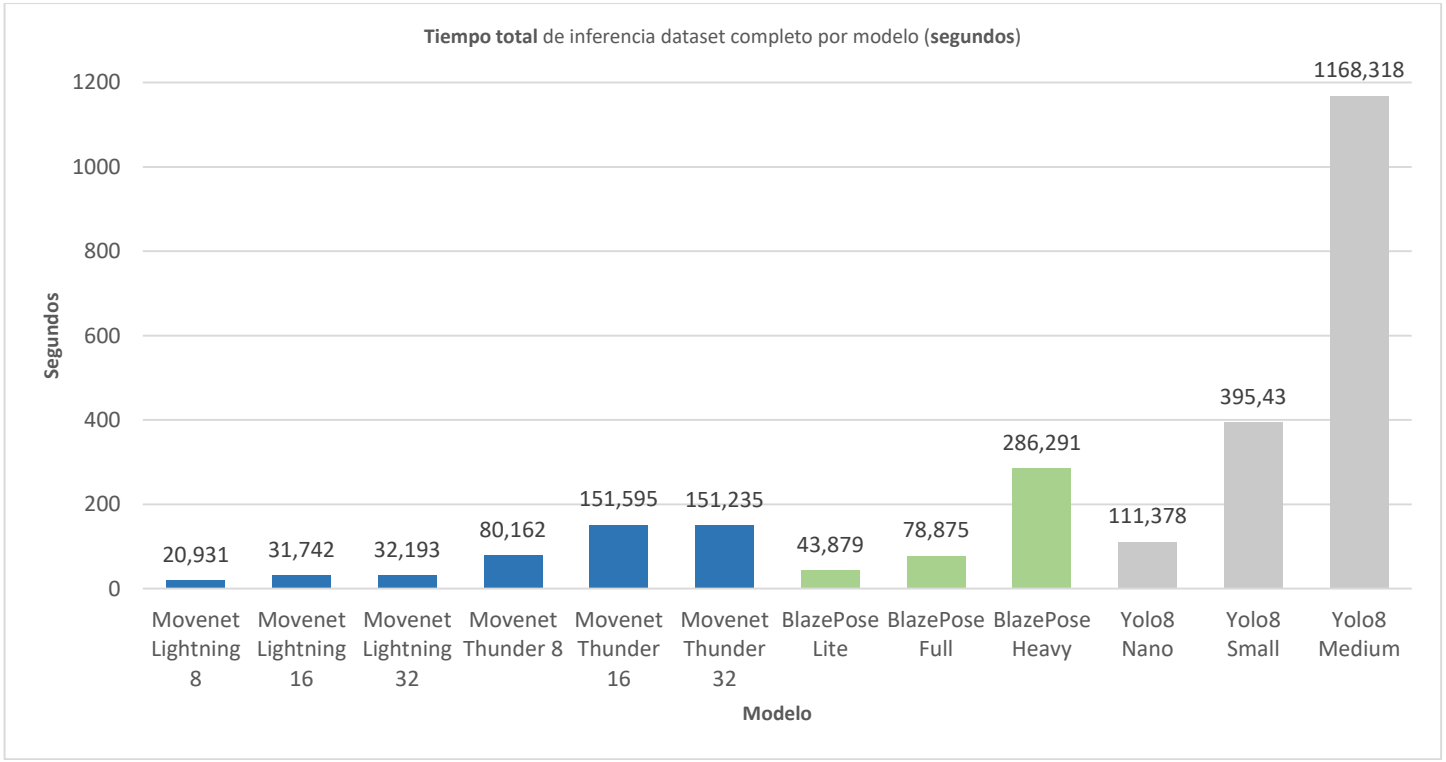


Imagen 42. Tiempo total inferencia Samsung Galaxy Tab A7 Lite

Gráfica con tiempo total de inferencia en el móvil **Samsung Galaxy M32**.

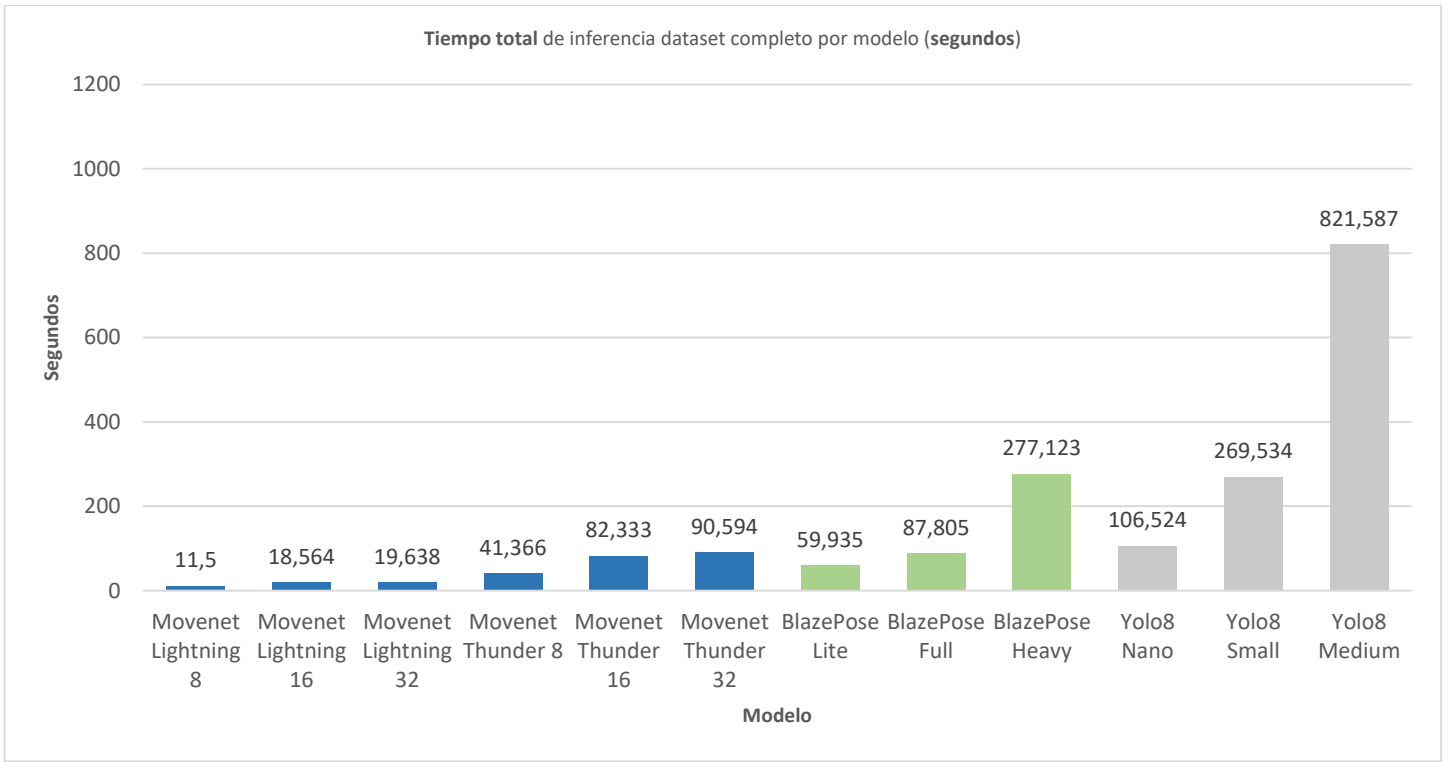


Imagen 43. Tiempo total inferencia Samsung Galaxy M32

Gráfica con tiempo total de inferencia en la tablet **Samsung Galaxy Tab A9**.

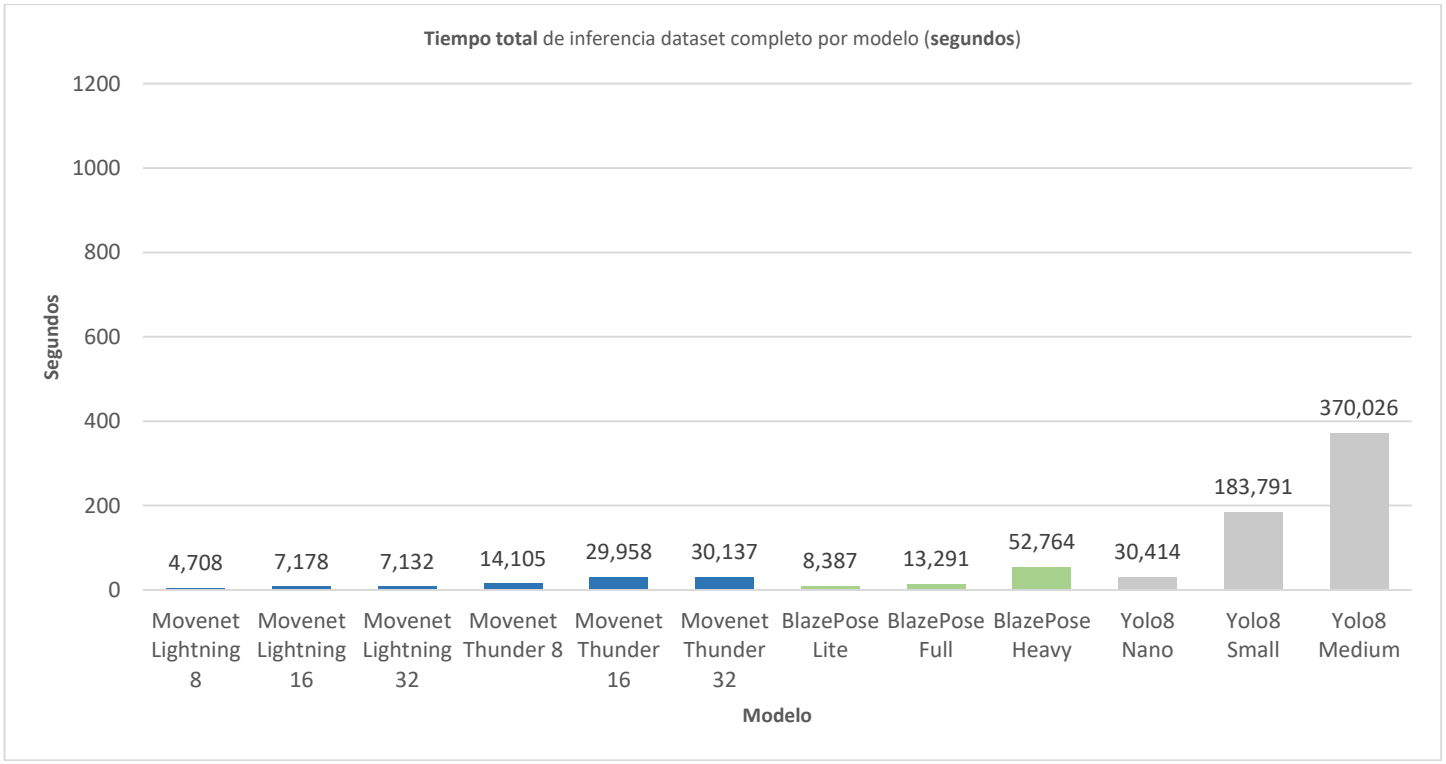
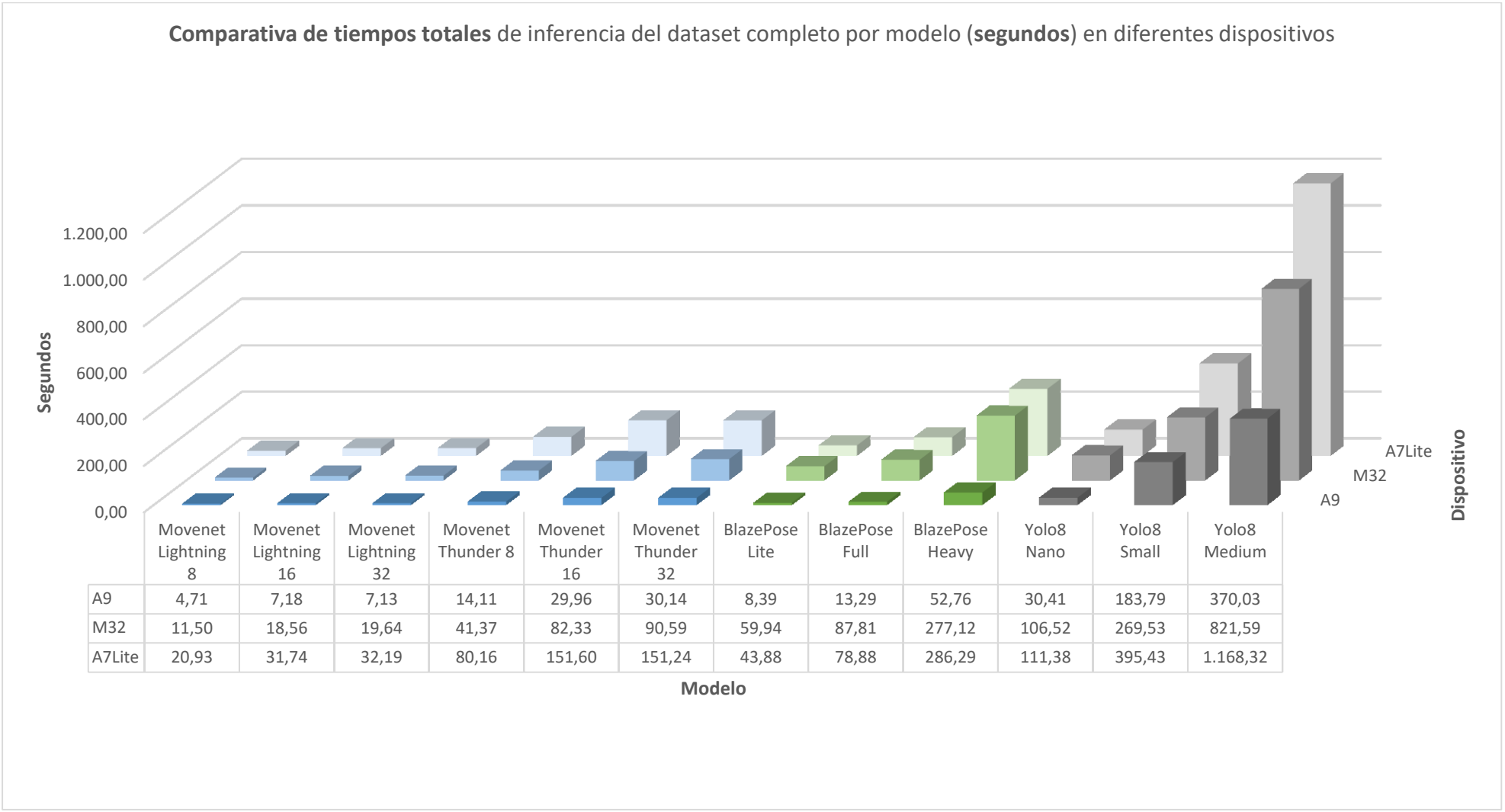


Imagen 44. Tiempo total inferencia Samsung Galaxy Tab A9

Gráfica comparativa de **tiempo TOTAL** de inferencia (en **segundos**) del **dataset general** por modelo en diferentes dispositivos.



	Movenet Lightning 8	Movenet Lightning 16	Movenet Lightning 32	Movenet Thunder 8	Movenet Thunder 16	Movenet Thunder 32	BlazePose Lite	BlazePose Full	BlazePose Heavy	Yolo8 Nano	Yolo8 Small	Yolo8 Medium
A9	4,71	7,18	7,13	14,11	29,96	30,14	8,39	13,29	52,76	30,41	183,79	370,03
M32	11,50	18,56	19,64	41,37	82,33	90,59	59,94	87,81	277,12	106,52	269,53	821,59
A7Lite	20,93	31,74	32,19	80,16	151,60	151,24	43,88	78,88	286,29	111,38	395,43	1.168,32

Imagen 45. Comparativa tiempo total inferencia por modelo por dispositivo

7.2.2. Análisis resultados rendimiento

La comparación de los rendimientos obtenidos por los modelos en los distintos dispositivos de prueba permite extraer varias conclusiones acerca de su comportamiento y eficiencia. En términos generales, los resultados muestran que, aunque todos los modelos tienen una tendencia esperada (el rendimiento de todos los modelos mejora de forma clara cuando se ejecutan en dispositivos con mayor capacidad de procesamiento), las diferencias entre ellos son notables y permiten establecer tres grupos en función de su rendimiento relativo:

- Modelos con **bajo rendimiento**. Se identifican algunos modelos que, independientemente del dispositivo utilizado, presentan tiempos de inferencia significativamente superiores al resto (Imagen 41 e Imagen 45). Dentro de este grupo se encuentran **YOLOv8-Pose Small, YOLOv8-Pose Medium y BlazePose Heavy**, cuya complejidad estructural los convierte en opciones poco adecuadas para dispositivos móviles o de recursos limitados.

- Modelos con **rendimiento intermedio**. Un segundo grupo lo conforman aquellos que ofrecen un rendimiento aceptable, aunque no sobresaliente. Entre ellos se encuentran las tres versiones de la familia **MoveNet Thunder, junto con BlazePose Full y YOLOv8-Pose Nano**. Estos modelos representan una solución equilibrada, con tiempos de inferencia moderados y una viabilidad de uso razonable en la mayoría de escenarios, aunque sin alcanzar la agilidad de los más eficientes.

- Modelos con **alto rendimiento**. Finalmente, destacan los modelos con los mejores tiempos de inferencia y mayor consistencia en todos los dispositivos. Este grupo está formado por las tres versiones de **MoveNet Lightning y por BlazePose Lite**, que se posicionan como las alternativas más ligeras y rápidas, adecuadas para aplicaciones en tiempo real y entornos de hardware limitado.

Familia MoveNet Lightning

Los modelos MoveNet Lightning se posicionan como **los más eficientes en cuanto a velocidad de ejecución**. En todos los dispositivos utilizados para las pruebas, independientemente de su capacidad de procesamiento, esta variante se mantuvo como la más rápida, confirmando su idoneidad para aplicaciones en tiempo real. Además, cabe resaltar que el rendimiento de Lightning es consistente: el cambio de dispositivo apenas afecta a los tiempos de inferencia, lo que indica una arquitectura altamente optimizada y con un coste computacional estable. Esto los convierte en candidatos idóneos para integraciones móviles donde los recursos de hardware son limitados.

Familia MoveNet Thunder

Los modelos MoveNet Thunder presentan un **rendimiento intermedio**. En particular, la versión cuantizada a int8 logra unos tiempos de inferencia competitivos, siendo la que muestra los mejores resultados dentro de esta familia. En contraste, las versiones en float16 y float32 empeoran ligeramente en rendimiento, probablemente debido al mayor coste computacional asociado al manejo de mayor precisión numérica. Aunque siguen siendo relativamente rápidas, muestran una sensibilidad más marcada a las limitaciones de hardware

que Lightning.

Familia BlazePose

Los modelos de la familia BlazePose obtienen un **rendimiento medio aceptable en las versiones Lite y Full**. Sin embargo, **la versión Heavy dispara sus tiempos de ejecución** de manera notable, situándose como la tercera peor opción por detrás de los modelos Small y Medium de Yolo. Esta diferencia refleja el alto coste computacional de las arquitecturas pesadas, que si bien pueden aportar mejoras de precisión en determinados escenarios, resultan menos prácticas para su uso en dispositivos con recursos limitados.

Familia YOLO-Pose (v8)

La familia YOLO muestra una marcada disparidad en su rendimiento según la variante, la versión **Nano** es la única que obtiene **tiempos de inferencia medio aceptables**, permitiendo pensar en posibles aplicaciones móviles con ciertas restricciones. La versión **Small** experimenta **tiempos de inferencia muy elevados**, alejándose de los valores prácticos requeridos para aplicaciones en tiempo real. La versión **Medium** presenta **tiempos de inferencia desproporcionados** en la mayoría de los dispositivos con respecto al resto de modelos. No obstante, se observa una mejora significativa en su rendimiento cuando se ejecuta en dispositivos con hardware más potente, lo que evidencia que este modelo está pensado para entornos de mayor capacidad de cómputo y no para hardware móvil estándar.

Resumen

El análisis de los tiempos de inferencia evidencia diferencias claras entre las familias de modelos evaluadas. Los modelos de la familia **MoveNet Lightning destacan como los más rápidos** y estables, siendo apenas sensibles al cambio de dispositivo, lo que los convierte en la opción más adecuada para aplicaciones móviles en tiempo real. Los modelos de la familia **MoveNet Thunder alcanzan un rendimiento intermedio aceptable** (en especial su versión cuantizada a int8), aunque las versiones de mayor precisión numérica se ven penalizadas en velocidad. Los modelos **BlazePose Lite y Full mantienen tiempos de ejecución razonables**, mientras que la versión Heavy resulta inviable por sus elevadas necesidades de cómputo. En la familia **YOLO-Pose, solo la variante Nano ofrece tiempos aceptables**, en cambio, Small y sobre todo Medium presentan inferencias muy lentas, aunque su rendimiento se ve más beneficiado con la utilización de dispositivos con mejor hardware.

7.3. Comparativa de resultados

La intención de esta comparativa no es únicamente señalar qué modelos son más precisos o más rápidos, sino identificar el **equilibrio** entre ambas dimensiones, lo que resulta fundamental en aplicaciones prácticas. En entornos de uso real, como dispositivos móviles, no basta con contar con una alta exactitud en las estimaciones sino que también es necesario que los tiempos de inferencia sean compatibles con un uso en tiempo real.

7.3.1. AP general vs. Tiempo medio

El análisis conjunto de precisión (Average Precision, AP) y tiempo de inferencia para el dataset general de imágenes que incluye el total de las imágenes de testeo seleccionadas para el estudio (316 imágenes), revela diferencias claras en la eficiencia relativa de las familias de modelos estudiados. En este apartado se muestra dicha comparativa para el resultado obtenido en el dispositivo con más prestaciones utilizado (Samsung Galaxy A9 Tab).

Se ha elaborado una **gráfica de dispersión** (Imagen 46) en la que cada punto representa a uno de los modelos evaluados. En este caso, el eje horizontal muestra la precisión obtenida (AP), mientras que el eje vertical refleja los tiempos medios de inferencia por imagen. Esta gráfica permite observar cómo aumenta el coste computacional conforme se avanza hacia modelos más precisos. Los modelos situados en la parte inferior derecha de la gráfica pueden considerarse los más ventajosos, al combinar altos niveles de precisión con tiempos de ejecución reducidos. En contraste, aquellos que se ubican en la zona superior derecha ofrecen buena precisión, pero a costa de tiempos de inferencia elevados. Por último, los modelos en la parte inferior izquierda muestran tiempos de inferencia bajos pero una precisión insuficiente.

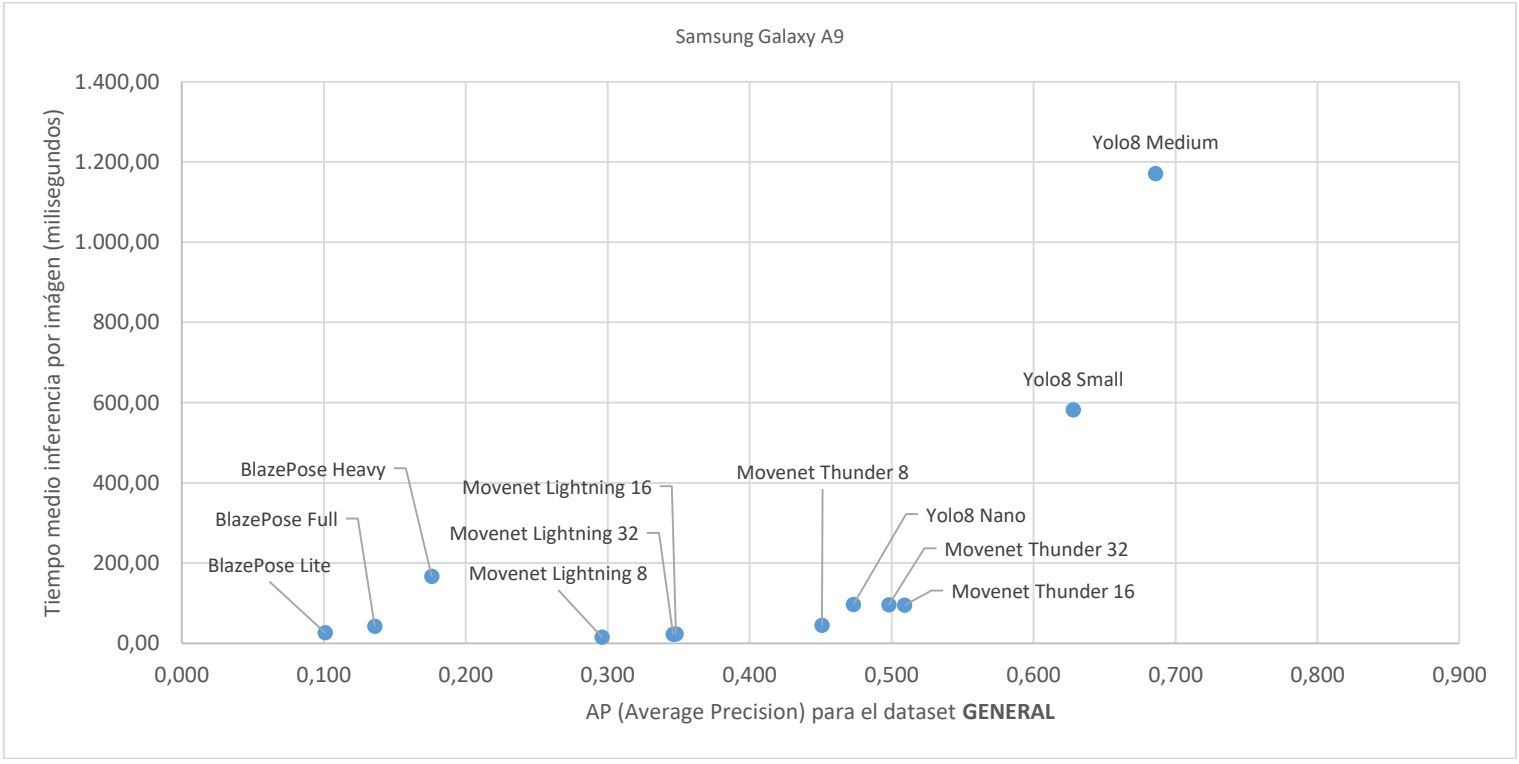


Imagen 46. Comparativa AP vs. Tiempo de inferencia dataset general

MoveNet Thunder (float16 y float32)

Estas versiones destacan como **las más equilibradas** en el conjunto de pruebas. Aunque no alcanzan las cifras absolutas de precisión de algunos modelos más complejos, logran una relación muy favorable entre la calidad de las predicciones y el coste computacional. En

consecuencia, ofrecen un **compromiso idóneo** para aplicaciones en dispositivos móviles donde la velocidad es importante pero no puede sacrificarse demasiado la exactitud.

YOLOv8-Pose Nano

También presenta una **relación adecuada entre precisión y rendimiento**. Su capacidad de mantener un nivel de exactitud aceptable con tiempos de inferencia moderados lo convierte en una opción práctica, aunque se sitúa por debajo de MoveNet Thunder en términos de equilibrio global.

YOLOv8-Pose Small y Medium

Estos modelos alcanzan los mejores valores de precisión del estudio, lo que los posiciona como referentes desde el punto de vista de la **exactitud en la estimación** de keypoints. Sin embargo, esta ventaja se ve contrarrestada por **tiempos de inferencia considerablemente altos**, que limitan su aplicabilidad a entornos con hardware de altas prestaciones. Su uso en dispositivos generales resultaría poco viable debido al coste temporal de la ejecución.

BlazePose

A pesar de ofrecer tiempos de inferencia reducidos, los resultados de precisión son notablemente inferiores al resto de familias. Este desequilibrio los hace **menos adecuados** para tareas en las que la calidad de la estimación es prioritaria, ya que la rapidez en el cálculo no compensa la baja fiabilidad de los resultados obtenidos.

7.3.2. AP por tipo de imagen (más adecuadas y menos adecuadas) vs. Tiempo medio

El análisis conjunto de precisión (Average Precision, AP) y tiempo de inferencia para los dos datasets (subconjuntos del dataset general de imágenes) que incluyen las imágenes consideradas más adecuadas (65 imágenes) y menos adecuadas para estimación de posturas humanas (61 imágenes) nos revela todavía más diferencias claras en la eficiencia relativa de las familias de modelos estudiados. En este apartado se muestran los resultados obtenidos con ambos subconjuntos del dataset de testeo en el dispositivo con más prestaciones utilizado (Samsung Galaxy A9 Tab).

Análisis AP/Tiempo de inferencia en subconjunto de imágenes adecuadas

La evaluación de los modelos de estimación de posturas humanas sobre el subconjunto de imágenes filtradas para su idoneidad (personas centradas y cercanas) permite observar tendencias significativas que complementan y amplían los hallazgos obtenidos con el dataset completo (Imagen 47). Este subconjunto, al ofrecer condiciones visuales óptimas para la estimación de posturas, permite medir el máximo potencial de precisión de cada modelo, al tiempo que se comparan sus tiempos de inferencia acumulados.

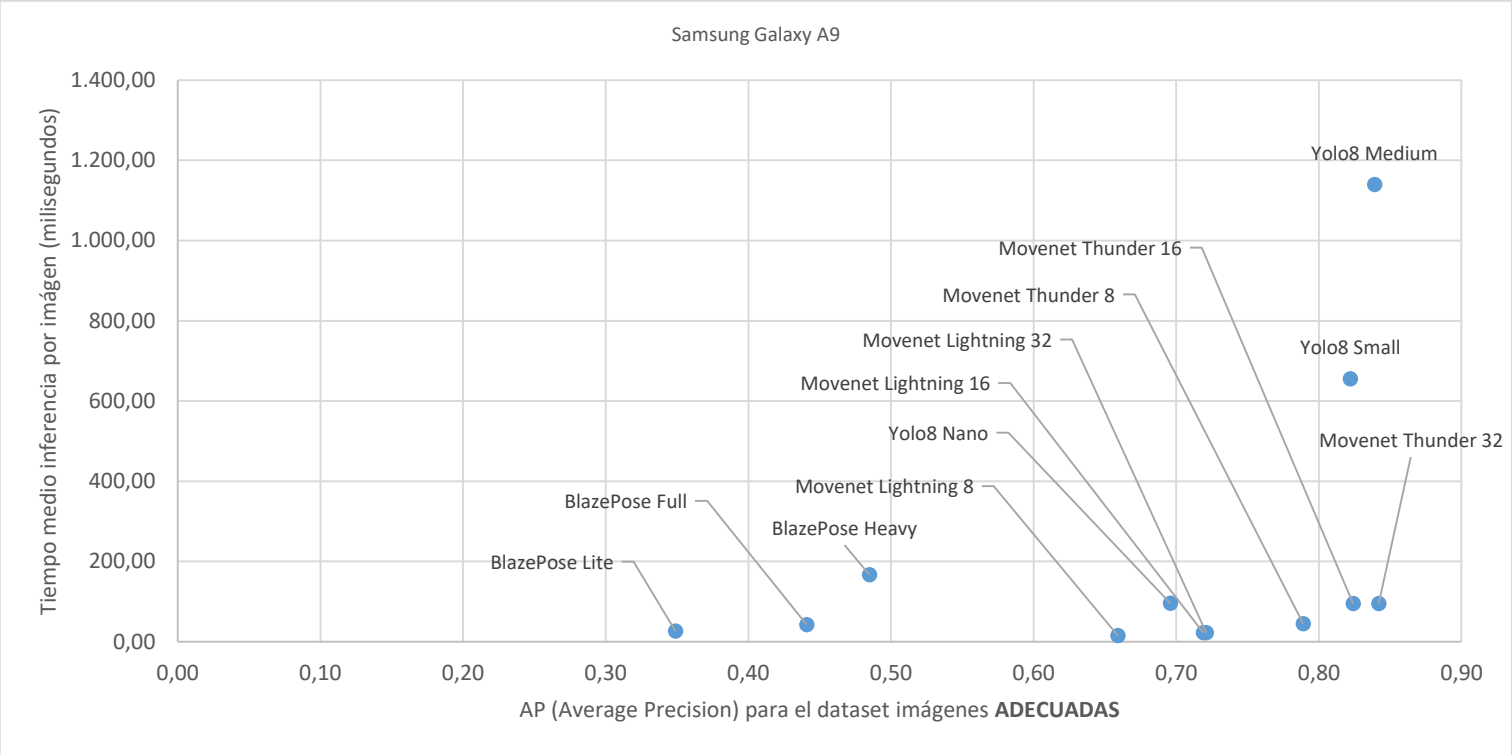


Imagen 47. Comparativa AP vs. Tiempo de inferencia dataset de imágenes adecuadas

En primer lugar, los modelos **MoveNet Thunder destacan de manera sobresaliente**. Las versiones float32 y float 16, y en menor medida la versión cuantizada int8, logran cifras de precisión espectaculares, cercanas al máximo teórico que cada arquitectura puede ofrecer. Además mantienen un rendimiento muy eficiente, igual al observado en el dataset completo, lo que evidencia la capacidad de estas arquitecturas para combinar exactitud y velocidad sin comprometer la inferencia. Esta combinación los posiciona como la opción más equilibrada para aplicaciones móviles o en tiempo real donde la exactitud es prioritaria.

Por otro lado, los modelos **YOLOv8-Pose Small y Medium** también alcanzan niveles de **precisión igualmente elevados**, reflejando su capacidad para detectar keypoints de manera **muy precisa** en condiciones visuales óptimas. No obstante, esta mejora en precisión mantiene un coste considerable en términos de rendimiento al igual que para las imágenes del dataset completo. La versión **Small** registra un **rendimiento bajo**, mientras que **Medium** alcanza tiempos de inferencia extremadamente altos, **desaconsejando su uso en dispositivos con recursos limitados**. Sin embargo, en entornos con hardware de alta gama, estas versiones pueden ser útiles cuando la prioridad absoluta es la precisión.

Los modelos **MoveNet Lightning y YOLOv8-Pose Nano** muestran un comportamiento equilibrado en este subconjunto, su precisión es buena, aunque ligeramente inferior a la de los modelos Thunder o las versiones más pesadas de YOLO, pero sus tiempos de inferencia son muy competitivos, posicionándolos como **alternativas fiables** para aplicaciones que requieren velocidad y respuesta en tiempo real. Su desempeño refleja la efectividad de estas arquitecturas ligeras para escenarios móviles sin comprometer

excesivamente la exactitud.

En cuanto a los modelos **BlazePose**, se observa un **aumento notable de la precisión, pero insuficiente** en comparación con el conjunto completo, lo que confirma que la idoneidad de las imágenes tiene un impacto positivo en su desempeño. Sin embargo, incluso en este subconjunto optimizado, BlazePose sigue siendo la familia que presenta mayor pérdida de precisión relativa respecto a las demás familias de modelos. Esta pérdida es más acentuada en la versión Lite, seguida por Full, mientras que la versión Heavy muestra una degradación algo menor, aunque no alcanza los niveles de precisión de MoveNet Thunder o YOLOv8-Pose Medium. En términos de rendimiento, BlazePose mantiene tiempos de inferencia similares a los observados en el dataset completo, reflejando una eficiencia estable que, no obstante, no compensa su menor exactitud.

Análisis AP/Tiempo de inferencia en subconjunto de imágenes inadecuadas

La evaluación de los modelos de estimación de posturas humanas sobre el subconjunto de imágenes filtradas menos adecuadas (con personas no centradas o lejanas) también permite sacar algunas conclusiones significativas complementarias a los hallazgos obtenidos con el dataset completo (Imagen 48). Este subconjunto con condiciones visuales negativas para la estimación de posturas nos permite observar cómo se defienden los modelos en estas condiciones mientras que se observa si sus tiempos de inferencia se ven influidos por las características de las imágenes inferidas.

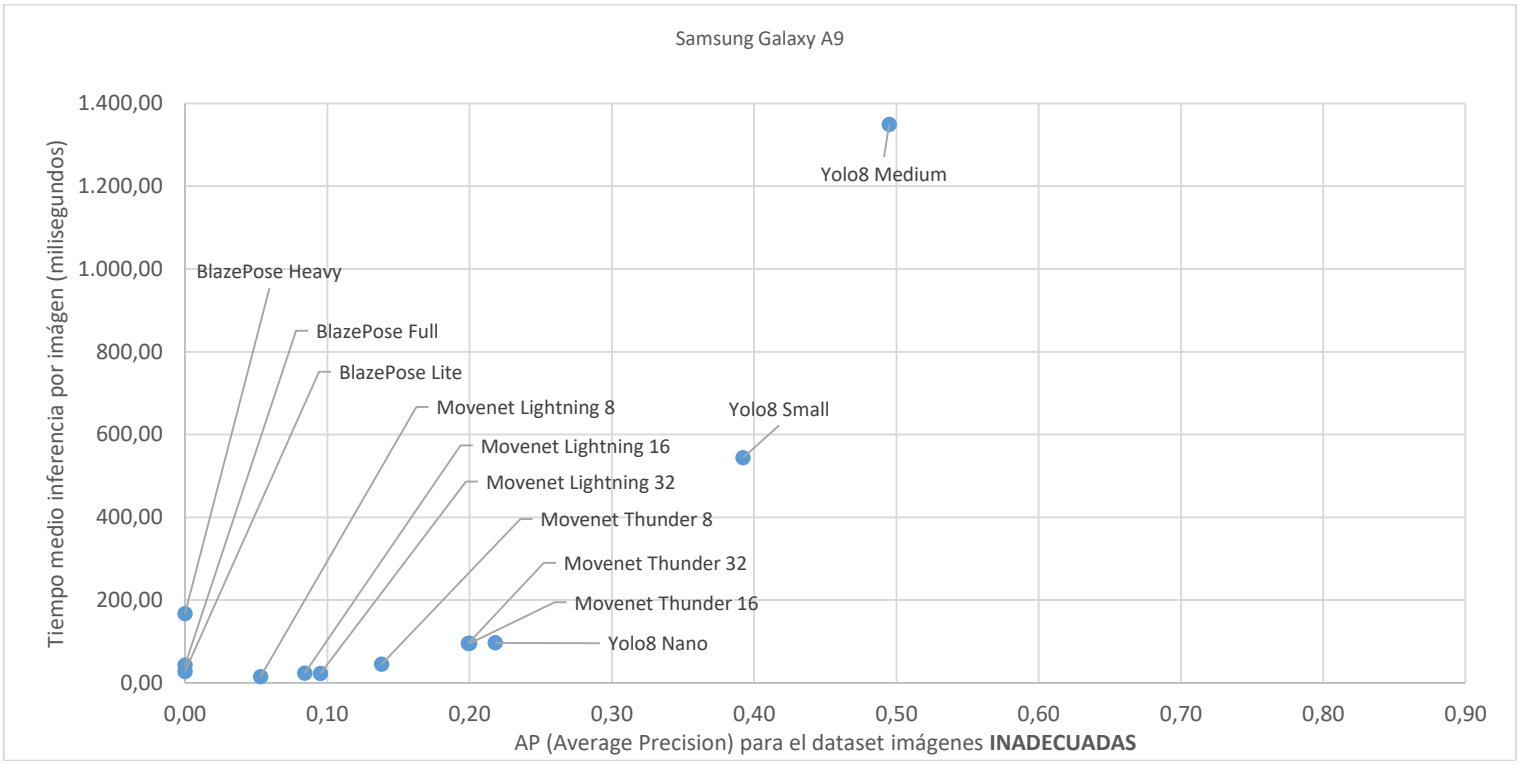


Imagen 48. Comparativa AP vs. Tiempo de inferencia dataset de imágenes inadecuadas

La evaluación de los modelos de estimación de posturas humanas sobre el subconjunto de imágenes clasificadas como inadecuadas (aquellas en las que la persona no se encuentra centrada o aparece lejana) aporta una perspectiva complementaria respecto al comportamiento de estas arquitecturas bajo condiciones adversas. Este análisis permite caracterizar las limitaciones de cada familia de modelos y comprobar su capacidad de generalización fuera de escenarios óptimos.

Un primer resultado destacable es el comportamiento de los modelos más pesados, en particular **YOLOv8-Pose Small y Medium**. A pesar de la dificultad inherente de este subconjunto, estas versiones **logran mantener una precisión media (aunque no elevada)**, lo que refleja la robustez de estas arquitecturas. En especial, la versión Medium demuestra una cierta resiliencia, probablemente atribuida a su mayor número de parámetros y profundidad de red, que le permiten manejar mejor escenarios con oclusiones o poses poco definidas. No obstante, esta ganancia relativa en precisión se produce **manteniendo los altos tiempos de inferencia** que caracterizan a estas versiones, lo que limita su aplicabilidad en dispositivos con restricciones de hardware.

Los modelos **MoveNet Thunder y YOLO Nano**, que en el subconjunto de imágenes adecuadas mostraban un desempeño notable en cuanto a la relación precisión/rendimiento, experimentan aquí una **penalización significativa en la precisión**. Aunque sus tiempos de inferencia se mantienen estables, la pérdida de exactitud en la localización de keypoints evidencia que en condiciones adversas, la eficiencia de estas arquitecturas no basta para compensar las dificultades en la detección. Este resultado indica que los modelos de tamaño intermedio o ligero (efectivos en entornos favorables) presentan mayor vulnerabilidad en la generalización.

Los modelos **MoveNet Lightning resultan ser los más afectados en este subconjunto**. Su precisión cae de manera drástica, incluso más que la de Thunder o Nano, lo que pone de manifiesto que el diseño ultraligero de esta familia, pensado para priorizar la velocidad en dispositivos móviles, conlleva un coste elevado en términos de robustez. La capacidad reducida de representación no permite a estas arquitecturas mantener un desempeño aceptable cuando las condiciones de la imagen no favorecen la tarea de estimación de poses.

Por último, los resultados de los modelos **BlazePose** son concluyentes: su precisión en este subconjunto es **prácticamente nula**, situándose en valores equivalentes a cero. En otras palabras, BlazePose se muestra **inadecuado** para la estimación de posturas humanas en contextos adversos, sin importar la variante (Lite, Full o Heavy). Esto refuerza lo observado en el dataset completo y en las imágenes adecuadas, donde BlazePose ya mostraba un rendimiento inferior respecto a otras familias.

En cuanto a los **tiempos de inferencia**, el análisis confirma un aspecto importante, estos **no se ven afectados por el tipo de imagen**. Al igual que en el dataset completo y en el subconjunto de imágenes adecuadas, los tiempos de ejecución se mantienen prácticamente idénticos. Esto implica que la carga computacional de los modelos depende únicamente de la arquitectura y del hardware disponible, no de la idoneidad de las imágenes procesadas. Por tanto, **el tipo de imagen influye de manera directa en la precisión pero no en el rendimiento** computacional.

PARTE III: DISCUSION Y CONCLUSIONES

8. DISCUSIÓN

En primer lugar se aborda la interpretación de los resultados principales, destacando los comportamientos diferenciales de los modelos estudiados en términos de precisión y rendimiento, así como el impacto que el tipo de dataset ha tenido en sus estimaciones. Este análisis permitirá evidenciar qué modelos ofrecen un equilibrio más adecuado entre exactitud y eficiencia y cuáles presentan limitaciones intrínsecas.

A continuación, se incluirán las limitaciones del estudio, entendidas como los factores metodológicos, técnicos o contextuales que han podido condicionar los resultados. Estas limitaciones abarcan desde la heterogeneidad en las estructuras de salida de los modelos hasta las restricciones impuestas por el dataset empleado y el hardware utilizado. Reconocer estos aspectos no solo aporta transparencia, sino que además abre la posibilidad de plantear futuras mejoras y ampliaciones del trabajo.

8.1. Interpretación de los resultados principales

Los resultados obtenidos permiten establecer varias conclusiones relevantes acerca del comportamiento de los modelos de estimación de posturas humanas bajo distintas condiciones. En primer lugar y como era de esperar se confirma que la **precisión** de los modelos muestra una alta sensibilidad a las condiciones visuales de las imágenes de entrada. En este sentido, los modelos de mayor complejidad (YOLOv8-Pose en sus versiones Small y Medium) exhiben una mayor capacidad de generalización, mientras que los modelos de menor tamaño sufren pérdidas significativas de exactitud en escenarios adversos. Esta degradación resulta particularmente acusada en la familia BlazePose, mientras que los modelos MoveNet mantienen un comportamiento relativamente más robusto.

En cuanto al **rendimiento** computacional, el análisis de los tiempos de inferencia revela diferencias sustanciales entre familias. Los modelos MoveNet Lightning se consolidan como los más rápidos y estables, apenas afectados por el dispositivo de ejecución, lo que los convierte en candidatos idóneos para aplicaciones móviles en tiempo real. Los MoveNet Thunder ofrecen un rendimiento intermedio: la versión cuantizada a int8 se aproxima a los tiempos de Lightning, mientras que las versiones en float16 y float32 sacrifican velocidad en favor de precisión. Por su parte, BlazePose Lite y Full mantienen tiempos razonables, mientras que la versión Heavy resulta computacionalmente inviable. En la familia YOLO-Pose, únicamente la variante Nano alcanza tiempos aceptables, frente a Small y, especialmente, Medium, que presentan inferencias lentas pero que escalan favorablemente en hardware de mayores prestaciones.

El **análisis conjunto de precisión y tiempos** de inferencia permite identificar aquellos modelos que presentan una mejor relación entre exactitud y eficiencia computacional, lo que resulta clave para su posible integración en aplicaciones prácticas de estimación de posturas humanas. **Los modelos de la familia MoveNet Thunder, especialmente en sus variantes 16 y 32, se posicionan como la opción más equilibrada**. Estos alcanzan niveles de precisión muy altos, comparables a los de arquitecturas de mayor tamaño, sin comprometer en exceso el tiempo de inferencia. Su rendimiento los hace especialmente adecuados para aplicaciones que requieran un compromiso sólido entre calidad de predicción y velocidad de procesamiento, incluso en dispositivos con recursos limitados. En un escalón próximo se sitúa el modelo **YOLOv8-Pose Nano**, que logra una precisión buena con tiempos de inferencia muy competitivos. Esta combinación también lo convierte en un candidato interesante para

aplicaciones móviles o embebidas, donde las restricciones de hardware son críticas. Los modelos **YOLOv8-Pose Small y Medium** destacan por su elevada precisión, pero sus elevados tiempos de inferencia **limitan su aplicabilidad** a escenarios en los que se disponga de hardware de alto rendimiento o donde la inferencia en tiempo real no sea un requisito estricto. Finalmente, los modelos **BlazePose**, pese a sus tiempos de ejecución reducidos, presentan **deficiencias importantes en términos de precisión**, lo que los sitúa en una muy clara desventaja frente a las demás familias para estimación de posturas humanas.

En síntesis, los **MoveNet Thunder y YOLO Nano representan las opciones más adecuadas para su incorporación en aplicaciones orientadas a la estimación de posturas humanas en tiempo real**, equilibrando correctamente precisión y rendimiento.

Por último, los tiempos de inferencia permanecen invariables ante el tipo de imagen procesada. Tanto en el dataset general como en los subconjuntos de imágenes adecuadas e inadecuadas, la duración del proceso de inferencia se mantiene en tiempos prácticamente constantes. Esto confirma que la carga computacional está determinada por la arquitectura del modelo y la capacidad del hardware, sin influencia de la idoneidad del contenido visual. En consecuencia, el tipo de imagen afecta de manera directa a la precisión de las estimaciones, pero no al rendimiento computacional.

8.2. Limitaciones del estudio

El estudio presenta una serie de limitaciones relacionadas principalmente con el alcance de la experimentación y la representatividad de los escenarios analizados:

- Una primera limitación es la **lista de modelos evaluados**. Si bien se han considerado arquitecturas representativas de las familias más relevantes (MoveNet, BlazePose y YOLO-Pose), el panorama actual de la estimación de posturas humanas es dinámico y en constante evolución. Existen otros modelos recientes, tanto ligeros como de mayor complejidad, que no fueron incluidos y cuya incorporación permitiría una visión más completa del estado del arte, así como una comparación más rica entre diferentes enfoques arquitectónicos.

- El **número reducido de imágenes disponibles** en el dataset utilizado, que asciende únicamente a 316. Si bien estas imágenes han permitido llevar a cabo una primera evaluación del comportamiento de los modelos, dicho volumen resulta limitado para extraer conclusiones con mayor robustez estadística y capacidad de generalización. Sería recomendable disponer de imágenes que reflejen ejercicios de rehabilitación de forma específica, ya que constituyen el contexto real en el que se prevé aplicar los modelos de estimación de posturas humanas. La inclusión de personas de diferentes edades, condiciones físicas y contextos demográficos (incluyendo diversidad racial y corporal) permitiría evaluar con mayor fidelidad la capacidad de generalización de las arquitecturas bajo estudio.

- Otra limitación importante se refiere al **soporte multiplataforma**, ya que los experimentos se han llevado a cabo exclusivamente en dispositivos Android. La ausencia de una evaluación en entornos iOS restringe la generalización de los resultados. Las diferencias entre las librerías de soporte, las optimizaciones específicas del sistema operativo y la gestión de hardware podrían alterar de manera

sustancial tanto la precisión como los tiempos de inferencia, por lo que este aspecto queda pendiente de ser explorado.

- Asimismo, el estudio se ve limitado por el **número y variedad de dispositivos** empleados en la experimentación. Aunque se han contemplado varias configuraciones con prestaciones diferenciadas, la muestra es insuficiente para reflejar la gran heterogeneidad existente en el ecosistema de hardware. No se incluyen, por ejemplo, dispositivos de gama muy baja, que representarían un escenario especialmente crítico para el despliegue de modelos en entornos con recursos limitados. Tampoco se han evaluado dispositivos de gama muy alta que podrían ofrecer un rendimiento significativamente superior, lo cual restringe la validez externa de las conclusiones en contextos más extremos.

Estas limitaciones condicionan la amplitud de las conclusiones y ponen de manifiesto la necesidad de extender el análisis en futuras investigaciones, tanto mediante la incorporación de más arquitecturas como ampliando la diversidad de plataformas y dispositivos evaluados.

9. CONCLUSIONES Y TRABAJO FUTURO

El trabajo permite evaluar de manera sistemática la precisión y el rendimiento de diversas arquitecturas de estimación de posturas humanas en dispositivos móviles, cumpliendo con el **objetivo de identificar modelos** adecuados para su implementación práctica en aplicaciones en tiempo real.

Además, este estudio ha puesto de manifiesto **áreas de mejora y posibles líneas de trabajo futuro**, tales como la ampliación de la lista de modelos evaluados, la incorporación de soporte para entornos iOS y la extensión de las pruebas a dispositivos con prestaciones más variadas. Estas propuestas buscan aumentar la robustez, generalización y aplicabilidad de los resultados, ofreciendo una base sólida para investigaciones posteriores y para la optimización de aplicaciones de estimación de posturas humanas en contextos reales.

9.1. Revisión objetivos principales del estudio

Según se describió en los apartados "1.3.1 Objetivo general" y "1.3.2 Objetivos específicos" el proyecto ha logrado cumplir de manera satisfactoria con los objetivos planteados, tanto en su dimensión general como en los objetivos específicos.

En cuanto al objetivo general, se ha llevado a cabo una búsqueda y selección de modelos de estimación de posturas humanas aptos para poder ser utilizados en dispositivos de *edge computing*, y la evaluación del rendimiento y la precisión de tres familias de ellos (MoveNet, BlazePose y YOLOv8-Pose) en dispositivos móviles. Esto ha permitido caracterizar el comportamiento de cada arquitectura estudiada bajo condiciones controladas y representativas, ofreciendo una visión comparativa de su aplicabilidad en entornos móviles.

Respecto a los objetivos específicos, el proyecto ha alcanzado los siguientes logros:

- Búsqueda y selección de modelos aptos para el estudio en base a las especificaciones definidas en la metodología.

- Implementación de un sistema de pruebas en Android capaz de ejecutar modelos en formato TensorFlow Lite (TFLite) y gestionar la inferencia sobre un conjunto filtrado de imágenes del dataset COCO.

- Medición y comparación de la precisión de los modelos, utilizando métricas estándar como Average Precision (AP@[0.50:0.95]), tanto sobre el dataset completo como en subconjuntos diferenciados por idoneidad de las imágenes para estimación de posturas.

- Medición y comparación de los tiempos de inferencia por imagen, evaluando el rendimiento de cada modelo en distintos dispositivos móviles.

- Establecimiento de la relación entre precisión y rendimiento, identificando qué modelos ofrecen el equilibrio más adecuado entre exactitud de predicciones y eficiencia computacional, y cuáles presentan limitaciones para su integración en aplicaciones móviles en tiempo real.

En conjunto, los resultados obtenidos confirman que los objetivos del proyecto se han cumplido, proporcionando una base sólida para seleccionar modelos óptimos según los requisitos de precisión y rendimiento, y ofreciendo directrices claras para futuras implementaciones en aplicaciones móviles de estimación de posturas humanas.

9.2. Propuestas de mejora y líneas futuras

El proyecto abre diversas líneas de mejora y expansión que permitirían aumentar su alcance y aplicabilidad en el ámbito de la estimación de posturas humanas en dispositivos móviles.

Ampliación para estudio de nuevos modelos emergentes

En primer lugar, se propone la ampliación del estudio a nuevos modelos ya existentes (como los evaluados en el punto "2.5. Modelos de estimación de posturas") y otros posibles emergentes, incorporando arquitecturas recientes que puedan ofrecer mejoras en precisión, eficiencia o robustez frente a condiciones adversas. Esto permitiría mantener el análisis actualizado y comparativo frente al estado del arte.

Ampliación para soporte multi-dispositivo (Apple iOS)

Otra línea relevante es la compatibilidad multiplataforma, extendiendo el soporte a dispositivos Apple iOS, lo que facilitaría la implementación de aplicaciones móviles de estimación de posturas humanas en un ecosistema más amplio y heterogéneo. Complementariamente, se plantea la evaluación en nuevos dispositivos, tanto de gama baja como alta, para validar la generalización de los resultados y conocer la influencia del hardware sobre la precisión y el rendimiento de los modelos.

Utilización de aceleradores de hardware específicos (GPU)

Se sugiere la inclusión en la aplicación Android de la utilización de aceleradores de hardware específicos, como GPUs o NPUs (descritos en el apartado "2.8.1. Hardware"), con el objetivo de mejorar los tiempos de inferencia y permitir la ejecución en tiempo real de modelos más complejos, ampliando las posibilidades de aplicación práctica en ámbitos más exigentes en términos de capacidad de computación.

Ampliación para estimación de posturas sobre entradas de vídeo

En términos de entrada de datos, se considera la ampliación para la estimación de posturas sobre secuencias de vídeo, lo que permitiría evaluar la estabilidad temporal de los modelos y su desempeño en aplicaciones dinámicas, como seguimiento en tiempo real o análisis de movimiento continuo.

Ampliación para reentrenamiento de modelos existentes

Por último, se contempla la posibilidad de reentrenamiento o *fine-tuning* de modelos existentes con datasets específicos, lo que permitiría adaptar los modelos a contextos particulares, mejorar su precisión y robustez frente a escenarios concretos, y optimizar su desempeño en aplicaciones móviles específicas.

10. REFERENCIAS BIBLIOGRÁFICAS

Nota: Todos los enlaces de las referencias bibliográficas han sido consultados a modo de comprobación por última vez a fecha 29 de agosto de 2025.

1. COCO Dataset. [Online]. Available from: https://cocodataset.org/#home.

2. MPII Human Pose Dataset. [Online]. Available from: https://www.mpi-inf.mpg.de/departments/computer-vision-and-machine-learning/software-and-datasets/mpii-human-pose-dataset/download.

3. Holian N. EDGE COMPUTING UNDERSTANDING THE USER EXPERIENCE. [Online].; marzo 2023. Available from: https://www.wipro.com/infrastructure/edge-computing-understanding-the-user-experience/.

4. Boesch G. Human Pose Estimation - Everything You Need to Know. [Online].; octubre 2023. Available from: https://viso.ai/deep-learning/pose-estimation-ultimate-overview/.

5. López OGT. Redes Neuronales Convolucionales para el reconocimiento de imágenes. [Online].; 30 agosto 2021. Available from: https://www.tecnoblog.org/desarrollo/cnn-para-el-reconocimiento-de-imagenes/.

6. Zewen Li WYSPFL. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. [Online].; abril 2020. Available from: https://arxiv.org/abs/2004.02806.

7. Omar Elharrouss YANASAM. Backbones-Review: Feature Extraction Networks for Deep Learning and Deep Reinforcement Learning Approaches. [Online].; junio 2022. Available from: https://arxiv.org/abs/2206.08016.

8. Carr T. Multi-Headed Networks. [Online].; febrero 2025. Available from: https://www.baeldung.com/cs/multi-headed-neural-nets.

9. Pendhari S. Connected Layer vs Fully Connected Layer. [Online].; enero 2022. Available from: https://medium.com/@sarahpendhari/connected-layer-vs-fully-connected-layer-32b4cbb29824.

10. What is Fully Connected Layer in Deep Learning? [Online].; junio 2025. Available from: https://www.geeksforgeeks.org/deep-learning/what-is-fully-connected-layer-in-deep-learning/.

11. Shuang Cong YZ. A review of convolutional neural network architectures and their optimizations. [Online].; junio 2022. Available from: https://www.researchgate.net/publication/361477855_A_review_of_convolutional_neural_network_architectures_and_their_optimizations.

12. Andrew G. Howard MZ. MobileNets: Open-Source Models for Efficient On-Device Vision. [Online].; 14 junio 2017. Available from: https://research.google/blog/mobilenets-open-source-models-for-efficient-on-device-vision/?hl=es-mx.

13. Kai Han YWQTJGCXCX. GhostNet: More Features from Cheap Operations. [Online].; 27 noviembre 2019, revisado 13 marzo 2020. Available from: https://arxiv.org/abs/1911.11907.

14. Kromydas B. Convolutional Neural Network (CNN): A Complete Guide. [Online].; enero 2023. Available from: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/.

15. Markus Nagel MFRAAYBMvBTB. A White Paper on Neural Network Quantization. [Online].; junio 2021. Available from: https://arxiv.org/abs/2106.08295.

16. Pérez S. Detección de poses humanas mediante Deep Learning. [Online].; febrero 2022. Available from: https://blog.damavis.com/deteccion-de-poses-humanas-mediante-deep-learning/.

17. An end-to-end platform for machine learning. [Online]. Available from: https://www.tensorflow.org/.

18. Khanh LeViet YhC. Pose estimation and classification on edge devices with MoveNet and TensorFlow Lite. [Online].; agosto 2021. Available from: https://blog.tensorflow.org/2021/08/pose-estimation-and-classification-on-edge-devices-with-MoveNet-and-TensorFlow-Lite.html.

19. Dave Bergmann CS. ¿Qué es PyTorch? [Online].; octubre 2023. Available from: https://www.ibm.com/es-es/think/topics/pytorch.

20. MediaPipe Vs. TensorFlow: Human Pose Estimation Giants. [Online].; enero 2024. Available from: https://medium.com/@codetrade/mediapipe-vs-c1a57a2fac7e.

21. Rath S. Deep Learning with OpenCV DNN Module: A Definitive Guide. [Online].; abril 2021. Available from: https://learnopencv.com/deep-learning-with-opencvs-dnn-module-a-definitive-guide/.

22. Using the SavedModel format. [Online]. Available from: https://www.tensorflow.org/guide/saved_model.

23. TensorFlow Lite. [Online].; septiembre 2021. Available from: https://www.tensorflow.org/lite/guide?hl=es-419.

24. Open Neural Network Exchange. [Online]. Available from: https://onnx.ai/.

25. Inkawhich M. Saving and Loading Models. [Online].; agosto 2018, última actialización junio 2025. Available from: https://docs.pytorch.org/tutorials/beginner/saving_loading_models.html.

26. Hua Q. [CVPR 2017] OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. [Online].; junio 2019. Available from: https://medium.com/data-science/cvpr-2017-openpose-realtime-multi-person-2d-pose-estimation-using-part-affinity-fields-f2ce18d720e8.

27. Hao-Shu Fang JLHTCXHZYXYLLCL. AlphaPose: Whole-Body Regional Multi-Person Pose Estimation and Tracking in Real-Time. [Online].; noviembre 2022. Available from: https://arxiv.org/abs/2211.03375.

28. Brown J. Real-Time Human Pose Estimation with PoseNet and Deep Learning. [Online].; septiembre 2024. Available from: https://33rdsquare.com/tech/ai/posture-detection-using-posenet-with-real-time-deep-learning-project/.

29. Boesch G. DensePose: Facebook's Breakthrough in Human Pose Estimation. [Online].; agosto 2024. Available from: https://viso.ai/deep-learning/densepose/.

30. Ronny Votel NL. Next-Generation Pose Detection with MoveNet and TensorFlow.js. [Online].; mayo 2021. Available from: https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html.

31. Valentin Bazarevsky IGKRTZFZMG. BlazePose: On-device Real-time Body Pose tracking. [Online].; junio 2020. Available from: https://arxiv.org/abs/2006.10204.

32. Documentación de Ultralytics YOLO. [Online]. Available from: https://docs.ultralytics.com/es/.

33. Shaoqing Ren KHRGJS. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. [Online].; 4 junio 2015. Available from: https://arxiv.org/abs/1506.01497.

34. Shivang Agarwal JODTFJ. Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks. [Online].; 10 septiembre 2018. Available from: https://arxiv.org/abs/1809.03193v2.

35. Xiao Sun JSSLYW. Compositional Human Pose Regression. [Online].; 1 abril 2017. Available from: https://arxiv.org/abs/1704.00159.

36. Guilhem Chéron ILCS. P-CNN: Pose-based CNN Features for Action Recognition. [Online].; 11 junio 2015. Available from: https://arxiv.org/abs/1506.03607.

37. Classification: Accuracy, recall, precision, and related metrics. [Online]. Available from: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall.

38. IoU and variants overview. [Online].; agosto 2024. Available from: https://medium.com/@cshyo1004/iou-and-variants-overview-a328acf177cd.

39. COCO Keypoint Evaluation. [Online]. Available from: https://cocodataset.org/#keypoints-eval.

40. Sharma A. Mean Average Precision (mAP) Using the COCO Evaluator. [Online].; mayo 2022. Available from: https://pyimagesearch.com/2022/05/02/mean-average-precision-map-using-the-coco-evaluator/.

41. MoveNet.SinglePose. [Online]. Available from: https://storage.googleapis.com/movenet/MoveNet.SinglePose%20Model%20Card.pdf.

42. A Guide on YOLO11 Model Export to TFLite for Deployment. [Online]. Available from: https://docs.ultralytics.com/integrations/tflite/#export-arguments.

43. MediaPipe BlazePose GHUM 3D. [Online]. Available from: https://developers.google.com/static/ml-kit/images/vision/pose-detection/pose_model_card.pdf.

11. ANEXOS

Anexo A. Resultados numéricos del estudio

En este anexo se incluyen los resultados obtenidos de todos los modelos en los distintos dispositivos de prueba en formato tabla.

Samsung Galaxy Tab A7 Lite (Tablet)

Modelo	Precisión obtenida AP [IoU=0.50:0.95] (dataset 316 imágenes)	Precisión obtenida AP [IoU=0.50:0.95] (dataset 65 imágenes adecuadas)	Precisión obtenida AP [IoU=0.50:0.95] (dataset 61 imágenes inadecuadas)	Tiempo medio inferencia por imagen (ms) (dataset 316 imágenes)
MoveNet Lightining 8	0,296	0,656	0,055	66,24
MoveNet Lightining 16	0,348	0,719	0,084	100,45
MoveNet Lightining 32	0,346	0,721	0,095	101,88
MoveNet Thunder 8	0,456	0,788	0,140	253,68
MoveNet Thunder 16	0,509	0,824	0,200	479,73
MoveNet Thunder 32	0,498	0,842	0,199	478,59
BlazePose Lite	0,101	0,349	0,000	138,86
BlazePose Full	0,136	0,441	0,000	249,60
BlazePose Heavy	0,176	0,487	0,000	905,98
Yolo8-pose Nano	0,473	0,696	0,218	352,46
Yolo8-pose Small	0,628	0,822	0,392	1.251,36
Yolo8-pose Medium	0,686	0,839	0,495	3.697,21

Tabla 19. Resultados numéricos Samsung Galaxy Tab A7 Lite

Samsung Galaxy M32 (Móvil)

Modelo	Precisión obtenida AP [IoU=0.50:0.95] (dataset 316 imágenes)	Precisión obtenida AP [IoU=0.50:0.95] (dataset 65 imágenes adecuadas)	Precisión obtenida AP [IoU=0.50:0.95] (dataset 61 imágenes inadecuadas)	Tiempo medio inferencia por imagen (ms) (dataset 316 imágenes)
MoveNet Lightining 8	0,293	0,656	0,055	36,39
MoveNet Lightining 16	0,348	0,719	0,084	58,75
MoveNet Lightining 32	0,346	0,721	0,095	62,15
MoveNet Thunder 8	0,456	0,788	0,140	130,91
MoveNet Thunder 16	0,510	0,824	0,200	260,55
MoveNet Thunder 32	0,498	0,842	0,199	286,69
BlazePose Lite	0,101	0,349	0,000	189,67
BlazePose Full	0,136	0,441	0,000	277,86
BlazePose Heavy	0,176	0,487	0,000	876,97
Yolo8-pose Nano	0,472	0,696	0,218	337,10
Yolo8-pose Small	0,628	0,822	0,392	852,96
Yolo8-pose Medium	0,686	0,839	0,495	2.599,96

Tabla 20. Resultados numéricos Samsung Galaxy M32

Samsung Galaxy Tab A9 (Tablet)

Modelo	Precisión obtenida AP [IoU=0.50:0.95] (dataset 316 imágenes)	Precisión obtenida AP [IoU=0.50:0.95] (dataset 65 imágenes adecuadas)	Precisión obtenida AP [IoU=0.50:0.95] (dataset 61 imágenes inadecuadas)	Tiempo medio inferencia por imagen (ms) (dataset 316 imágenes)
MoveNet Lightining 8	0,296	0,659	0,053	14,90
MoveNet Lightining 16	0,348	0,719	0,084	22,72
MoveNet Lightining 32	0,346	0,721	0,095	22,57
MoveNet Thunder 8	0,451	0,789	0,138	44,64
MoveNet Thunder 16	0,509	0,824	0,200	94,80
MoveNet Thunder 32	0,498	0,842	0,199	95,37
BlazePose Lite	0,101	0,349	0,000	26,54
BlazePose Full	0,136	0,441	0,000	42,06
BlazePose Heavy	0,176	0,485	0,000	166,97
Yolo8-pose Nano	0,473	0,696	0,218	96,25
Yolo8-pose Small	0,628	0,822	0,392	581,62
Yolo8-pose Medium	0,686	0,839	0,495	1.170,97

Tabla 21. Resultados numéricos Samsung Galaxy Tab A9

Anexo B. Ejemplos de visualización de keypoints estimados sobre imágenes

En las imágenes de este apartado se muestran ejemplos de imágenes procedentes del conjunto de validación del dataset de datos COCO sobre las cuales se representan los keypoints estimados por distintos modelos de redes convolucionales evaluados en este estudio, superpuestos a las anotaciones de referencia (ground truth) proporcionadas por el dataset.

Los puntos en **rojo** corresponden a las posiciones exactas de articulaciones definidas en las anotaciones de COCO, que incluyen 17 localizaciones corporales como hombros, codos, muñecas, caderas, rodillas y tobillos. En contraste, los keypoints predichos por los modelos aparecen en colores **verde, azul y cian**, permitiendo una comparación visual entre la predicción automática y la verdad de referencia.

Para la representación de los keypoints sobre las imágenes se ha utilizado Jupyter Notebook y, además de las librerías ya descritas en el apartado "7.1. Resultados obtenidos de precisión" (pycocotools.coco, json), se han utilizado las siguientes librerías de Python:

- **skimage.io**. Permite leer y mostrar imágenes en distintos formatos directamente desde archivos o URLs. Es útil para cargar imágenes del dataset COCO.

- **matplotlib.pyplot**. Herramienta de visualización que permite mostrar imágenes y superponer elementos gráficos, como los keypoints predichos o anotaciones de referencia.

- **PIL.Image**. Parte de la librería Pillow, utilizada para abrir, procesar y manipular imágenes de manera flexible.

- **numpy**. Biblioteca fundamental para el cálculo numérico en Python. Se utiliza para manejar arreglos multidimensionales, coordenadas y operaciones matemáticas asociadas a imágenes y keypoints.

- **pylab**. Entorno de visualización que combina funcionalidades de matplotlib y numpy, útil para configurar parámetros gráficos y mostrar imágenes con anotaciones.

Ejemplos de estimaciones para imágenes **adecuadas (personas centradas y cercanas):**

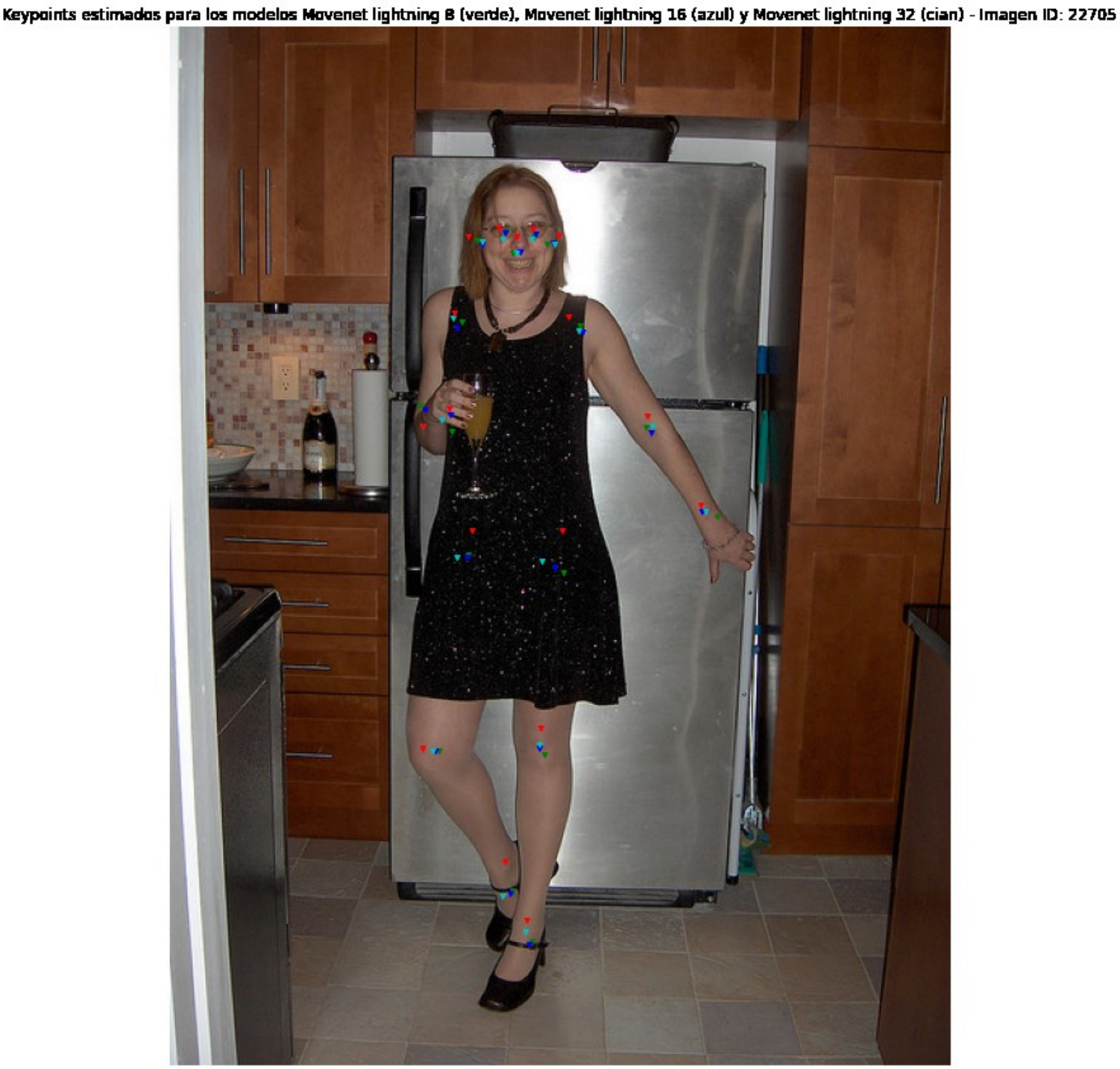


Imagen 49. Keypoints estimados por MoveNet Lightning para imagen 22705

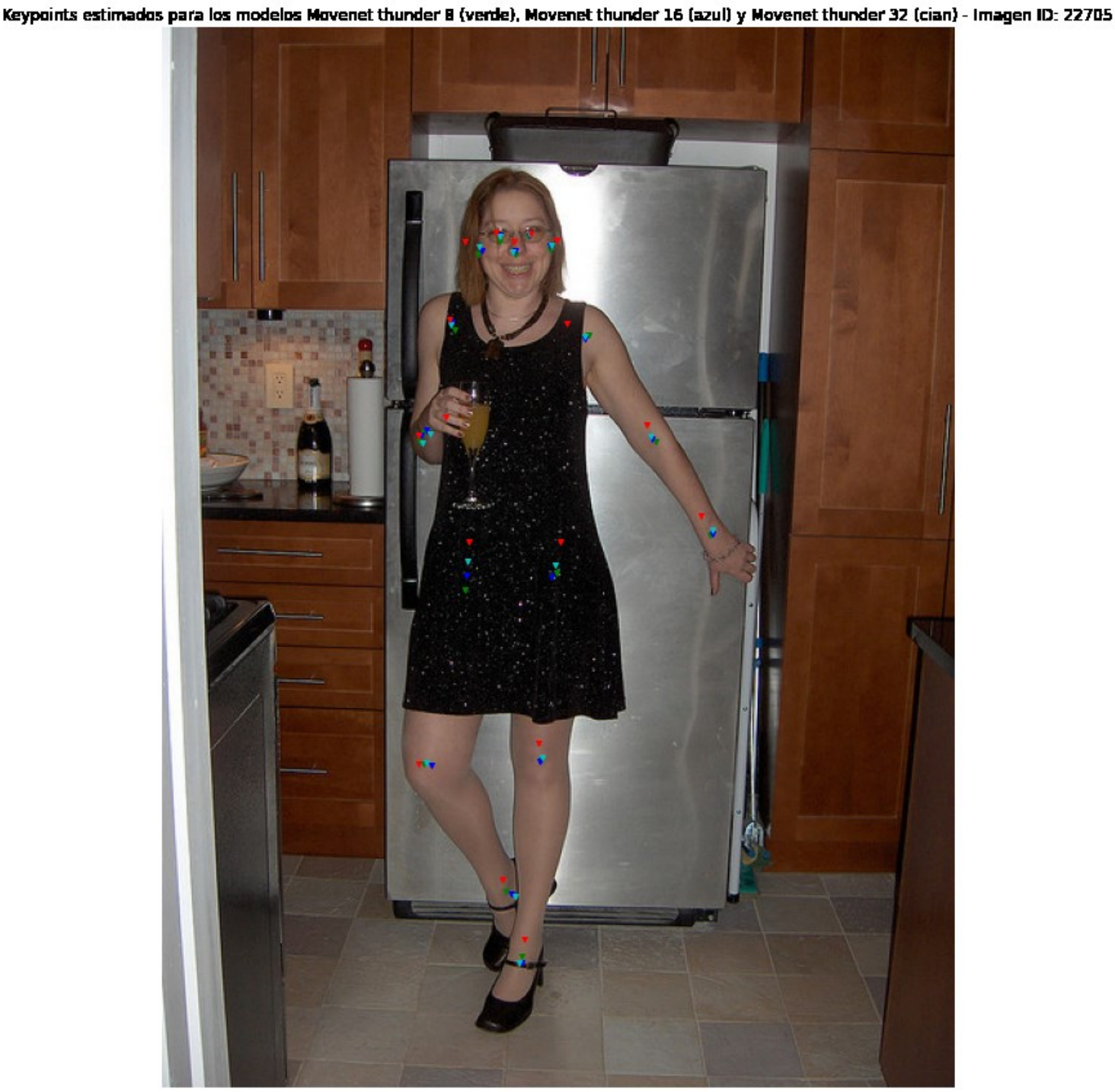


Imagen 50. Keypoints estimados por MoveNet Thunder para imagen 22705

Imagen 51. Keypoints estimados por BlazePose para imagen 22705

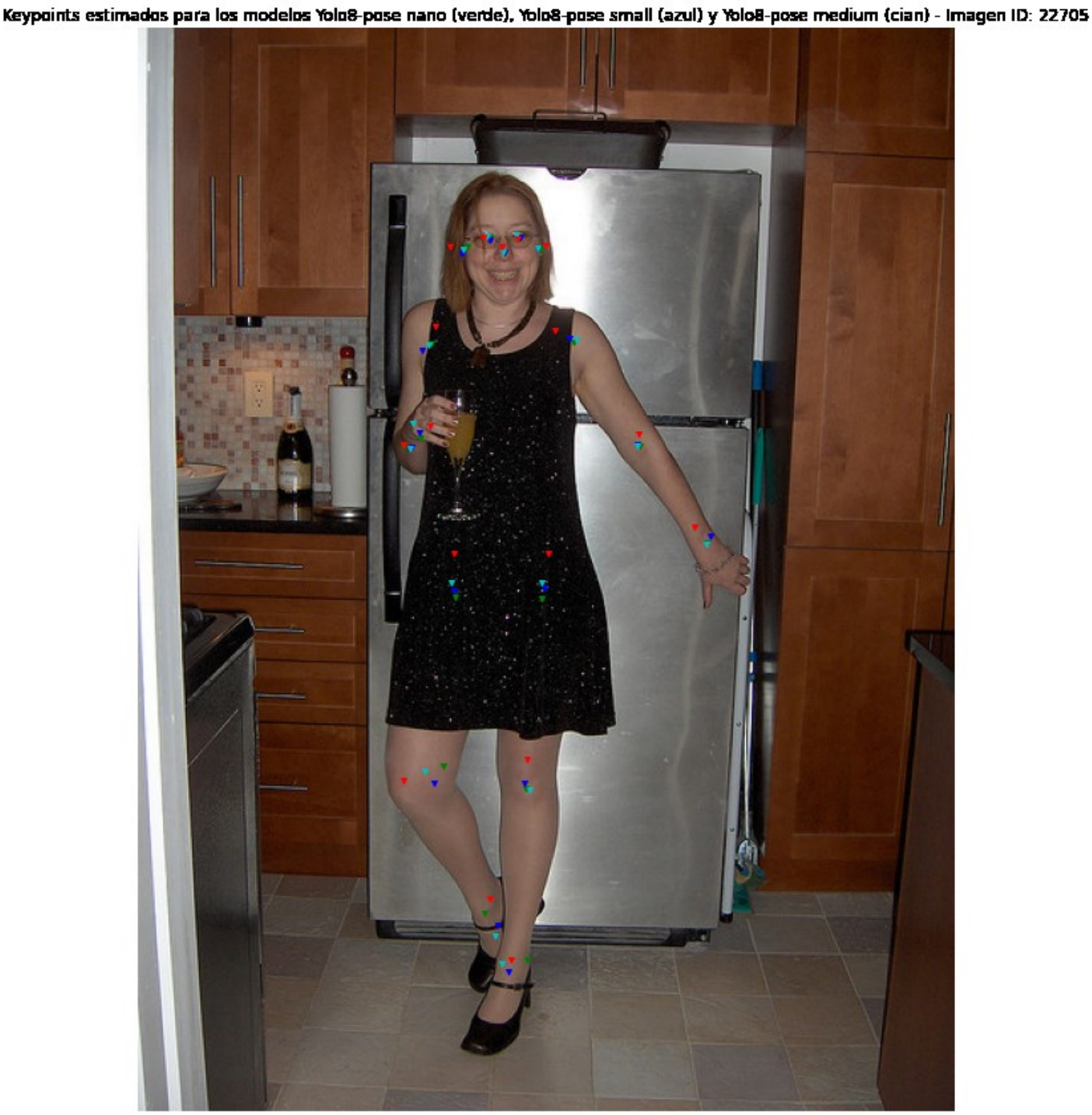


Imagen 52. Keypoints estimados por Yolo8-pose para imagen 22705

Imagen 53. Keypoints estimados por MoveNet Lightning para imagen 65736

Imagen 54. Keypoints estimados por MoveNet Thunder para imagen 65736

Imagen 55. Keypoints estimados por BlazePose para imagen 65736

Imagen 56. Keypoints estimados por Yolo8-pose para imagen 65736

Ejemplos de estimaciones para imágenes **inadecuadas (personas no centradas o lejanas):**

Imagen 57. Keypoints estimados por MoveNet Lightning para imagen 347265

Imagen 58. Keypoints estimados por MoveNet Thunder para imagen 347265

Imagen 59. Keypoints estimados por BlazePose para imagen 347265

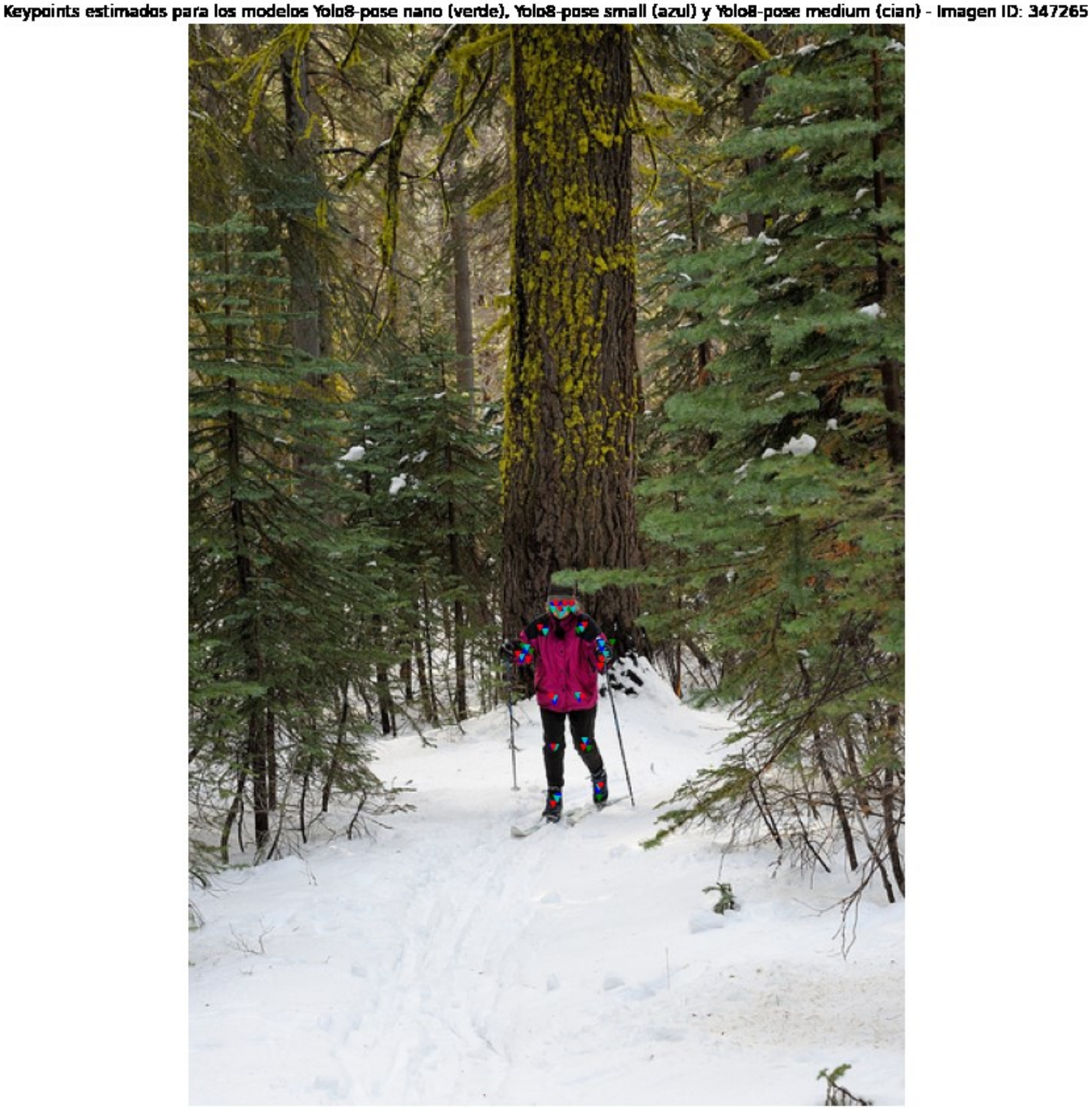


Imagen 60. Keypoints estimados por Yolo8-pose para imagen 347265

Imagen 61. Keypoints estimados por MoveNet Lightning para imagen 161879

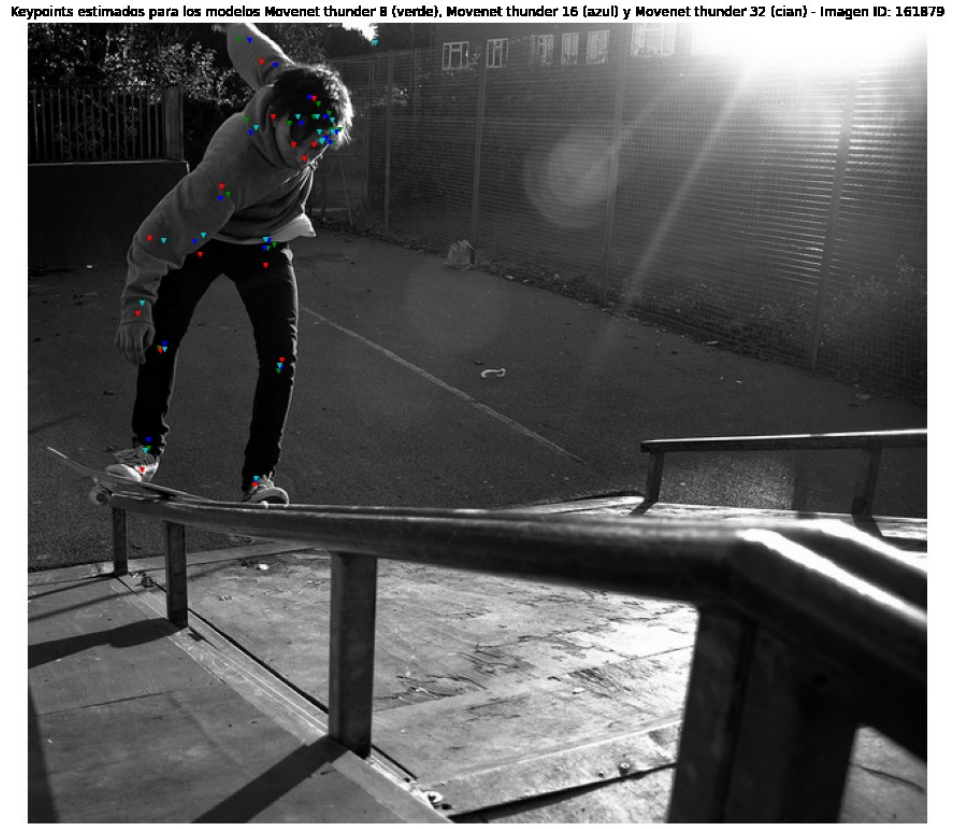


Imagen 62. Keypoints estimados por MoveNet Thunder para imagen 161879

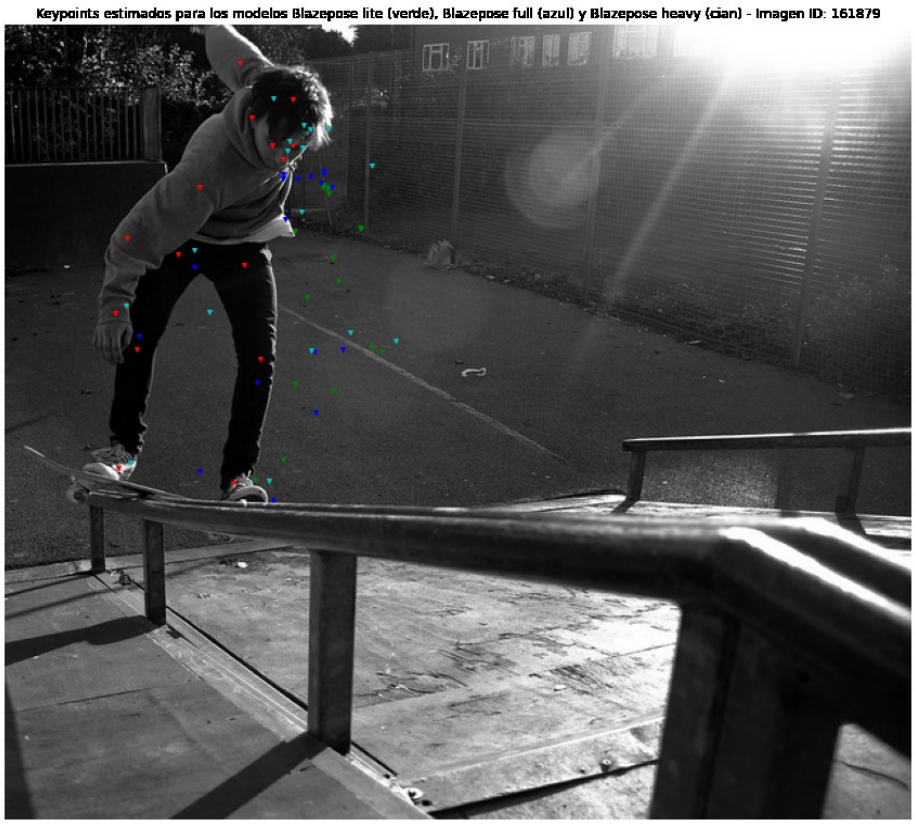


Imagen 63. Keypoints estimados por BlazePose para imagen 161879

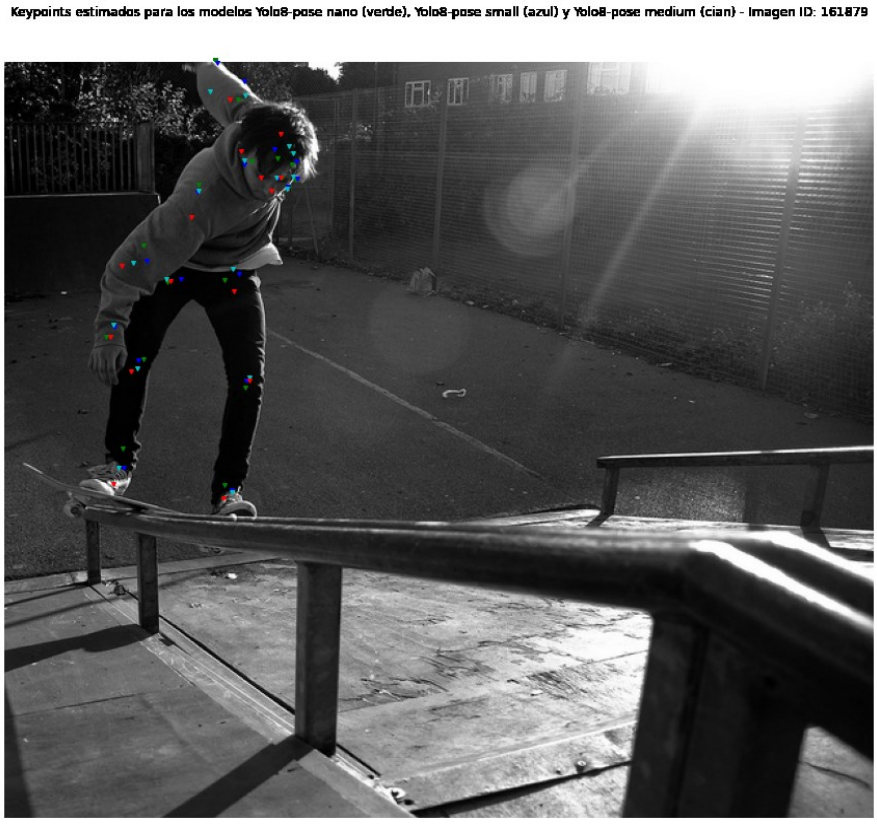


Imagen 64. Keypoints estimados por Yolo8-pose para imagen 161879