UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Clasificación y corrección de posturas humanas en dispositivos móviles

TRABAJO FIN DE MÁSTER
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster Universitario en Inteligencia Artificial,
Reconocimiento de Formas e Imagen Digital

Autor: Ferran Pérez Esteve

Tutores: Roberto Paredes Palacios
Juan Maroñas Molano

Curso 2020-2021

Resum

L'objectiu d'aquest treball és desenvolupar una app mòbil que siga capaç de classificar en temps real diverses postures humanes i de localitzar diferents parts del cos, les imatges que es facen a l'app s'utilitzaran per a una reconstrucció 3D del subjecte. El sistema identificarà diverses característiques fonamentals com són la postura de les dues mans , l'orientació del cap respecte al seu propi cos, la posició dels braços, les cames i els peus. S'utilitzaran mètodes d'aprenentatge automàtic com són les xarxes neuronals per a classificar punts 3D i per a localitzar elements en les imatges, per la qual cosa l'app serà capaç d'identificar i corregir les postures de l'usuari. El treball inclou també la creació d'una base de dades antropomètrica, que contindrà imatges de persones amb diferents característiques realitzant les postures a classificar.

Paraules clau: Classificació de punts 3D, Aprenentatge Profund, Xarxes Neuronals, Android, Postura Humana

Resumen

El objetivo de este trabajo es desarrollar una app móvil que sea capaz de clasificar en tiempo real diversas posturas humanas y de localizar diferentes partes del cuerpo, las imágenes que se tomen en la app se utilizarán para una reconstrucción 3D del sujeto. El sistema identificará varias características fundamentales como son la pose de ambas manos , la orientación de la cabeza respecto a su propio cuerpo, la posición de los brazos, las piernas y los pies. Se utilizarán métodos de aprendizaje automático como son las redes neuronales para clasificar puntos 3D y para localizar elementos en las imágenes, por lo que la app será capaz de identificar y corregir las posturas del usuario. El trabajo incluye también la creación de una base de datos antropométrica, que contendrá imágenes de personas con diferentes características realizando las posturas a clasificar.

Palabras clave: Clasificación de puntos 3D, Aprendizaje Profundo, Redes Neuronales, Android, Postura Humana

Abstract

The aim of this work is to develop a mobile app that is capable of classifying various human postures in real time and locating different parts of the body, the images that are taken in the app will be used for a 3D reconstruction of the subject. The system will identify several fundamental characteristics such as the pose of both hands, the orientation of the head with respect to its own body, the position of the arms, legs and feet. Machine learning methods such as neural networks will be used to classify 3D points and to locate elements in the images, so the app will be able to identify and correct the user's postures. The work also includes the creation of an anthropometric database, which will contain images of people with different characteristics performing the positions to be classified.

Key words: 3D point classification, Deep Learning, Neural Networks, Android, Human Pose

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Visión general

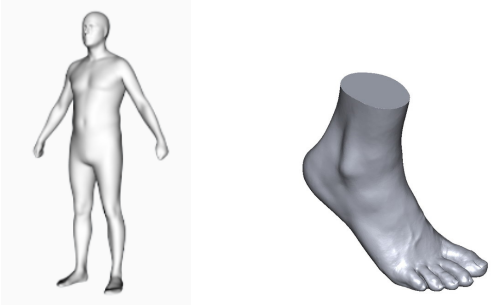
El ámbito de este trabajo incluye la creación y aplicación de sistemas que puedan localizar diferentes partes del cuerpo e identificar su postura en tiempo real a partir de una imagen o un vídeo. El objetivo de controlar estas posturas es realizar una reconstrucción 3D posterior del sujeto, por lo que las posturas a identificar serán bastante específicas. Debido a lo concreto que es el dominio del problema, surge la necesidad de crear una base de datos antropométrica que se pueda utilizar para el entrenamiento de sistemas de aprendizaje automático.

Las posturas y partes del cuerpo que se tienen que identificar presentan una gran variabilidad, debido a múltiples factores como la forma corporal del sujeto, su orientación, distancia con la cámara, condiciones de iluminación etc. Debido a esta complejidad, son necesarias técnicas de deep learning que incluyen la obtención de esqueletos humanos en 3D, detección de objetos en tiempo real y clasificación de puntos 3D. La finalidad del proyecto es integrar estas técnicas de machine learning en una app para dispositivos móviles. Debido a las limitaciones computacionales de estos dispositivos, el trabajo explora que librerías y métodos son los más eficientes para realizar esta clasificación.

1.2 Descripción del problema

La reconstrucción de objetos 3D a partir de imágenes 2D ha recibido una amplia atención en los últimos años, especialmente debido a los avances en el procesamiento de gráficos por computador y a la realidad virtual. Dentro de esta área, tiene especial interés la construcción de sistemas que sean capaces de representar cuerpos humanos en 3D a partir de imágenes bidimensionales, en [1.1] aparecen un cuerpo y un pie reconstruidos a partir de imágenes. Para que estas reconstrucciones tengan un alto de grado de precisión es necesario que las fotografías sean tomadas en unas condiciones específicas, con el objetivo de que la calidad de los datos sea adecuada. En este trabajo no se detallará como se realiza la reconstrucción 3D, sino que se centrará en la creación de un sistema que sea capaz de detectar automáticamente que estas condiciones se cumplen.

Para realizar una reconstrucción en 3D es necesario contar con varias fotografías que muestren diferentes perspectivas del mismo *objeto*, la orientación y el número de imágenes depende del elemento que se vaya a reconstruir.



(a) Reconstrucción 3D cuerpo

(b) Reconstrucción 3D pie

Figura 1.1: Ejemplos de modelos reconstruidos

En el ámbito de nuestro problema, tenemos dos elementos a reconstruir de forma *individual*, el cuerpo y el pie de una persona. El pie tiene una reconstrucción propia y no va incluido en la reconstrucción del cuerpo.

- **Cuerpo:** En la generación de un modelo tridimensional de un cuerpo se necesitan dos imágenes, una en la que la persona aparezca en una vista frontal y otra lateral. En cada una de estas vistas, la persona tiene que realizar una postura específica para que la reconstrucción 3D sea realizada con éxito. La calidad del modelo generado será mucho mejor si la persona realiza de forma correcta cada postura. Cada una de estas posturas tiene una serie de elementos que se tienen que cumplir para que sea considerada correcta, esto incluye la posición de la cabeza, las manos, las piernas, etc.

- **Pie:** Para generar un modelo 3D de un pie se necesitan tres imágenes, dos vistas laterales y una frontal. El pie tiene que estar colocado sobre un folio, que se utiliza como una referencia de medida en el mundo real. En cada imagen el folio tiene que aparecer de forma completa, sin que ninguna esquina esté tapada. Por otro lado, el pie tiene que estar en la posición correcta en cada imagen mostrando la parte del pie que corresponda. Igual que en el caso del cuerpo, estas imágenes tienen que cumplir un conjunto de condiciones para que sea válida, por ejemplo las imágenes tienen que contener el mismo pie(izquierdo o derecho) y el folio y el pie deben ser visibles.

1.3 Objetivos

En esta sección vamos a detallar cuáles son los objetivos principales a alcanzar durante el desarrollo del trabajo:

- Hacer uso de las distintas tecnologías y librerías que se usan actualmente en la implementación de sistemas de machine learning para dispositivos móviles, entre las que se incluyen Keras y TensorFlow Lite.
- Creación de una base de datos antropométrica que contenga imágenes a cuerpo completo y de los pies, las imágenes deberán contener las posturas que se tienen que identificar y estar etiquetadas correctamente.

- Estudiar e implementar las diferentes librerías y técnicas que se usan para obtener un esqueleto 3D del cuerpo de una persona.
- Desarrollar un clasificador que reciba puntos en 3D pertenecientes al esqueleto tridimensional de una persona y sea capaz de discernir si una postura es correcta o no.
- Implementar un detector que sea capaz de reconocer y localizar pies en imágenes, además debe ser capaz de diferenciar los pies según su orientación en la imagen e identificar si es el pie izquierdo o el derecho.
- Realizar una comparación experimental entre los diferentes modelos que se han probado, teniendo en cuenta diversos parámetros como el coste compucional o el acierto. Se analizarán las ventajas y los incovenientes de cada uno.
- Implementación de una aplicación de Android que incluya todos los sistemas de aprendizaje automático desarrollados, esta app tiene que ser capaz de ser ejecutada en tiempo real en un dispositivo móvil y ser visual para el usuario.

1.4 Estructura del documento

Los capítulos de la memoria se esctructuran de la siguiente forma:

Comenzaremos con un repaso del estado del arte referente a las técnicas de obtención de poses humanas en 3D, la clasificación de puntos 3D y la detección de objetos en tiempo real. En el siguiente capítulo se explica la metodología del proyecto, donde se detalla cuál ha sido el proceso de desarrollo y análisis de la solución, así como las herramientas de software y de hardware que se han utilizado para la implementación.

A continuación se aborda el desarrollo, en este apartado se incluye un estudio minucioso de los datasets utilizados para entrenar los sistemas de aprendizaje automático. Además se introducen las descripciones y análisis de los diferentes métodos utilizados para resolver la validación de imágenes. También aparece una explicación detallada de como se ha llevado a cabo la integración de estos sistemas en una app para dispositivos móviles. El capítulo por lo tanto está divido en tres partes bien diferenciadas, la validación de fotografías del cuerpo completo, la validación de fotografías del pie y la implementación de la app.

En el siguiente capítulo aparecen los resultados de los diferentes experimentos llevados a cabo durante el desarrollo del proyecto. En la sección se presentan comparaciones entre las diferentes técnicas y modelos utilizados para clasificar puntos 3D y para la detección de objetos en tiempo real. Se usan diversas métricas para hacer un balance de estos modelos y analizar las ventajas e inconvenientes de cada uno de ellos. Por último, la memoria finaliza con unas conclusiones que mimetizan todo lo conseguido durante el desarrollo del proyecto, así como las posibles mejoras en un trabajo futuro.

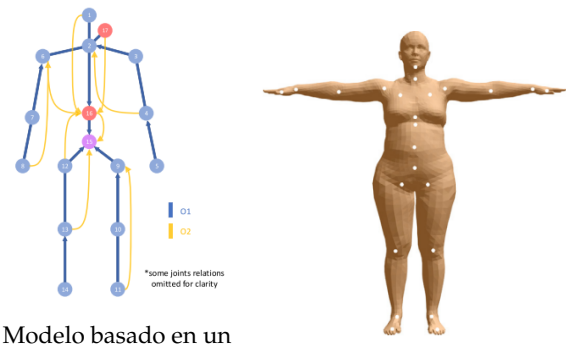
CAPÍTULO 2
Estado del arte

Este capítulo explora las diferentes alternativas que se utilizan en la actualidad para resolver los dos problemas fundamentales de nuestro proyecto: la obtención de la pose humana en 3D y la detección de objetos en tiempo real.

2.1 Pose humana 3D

La estimación de la pose humana 3D es la tarea de predecir las ubicaciones de puntos de referencia en 3D de un cuerpo humano a partir de una imagen o una secuencia de imágenes de esa persona. Debido a su amplia gama de aplicaciones, la estimación de la pose humana es un área de investigación muy activa en el campo de visión artificial. En [1] se hace un repaso de las técnicas de deep learning que han permitido un gran avance en la resolución de este problema. En el artículo se exponen diferentes modelos de representación para la estimación de poses humanas en 3D:

- **Modelo basado en esqueletos:** En este modelo se usa una estructura de árbol con puntos clave del cuerpo humano como representación. Es el tipo de modelo más extendido actualmente, en la figura 2.1a tenemos un posible ejemplo de este modelo.
- **Modelo SMPL:** En el modelo SMPL(skinned multi-person linear) el cuerpo humano se representa como una malla triangulada de 6890 vértices. Esta malla está parametrizada con parámetros de forma y pose. Un ejemplo de modelo SMPL aparece en 2.1b.



(a) Modelo basado en un esqueleto

(b) Modelo SMPL

Figura 2.1: Modelos de representación - Pose Humana 3D

Otro de los factores claves es que métrica se usa para evaluar como de bueno es un modelo, en el artículo se mencionan los más significativos:

- **Mean Per Joint Position Error (MPJPE).** Distancia euclídea entre las articulaciones(*joints*) de la pose 3D estimada y de la real. Se calcula con:

$$E_{\text{MPJPE}}(f,S) = \frac{1}{N_s}\sum_{i=1}^{N_s}\left\| P_{f,S}^{(f)}(i) - P_{gt,S}^{(f)}(i)\right\|_2$$

  Donde f denota un frame y S el correspondiente esqueleto. $P_{f,S}^{(f)}(i)$ es la posición estimada del *joint*, y $P_{gt,S}^{(f)}(i)$ es la posición correcta. Se hace la media con los MPJPEs de todos los frames.

- **Percentage of Correctly estimated Parts (PCP).** Mide el porcentaje de partes del cuerpo con una predicción correcta, una parte del cuerpo se considera correcto si:

$$\frac{\|s_n - \hat{s}_n\| + \|e_n - \hat{e}_n\|}{2} \leq \alpha\|s_n - e_n\|$$

  Donde s_n y e_n son la ubicación inicial y final real respectivamente, mientras que $\hat{s}_n$ y $\hat{e}_n$ representan las ubicaciones estimadas correspondientes, α es un parámetro umbral.

- **Mean of the Root Position Error (MRPE).** Esta métrica evalúa la precisión de la ubicación absoluta de la pose humana 3D estimada.

$$MRPE = \frac{1}{N}\sum_{i=1}^{N}\left\| R^{(i)} - R^{(i)*}\right\|_2$$

  Donde $R^{(i)}$ y $R^{(i)*}$ son el valor real y el estimado de las localizaciones del ejemplo i.

2.1.1. PoseNet

PoseNet [2] [3] es uno de los modelos más utilizados y estudiados actualmente para la estimación de pose 3D. Predice un mapa de calor de probabilidad para cada articulación(*joint*), la articulación obtenida será el punto con la máxima verosimilitud en el mapa. Dado un mapa de calor H_k para el k^{th} joint, cada localización en el mapa representa la probabilidad de que el *joint* esté en esa ubicación. Las coordenadas finales del *joint* J_k se obtiene de la localización p con la máxima verosimilitud:

$$J_k = \arg\max_p H_k(p)$$

El modelo consiste en dos partes. La primera parte es una red neuronal convolucional(CNN) que extrae un mapa de características de la imagen. La seguda parte, toma este mapa de características y usa tres capas deconvolucionales con *batch normalization* para predecir el mapa de calor con los *joints*.

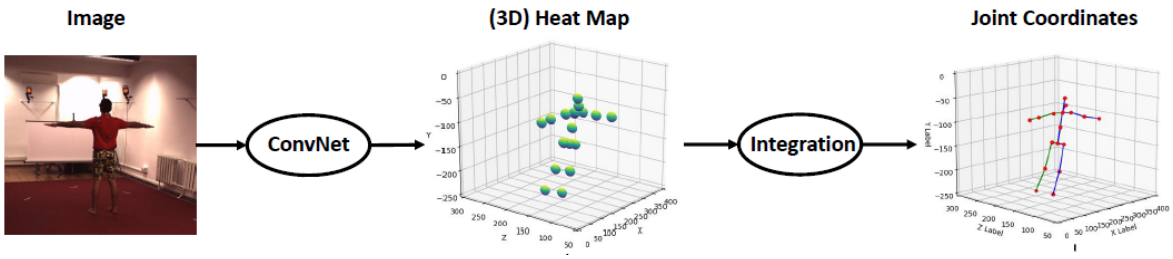


Figura 2.2: Pipeline básico Posenet

En la figura 2.2 se representa el pipeline básico de PoseNet, pueden haber variaciones, dependiendo de la CNN que se use como extractor de características. También se puede usar un detector de personas, de este modo le pasamos a la CNN la imagen recortada que contiene a la persona.

2.1.2. BlazePose

BlazePose [4] [5] es el modelo de predicción de pose 3D que usa el framework Mediapipe, este mismo modelo con algunas modificaciones menores se utiliza para predicir la pose de las manos o de la cara entrenándolo con los datasets apropiados.

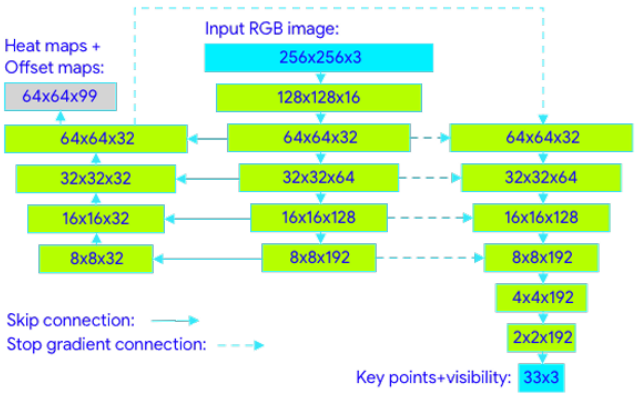


Figura 2.3: Arquitectura Blazepose

Este modelo tiene una diferencia fundamental respecto al PoseNet, ya que en este caso se usa un enfoque combinado de mapa de calor y de regresión. La pérdida del mapa de calor solamente se usa en la etapa de entrenamiento, y su correspondiente capa de salida se elimina al ejecutar la inferencia, por lo que se queda solo con la regresión, eliminar los mapas de calor es un punto clave, ya que hace al modelo lo suficientemente ligero para que se puede ejecutar en un teléfono móvil. Otro punto interesante de esta red es que se utilizan *skip-connections* para tener características de alto y de bajo nivel, como se ve representado en 2.3, donde aparece la arquitectura de la red.

Durante la inferencia, se ejecuta el detector hasta que se localiza a una persona. Una vez se detecta, se ejecuta el modelo de estimación de pose 3D. Este modelo tiene como salida la ubicación de los *keypoints* del cuerpo, si se detecta o no una persona, la nueva ROI(Region of interest) calculada a partir de la ROI anterior y la predicción de la pose. Si la red indica que no hay ninguna persona, se vuelve a ejecutar el detector. En la figura 2.4 aparece de forma gráfica el *pipeline* que acabamos de explicar, este mismo procedimiento se puede utilizar para obtener la pose de las manos y la cara.

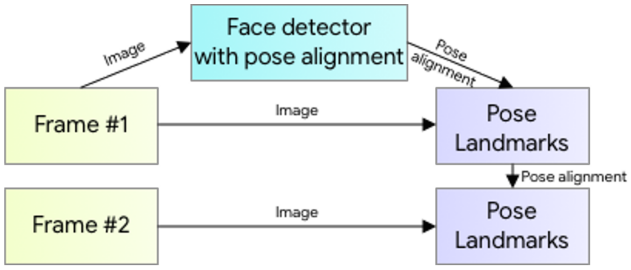


Figura 2.4: Pipeline Detector + BlazePose

2.2 Detección de objetos en tiempo real

La detección de objetos tiene como objetivo identificar y localizar objetos en una imagen o video. Este tipo de modelos se puede utilizar para contar objetos en una escena y determinar sus ubicaciones precisas, además de etiquetarlos con la clase correspondiente. Debido a las grandes variaciones en los puntos de vista, poses, oclusiones y condiciones de iluminación, es difícil conseguir una detección de objetos que funcione perfectamente.

Se usan diversas métricas para la detección de objetos en tiempo real, las más importantes son:

- **Intersection over union (IoU)**: IoU mide el grado de similitud entre 2 conjuntos. En el numerador calculamos el área de superposición entre el *bounding box* predicho y el real. El denominador contiene el área de unión de ambas *bounding boxes*:

$$J(A,B) = \frac{A \cap B}{A \cup B}$$

- **Average precision:** Es una de las métricas más usadas para *object detection*. Se establece un umbral (por ejemplo Iou>0.5), para que una predicción se considere un verdadero positivo. A continuación se calculan la *Precision* y el *Recall* de la siguiente forma:

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

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

  Donde TP son los verdaderos positivos, FP los falsos positivos y FN los falsos negativos. Para obtener el *Average Precision* se calcula el AUC(área bajo la curva) de la curva precision-recall.

- **Mean average precision:** Para calcular esta métrica, se promedia el AP para todas las clases, obteniendo el MaP del modelo. Cuanto mayor sea este valor, mejor será el modelo.

Una vez definidas las métricas, vamos a explicar las principales 'familias' de modelos que se utilizan para resolver esta tarea, los modelos en los que vamos a profundizar son los basados en *R-CNN*, *YOLO* y *SSD*.

2.2.1. R-CNN

Los modelos basados en R-CNN(*Region Based Convolutional Neural Networks*) [7] [8] utilizan una búsqueda selectiva para extraer regiones de interés(ROI), a partir de estas ROI se computan una serie de características mediante una red convolucional y se clasifica con su etiqueta.

En [6] se hace un resumen del funcionamiento de este tipo de modelos, básicamente consta de tres partes:

- **Propuesta de regiones:** Se usa un búsqueda selectiva para generar aproximadamente 2.000 propuestas de regiones para cada imagen.

- **Extracción de características basada en CNN:** En esta fase, cada región propuesta se recorta a una resolución fija y la red CNN se utiliza para extraer un vector de características de 4096 dimensiones como representación. Debido a la gran capacidad de aprendizaje de estas redes, se puede conseguir una representación robusta y semánticamente rica de cada región propuesta.
- **Clasificación:** Se utilizan Support Vector Machines(SVM) previamente entrenadas en múltiples clases para realizar la clasificación, recibe como entrada las características extraídas por las CNN. Finalmente, se usa *non maximum suppression* (NMS) para obtener las *bounding boxes* finales.

En la figura 2.5 se puede observar de forma resumida y gráfica el prodimiento explicado para detectar objetos en R-CNN.

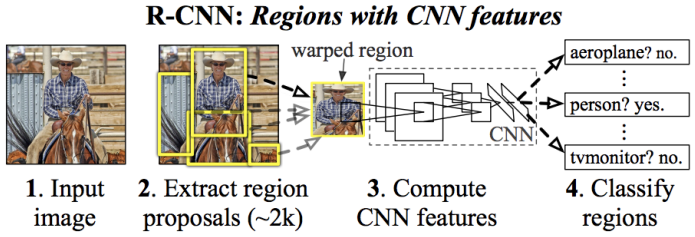


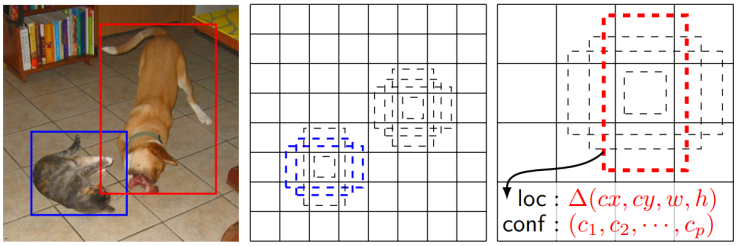
Figura 2.5: Pipeline R-CNN

A partir de la R-CNN, se han propuesto modelos que se basan en los mismos principios pero que introducen mejoras respecta a la versión original, por ejemplo, **Fast R-CNN** ejecuta la red neuronal una única para cada imagen, mientras que anteriormente el vector de características de cada ROI se calculaba de forma individual. Posteriormente, **Faster R-CNN** integra la propuesta de regiones dentro de la propia red neuronal.

2.2.2. SSD

Single Shot MultiBox Detector(SSD)[9] es un método para detectar objetos en imágenes utilizando una única red neuronal. SSD discretiza el espacio de salida de los *bounding boxes* en un conjunto de *bounding boxes* 'por defecto' en diferentes proporciones y escalas en el mapa de características. Además, la red combina predicciones de múltiples mapas de características con diferentes resoluciones para manejar de forma natural objetos de varios tamaños. El modelo SSD es bastante simple en relación con los métodos basados

en propuestas de regiones ya que encapsula todos los cálculos en una sola red. Las mejoras de este sistema incluyen el uso de filtros convolucionales pequeños para predecir las categorías de los objetos y los *offsets* en los localizaciones de los *bounding box*, usando predictores separados para detecciones con diferentes relaciones de aspecto.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figura 2.6: Detección SSD

En la figura 2.6 vemos como funciona la detección de objetos en este tipo de modelos. En cada celda del mapa de características, se predicen los *offsets* en relación con las *bounding box* predeterminadas de la celda. También se predicen las puntuaciones de cada clase, que indican la presencia de una clase en cada uno de esos cuadros. Específicamente, para cada una de las K *bounding boxes* predeterminadas en una ubicación dada, se calcula la puntuación de la clase c y los 4 *offsets* relativos al original $\triangle(cx, cy, w, h)$.

2.2.3. YOLO

You Only Look Once(YOLO) [10] es un de los sistemas de detección de objetos más usados en la actualidad. Hace uso de una única red neuronal convolucional para detectar objetos en imágenes. El sistema trata la detección de objetos como un problema de regresión, divide la imagen en un *grid* de SxS y para cada una de ellas predice B *bounding boxes*, su valor de confianza y C probabilidades de clases. Estas predicciones se codifican con un tensor de $SxS(B * 5 + C)$.

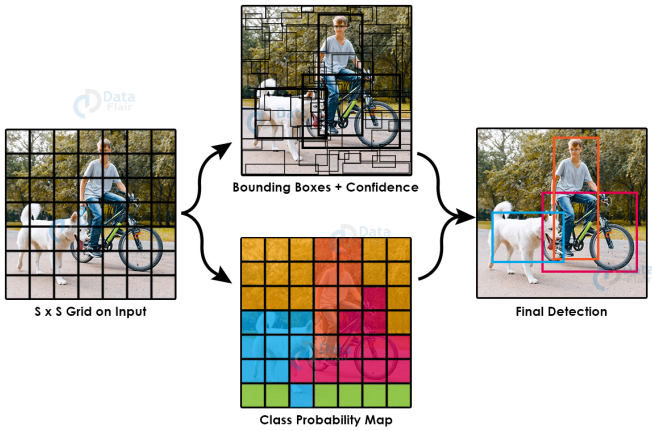


Figura 2.7: Detección mediante Yolo

Cada *bounding box* consiste en 5 predicciones: *x*, *y*, *w*, *h* y un valor de confianza. Las coordenadas (x,y) representan el centro del *bounding box* relativo a los límites del *grid*. El ancho y el alto se predicen de forma relativa a toda la imagen. El valor de confianza representa el IOU entre la predicción del *bounding box* y su valor real. En la figura 2.7 vemos como *YOLO* realiza la detección, primero se realiza la división en *SxS Grids*, a

continuación se obtienen las *bounding boxes* junto con su valor de confianza y mapa de probabilidades de clase. Con esta información se realiza la detección final, se usa *Non-maximal suppression* para arreglar las detecciones múltiples.

YOLOv2 [11] [12] es una versión mejorada del modelo original que tiene como objetivo paliar los puntos débiles más significativos. Comparado con *Fast R-CNN*, YOLO comete un elevado número de errores de localización, además tiene un *recall* relativamente bajo en comparación con modelos basados en *R-CNN*. En [12] se hace un resumen de algunas de las principales mejoras que introduce esta nueva versión:

- **Batch Normalization(BN):** Usar *Batch normalization* mejora la convergencia y elimina la necesidad de otras formas de regularización, añadiéndolo en todas las capas convolucionales se consigue una mejora de aproximadamente un 2 % en el mAP.

- **Clasificación de alta resolución:** La versión de YOLO original entrena la red de clasificación a a 224x224 y incrementa la resolución a 448 para la detección. En *YOLOv2* se hace un *fine tunning* durante 10 épocas en la red de clasificación con una resolución de 448x448. Esto mejora significativamente el mAP.

- **Anchor boxes:** YOLO predice las coordenadas de los *bounding boxes* directamente, mientras que *YOLOv2* usa *Anchor boxes* para realizar la predicción. De este modo se desacopla la predicción de classes de la localización espacial del objeto.
- **Fine-grained features:** Mapas de características de 13 × 13 y 26x26, mejora en objetos pequeños.
- **Multi-Scale Training:** Cada 10 *batches* la red elige aleatoriamente una nueva dimensión de imagen, desde 320x320 a 608x608. De este modo la red aprende a predecir con diferentes dimensiones de entrada.

CAPÍTULO 3
Metodología

En este capítulo explicaremos cuál ha sido la metodología que se ha usado durante el desarrollo del proyecto, así como las herramientas de software y de hardware que se han utilizado para su implementación.

3.1 Pasos en el desarrollo del proyecto

1. **Estudio de las diferentes arquitecturas de predicción de Pose 3D y de los modelos de detección de objetos**

   El primer paso ha sido realizar un estudio de las técnicas que se usan actualmente en la obtención de la Pose 3D a partir de una única imagen, poniendo especial atención en aquellos métodos que puede usarse en tiempo real en dispositivos móviles. También se han estudiado los diferentes modelos de detección de objetos, en este caso también prestando atención a su rendimiento en móviles.

2. **Análisis y creación del conjunto de datos**

   Uno de los puntos claves en el aprendizaje supervisado son los datos usados para el entrenamiento. Por ello, se ha llevado a cabo un análisis exhaustivo de los datasets existentes relacionados con la pose humana. No obstante, debido a lo concreto de nuestro problema, ha sido necesario el uso de datasets propios para entrenar nuestros modelos. Durante esta fase del proyecto es importante detectar cuáles son los potenciales problemas e inconvenientes que tiene este conjunto de datos para su posterior uso.

3. **Implementación del procesado de datos y de la arquitectura**

   Una vez realizada la parte más analítica, se debe proceder a la implementación del modelo. En esta parte se incluyen las técnicas utilizadas para procesar y acceder al conjunto de datos de forma eficiente. También se deben implementar todas las arquitecturas que posteriormente se utilizarán y validarán.

4. **Entrenamiento y experimentación**

   En esta fase se han probado diferentes arquitecturas y técnicas con el objetivo de compararlas y obtener el máximo rendimiento. Los experimentos se han realizado cambiando múltiples parámetros, entre los que se varias arquitecturas de deep learning y transformaciones en los datos.

5. **Implementación de la app móvil e integración de los sistemas de machine learning**

   El último paso en el proyecto es el desarrollo de una aplicación que pueda funcionar en dispositivos *Android*, es importante que la aplicación sea simple e intuitiva de cara a un posible usuario. Además en esta app se tiene que integrar todos los modelos previamente entrenados para su inferencia en tiempo real.

3.2 Herramientas utilizadas

3.2.1. Software

En este punto se va a describir brevemente el software que se ha utilizado para el desarrollo del proyecto, primero se exponen las *librerías* usadas:

Keras[1]**:** Es una API de redes neuronales de alto nivel, escrita en Python y capaz de ejecutarse sobre TensorFlow, CNTK y Theano. Fue desarrollada con el objetivo de permitir la experimentación rápida en redes convolucionales y redes recurrentes. Es capaz de ejecutarse tanto en CPU como en GPU.

TensorFlow Lite[2]: TensorFlow es una plataforma de código abierto enfocada a la implementación de sistemas de aprendizaje automático. Cuenta con un framework llamado *TensorFlow Lite*, que se especializa en la implementación de estos sistemas para su inferencia en dispositivos móviles.

Mediapipe [3]: MediaPipe es un framework para la construcción sistemas de aprendizaje automático aplicada a diferentes ámbitos(obtención de Pose3D, detección de rostros, segmentación de pelo, detección de objetos en 3D). Es multiplataforma (Android, iOS, web, dispositivos periféricos). Con MediaPipe, se puede construir un *pipeline* como un gráfico de componentes modulares e incluir modelos de inferencia (por ejemplo, TensorFlow, TFLite).

A continuación se muestras los *entornos de desarrollo* que se han utilizado:

Android Studio[4]: Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android, está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Cuenta con un sistema de compilación flexible basado en Gradle y un emulador de Android entre otras características.

Google Colab[5]: Es un entorno colaborativo de Google que permite trabajar con Notebooks y el lenguaje de programación de Python. Permite almacenar dichos cuadernos y trabajar con datos que tengas almacenados en Google Drive. Colab ofrece acceso gratuito a recursos informáticos, como GPUs, por ello es especialmente adecuado para tareas de aprendizaje automático y análisis de datos.

[1] https://keras.io/
[2] https://www.tensorflow.org/
[3] https://mediapipe.dev/
[4] https://developer.android.com/studio
[5] https://colab.research.google.com

3.2.2. Hardware

Para la implementación de sistemas de *deep learning* en dispositivos móviles, es necesario que estos dispositivos cuenten con GPUs que nos permitan ejecutar cálculos en paralelo y obtener unos tiempos de inferencia admisibles. Para ello, se han utilizado los siguientes dispositivos en las pruebas realizadas durante el desarrollo de la app:

POCO X3	
SO	Android 10
CPU	CPU Qualcomm Kryo 470, ocho núcleos, 2.3 GHz
RAM	6 GB LPDDR4
GPU	Adreno 630

Tabla 3.1: Características POCO X3

SAMSUNG S9	
SO	Android 10
CPU	Samsung Exynos 9810, octa-core, 2.9 GHz
RAM	4 GB
GPU	Adreno 618

Tabla 3.2: Características SAMSUNG S9

CAPÍTULO 4
Desarrollo

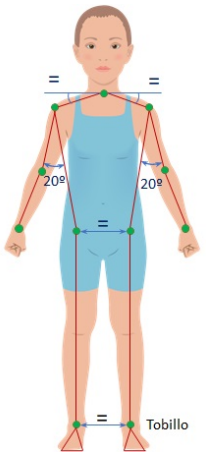
El desarrollo está dividido en tres partes que constituyen el esqueleto del proyecto: validación de imágenes del cuerpo, validación de imágenes del pie e implementación de la app móvil.

4.1 Validación de imágenes del cuerpo

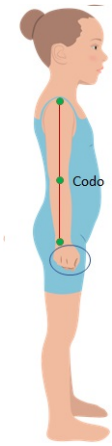
Como ya se ha comentado en la introducción, el objetivo de tomar imágenes del cuerpo es realizar una reconstrucción 3D posterior, pero para que esa reconstrucción sea lo más precisa posible es necesario tener imágenes en unas ciertas condiciones. En esta sección se explicará paso por paso como se lleva a cabo la validación de imágenes del cuerpo, pasando por los requisitos de la postura, el conjunto de datos utilizados, la predicción de la pose 3D y la clasificación de puntos 3D.

4.1.1. Requisitos de la postura

En este apartado vamos a explicar con detalle los requisitos que tiene que tener la postura del usuario. Hacen falta dos imágenes para realizar la reconstrucción, una en la que la persona aparece de frente y otra de forma lateral. En la figura 4.1 aparecen las dos posturas que tiene que realizar el usuario.



(a) Postura frontal de la persona



(b) Postura lateral de la persona

Figura 4.1: Posturas usadas para la reconstrucción 3D del cuerpo

A continuación detallaremos los requisitos que tienen que tener estas posturas para que sean consideradas correctas para ambas posturas, empezaremos por la **postura frontal**:

- **Aparece todo el cuerpo de forma completa:** Es muy importante que en la imagen todo el cuerpo sea visible y no haya ninguna parte del mismo cortada o tapada.
- **Cuerpo y cabeza bien orientados:** Tanto la cabeza como el cuerpo tienen que estar orientados hacia la cámara. Es importante que la cabeza esté completamente recta y mirando al frente. Además, la espalda tiene que estar completamente erguida.
- **Apertura brazos:** Los brazos tienen que estar extendidos y completamente rectos, tienen que formar aproximadamente un ángulo de 20 grados con la vertical que forma el propio cuerpo.
- **Apertura piernas:** Las piernas deben estar completamente rectas y alineadas con los hombros del usuario.
- **Puños cerrados:** Ambos puños tienen que estar bien cerrados y orientados de forma correcta hacia la cámara. En la figura 4.2 vemos cual es la orientación correcta que tiene que tener la mano cerrada para que sea correcta.

(a) Postura bien orientada **(b)** Puño mal orientado

Figura 4.2: Orientación correcta mano - Postura frontal

Una vez descritos los requisitos que tiene que tener la postura frontal vamos a continuar con la **postura lateral**:

- **Aparece uno de los laterales de forma completa:** Igual que en la postura frontal, no pueden haber partes tapadas o cortadas, pero en este caso, como la vista es lateral solamente se tiene que ver un lateral del cuerpo(un brazo, una mano, un lateral de la cara...).
- **Cuerpo y cabeza bien orientados de forma lateral:** En este caso también es importante la orientación del cuerpo y la cabeza, que tiene que tener la orientación que aparece en la figura 4.1b.
- **Posición brazo:** El brazo que aparece en lateral tiene que estar completamente recto y extendido, también debe estar pegado al cuerpo.
- **Posición pierna:** Las piernas tiene que estar rectas y extendidas, aunque en esta posición no es muy relevante controlar esta parte del cuerpo, por lo que hay un grado de flexibilidad mayor.
- **Mano extendida:** En este caso la mano no tiene que estar cerrada, sino que tiene que estar extendida, con el dorso de la mano apuntando a la cámara y los dedos juntos.

4.1.2. Conjunto de datos

El primer paso que se ha realizado para ver que *dataset* usar es explorar los ya existentes. Hay una gran cantidad de *datasets* relacionados con posturas humanas, por ejemplo, tenemos el *MPII Human Pose Dataset*[1], que incluye unas 25.000 imágenes que cubren unas 410 actividades humanas, en la figura 4.3 vemos algunos ejemplos de estas actividades.

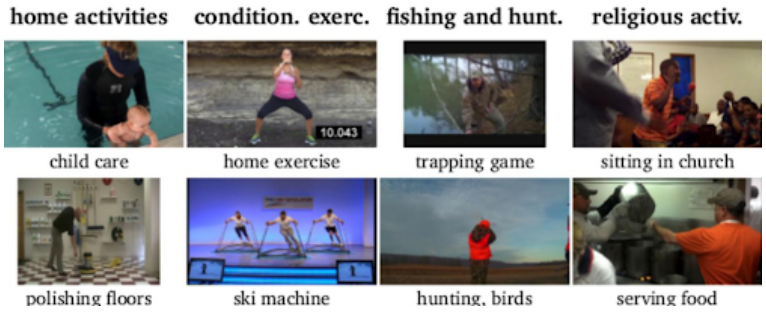


Figura 4.3: Human Pose Dataset

Otro *dataset* interesante relacionado con posturas humanas es el *Yoga-82*[2], que continen 82 posturas de Yoga con una gran diversidad de personas, orientaciones y variaciones en las posiciones del cuerpo. En la figura 4.4 aparecen algunas de las posiciones de yoga que contiene el conjunto de datos.

Figura 4.4: Yoga-82

Debido a lo concreto que son las posturas que hemos descrito anteriormente los *datasets* ya existentes **no nos sirven**, porque no se adaptan a las necesidades del problema. Por lo tanto es necesario utilizar un conjunto de imágenes propio, que esté diseñado y adaptado para este problema. A continuación vamos a describir el *dataset* que se ha creado exclusivamente para la resolución de este problema. Las imágenes no solo tienen que contener a personas realizando la postura de forma correcta, sino que es imprescindible que en las imágenes también haya personas realizando la postura de forma incorrecta, ya que los sistemas de aprendizaje automático necesitan aprender también de ejemplos negativos. Como sería muy complicado cubrir todos los posibles errores, el conjunto datos intenta cubrir los errores más comunes.

En total hay 2300 imágenes donde aparecen personas realizando las posturas descritas, en la tabla 4.1 vemos como se reparten las imágenes entre la postura frontal y lateral, y entre posturas correctas e incorrectas, una postura se considera correcta si se cumplen **todos** los requisitos descritos anteriormente para cada postura.

[1] http://human-pose.mpi-inf.mpg.de/
[2] https://sites.google.com/view/yoga-82/home

Conjunto imágenes	
Postura	**Num. Imágenes**
Frontal correcta	600
Frontal incorrecta	535
Lateral correcta	600
Lateral incorrecta	565

Tabla 4.1: Distribución conjunto de datos - Postura

Postura frontal correcta: En la figura 4.5 tenemos ejemplos de imágenes que contienen a una persona haciendo la postura frontal de forma correcta, como podemos ver, se cumplen todos los requisitos descritos anteriormente. En total, hay 600 imágenes de este estilo, para que el conjunto de datos sea significativo, las imágenes contiene a personas con diferentes formas corporales, condiciones de ilumunación, ropa etc. Las caras de las personas que aparecen en las fotografías están **pixeladas por cuestiones de privacidad**.

(a) Frontal correcta(1)

(b) Frontal correcta(2)

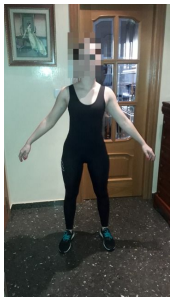
(c) Frontal correcta(3)

(d) Frontal correcta(4)

Figura 4.5: Imágenes postura frontal correcta

Postura frontal incorrecta: En la figura 4.6 tenemos imágenes que contienen al menos un error en la postura frontal. En la imagen 4.6a la orientación de la mano es incorrecta, ya que el dorso de la mano está apuntando hacia delante. En 4.6b las manos están mal, ya que están ligeramente abiertas. En la fotografía 4.6c, la orientación del cuerpo no es correcta, ya que tanto la cabeza como el cuerpo no están de frente a la cámara, además las manos no están bien cerradas ni orientadas. Por último, en la imagen 4.6d ni la orientación del cuerpo ni la apertura de los brazos está bien. En total, tenemos 535 imágenes con algún error en la postura frontal.

(a) Frontal incorrecta(1)



(b) Frontal incorrecta(2)

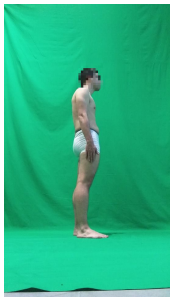
(c) Frontal incorrecta(3)

(d) Frontal incorrecta(4)

Figura 4.6: Imágenes postura frontal incorrecta

Postura lateral correcta: En la figura 4.7 aparecen ejemplos de posturas laterales realizadas de forma correcta, igual que en los casos anteriores, se han tomado tomando en cuenta variaciones en la formas corporales. En total tenemos 600 imágenes de este tipo.

(a) Lateral correcta(1)



(b) Lateral correcta(2)

(c) Lateral correcta(3)

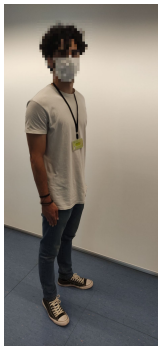
(d) Lateral correcta(4)

Figura 4.7: Imágenes postura lateral correcta

Postura lateral incorrecta: En la figura 4.8 aparecen ejemplos de posturas laterales realizadas de forma incorrecta. En la imagen 4.8a la orientación del cuerpo y la cabeza está bien, pero la mano está cerrada. En 4.8b tanto la orientacioón del cuerpo-cabeza como la postura de las manos es incorrecta, ya que vemos que el cuerpo está un poco ladeado. Respecto a la mano, debería tener los dedos juntos, pero en la imagen el usuario los tiene separados. En la fotografía 4.5c la postura de la mano es correcta, pero la orientación del cuerpo no. Por último, en 4.8d la mano del usuario está cerrada. El conjunto de datos contiene 565 imágenes de este estilo.

(a) Lateral incorrecta(1)

(b) Lateral incorrecta(2)



(c) Lateral incorrecta(3)

(d) Lateral incorrecta(4)

Figura 4.8: Imágenes postura lateral incorrecta

En casi todas las imágenes los errores se deben a que la orientación del cuerpo-cabeza o la postura de las manos, por ello, se decidió añadir información relacionada con estos errores para saber que está fallando concretamente en cada fotografía. De este modo, si una imagen se etiqueta como **incorrecta** se añade más información, concretamente los siguientes elementos los etiquetaremos como correctos o incorrectos:

- **Postura frontal:** Mano izquierda, mano derecha, orientación cuerpo-cabeza.
- **Postura lateral:** Mano lateral [3], orientación cuerpo-cabeza.

Por ejemplo, en la figura 4.6b marcaríamos la mano derecha e izquierda como erróneas y la orientación como correcta. En la imagen 4.8c sería al contrario, la mano se etiquetaría como correcta y la orientación como incorrecta. Esta información nos permite afinar mejor donde está error y le da mayor riqueza al conjunto de datos.

[3]Se considera *mano lateral* a la mano que esté orientada hacia la cámara.

Postura Frontal	
Etiqueta	**Num. Elementos**
Mano correcta	1357
Mano incorrecta	914
Orientación correcta	738
Orientación incorrecta	427

Tabla 4.2: Distribución datos postura frontal

En la tabla 4.2 tenemos la distribución de datos para la postura frontal, como se puede observar, hay un desbalanceo entre los elementos etiquetados como correctos y los erróneos, debido a que aún intentando balancear las imágenes es mucho más común realizar la postura de forma correcta. Tenemos más de 2000 etiquetas de manos debido a que de cada imagen se extraen 2 manos en la postura frontal.

Postura Lateral	
Etiqueta	**Num. Elementos**
Mano correcta	663
Mano incorrecta	502
Orientación correcta	632
Orientación incorrecta	533

Tabla 4.3: Distribución datos postura lateral

En 4.3 aparece la distribución de datos en la postura lateral, en este caso sigue habiendo desbalanceo entre las clases, pero es un poco menos significativo que en el caso anterior. Hay menos manos porque en la postura lateral solo tenemos en cuenta una mano.

4.1.3. Predicción de la pose 3D

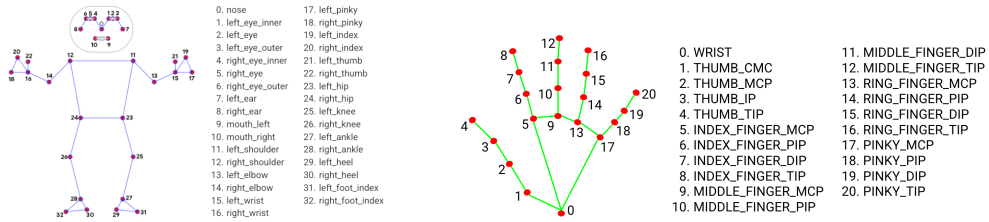
La obtención de la pose 3D de una persona a partir de imágenes o vídeos nos sirve de punto de partida para determinar si su postura es correcta. Es imprescindible que esta predicción pueda ejecutarse en tiempo real en dispositivos móviles, intentando conseguir la mayor precisión posible. Debido a la complejidad del problema y el nivel de optimización necesarios para resolver esta tarea, se han buscado librerías o modelos que nos sirvan como base para resolver el problema y poder implementarlo dentro de una app.

Las dos principales opciones que se han barajado para la obtención de la Pose 3D han sido **PoseNet(Tensorflow)** y **BlazePose(Mediapipe)**. El funcionamiento básico de los modelos ya ha sido explicado en el apartado de estado del arte. Ambos sistemas tiene ventajas e inconvenientes que se han valorado para hacer la elección final:

- **PoseNet:** La gran ventaja que tiene PoseNet es la facilidad que tiene su implementación, ya que está incluido dentro de *Tensorflow Lite*, por lo que su uso e instalación es bastante sencilla. No obstante, la representación de la pose humana es bastante limitada, con únicamente 16 puntos, además su rendimiento no está tan optimizado como lo está el modelo BlazePose en Mediapipe.

- **BlazePose:** El modelo BlazePose cuenta con una representación muy rica del cuerpo humano, con 33 puntos para el cuerpo y posibilidad de realizar una predicción precisa sobre ambas manos(21 puntos 3D en cada una) y sobre la cara. Además, su nivel de optimización en dispositivos móviles es muy alta. Sin embargo, Mediapipe es una librería bastante nueva y experimental, por lo que la implementación de estos modelos es bastante más complicada que la de PoseNet, teniendo que construir el *pipeline* del sistema dependiendo de nuestras necesidades.

Debido a la gran riqueza que tiene la representación 3D de Mediapipe, al final se ha optado por usar *BlazePose* para la implementación en nuestra app. A continuación, vamos a detallar con más precisión el funcionamiento de este sistema. Con *Mediapipe*, el cuerpo humano se representa con 33 puntos 3D que marcan diferentes partes del cuerpo, por otro lado, cada mano se representa con 21 puntos 3D. En la figura 4.11 aparece la representación de los *keypoints* que acabamos de comentar.



(a) Puntos 3D cuerpo) **(b)** Puntos 3D mano

Figura 4.9: Representación del cuerpo y la mano en Mediapipe

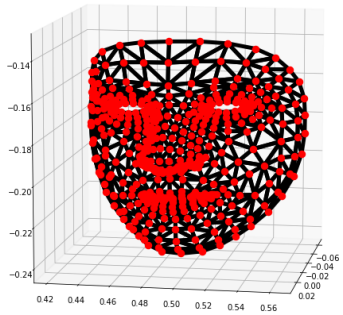
Para ver de forma más clara los resultados que arrojan los modelos vamos a utilizar la figura 4.10. Se ha usado la fotografía 4.10a como ejemplo para extraer la pose 3D, en la imagen un usuario está realizando la postura frontal de forma correcta. Como vemos en la figura, con *Mediapipe* podemos obtener una representación 3D de la cara, el cuerpo y las manos.

- **Cara 3D:** Con *Mediapipe* podemos obtener 468 *keypoints* 3D, en la figura 4.10b vemos la predicción que se consigue a partir de la imagen original. En principio, la representación de los puntos 3D está en valor absoluto respecto al tamaño de la imagen, pero en nuestro caso se ha normalizado de forma relativa al *bounding box* que detecta la cara, por lo que los valores de los *landmarks* están entre 0 y 1. Como tenemos muchos puntos, se utilizarán únicamente los puntos más significativos. Se escogen los puntos que forman el contorno de la cara, los infraorbitales, los de abajo de la boca y los cercanos a la nariz. Se escogen estos puntos porque son los que mejor representan la orientación y la postura de la cabeza independientemente de la forma de la cara del usuario. Se puede acceder a ellos directamente automáticamente, ya que los keypoints de la malla están ordenados.

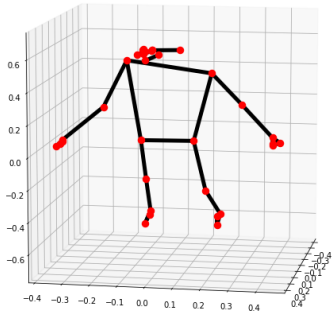
- **Pose cuerpo 3D:** En la imagen 4.10c tenemos la predicción que se ha hecho sobre el cuerpo, con un total de 33 *landmarks*. En este caso, los puntos 3D están normalizados entre el ancho y la altura de la imagen, teniendo de nuevo valores entre 0 y 1. Como vemos, aunque la predicción es bastante buena, no es del todo precisa, por lo que los sistemas que entrenemos tienen que paliar estas imprecisiones.

- **Mano 3D:** En la figura 4.10d vemos la predicción que ha hecho el modelo sobre la mano izquierda, igual que en el caso de la cara, los puntos 3D están normalizados respecto al *bounding box* que detecta la mano. La predicción se ajusta bastante a la posición que vemos en la imagen, excepto en el dedo gordo, que está más cerca de los otros dedos de lo que marca la predicción. En total hay 21 *keypoints* 3D.

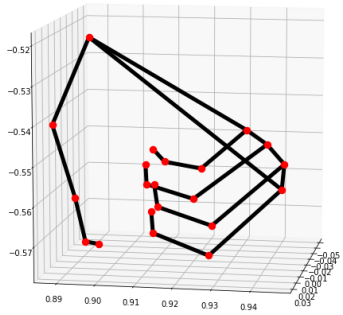
(a) Foto original



(b) Cara 3D



(c) Pose cuerpo 3D



(d) Mano derecha 3D

Figura 4.10: Resultados pose 3D - Mediapipe

Una vez ya hemos visto de forma general el tipo de predicciones que obtenemos con *Mediapipe* nos centraremos en la construcción del *pipeline* de estos modelos. Nos basaremos en el esquema 4.11 para realizar la explicación, el primer paso es detectar si hay alguna persona y donde está, para ello se ejecuta un detector, que devuelve el centro y radio del círculo que mejor se ajusta a la persona. Una vez detectada la persona, lo siguiente es realizar la predicción de la pose del cuerpo 3D, en la que obtenemos 33 puntos 3D. A continuación, se realiza la predicción de la pose de ambas manos y de la cara. En esta parte no es necesario ejecutar la detección de estas partes, ya que el sistema sabe donde está situada cada parte usando los puntos de la predicción del cuerpo. Prescindir de los detectores de la cara y las manos ahorra una gran cantidad de computación al sistema.

La operación de detección del cuerpo es bastante costosa, por lo solamente la vamos a ejecutar **hasta que encuentre una persona**. Una vez haya encontrado la detección se utiliza la información de la predicción de la pose 3D para calcular la posición del siguiente frame, esta opción solamente se puede usar en vídeo debido a que las imágenes están correlacionadas temporalmente. Esta opción limita que el usuario pueda moverse muy rápido, ya que se pierde la posición de la persona y el detector tiene que volver a ejecutarse.

Debido a que Mediapipe es un framework experimental, el proceso de compilación e integración de los modelos se tiene crear dependiendo de nuestras necesidades. Para su compilación debemos construir un grafo en el que vamos indicando los modelos a usar y el flujo de información, aquí también se incluye si queremos ejecutar el detector solo al principio.

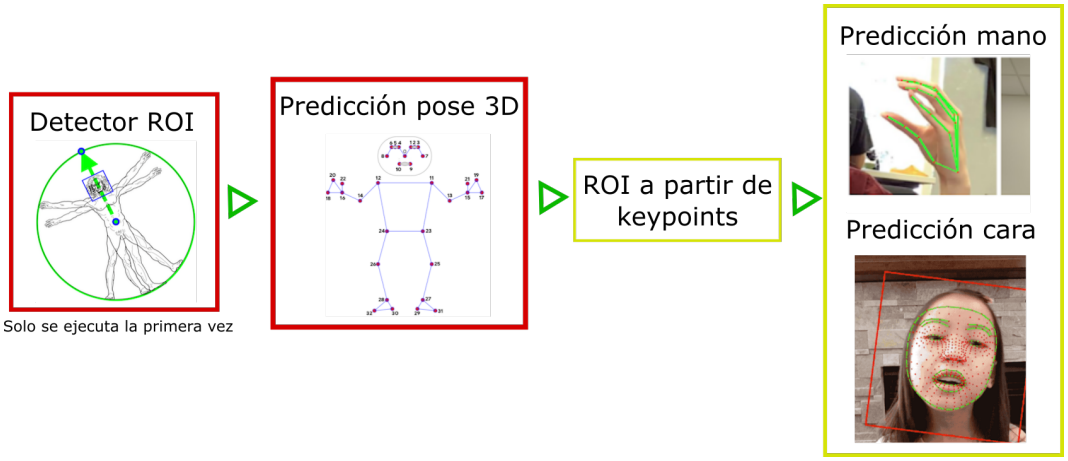
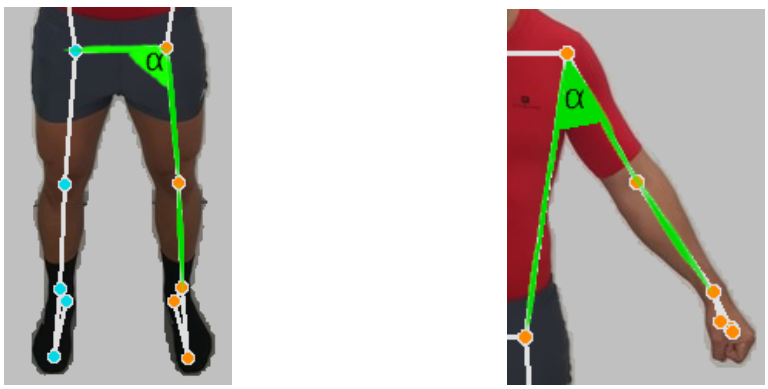


Figura 4.11: PipeLine Mediapipe

4.1.4. Clasificación de puntos 3D

Los puntos 3D que representan la pose del usuario tienen que interpretarse para identificar si la postura que se está realizando es correcta. Aunque la información de los puntos es muy relevante de por sí, hay diversos factores que complican la identificación de la postura. Entre estos factores encontramos la variación en las formas corporales, la distancia a la que la persona puede estar de la imagen o los errores de precisión del sistema de Pose 3D. Debido a esto, hay partes de la postura que se controlarán con modelos basados en redes neuronales y otras partes que se controlarán trabajando directamente con los puntos 3D.

Concretamente, la apertura de los brazos y las piernas se controlarán directamente calculando el ángulo que forman los *keypoints* detectados. Mientras que la postura de ambas manos y la orientación de la cabeza y el cuerpo se clasificarán usando redes neuronales que reciban como entrada puntos 3D.



(a) Ángulo pierna **(b)** Ángulo brazo

Figura 4.12: Ángulo usado para comprobar la apertura de los brazos y las piernas

Brazos y piernas: Comprobar si la apertura de los brazos y piernas son correctas es el problema más *sencillo*, lo abiertas que estén las extremidades tampoco es muy relevante dentro de un mínimo. Debido a que tampoco buscamos un grado de precisión muy alto en este requisito se ha optado por calcular directamente el ángulo que forman los puntos 2D del brazo y de la pierna, en la figura 4.12 hemos marcado en verde el ángulo que se toma como referencia para comprobar la apertura de las extremidades.

Manos y orientación cuerpo-cara: Tanto las manos como la orientación del cuerpo y la cara son requisitos más difíciles de comprobar que los anteriores, en este caso se ha optado por un método diferente, utilizar redes neuronales que reciban como entrada los puntos 3D. Estas redes se entrenarán con los *keypoints* que se obtengan de las imágenes de entrenamiento del conjunto de datos que hemos explicado anteriormente. En la figura 4.13 tenemos un resumen del *pipeline* de la clasificación, primero se obtienen los *keypoints 3D* del cuerpo, y después una red neuronal recibe estos puntos y clasifica la postura como correcta e incorrecta. Habrá dos redes, una para las manos y otra para la orientación cuerpo-cara.

Figura 4.13: PipeLine Clasificación

4.1.5. Entrenamiento red neuronal - Keypoints

En este apartado se detallará como se ha llevado a cabo el entrenamiento de las redes neuronales que reciben como entrada *keypoints* del cuerpo, y que sirven para clasificar posturas. Tenemos 4 problemas de **clasificacion binaria**: mano frontal, mano lateral, orientación frontal y orientación lateral. Cada una de estas posturas se clasificará como correcta o incorrecta. Para el entrenamiento se va a usar el conjunto de datos que hemos explicado anteriormente, en total se van a entrenar 4 redes para la clasificación de *kyepoints*, en el entrenamiento de todas ellas se realiza una partición de los datos de un 60 % para el entrenamiento, un 20 % para validación y un 20 % para test.

Los *keypoints* se obtienen realizando predicciones con Mediapipe sobre las imágenes del conjunto de datos, los puntos 3D obtenidos se anotan en un archivo *CSV* junto con la etiqueta correspondiente. Para el caso de las manos se anotan 21 puntos 3D por cada una, que se corresponde con un vector de entrada de tamaño 63, este vector será la entrada de la red neuronal. En la figura 4.14 tenemos algunas de las predicciones que se han sacado a partir del conjunto de datos.

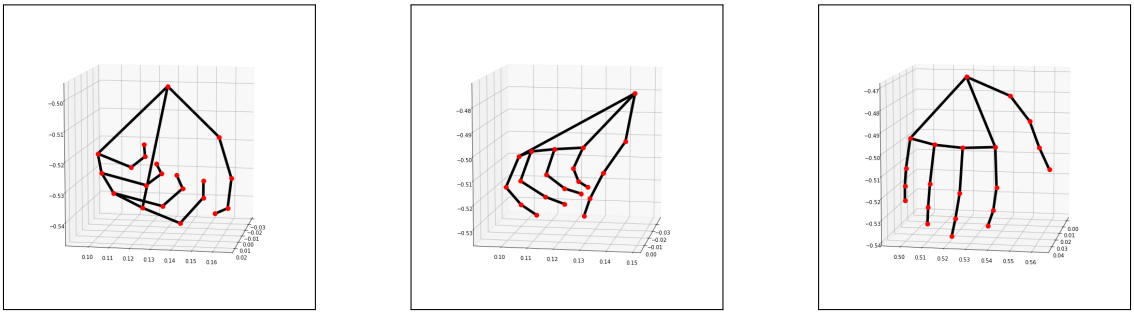
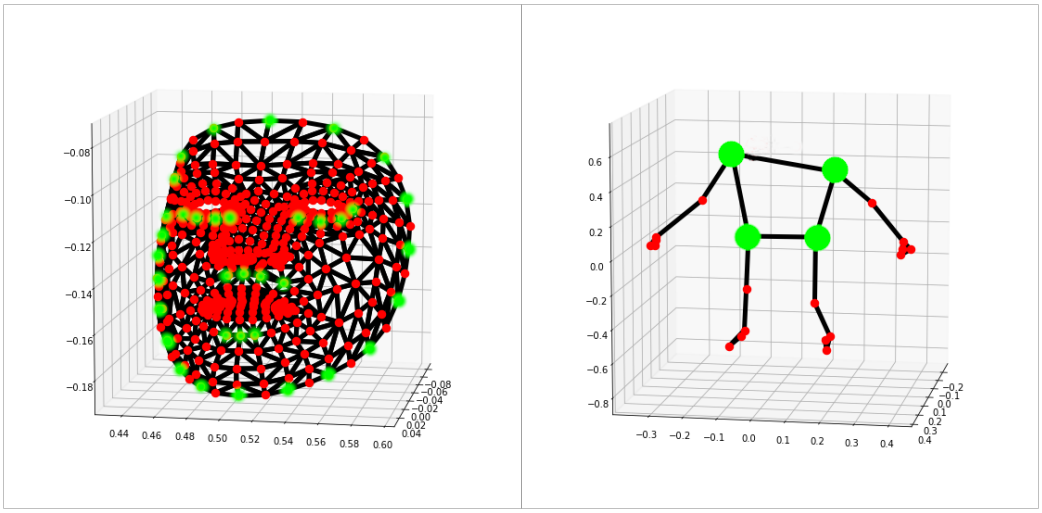


Figura 4.14: Ejemplos de predicción de manos en 3D

En el caso de la orientación es diferente, ya que la red neuronal recibe como entrada puntos 3D pertenecientes a la cabeza y al cuerpo. Como para la cabeza tenemos muchos puntos(468 en total), se ha reducido el número de puntos a tener en cuenta escogiendo únicamente los más significativos, cogiendo entre los puntos de la cabeza y el cuerpo un total de 40. En la figura 4.15a tenemos marcados en verde los puntos 3D que se toman como entrada para clasificar la orientación de la cabeza. Se han escogido los puntos que marcan el contorno de la cabeza, los infraorbitales(debajo de los ojos) y los puntos que bordean la nariz y la boca. Del cuerpo se usan los puntos que corresponden a los hombros y la cadera del usuario, están marcados en verde en la figura 4.15b, con estos puntos y los de la cabeza como entrada se entrenará a la red neuronal para que clasifique si la orientación es correcta o no.



(a) *Keypoints* 3D que se usan de la cabeza **(b)** *Keypoints* 3D que se usan del cuerpo

Figura 4.15: Keypoints usados para la orientación cabeza-cuerpo

Data Augmentation. Debido a la poca cantidad de datos con los que contamos es necesario el uso de técnicas de que puedan paliar este problema, además también existe un desbalanceo entre clases, teniendo más muestras en algunas categorías que en otras. Por ello, se ha optado por aplicar *Data Augmentation* directamente a los datos 3D. Para aplicar esta técnica se debe tener cuidado, ya que la validez de los datos se puede ver afectada, a continuación se va a explicar con detalle:

- **Flip Horizontal:** Igual que si fuera una imagen, aquí también se puede hacer un volteado horizontal de los *keypoints 3D*. En el caso de las manos por ejemplo, es como transformar una mano derecha en una mano izquierda. No se puede hacer un volteado vertical porque entonces los datos ya no serían válidos.

- **Escalar puntos:** Cada cuerpo, cabeza o mano tiene unas proporciones y tamaño diferente. Es importante que nuestro modelo sea capaz de adaptarse a diversas fisionomías, por ello se puede usar como *data augmentation* escalar la posición de los puntos 3D, aumentando o dismuniyendo el tamaño de una parte del cuerpo.
- **Añadir ruido en los puntos:** Otra de las técnicas que se han usado es sumar o restar cada *keypoint* por un número aleatorio pequeño, provocando leves variaciones en los datos. Si el *ruido* que metemos es demasiado alto, los datos dejan de ser válidos al estar *contaminados*.
- **Ocultar aleatoriamente *keypoints*:** Las predicciones que realiza Mediapipe no son siempre correctas, y a veces algunos *keypoints* tienen valores erróneos o sin sentido. El modelo que entrenemos tiene que ser capaz de funcionar incluso aunque la predicción de algunos de estos puntos no sea correcta. Para que sistema sea robusto frente a este problema, se la da algunos a *keypoints* un valor completamente aleatorio. Si hacemos esto con demasiados puntos los datos pierden sentido, por lo que solo se puede hacer con unos pocos. Si tuviéramos un problema con imágenes, lo que estamos haciendo sería equivalente a tapar una parte de la imagen de negro para que el modelo sea capaz de generalizar.

La magnitud del escalado en los puntos, la potencia del ruido y el número de *keypoints* que se ocultan son parámetros cuyo valor se ha asignado de forma totalmente experimental. Por ejemplo, si el ruido es demasiado alto y el acierto en validación baja, se va ajustando el valor de ese parámetro.

Modelo usado. El sistema tiene que ser capaz de ejecutarse en tiempo real en un teléfono móvil, con la obtención de los puntos 3D ya tenemos un coste computacional elevado, por lo tanto el modelo que usemos debe ser muy ligero. La red que se ha usado tiene aproximadamente 53k parámetros, en la figura 4.16 vemos el esquema básico. GN significa *Gaussian Noise*, mientras que BN es *Batch Normalization*. Como vemos, cuenta con tres capas densas progresivamente más pequeña. Por último, tenemos una capa *softmax* para realizar la clasificación.

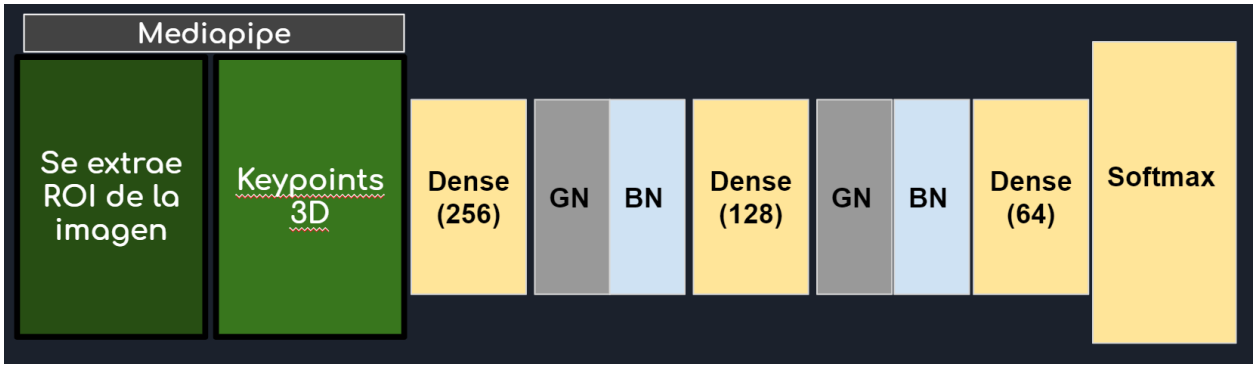


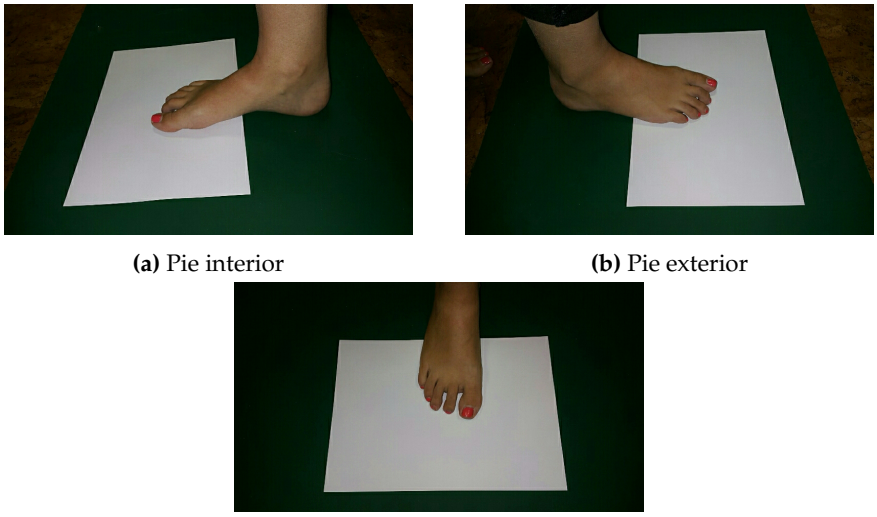
Figura 4.16: Modelo Red Neuronal

4.2 Validación de imágenes del pie

Igual que con el cuerpo, el propósito de realizar fotografías de los pies es conseguir una reconstrucción 3D. En este caso también se tienen que cumplir diferentes requisitos para la que postura sea considerada correcta. En esta sección se va a detallar el proceso de validación de imágenes del pie, incluyendo los requisitos de la postura, las imágenes usadas y la detección de objetos. El enfoque en el caso del pie va a ser muy diferente al cuerpo, ya que no se van a usar mallas de puntos 3d. Esto se debe a que la orientación que tiene que tener el pie es mucho menos restrictiva que el cuerpo, por ello, es suficiente la detección del pie en la imagen con una de las tres posturas(exterior,interior y frontal). Además, tampoco se encontró ninguna librería o modelo con el que sacar la malla 3d del pie y que pudiera funcionar en un teléfono.

4.2.1. Requisitos de la postura del pie

Lo primero que haremos será detallar cuáles son las condiciones y las características que tiene que tener una imagen para que sea considerada correcta. Son necesarias tres fotografías del pie para realizar la reconstrucción, una en la que el pie aparezca de forma frontal, otra en la que se vea la parte interior del pie y otra en la que aparezca la exterior. En todas las fotografías el pie tiene que estar sobre un folio blanco. En la figura 4.17 tenemos un ejemplo de cada tipo de fotografía.



(a) Pie interior **(b)** Pie exterior

(c) Pie frontal

Figura 4.17: Posturas usadas para la reconstrucción 3D del pie

A continuación vamos a especificar cuáles son las condiciones para que una imagen se considere correcta:

- **El folio no aparece cortado:** Es importante que el folio aparezca de forma completa y que no salga cortado en la imagen. En la figura 4.18 vemos la diferencia entre una imagen incorrecta con el folio cortado y otra que no.
- **No aparecen todas las esquinas del folio:** Otro error común es que alguna de las esquinas del folio esté tapada por el propio pie. En la figura 4.19 vemos un ejemplo de este error, ya que el pie oculta la esquina superior derecha del folio.
- **Aparece otro pie:** En cada imagen únicamente puede aparecer un solo pie, si aparecen los dos pies la fotografía no es correcta.

- **La posición del pie no es correcta:** Hay que detectar si la posición del pie no se corresponde con la posición requerida por el sistema. Por ejemplo, el usuario tiene que tomar una foto de la parte exterior del pie pero la hace de la parte interior.
- **No es el mismo pie en todas las fotos:** Es importante que el pie que aparece en las fotos sea siempre el mismo, por lo que habrá que saber distinguir entre pie derecho e izquierdo.
- **Detección de pie y de folio:** Para la recontrucción 3D posterior se debe obtener también donde están localizados tanto el pie como el folio.

(a) Folio cortado **(b)** Folio no cortado

Figura 4.18: Diferencia entre folio cortado y no cortado

Figura 4.19: Pie con la esquina tapada

4.2.2. Conjunto de datos

Igual que en el caso del cuerpo, los requisitos del problema nos han llevado a la utilización de un *dataset* propio que cumpla con las características necesarias para poder entrenar a modelos de detección de objetos. Las imágenes del conjunto de datos contendrán a usuarios realizando la postura con el pie descrita en la sección anterior. Es importante también tener imágenes que contengan folios mal colocados, con el objetivo de que el sistema pueda clasificarlos correctamente. Cada una de las imágenes tiene anotados los *bounding boxes* referentes a los pies y folios presentes en las fotografías, además cada uno de ellos también está etiquetado con la clase correspondiente. A continuación vamos a detallar cada una de las clases presentes en el *dataset*.

En la tabla 4.4 tenemos las 8 clases a detectar junto a su descripción. La clase *FolioNoVisible* podría dividirse en dos clases, diferenciando si el folio está cortado o tapado por el pie, no obstante, como tenemos pocas imágenes de estas categorías se ha optado por juntarlas en una clase. La figura 4.21 contiene 6 imágenes que contienen todas las clases del conjunto de datos, también aparecen los *bounding boxes* anotados junto a su clase. En

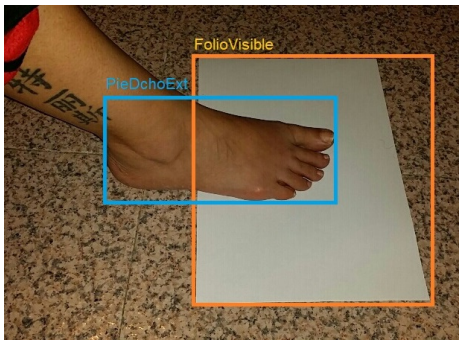
total contamos con 3243 imágenes con unas 6486 anotaciones marcando los *bounding boxes*. En la tabla 4.5 tenemos el número de elementos que tiene cada clase, todas las clases referentes al pie están balanceadas, si nos fijamos, vemos que dentro de las categorías de cada pie tenemos exactamente el mismo número de elementos, esto se debe a que de cada usuario que se hacía foto de un pie se tomaban las 3 fotografías correspondientes. Por lo que respecta al folio, hay un gran desbalanceo entre las clases *FolioVisible* y *FolioNoVisible* como ya se ha comentado anteriormente.

Clases - Imágenes Pie	
Etiqueta	**Descripción**
FolioVisible	Folio bien colocado en los que todas las esquinas son visibles [4.20a, 4.20c, 4.20e, 4.20f]
FolioNoVisible	Folio que contiene algún error, puede ser que aparezca cortado o que alguna de las esquinas esté tapada por el pie [4.20b, 4.20d]
PieIzdoInt	Aparece el pie izquierdo con el interior del pie orientado hacia la cámara(dedo gordo)[4.20d]
PieIzdoExt	Aparece el pie izquierdo con el exterior del pie orientado hacia la cámara(dedo meñique)[4.20e]
PieIzdoFront	Aparece el pie izquierdo orientado de forma frontal hacia la cámara [4.20f]
PieDchoInt	Igual que *PieIzdoInt* pero con el pie derecho [4.20b]
PieDchoExt	Igual que *PieIzdoExt* pero con el pie derecho[4.20a]
PieDchoFront	Igual que *PieIzdoFront* pero con el pie derecho[4.20c]

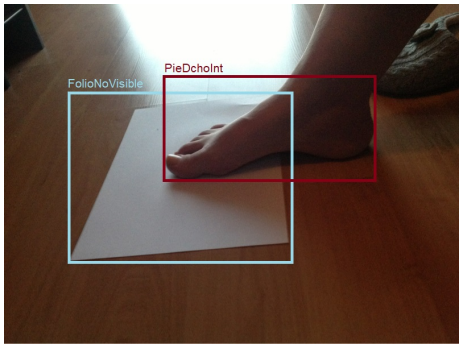
Tabla 4.4: Distribución conjunto de datos - Postura

Distribución clases - Imágenes Pie	
Etiqueta	**Num. Elementos**
FolioVisible	2639
FolioNoVisible	604
PieIzdoInt	571
PieIzdoExt	571
PieIzdoFront	571
PieDchoInt	510
PieDchoExt	510
PieDchoFront	510

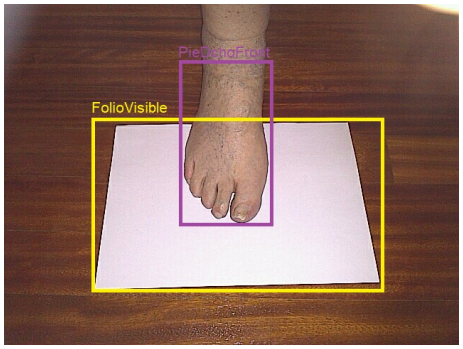
Tabla 4.5: Distribución datos pie



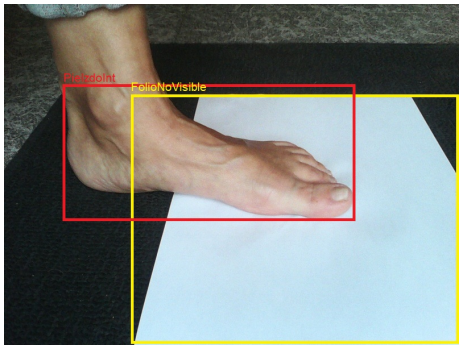
(a) Pie derecho exterior



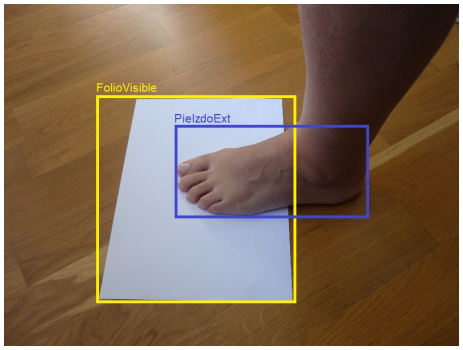
(b) Pie derecho interior



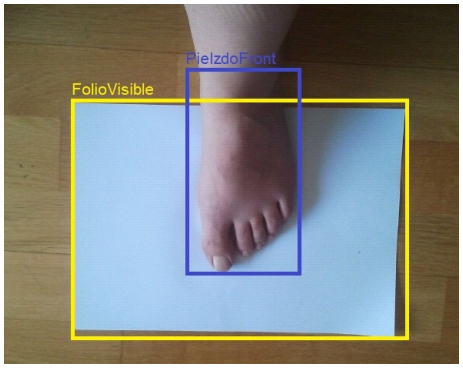
(c) Pie derecho frontal



(d) Pie izquierdo interior



(e) Pie izquierdo exterior



(f) Pie izquierdo frontal

Figura 4.20: Imágenes del conjunto de datos

4.2.3. Detección de objetos

Modelo. La detección de objetos se realizará con la familia de modelos *EfficientDet-Lite*, una arquitectura diseñada exclusivamente para la detección de objetos en dispositivos móviles, las bases de su funcionamiento proceden de la arquitectura EfficientDet[13]. En la tabla 4.6 aparecen las características de los distintos modelos, a medida que aumenta la complejidad del modelo aumenta la precisión media. No obstante, este aumento de precisión tiene costes, ya que tanto el tamaño como el tiempo de latencia también aumenta. Durante la experimentación, se probarán los diferentes modelos para comprobar hasta que punto mejoran la precisión media en nuestro problema.

La *EfficientDet-Lite* no detectará solamente el folio y el pie, sino que directamente detecta las diferente clases del pie y si el folio es visible o no. La idea original era localizar el pie y el folio, y después pasarle la imagen recortada a una CNN para clasificar la postura del pie o si el folio era correcto. Usando únicamente el modelo de detección los resultados son muy positivos, por lo que no se ha llegado a probar el otro método.

Modelos detección objetos			
Arquitectura modelo	**Tamaño (MB)**	**Latencia (ms)***	**Precisión media****
EfficientDet-Lite0	4.4	37	25.9 %
EfficientDet-Lite1	5.8	49	30.55 %

Tabla 4.6: Arquitecturas detección de modelos

**Latencia medida en Pixel 4 con 4 subprocesos en la CPU*

***La precisión media es el mAP (precisión media media) del conjunto de datos de validación de COCO 2017.*

Data Augmentation. Debido a la falta de datos y el desbalanceo presente en algunas clases es necesario implementar alguna técnica que nos permita aumentar la cantidad de datos disponibles. Como estamos trabajando con *bounding boxes*, cualquier rotación, escalado o desplazamiento que cambiemos en la imagen tendremos que hacerlo también en el *bounding box* correspondiente. Para realizar el aumentado de datos se ha usado la librería *imgaug* [4]. A continuación vamos a detallar las diferentes formas de *data augmentation* aplicadas:

- **Flip horizontal:** Voltear la imagen horizontal es una de las técnicas más clásicas para aumentar datos. En este caso tenemos que tener especial cuidado, ya que hay que voltear también la posición del *bounding box* y cambiar la etiqueta del pie, si en la imagen original el pie es el derecho habrá cambiarlo al izquierdo, y viceversa.

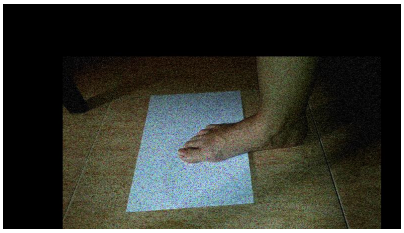
- **Brillo, contraste y ruido gaussiano:** Otro de los métodos que se usan para aumentar los datos consiste en variar de forma aleatoria los valores de brillo y contraste de la imagen, en 4.21a tenemos un ejemplo en el que se ha aumentado el brillo. Otra posible técnica es puede añadir ruido gaussiano a la imagen, en 4.21c y 4.21d tenemos dos ejemplos. Con este tipo de cambios no hace falta cambiar la posición del *bounding box*.

[4] https://imgaug.readthedocs.io/en/latest/

- **Escalado y rotación:** Las últimas técnicas que se han usado para el aumentado de datos son escalar y rotar la imagen, en este caso tenemos que recolocar el *bounding box* a su posición correspondiente. En 4.21b, 4.21c y 4.21d tenemos 3 ejemplos.

(a) Data augmentation(1)

(b) Data augmentation(2)



(c) Data augmentation(3)

(d) Data augmentation(4)

Figura 4.21: Ejemplos *data augmentation*

No se aplican siempre **todas** las transformaciones descritas a la imagen. Lo que se hace es darle una cierta probabilidad a cada técnica, por lo que la combinación de transformaciones que se aplican a la imagen es aleatoria. Además, el orden en el que se aplican también es completamente aleatorio, de este modo la capacidad de generar nuevas imágenes es mucho es más rica.

Entrenamiento. El conjunto de datos de fotografías de pies se ha divido en un 60 % para entrenamiento, 20 % para validación y 20 % para test. Se han probado dos formas de entrenar al modelo, utilizando un modelo pre entrenado con 90 clases y realizando un *fine-tunning* y entrenándolo desde 0. Durante las pruebas se ha probado con los modelos descritos anteriormente. El modelo de TensorFlow usa supresión no máxima (NMS) por clase para el postprocesamiento, no obstante, el modelo que se usará en móviles se tiene que pasar TFlite. Esta versión del modelo usa NMS global que es mucho más rápido pero menos preciso. Con TensorFlow generamos un máximo de 100 detecciones mientras que con tflite se generan un máximo de 25 detecciones, esto no afecta a nuestro problema, ya que no tendremos tantas detecciones simultáneas.

4.3 Implementación de la aplicación

En esta sección vamos a hablar de los detalles más relevantes de la implementación de la aplicación, especialmente la parte que tiene ver que con la incorporación de los sistemas de *machine learning*. En la *app* tenemos dos opciones, o entrar en la parte de la postura del cuerpo o del pie, en la figura 4.22 tenemos la vista del menú donde escogemos estas opciones.

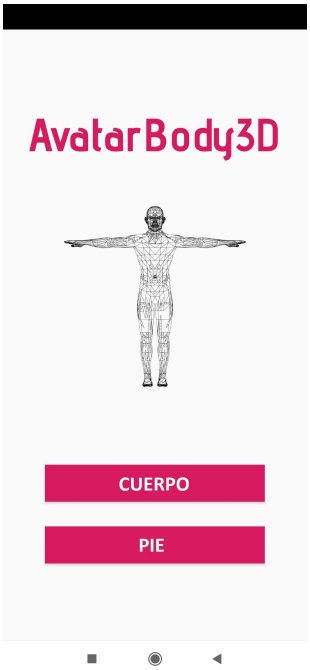


Figura 4.22: Menú principal app

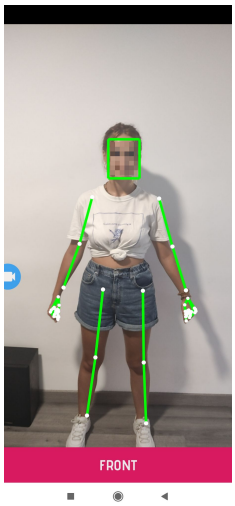
En la parte del **cuerpo** podemos alternar entre la vista frontal y la lateral, donde en cada una se ejecutarán los modelos pertinentes de cada postura, a continuación vamos a explicar los modelos que se ejecutan en cada postura:

- **Postura frontal:** Se ejecuta el modelo de NN[5] de las manos dos veces(una por cada mano) y el modelo de NN para orientación frontal. Para los brazos y las piernas, se obtiene el ángulo tal y como se ha explicado anteriormente a partir de los puntos de Mediapipe.

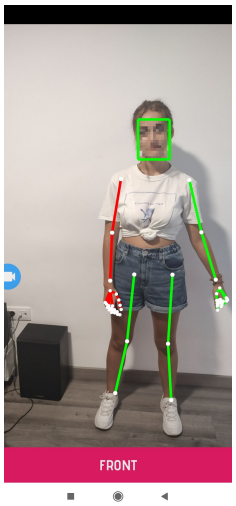
- **Postura lateral:** Se escoge la mano más cercana a la cámara(usando la coordenada z de las manos), y se infiere la predicción de esa mano con el modelo NN para la mano lateral. También se ejecuta el modelo de la orientación lateral. En esta postura únicamente nos fijaremos en el ángulo de un brazo(el más cercano) para ver que está recto.

[5]Neural Network

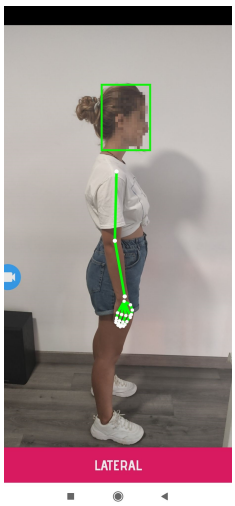
Una vez sabemos que parte de la postura es correcta o incorrecta es importante ver como se va a **representar** esta información. La idea básica es 'pintar' cada parte del cuerpo en verde o rojo dependiendo si la postura está bien hecha. En la figura 4.23 vemos como se representa en la *app*, la parte de la orientación cabeza-cuerpo se representa como un cuadro en la cabeza indicando si es correcta o no. En la imagen 4.23a la postura frontal es correcta, por lo que todo está marcado en verde. En 4.23b en cambio, tanto la mano como el brazo derecho están posicionados de forma incorrecta. En 4.23c, 4.23d también tenemos ejemplos similares pero con la vista lateral.



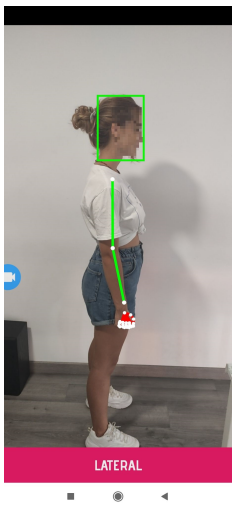
(a) Vista app(1)



(b) Vista app(2)



(c) Vista app(3)



(d) Vista app(4)

Figura 4.23: Aplicación - Parte cuerpo

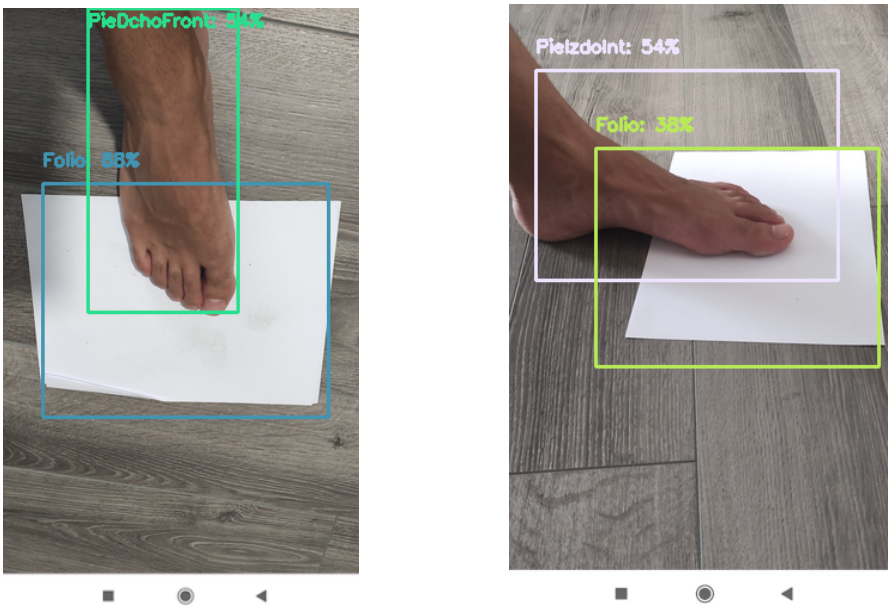
Otro detalle importante es saber el coste computacional que tiene este sistema en un dispositivo móvil. Los móviles que hemos probado han sido POCO X3 y el SAMSUNG S9, el objetivo principal del proyecto es que sea capaz de ejecutarse en tiempo real, o con una tasa cercana que permita al usuario corregir su postura. En la tabla 4.7 tenemos la tasa conseguida, como vemos, son datos muy buenos teniendo en cuenta la capacidad de computación de los móviles y la *complejidad* de nuestro sistema. En un dispositivo con menos capacidad o sin GPU esta tasa de frames descenderá notablemente.

Frames por segundo - Dispositivos móviles	
Dispostivo	**Frames por segundo**
POCO X3	24
SAMSUNG S9	20

Tabla 4.7: Frames por segundo en móviles

A continuación vamos a continuar con la parte del **pie**. En esta sección de la app lo que hacemos se realiza la inferencia del detector y se dibujan en pantalla los *bounding boxes* junto por su correspondiente clase. En la figura 4.24 tenemos dos capturas de la aplicación con el modelo realizando la predicción, en 4.24a tenemos un pie derecho frontal junto con su folio y en 4.24b aparece la parte interior de un pie izquierdo.

En la parte del pie únicamente se ejecuta el detector, pero realmente para cubrir los requisitos descritos haría falta que hubiera tres *modos*, uno por cada fotografía del pie. En cada uno de estos *modos* la postura se marcaría como correcta si se detecta un único pie con la clase correspondiente y también se detecta un folio marcado como visible. Como la parte que más nos interesa del trabajo es el desarrollo de los sistemas de aprendizaje automático, no se ha considerado demasiado relevante acabar esta parte de la app.



(a) App pie (1) **(b)** App pie (2)

Figura 4.24: Inferencia del detector en al app

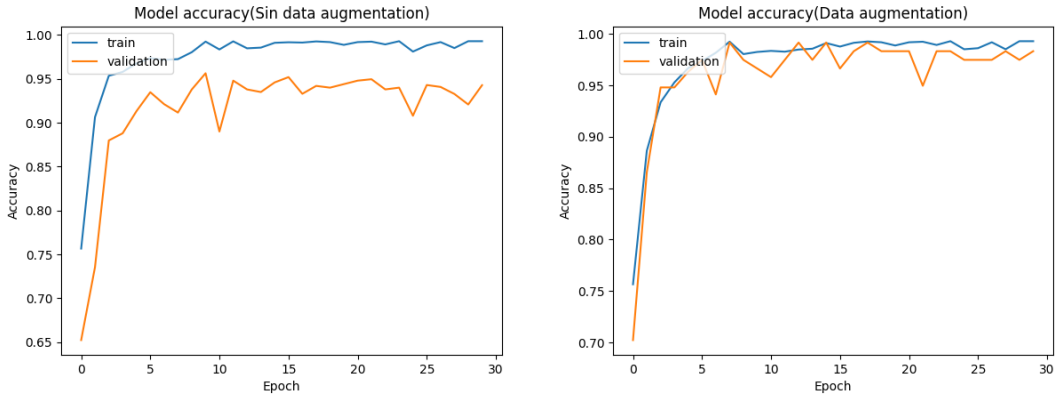
CAPÍTULO 5
Experimentación y resultados

En este capítulo se mostrarán los resultados de todos los experimentos hechos durante el desarrollo del proyecto. En cada experimento se explicarán los parámetros usados, también se ilustrarán los resultados con gráficos y tablas para que su visualización sea lo más gráfica posible. Igual que en el desarrollo, la experimentación se dividirá en dos secciones principales: cuerpo y pie.

5.1 Experimentación - cuerpo

Para la red encargada de clasificar posturas humanas hemos realizado diferentes experimentos por cada parte del cuerpo. La mayoría de experimentos se han hecho con la red descrita en la sección de desarrollo, también se ha probado con alguna red con más parámetros, pero la mejora no ha sido muy significativa. En cada parte del cuerpo se ha experimentado con *Data Augmentation* y sin él, para ver si mejora de alguna forma el acierto.

Mano frontal. En la figura 5.1 tenemos los resultados del entrenamiento para esta postura, en las gráficas aparece el acierto sobre el conjunto de entrenamiento y el de validación. El acierto conseguido sin aumentado de datos es del **95 %**, mientras que si aplicamos esta técnica se alcanza el **98.2 %** en validación. Como vemos en las gráficas, el acierto en entrenamiento es muy similar en ambos casos, pero con aumentado de datos el modelo es capaz de generalizar mejor.

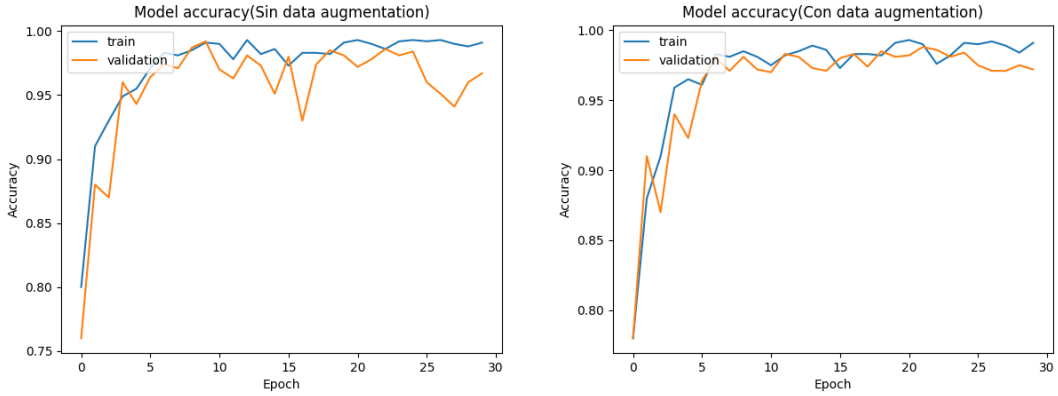


(a) Mano frontal sin data augmentation

(b) Mano frontal con data augmentation

Figura 5.1: Acierto red - Mano frontal

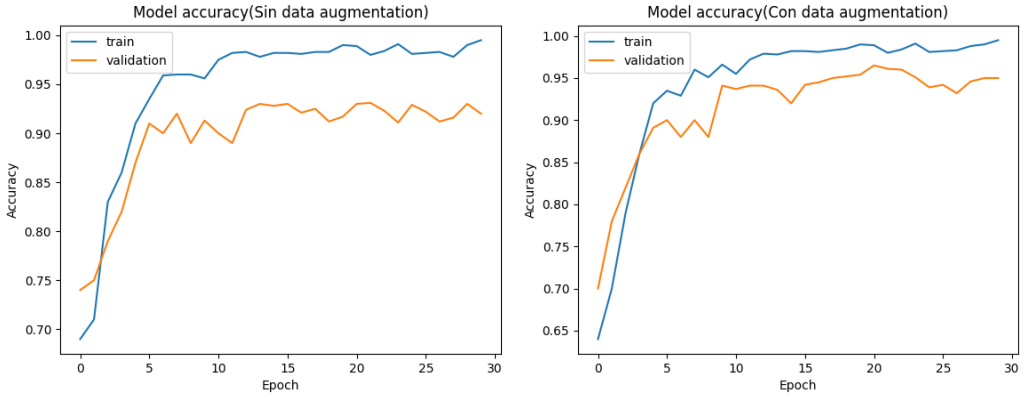
Mano lateral. La figura 5.2 contiene los resultados correspondientes al entrenamiento en la postura de la mano lateral, esta postura en principio es más sencilla que la anterior porque todos los dedos de la mano aparecen y no hay ninguno tapado. En este caso el *data augmentation* no mejora tanto como en el caso anterior, porque sin él ya se consigue un buen resultado. Con esta técnica tenemos un **98.5 %** en validación y sin ella un **98.2 %**, prácticamente el mismo resultado.



(a) Mano lateral sin data augmentation **(b)** Mano lateral con data augmentation

Figura 5.2: Acierto red - Mano lateral

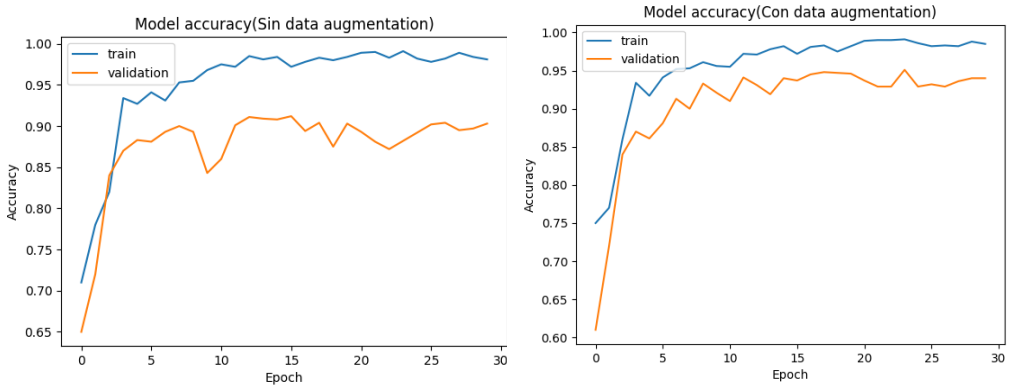
Orientación frontal. El acierto conseguido en la orientación frontal es un poco menor que en el caso de las manos, también en parte debido a que algunos usuarios están 'mal' orientados únicamente por pequeños detalles. El aumentado de datos sirve para conseguir una mejora y pasar del **92.7 %** al **95.3 %** en validación.



(a) Orientación frontal sin data augmentation **(b)** Orientación frontal con data augmentation

Figura 5.3: Acierto red - Orientación frontal cabeza-cuerpo

Orientación lateral. La orientación lateral es la postura más díficil de clasificar de todas, ya que como únicamente se ve una parte del cuerpo y la predicción de los *keypoints* tiene bastantes errores. Aún obteniendo los peores resultados en todas las posturas, se consigue un **91 %** sin **data augmentation** y un **94.5 %** con esta técnica en validación. En la figura 5.4 tenemos las dos gráficas donde aparecen los resultados.



(a) Orientación lateral sin data augmentation **(b)** Orientación lateral con data augmentation

Figura 5.4: Acierto red - Orientación lateral cabeza-cuerpo

La tabla 5.1 contiene los resultados en test conseguidos en las diferentes posturas, en todas se ha alcanzado al menos un 95 % en este conjunto, en estos experimentos se ha usado *data augmentation*.

Acierto en test	
Postura	**Acierto(%)**
Mano frontal	98.2
Mano lateral	98.8
Orientación frontal	95.7
Orientación lateral	95.1

Tabla 5.1: Resultados acierto en test - Postura corporal

5.2 Experimentación - pie

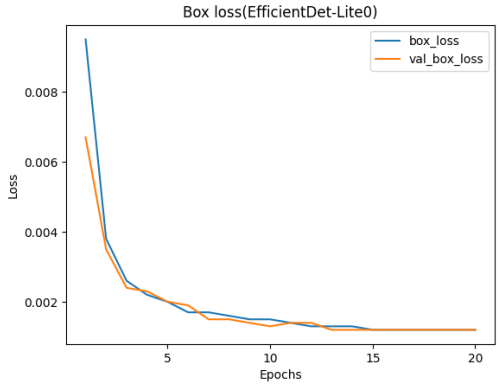
En la experimentación del pie se mostrarán los resultados obtenidos en los diferentes modelos de detección entrenados, para cada experimento realizado se detallarán los parámetros usados. Las principales diferencias en cada experimento son las arquitecturas usadas, el uso de *data augmentation* y el *fine-tunning*. El box loss indica lo bueno que es el modelo en ajustar el *bounding box* al objeto, mientras que el class loss muestra lo bien que diferencia entre las clases. El detection loss es el loss 'final' que se obtiene combinando el class loss y el box loss, en esta combinación se le da más peso al loss que peores resultados tiene(por eso en las gráficas veremos que va muy parejo al class loss).

EfficientDet-Lite0 sin aumentado de datos. El primer experimento que se ha hecho ha sido el más básico, se ha usado el modelo más simple de todos y el más rápido sin ningún tipo de aumentado de datos. El modelo se ha entrenado de 0. En la figura 5.5 tenemos las gráficas que muestran la evolución del entrenamiento, comparando los resultados de los datos de entrenamiento con los de validación. La *Class Loss* nos indica lo bueno que es el modelo en identificar la clase apropiada, el *Box Loss* indica el error en la predicción del *bounding box* del objeto y el *Detection loss* muestra lo bueno que es el modelo detectando objetos. Tanto el *class loss* como la *detection loss* bajan tanto en entrenamiento como en validación hasta la época 12, donde el error empieza aumentar y el modelo deja de mejorar. En cambio la *box loss* va mejorando hasta la última época, y los resultados en entrenamiento y validación van muy parejos.

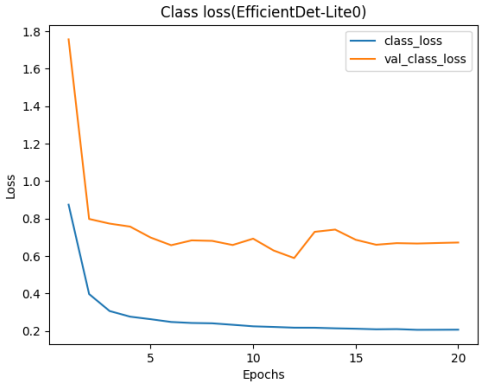
Average Precision	
mAP	0.389
AP/Folio	0.399
AP/FolioNoVisible	0.324
AP/PieDchoFront	0.509
AP/PieDchoInt	0.269
AP/PieDchoExt	0.372
AP/PieIzdoInt	0.487
AP/PieIzdoFront	0.472
AP/PieIzdoExt	0.282

Tabla 5.2: AV en EfficientDet-Lite0 sin aumentado de datos

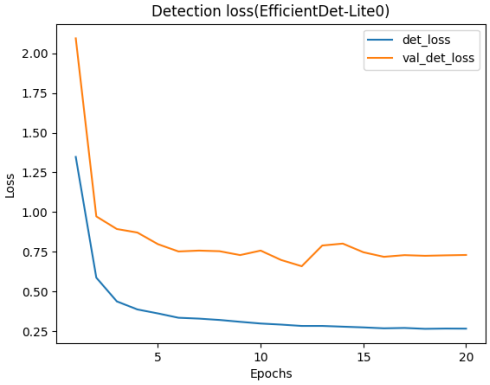
En la tabla 5.2 tenemos el *average precision* total y por clase obtenido sobre el conjunto de validación. Las clases que peor AP presentan son *PieDchoInt* con 0.269 y *PieIzdoExt* con 0.282. El mAP es de 0.389, teniendo en cuenta que no estamos utilizando *data augmentation* ni el modelo más avanzado los resultados son ya bastante decentes, aunque aún es un mAP bajo.



(a) Box loss



(b) Class loss



(c) Detection loss

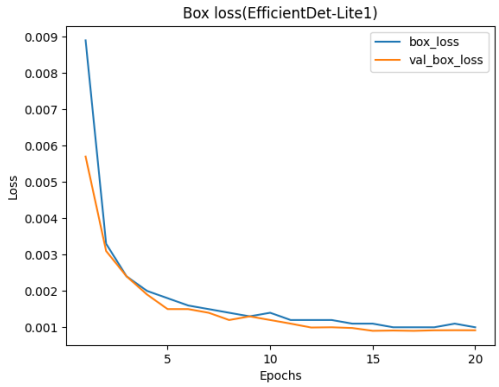
Figura 5.5: Resultados EfficientDet-Lite0 sin aumentado de datos

EfficientDet-Lite1 sin aumentado de datos. En este caso vamos a utilizar el mismo esquema que en el caso anterior pero con un modelo más avanzado. En la tabla 5.3 tenemos el *average precision* conseguido, mejora ligeramente los resultados de la versión 0 del modelo. Prácticamente el AP de todas las clases aumenta en esta versión.

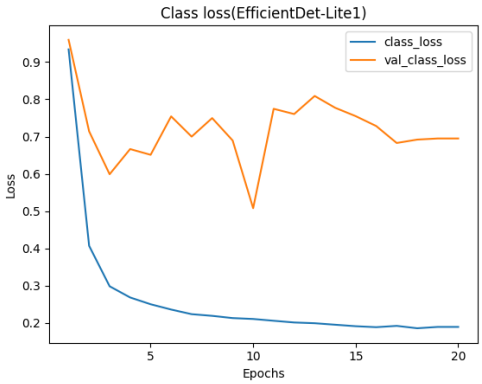
Average Precision	
mAP	0.401
AP/Folio	0.412
AP/FolioNoVisible	0.331
AP/PieDchoFront	0.494
AP/PieDchoInt	0.296
AP/PieDchoExt	0.411
AP/PieIzdoInt	0.491
AP/PieIzdoFront	0.441
AP/PieIzdoExt	0.327

Tabla 5.3: AV en EfficientDet-Lite1 sin aumentado de datos

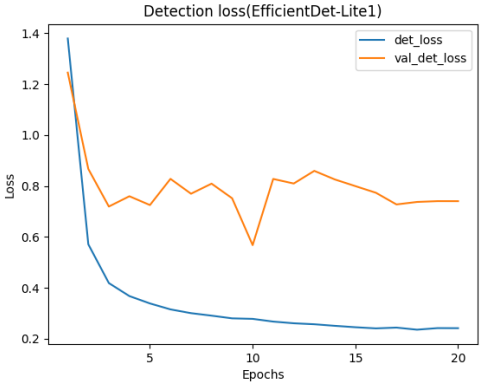
En la figura 5.6 tenemos las gráficas pertinentes del modelo EfficientDet-Lite1 sin aumentado de datos, la forma de las gráficas es muy similar a la anterior, aunque aquí, obtenemos en la época 11 un modelo con un loss muy bajo tanto en la clase como en la detección y después empeora de forma muy significativa.



(a) Box loss



(b) Class loss



(c) Detection loss

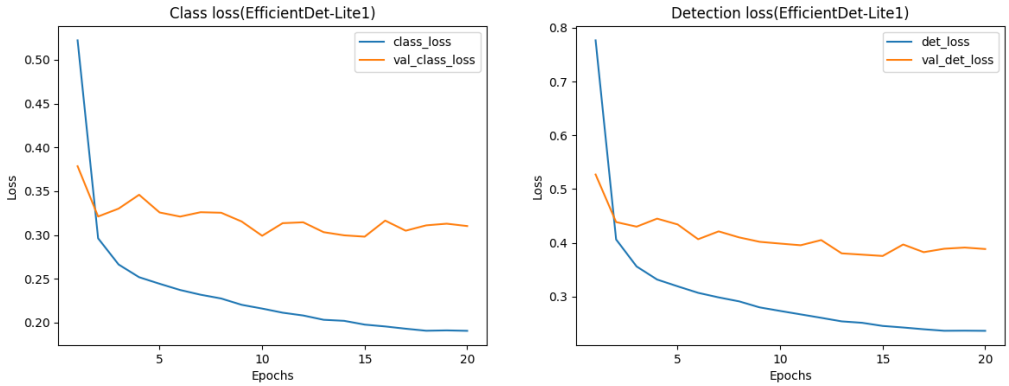
Figura 5.6: Resultados EfficientDet-Lite1 sin aumentado de datos

EfficientDet-Lite1 con aumentado de datos. Como la versión 1 del modelo nos ha dado mejor resultado, vamos a continuar la experimentación con esta arquitectura. En la figura 5.8 tenemos el *detection loss* y la *class loss*, no aparece la gráfica de la *box loss* porque es el menos significativo en este caso. Vemos en la gráfica 5.7a que el error en validación en la *class loss* es bastante más estable que sin aumentado de datos, ademas el valor del *loss* es más bajo que en los experimentos anteriores, porque lo que ya a primera vista se aprecia una mejora notable.

En la tabla 5.4 tenemos el *Average Precision* de todas las clases igual que en los experimentos anteriores. Lo primero que observamos es una mejora muy importante en el mAP, que pasa de 0.401 a 0.537 usando *data augmentation*, si nos paramos a mirar clase a clase observamos que la mejora más importante ha sido con el folio, que ha pasado a un AP de 0.913 y un AP de 0.816 para los folios no visibles. De forma general, también mejora en las clases de los pies, aunque es cierto que algunos de ellos empeora ligeramente, como en el caso de la clase *PieIzdoFront*.

Average Precision	
mAP	0.537
AP/Folio	0.913
AP/FolioNoVisible	0.816
AP/PieDchoFront	0.441
AP/PieDchoInt	0.361
AP/PieDchoExt	0.386
AP/PieIzdoInt	0.583
AP/PieIzdoFront	0.403
AP/PieIzdoExt	0.391

Tabla 5.4: AV en EfficientDet-Lite1 con aumentado de datos



(a) Class loss **(b)** Detection loss

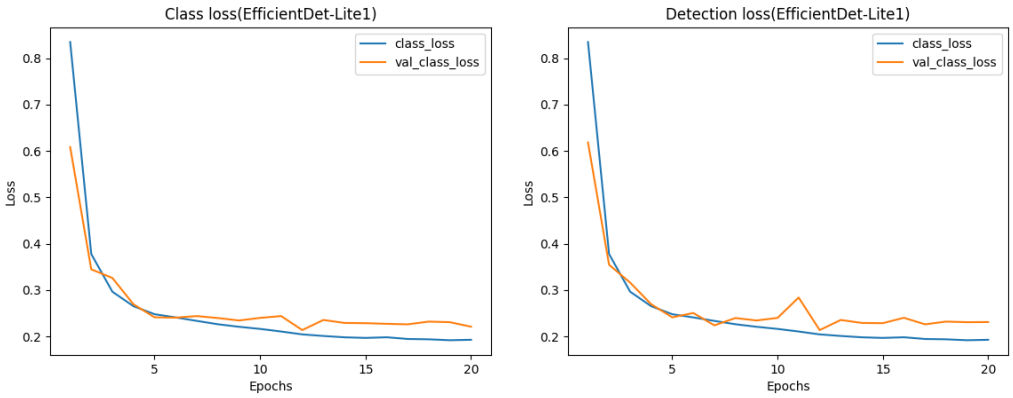
Figura 5.7: Resultados EfficientDet-Lite1 con aumentado de datos

EfficientDet-Lite1 con aumentado de datos y fine-tunning. El fine-tunning se ha hecho congelando las capas etiquetadas con el prefijo *efficientnet*(que son las primeras del modelo) y dejando las demás capas libres para entrenar. Se ha usado el modelo EfficientDet-Lite1 con aumentado de datos. En la tabla 5.5 tenemos el AP por cada clase obtenido en el conjunto de validación, como vemos el *fine tunning* mejora notablemente los resultados, consiguiendo un mAP de 0.638, que es un resultado muy positivo teniendo en cuenta el modelo que estamos usando y la poca cantidad de datos disponibles. Prácticamente todas las clases del pie aumenta su AP.

En la figura 5.8b tenemos las gráficas correspondientes al entrenamiento del modelo, es especialmente significativa la gráfica 5.8a con el *loss* de la clase, se puede observar que en este caso el *loss* de la validación está muy cercano al de entrenamiento, lo que significa que el modelo aprende y generaliza, ya que su acierto en datos que no ha visto nunca es mayor.

Average Precision	
mAP	0.638
AP/Folio	0.910
AP/FolioNoVisible	0.817
AP/PieDchoFront	0.672
AP/PieDchoInt	0.525
AP/PieDchoExt	0.432
AP/PieIzdoInt	0.509
AP/PieIzdoFront	0.633
AP/PieIzdoExt	0.623

Tabla 5.5: AV en EfficientDet-Lite1 con aumentado de datos y fine-tunning



(a) Class loss **(b)** Detection loss

Figura 5.8: Resultados EfficientDet-Lite1 con aumentado de datos y fine-tunning

En 5.6 tenemos una breve tabla con el mAP conseguido en test:

mAP	
Det0-noAug	0.393
Det1-noAug	0.423
Det1-Aug	0.524
Det1-Aug-FN	0.652

Tabla 5.6: Comparación final modelos detección

Las diferentes mejoras, como cambiar la arquitectura del modelo a uno más avanzado, el uso de aumentado de datos y el fine tunning van mejorando paulatinamente la precisión del modelo. **El máximo mAP** se consigue con el modelo que usa una EfficientDet-Lite1 con aumentado de datos y fine-tunning.

CAPÍTULO 6
Conclusiones

Como conclusión de este TFM, se va a hacer un repaso punto por punto de todo lo conseguido. Además, también se incluyen algunas reflexiones sobre lo aprendido a lo largo del proyecto.

Estos son los resultado obtenidos expuestos de forma sintética:

- Se ha conseguido crear una base de datos con diferentes posturas humanas y sus correspondientes etiquetas, este banco de de datos se ha demostrado útil para el entrenamiento de sistemas de *deep learning*.
- Implementación de un modelo capaz de inferir la pose 3D a partir del framework *Mediapipe*. Además, este modelo se ha ajustado para que puede ejecutarse en tiempo real en dispositivos móviles. Tiene algunas limitaciones, como que el usuario no puede moverse muy rápido.
- Se ha entrenado una red neuronal capaz de recibir *keypoints 3D* e identificar si la postura adoptada por el usuario se ajusta a la postura objetivo. Para realizar este entrenamiento se ha usado la base de datos con imágenes de usuario realizando las posturas de forma correcta e incorrecta. El modelo de la red neuronal elegido es ligero para ejecutarlo junto a la predicción de la pose 3D.
- Se ha implementado y entrenado un detector de objetos capaz de detectar y diferenciar diferentes posturas de pies. También es capaz de detectar folios y si estos están bien colocados. Se ha entrenado con *bounding boxes* anotadas en una base de datos propia.
- Desarrollo de una aplicación en dispositivos móviles que es capaz de integrar los sistemas de *machine learning* explicados en la *app*.

Uno de los puntos del proyecto a mejorar es la cantidad de datos, ya que el número de imágenes se queda bastante ajustado para entrenar a las redes, con un mayor número de elementos el resultado sería aún mejor. También haría falta acabar de pulir la aplicación, ya que aunque los sistemas de *deep learning* están integrados aún habría que acabar la parte de la *app* de cara a un posible usuario. A pesar de estos puntos, considero que el resultado del trabajo es muy satisfactorio, ya que se ha conseguido solucionar el problema de identificación de posturas con datos propios e integrarlos en una *app* para dispositivos móviles.

Bibliografía

[1] J. Wang, S. Tan, X. Zhen, S. Xu, F. Zheng, Z. He and L. Shao. *"Deep 3D human pose estimation: A review"*, *Computer Vision and Image Understanding, Volume 210*, 2021.

[2] X. Sun, B. Xiao, F. Wei, S. Liang and Y. Wei *'Integral Human Pose Regression'*. November 2017.

[3] G. Moon, J. Chang, K. Lee *'Camera Distance-aware Top-down Approach for 3D Multi-person Pose Estimation from a Single RGB Image'*. ICCV 2019

[4] V. Bazarevsky, I. Grishchenk, K. Raveendran,T. Zhu, F. Zhang and M. Grundmann. *"BlazePose: On-device Real-time Body Pose tracking"*. June 2020.

[5] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C. Chang and M. Grundmann. *'MediaPipe Hands: On-device Real-time Hand Tracking'*. CVPR Workshop on Computer Vision for Augmented and Virtual Reality, 2020

[6] Z. Zhao, P. Zheng, S. Xu and X. Wu *'Object Detection with Deep Learning: A Review'*. IEEE Transactions on Neural Networks and Learning Systems PP(99): 1-21, January 2019

[7] R. Girshick, J. Donahue, T. Darrell and J. Malik *'Rich feature hierarchies for accurate object detection and semantic segmentation'*. CVPR 2014.

[8] S. Ren, K. He, R. Girshick and J. Sun *'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks'*. NeurIPS 2015.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu and A. Berg *'SSD: Single Shot MultiBox Detector'*. ECCV 2016.

[10] J. Redmon, S. Divvala, R. Girshick and A. Farhadi *'You Only Look Once: Unified, Real-Time Object Detection'*. 2015.

[11] M. Shafiee, B. Chywl, F. Li, A. Wong. *'Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video'*. 2017.

[12] J. Redmon and A. Farhadi. *'YOLO9000: Better, Faster, Stronger'*. University of Washington, Allen Institute for AI, 2017

[13] M. Tan, R. Pang, Q. V. Le. *'EfficientDet: Scalable and Efficient Object Detection'*. CVPR 2020.