UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

GRADO EN INGENIERÍA DEL SOFTWARE

TRABAJO DE FIN DE GRADO

Curso 2020 - 2021

Técnicas inteligentes con imágenes para dispositivos móviles

Intelligent imaging techniques for mobile devices

Autor
Yhondri Josué Acosta Novas

Director
Gonzalo Pajares Martinsanz

Resumen

Los dispositivos móviles inteligentes tales como *smartphones* o *tablets* han evolucionado notablemente tanto en su forma física como en sus funcionalidades y capacidades de procesamiento. Esta evolución ha permitido que las personas puedan realizar más y más tareas con un dispositivo que cabe en la palma de sus manos y que antes sólo se podía llevar a cabo con dispositivos de gran tamaño, como por ejemplo un ordenador personal, o una videoconsola.

En el presente trabajo se propone el diseño de una aplicación para dispositivos móviles iOS que permite el reconocimiento de cinco actividades físicas realizadas por personas que portan el dispositivo.

Para ello, se propone un modelo de Red Neuronal Residual (ResNet) dentro del Aprendizaje Profundo. La red recibe como entradas secuencias de imágenes o vídeos, captados con el dispositivo móvil con los que se realiza el proceso de entrenamiento y decisión. Con tal propósito, se ha diseñado una estrategia que integra el modelo de red indicado, bajo la cobertura de una aplicación informática, para realizar el procesamiento a nivel local, lo que representa un avance importante desde el punto de vista de la privacidad de los datos, a la vez que se mantiene la efectividad de procesamiento. Los resultados obtenidos respecto de la toma de decisiones corroboran esta circunstancia.

Palabras claves

Inteligencia artificial

Red Neuronal Residual (ResNet)

Aprendizaje profundo

iOS

Desarrollo aplicación móvil

Abstract

Smartphones or tablets have notably evolved in their physical form, in their functionalities and processing capabilities. This evolution has allowed people to perform more and more tasks on a device that fits in the palm of their hands that previously could only be done on large devices such as a personal computer or a videogame console.

This work, propose the design of an application for iOS mobile devices that allows the recognition of five physical activities carried out by people who carry the device.

For this, a Residual Neural Network (ResNet) model is proposed within Deep Learning. The network receives as inputs sequences of images or videos, captured with the mobile device with which the training and decision process is carried out. For this purpose, a strategy has been designed that integrates the indicated network model, under the cover of a computer application, to perform the processing at the local level, which represents an important advance from the point of view of data privacy. , while maintaining processing effectiveness. The results obtained regarding decision-making corroborate this circumstance.

Keywords

Artificial intelligence

Residual Neural Network

Deep learning

iOS

Mobile application development

Índice general

Capítulo 1

Introducción

1.1 Antecedentes

Los avances tecnológicos desde la aparición de los primeros teléfonos inteligentes o *smartphones* han aumentado considerablemente en las últimas dos décadas, permitiendo crear dispositivos cada vez más pequeños con grandes capacidades de proceso que antes requerían el uso de sistemas de gran tamaño.

Un ejemplo de ello son los avances que se están producción en el campo de la Inteligencia Artificial. En el pasado, compañías de videojuegos como Microsoft a través de su rama Xbox y Sony con PlayStation, utilizaron junto a sus consolas dispositivos con cámaras externas para poder detectar las acciones que realizaban los jugadores, permitiendo desarrollar juegos de realidad aumentada con las que el jugador podía interactuar mediante gestos y movimientos con los elementos que aparecían en la pantalla. Estos dispositivos requerían necesariamente estar conectados físicamente a las consolas.

Apple y Google, compañías que desarrollan los dos sistemas operativos predominantes en los teléfonos inteligentes, han desarrollado sus propios sistemas de Inteligencia Artificial. Estos sistemas han permitido el desarrollo de aplicaciones que integran tecnologías de realidad aumentada, detección de objetos y detección de movimiento, entre otras, en dispositivos móviles que caben en la palma de la mano. Además, el desarrollo tecnológico también ha permitido y facilitado las conexiones inalámbricas de cualquier tipo,

incluyendo conexiones *Bluetooh*, *Wi-Fi*, 4G (en el futuro 5G) u otras. Todo ello, junto con la incorporación de técnicas inteligentes ha incrementado considerablemente su amplia gama de utilidades y posibilidades, además de su potencialidad.

Por otra parte, la última conferencia de desarrolladores de Apple, *World Wide Developers Conference* del 2020 (WWDC20) [1], celebrada en Junio de 2020 ha supuesto un punto de partida de enorme interés como elemento motivador del presente trabajo, ya que ha establecido un ámbito de apoyo y soporte desde el punto de vista de los desarrolladores a nivel mundial, con amplias posibilidades de intercambio de información a través de los foros abiertos y documentación asociada. La orientación hacia la integración de técnicas inteligentes en dispositivos móviles constituye una de sus grandes contribuciones para la comunidad de desarrolladores y científica en este caso desde una perspectiva más avanzada.

Dentro del amplísimo abanico de técnicas inteligentes integradas en dispositivos móviles, bajo el sistema operativo iOS de Apple, la detección o identificación de Acciones Humanas (*HAR, Human Action Recognition*) en secuencias de imágenes, o sea, a través de vídeos captados por dispositivos móviles constituye un segundo elemento motivacional de interés. En efecto, la posibilidad de detectar acciones humanas resulta de gran utilidad, por ejemplo, en videovigilancia con fines de prevención o seguimiento, monitorización en sesiones de rehabilitación, comunicación por signos, por mencionar sólo algunas de las expuestas.

En Asadi-Aghbolaghi y col.[2] se realiza una revisión de métodos en el ámbito de HAR, donde se analizan una serie de modelos con tal propósito, destacando los siguientes:

1. Redes Neuronales Convolucionales.

2. Redes de dos flujos, con separación de la parte espacial y temporal.

3. Redes de segmentación temporal (Temporal Segment Networks).

El presente trabajo se orienta específicamente en el enfoque planteado en el primer punto.

1.2 Modelo de la aplicación

Una vez establecidas las premisas anteriores, en el presente trabajo se propone la utilización de un entorno iOS para reconocimiento de acciones humanas a través de secuencias de imágenes o vídeos captados por un dispositivo móvil y utilizando una Red Neuronal Residual (ResNet).

Así pues, siguiendo el esquema de la figura 1.1, dado un vídeo de entrada con *n frames*, se utiliza una Red Neuronal Convolucional, previamente entrenada. La red recibe como entrada la secuencia de frames indicada, además de las localizaciones (x,y) de los siguientes puntos característicos (ojos, oídos, nariz, cuello, hombros, codos, muñecas, caderas, rodillas, tobillos) junto con el grado de confianza de estos puntos.

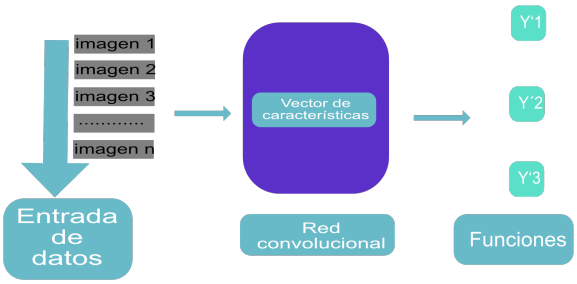


Figura 1.1: Modelo entrada y procesamiento de datos

La aplicación podrá detectar las siguientes actividades físicas:

- Elevación de rodillas o *High knees run in place*: Consiste en elevar las piernas hasta la altura de la cadera y mover los brazos de forma natural coordinando el movimiento de piernas y brazos.
- *Jumping jack*: Consiste en realizar repeticiones de movimientos para alcanzar las siguientes posiciones. En la posición inicial los pies están juntos y los brazos están pegados al cuerpo. La segunda posición consiste en abrir las piernas y elevar los brazos por encima de la cabeza.
- Plancha o *Plank*: En este ejercicio hay que mantener el cuerpo elevado durante un tiempo determinado con los codos y las puntas de los pies posicionados en el suelo.

- Sentadillas o *Sumo squat*: Para realizar este ejercicio en primer lugar hay que colocar los pies alejados entre sí y mantener una posición erguida. A continuación hay que realizar un movmiento de flexión como si fuese a sentarse sobre un lugar. Para mantener el equilibrio se puede ayudar intentando juntar las manos. La ejecución del ejercicio consiste, por tanto, en realizar tantas repeticiones como se pueda en un tiempo determinado de los dos movimientos.

- Sentadilla contra la pared o *Wall sit*: En este ejercicio hay que mantener la posición que se describe a continuación durante un tiempo determinado. Para llevar a cabo la posición, hay que pegar el cuerpo contra la pared y a continuación flexionar las rodillas hasta alzancar una postura como si se estuviese sentado. Las piernas han de estar separadas y los brazos pueden estar o apoyados en las piernas o libres mirando hacía el suelo.

1.3 Objetivos

Como objetivo general se plantea desarrollar una aplicación para dispositivos iOS (sistema operativo de los teléfonos móviles y tablets de Apple) que permita detectar la actividad que está llevando a cabo una persona delante de un dispositivo móvil, basándose en secuencias de acciones capturadas con su cámara.

Para ello se establecen los siguientes subobjetivos:

1. Desarrollar un módulo que capture, mediante vídeo en tiempo real, la actividad de una persona.

2. Adapatar un modelo de ResNet pre-entrenado para extraer caracterísiticas de cada frame o imagen del vídeo.

3. Integrar los diferentes módulos que componen la Aplicación.

4. Verificar los resultados tanto a nivel de módulo o unidad como de su integración en la Aplicación.

5. Mantener la privacidad del usuario lo máximo posible realizando el procesamiento de datos de forma local sin enviarlo a ningún servidor externo.

1.4 Plan de trabajo

Para llevar a cabo los objetivos previos se plantean las siguientes tareas:

1. **Análisis del alcance de los dispositivos y Frameworks a utilizar:** Consiste en estudiar la documentación disponible (ya sea mediante vídeos de conferencias para desarrolladores como la WWDC, o mediante la documentación de desarrolladores de Apple [3]).

2. **Estudiar las herramientas disponibles para crear el modelo** Analizar el funcionamiento de la herramienta a utilizar para crear el modelo, en este caso CreateML [4].

3. **Crear un primer modelo** básico para comprobar el correcto funcionamiento de la herramienta CreateML.

4. **Implementar un módulo que permita capturar vídeo:** Este módulo se encargará de la captura de datos así como su procesamiento para mostrar la actividad que realiza el usuario.

5. **Crear modelo completo:** Reunir los datos necesarios para crear el modelo con las actividades a detectar por la aplicación.

6. **Implementar el módulo de ejercicio:** Evolucionar el módulo que hasta ahora sólo captaba y procesaba el vídeo para que ahora procese todas las actividades, permita pausar y reanudar el entramiento y además muestre el resultado final del entrenamiento.

7. **Crear la base de datos:** Estudiar los *frameworks* de base de datos disponible en forma de iOS e implementar el modelo que permita guardar las estadísticas de los entrenamientos realizados por el usuario.

8. **Desarrollar módulo para crear entrenamientos:** Este módulo permitirá al usuario seleccionar ejercicios para realizar un entrenamiento.

9. **Implementar módulo de ejercicios:** Mostrará los ejercicios disponibles en el aplicación y además permitirá al usuario realizar directamente cada ejercicio sin tener que pasar por el módulo de entrenamiento.

10. **Implementar módulo de gráficos:** Este módulo recogerá los datos en forma de base de datos de la actividad realizada por el usuario y la mostrará, en formato de gráficos de barra.

11. **Pruebas:** Realizar pruebas con usuarios reales.

Capítulo 2

Introduction

2.1 Previous work

Technology has come a long way since the first smartphones appeared. In the last two decades has allowed to create small device with a big processing power.

An example of this is Artificial Intelligence. In the past, videogames companies like Microsoft (Xbox) and Sony (Playstation) used with their consoles an a camera (Kinect or PlayStation Camera) to allow consoles to detect persons and their actions. This allowed to create Augmented Reality videogames where the players could move objetcts on the videogames through gestures.

Apple and Google are creators of the two principals operating systems (Android and iOS) of the smartphones. These companies have developed their own Artificial Intelligence systems. These systems have allowed the development of applications that integrate technologies of augmented reality, object detection and motion detection, among others, in mobile devices that fit in the palm of the hand. Furthermore, technological development has also allowed and facilitated wireless connections of any kind, including Bluetooh, Wi-Fi, 4G (in the future 5G) or others.

On the other hand, the last Apple developers conference, WorldWide Developers Conference of 2020 (WWDC20)[1], held in June 2020, has been a starting point of great interest as a motivating element for this work, sin-

ce it has established an area of help and support from the point of view of developers worldwide, with wide possibilities for exchanging information through open forums and associated documentation. The orientation towards the integration of intelligent techniques in mobile devices constitutes one of its great contributions for the community of developers and scientists in this case from a more advanced perspective.

Within the wide range of intelligent techniques, integrated into mobile devices, under Apple's iOS operating system, the detection or identification of Human Actions (HAR) in image sequences, that is, through videos captured by mobile devices constitutes a second element of motivational interest. Indeed, the possibility of detecting human actions is very useful, for example, in video surveillance for prevention or follow-up purposes, monitoring in rehabilitation sessions, communication by signs, to mention just some of those exposed.

In Asadi-Aghbolaghi et al. [2] a review of methods is carried out in the field of HAR, where different models are analyzed for this purpose, highlighting the following:

1. Convolutional Neuronal Networks.
2. Two-flow networks, with separation of the spatial and temporal part.
3. Temporal Segment Networks.

The present work is specifically oriented on the approach raised in the first point.

2.2 App model

Once the previous premises have been established, in this work the use of an iOS environment is proposed for the human action recognition through sequences of images or videos captured by a mobile device and using a Residual Neural Network (ResNet).

Following the scheme in figure 1.1, given an input video with n frames, a previously trained Convolutional Neural Network is used. The network

receives as input the sequence of frames indicated, in addition to the locations (x,y) of the following characteristic points (eyes, ears, nose, neck, shoulders, elbows, wrists, hips, knees, ankles) together with the degree trust of these points.

The application will be able to detect the following physical activities:

- High knees run in place: It consists of raising the legs to the height of the hips and moving the arms in a natural way, coordinating the movement of the legs and arms.
- Jumping jack: It consists of repeating movements to reach the following positions. In the starting position the feet are together and the arms are close to the body. The second position is to spread your legs and raise your arms above your head.
- Plank: In this exercise you have to keep your body elevated for a certain time with the elbows and the tips of your feet positioned on the ground.
- Sumo squat: To perform this exercise, you must first place your feet away from each other and maintain an upright position. Next, you have to perform a flexion movement as if you were going to sit on a place. To maintain balance you can help trying to put your hands together. The execution of the exercise therefore consists of performing as many repetitions as possible in a given time of the two movements.
- Wall sit: In this exercise, you have to maintain the position described below for a certain time. To carry out the position, you have to press your body against the wall and then bend your knees until you reach a posture as if you were sitting. The legs must be separated and the arms can be either supported on the legs or free facing the ground.

2.3 Objectives

As a general goal, it is proposed to develop an application for iOS devices (operating system of Apple mobile phones and tablets) that allows detecting the activity that a person is carrying out in front of a mobile device, based on sequences of actions captured with its camera.

For this, the following sub-objectives are established:

1. Develop a module that captures, through video in real time, the activity of a person.
2. Adapt a pre-trained ResNet model to extract characteristics from each frame or image of the video.
3. Integrate the different modules that make up the Application.
4. Verify the results both, at the module or unit level, as well as according to their integration in the Application.
5. Maintain user privacy as much as possible by processing data locally without sending it to any external server.

2.4 Work plan

To carry out the previous objectives, the following tasks are proposed:

1. **Analysis of the scope of the devices and Frameworks to be used:** It consists of studying the available documentation (either through videos of conferences for developers such as WWDC, or through Apple's developer documentation[3]).
2. **Study the tools available to create the model:** Analyze the operation of the tool to be used to create the model, in this case CreateML [4].
3. **Create a first basic model:** To check the correct operation of the CreateML tool.
4. **Implement a module that allows to capture video:** This module will be in charge of capturing data as well as its processing to show the activity carried out by the user.
5. **Create complete model:** Gather the necessary data to create the model with the activities to be detected by the application.
6. **Implement the exercise module:** Evolve the module that until now only captured and processed the video so that it now processes all the activities, allows pausing and resuming the training and also shows the final result of the training.

7. **Create the database:** Study the database frameworks available in the form of iOS and implement the model that allows saving the statistics of the trainings carried out by the user.

8. **Develop module to create workouts:** This module will allow the user to select exercises to carry out a training.

9. **Implement exercise module:** It will show the exercises available in the application and also allow the user to directly perform each exercise without having to go through the training module.

10. **Implement graphics module:** This module will collect the data in the form of a database of the activity carried out by the user and display it, in the format of bar graphs.

11. **Testing:** Perform tests with real users.

Capítulo 3

Revisión de métodos y técnicas

El pilar básico de la aplicación es el Aprendizaje Automático que es la capacidad que obtienen las máquinas de adquirir conocimiento encontrando patrones a partir de un conjunto de datos datos.

A continuación se realiza una revisión de los distintos conceptos involucrados en el desarrollo del presente trabajo.

Aprendizaje profundo o *Deep learning*, es uno de los métodos de Aprendizaje. Esta metodología imita el cerebro humano para detectar patrones y crear sus propias reglas sobre un conjunto de datos. Para el desarrollo de la aplicación se han utilizado Redes Convolucionales, en concreto una ResNet, una técnica de Aprendizaje profundo [5].

Red Neuronal Artificial: Es una estructura compuesta por unidades básicas, llamadas neuronas, conectadas entre sí a través de lo que se denominan sinapsis, estas conexiones se caracterizan por tener lo que se conocen como pesos asociados, que son los parámetros que aprende el modelo durante la fase de entrenamiento[6].
El modelo consta de una capa de entrada, que recibe los datos, y una serie de capas ocultas para finalizar con la capa de salida. Todas las capas están formadas por las mencionadas neuronas interconectadas entre las neuronas de las capas previas y subsiguientes.
Las neuronas se caracterizan además, por poseer un mecanismo de activación, dependiendo de la entrada a la misma.
En definitiva, cuando un determinado dato es recibido por la capa de entrada,

éste se propaga a través de las capas ocultas, con la correspondiente activación de las neuronas hasta llegar a la capa de salida, donde se determina la clase a la que pertenece la entrada, cuando el modelo es de clasificación.

Teniendo en cuenta el marco teórico expuesto y con referencia a la figura 1.1, a continuación se describen las estructuras estructuras y operaciones básicas que conforman la estructura del modelo ResNet, junto con el propio modelo en su conjunto.

3.1 Operaciones en redes neuronales convolucionales

3.1.1 Convolución

Una red Neuronal Convolucional es un tipo de Red Neuronal que aplica operaciones de Convolución (de ahí su nombre) en determinadas capas para el procesamiento de datos. La Convolución es una operación matemática que permite combinar dos funciones, que involucran datos a partir de la cual se genera un nuevo resultado. Dada una señal o imagen de entrada, sobre los valores de la misma se aplica una operación matemática lineal a través de lo que se conoce como núcleo de convolución. Este núcleo consta de unas serie de valores o pesos que son los que se aprenden durante el proceso de aprendizaje, siendo ésta la clave de este tipo de redes.[7]

La convolución es una operación matemática definida como sigue en la ecuación (3.1) en el contexto del procesamiento de imágenes 2D

$$C(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{3.1}$$

Las coordenadas *C(i,j)* es el resultado de la operación de convolución en relación a un píxel o unidad dada *(i,j)* sobre una imagen de entrada I o tensor aplicando un núcleo definido por sus correspondientes parámetros, que son los pesos que se estiman o aprenden durante el proceso de entrenamiento de la red.

La figura 3.1 muestra un ejemplo de convolución con el núcleo K, aplicado sobre un tensor 2-D que bien puede representar una imagen I, como se ha indicado previamente. La convolución se representa con solapamiento total del

núcleo y obteniendo una imagen resultante cuyos bordes externos se quedan sin valores, generando así una imagen de menor dimensión que la original; en el caso de la figura se pasa de tener dimensión 6x7 a 4x5. No obstante, si se desea obtener una imagen de la misma dimensión lo que hay que hacer es ampliar la imagen original en dos filas (arriba y abajo) y dos columnas (izquierda y derecha) con ceros, procesándola con el núcleo solapado. Esta operación es lo que se conoce como relleno con ceros o *zero-padding* y generalmente se indica en términos de implementación "same". Por el contrario, si no se realiza tal operación, es decir, no se añaden ceros, en implementación se indica como "valid".
En referencia a este último caso, más adelante se indica que la operación es sin relleno con ceros (*no zero-padding*). El modelo ResNet mostrado en la figura 3.6, el valor de p indica el *padding* aplicado.

En las convoluciones, el campo receptivo se define como la región de entrada que contribuye a la salida generada por el filtro. En la figura 2-2 se muestran sendos campos receptivos de la imagen I que contribuyen a las salidas P y Q generadas por el filtro K.

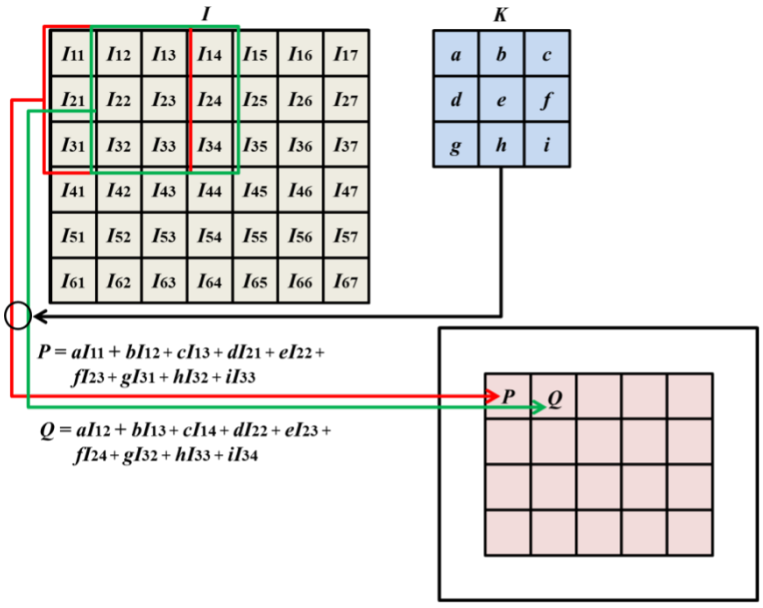


Figura 3.1: Ejemplo convolución 2-D

3.1.2 Unidad Lineal Rectificada o Rectified Linear Unit

Se trata de una función de activación que sirve para indicar de forma matemática el grado en que se posee una característica[8]. Es muy frecuente en los modelos de redes que, tras cada capa convolucional, se aplique una función de transformación que rectifica el valor de entrada devolviendo 0 para valores negativos y el propio valor de entrada para valores positivos. En la ecuación 3.2 se muestra la función ReLU

$$f(x) = x^+ = max(0, x) \tag{3.2}$$

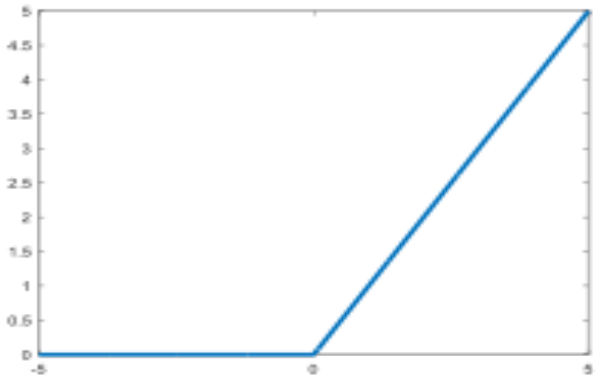


Figura 3.2: Ejemplo gráfica ReLU

3.1.3 Pool Overlapping

La operación de pooling se aplica para reducir la dimensionalidad, de forma que en cada paso, la posición de la ventana se actualiza de acuerdo a los desplazamientos (*strides*)[9]. Cuando se sitúa la ventana en las proximidades de los píxeles, de borde algunos de los elementos de la ventana pueden quedar fuera de los elementos de entrada. Por tanto, para obtener los valores de las regiones de borde, la entrada puede extenderse con valores de cero, lo que previamente se ha denominado *zero-padding*. Si bien, puede optarse por no aplicar este tipo de extensión, en cuyo caso se dice que la operación que se realiza es no *zero-padding*. Al igual que en el caso de la convolución y desde el punto de vista de la implementación, cuando se aplica *zero-padding* se suele indicar como "same" y en caso contrario como "valid". En determinadas representaciones gráficas de capas de red, esta última operación se suele indicar

con p = 0 o equivalentemente con el símbolo "V" de "valid". Por el contrario, en el primer caso, se indica exactamente el valor de p sin más especificación de simbología y por tanto, rara vez aparece "S" refiriéndose a "same".

En la figura 3.3 se muestra un ejemplo ilustrativo de carácter pedagógico de agrupamiento máximo (*max pooling*) sin relleno con ceros (*no zero padding*) (V, p = 0) y desplazamiento s = 2. En este caso, la ventana se desplaza de dos en dos posiciones en horizontal y vertical a partir del inicio, si bien al ser una operación de tipo V, la columna más a la derecha no interviene, dado que las ventanas no se pueden solapar sobre ella.

En la figura 3.3 se muestra también la misma operación, si bien ahora con relleno con ceros (*zero padding*) (S, p = 1 en horizontal) y s = 2). En este caso la ventana mediante su desplazamiento en saltos de dos en dos ya puede incorporar a la columna de la derecha.

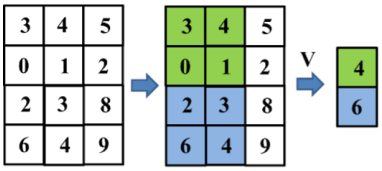


Figura 3.3: Max pooling, no zero-padding (V), s = 2

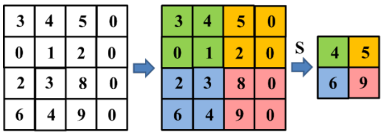


Figura 3.4: Max pooling, zero-padding (S), s = 2

3.1.4 Dropout

Es una técnica desarrollada por Hilton y col.[10] y se trata de una técnica que resuelve el problema de sobreajuste (*overfitting*). La idea subyacente detrás de esta técnica consiste en anular de manera aleatoria unidades (junto a sus conexiones) de la red neuronal durante el entrenamiento para evitar cálculos que den como soluciones polinomios de un grado muy alto.

En Ioffe y Szegedy[11] se propone la siguiente estrategia para la normalización por lotes (*batch normalization*). Con tal propósito se realizan dos simplificaciones. La primera es que en lugar de aplicar la normalización conjuntamente en las capas de entrada y salida, esta se realiza independientemente sobre cada mapa de características escalares, haciendo que tomen como media cero y varianza uno. Para una capa con entrada d-dimensional, x =x(1),. . . ,x(d), se normaliza cada dimensión según la ecuación 3.3 siguiente,

$$\hat{x} = \gamma \frac{x - m}{\sigma} + \beta \tag{3.3}$$

donde la media (m) y la desviación estándar (σ) se calculan sobre el conjunto de datos, mientras que γ y β son sendos factores de ajuste y desplazamiento respectivamente. El concepto batch o lote se refiere a que la normalización no se aplica para cada imagen sino tras el paso de un conjunto especificado de imágenes que constituyen el lote.

3.1.5 Softmax

Consistente en proyectar los valores de la salida en la capa final al rango [0,1], proporcionando así una especie de probabilidad de pertenencia a cada una de las clases para una imagen de entrada dada. Su función se define según la siguiente expresión[12].

$$softmax(x)_{\mathrm{i}} = \frac{exp(x_{\mathrm{i}})}{\sum_{j=1}^{n}} exp(x_{\mathrm{j}}) \tag{3.4}$$

3.1.6 Gradiente descendente

Es una técnica de minimización utilizada para el ajuste de los pesos involucrados en los modelos de las redes durante el proceso de retro-propagación. La función a optimizar se conoce como función *objetivo* y cuando se está minimizando, se le denomina también *loss function* o *error function*. Se trata de minimizar una función objetivo que tiene la forma,

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} J_{\mathrm{i}}(w) \tag{3.5}$$

donde el parámetro w que minimiza *J(w)* debe estimarse, constituyendo el objetivo principal del aprendizaje. J_i se asocia con la *i-ésima* observación

en el conjunto de datos utilizados para el entrenamiento (ajuste). El gradiente descendente se usa para minimizar esta función de forma iterativa, con iteraciones t,

$$w(t+1) = w(t) - \epsilon \frac{1}{n} \sum_{i=1}^{n} \nabla J_{\mathrm{i}}(w) \tag{3.6}$$

De esta forma iterativa el método recorre el conjunto de entrenamiento y realiza la actualización anterior para cada muestra de entrenamiento. Se pueden realizar varios pasos sobre el conjunto de entrenamiento hasta que el algoritmo converja. Estas muestras así seleccionadas constituyen lo que se denomina *batch*, como se describe más adelante, a la hora de seleccionar los parámetros de entrenamiento. En este sentido, es habitual utilizar exactamente la técnica conocida como Gradiente Descendente Estocástico (SGD, Stochastic Gradient Descent)[13].

Las implementaciones típicas pueden usar una razón de aprendizaje adaptativa para que el algoritmo consiga una mejor convergencia. En pseudocódigo, el método de gradiente descendente estocástico es como sigue,

1. Elegir un vector inicial de parámetros w (puede ser aleatoriamente) y razón de aprendizaje ϵ.
2. Repetir hasta que se consigue un mínimo aproximado
   2.1. Seleccionar aleatoriamente ejemplos en el conjunto de entrenamiento
   2.2. Para i = 1,2,..,n, hacer $w(t+1)=w(t)-\epsilon \nabla J_i(w)$

3.2 RESNET

La motivación para plantear este tipo de estructuras en las Redes Neuronales Convolucionales surge a partir de las experiencias realizadas por He y col.[14] en relación con el número de capas de los diferentes modelos, ya que cuanto más profundas sean las redes mayor dificultad existe para el entrenamiento. Es precisamente en relación con el hecho de conseguir una mayor facilidad para realizar el entrenamiento, esto es, mayor facilidad en el proceso de optimización, donde surge la propuesta formulada bajo el paradigma *Deep*

Residual Learning, que desemboca en el marco de referencia propuesto en He y col.[14]. Según dicho trabajo, a veces ocurre que cuando la profundidad de la red aumenta, la precisión se satura llegando a degradarse rápidamente, no necesariamente causada por *overfitting*, y a veces el hecho de añadir más capas a la red conduce a un mayor error en el entrenamiento.

La idea de la red neuronal residual (ResNet, *Residual neural network*) consiste en la introducción de lo que se conoce como cortocircuito, o en términos originales *identity shortcut connection*, consistente en saltar una o más capas de la red original.

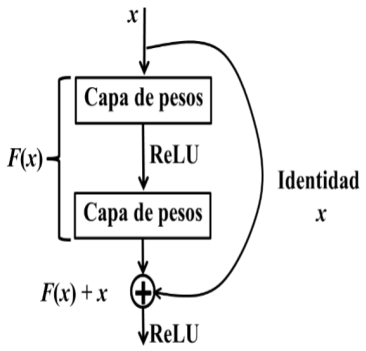


Figura 3.5: Bloque ResNet

En lugar de esperar a que cada pocas capas apiladas encajen o se acoplen directamente hacia una proyección subyacente deseada, se deja explícitamente que estas capas se ajusten a una proyección residual. Formalmente, denotando la proyección subyacente deseada como $H(x)$, se permite que las capas no lineales apiladas se ajusten a otra proyección $F(x) = H(x)$ - x; de esta forma, la proyección original se plantea como $F(x) + x$ bajo la hipótesis de que si una proyección de identidad es óptima, resultaría más fácil llevar el residuo, $F(x)$, a cero que ajustar una proyección de identidad mediante una pila de capas no lineales. Dicho de otra forma, es más fácil encontrar una solución tal que $F(x) = 0$ en lugar de $F(x) = x$ usando una pila de capas no lineales como función. Como se ha indicado, la función $F(x)$ es lo que se denomina residuo. Bajo esta formulación se argumenta que las capas apiladas no deberían degradar el rendimiento de la red, porque simplemente se pueden apilar asignaciones de identidad; es decir, una capa que no hace nada, en la red actual, y la arquitectura resultante funcionaría de la misma

manera. En cualquier caso, conviene dejar claro que la red posee tanto las capas apiladas no lineales que producen la salida $F(x)$ como el cortocircuito que produce la identidad $\boldsymbol{x}$.

La formulación de $F(x) + x$ puede implementarse mediante redes neuronales con propagación hacia adelante (*feedforward*) con conexiones cortocircuitadas que saltan una o más capas. En el modelo de la figura 3.5 las conexiones cortocircuitadas realizan simplemente una proyección de identidad, de forma que sus salidas se añaden a la salida de las capas apiladas. En definitiva, la idea consiste en que a la hora de realizar la operación de identidad, $H(x) = F(x) + x$, el objetivo es que la red aprenda cualquier función $F(x)$ de forma que esta sea cero, $F(x) = 0$.

Conviene señalar, que las conexiones de identidad cortocircuitadas no añaden parámetros extra ni complejidad computacional. La red completa puede entrenarse de principio a fin utilizando el Gradiente Descendente Estocástico (SGD) mediante retro-propagación para conseguir lo indicado anteriormente, descrito en el capítulo tres, sección 3.1.6.

Así pues, considérese $H(x)$ como una proyección subyacente para ser ajustada por unas pocas capas apiladas subyacentes, y por tanto no necesariamente la red completa, con x siendo las entradas a la primera de esas capas. Para las capas apiladas se aplica el aprendizaje residual, obteniendo un bloque completo como el mostrado en la figura 3.5, que se define como sigue,

$$y = F(x, \{W_{\mathrm{i}}\}) + x \tag{3.7}$$

Siendo x e y los vectores de entrada y salida de las capas consideradas. Como se ha indicado anteriormente la función $F(x, \{\mathrm{W_i}\})$ representa la proyección residual que debe aprenderse. Por ejemplo, siguiendo el trabajo de He y col.[14] y considerando la figura 3.5 que tiene dos capas, $F \equiv W_2\sigma(W_1x)$, donde σ indica la capa ReLU, omitiendo los pesos bias por simplicidad. La operación $F(x) + x$ se realiza por una adición elemento a elemento. En el ejemplo de la mencionada figura, la segunda operación no lineal, ReLU, se lleva a cabo después de la suma.

Un aspecto de relevancia a la hora de realizar la adición es que las dimensiones de $\boldsymbol{F}$ y $\boldsymbol{x}$ deben ser iguales. Si esto no se cumple, es necesario

realizar una proyección lineal W_s para las conexiones cortocircuitadas para establecer la deseada correspondencia quedando la siguiente expresión,

$$y = F(x, \{W_\mathrm{i}\}) + W_\mathrm{s}x \tag{3.8}$$

En cualquier caso, en la ecuación 3.8 se pondría igualmente una matriz Ws, aunque según He y col., consideran que esto es inefectivo en base a los resultados obtenidos en los experimentos, siendo suficiente la identidad. Siguiendo a estos mismos autores, la forma de la función residual F es flexible en extensión, pudiendo abarcar dos o tres capas o incluso más. En cualquier caso, una única capa es similar a una capa lineal $y = W_1x + x$ por lo que no se observan ventajas significativas.

Por otra parte señalar que aunque por simplicidad las notaciones anteriores se refieren a capas totalmente conectadas, esto es aplicable a capas convolucionales. La función $F(x, \{\mathrm{W_i}\})$ puede representar múltiples capas convolucionales. La operación de adición elemento a elemento se realiza sobre los dos mapas de características canal por canal.

Con carácter ilustrativo en la figura 3.6 se muestra un ejemplo didáctico de un tramo de red con distintos bloques y con dos cortocircuitos, A y B, de suerte que la correspondiente función residual comprende 2 y 3 capas respectivamente. Se consideran cinco volúmenes numerados del 1 al 5 con las dimensiones y las operaciones que se indican. Para cada volumen se identifican las correspondientes dimensiones, siendo respectivamente las siguientes: 55x55x96, 27x27x256, 13x13x256 y 13x13x384. Al lado de cada volumen se indican los parámetros utilizados en las operaciones definidas por las relaciones indicadas, concretamente se trata de la dimensión de los núcleos de convolución dxd, *stride* (s), *padding* (p) y número de núcleos (K) generando los volúmenes con las dimensiones correspondientes mediante las operaciones indicadas en dichas relaciones. Por otro lado, en el cortocircuito A se aplica la relación 6, definida posteriormente, sobre el volumen 1 con los parámetros indicados para obtener a la salida (antes de la operación suma) un volumen de dimensión 27x27x256, cuyas dimensiones se corresponden exactamente con las del volumen 3, pudiéndose en este caso realizar la operación de suma elemento a elemento entre este volumen resultante y el volumen 3 en cada uno de los 256 canales de sendos volúmenes, obteniendo como resultado de esta suma un bloque de volumen 27x27x256, es decir, de las mismas dimensio-

nes de los bloques que conforman los sumandos. En el caso del cortocircuito B se aplica también la relación 6, con los parámetros indicados, para obtener un volumen de salida de dimensión 13x13x256, cuyas dimensiones se corresponden en este caso exactamente con las del volumen 4, pudiéndose ahora realizar también la operación de suma elemento a elemento entre este volumen resultante y el volumen 4, de nuevo para los 256 canales.

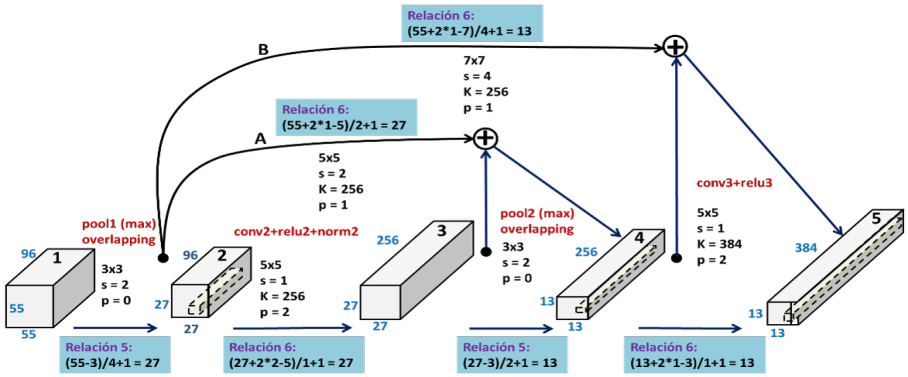


Figura 3.6: ResNet ejemplo ilustrativo

Por otra parte, el núcleo de convolución o las ventanas de *padding*, definidos previamente, se puede desplazar a través de la imagen con saltos de celda de uno en uno o con saltos más altos. También, como se ha indicado previamente, este movimiento se define mediante el parámetro s que aparece en el esquema del modelo ResNet, conocido técnicamente como *stride*. Así, cuando es uno, el desplazamiento es de una unidad y cuando es por ejemplo dos de dos unidades.

Finalmente, en la arquitectura del modelo ResNet aparecen sendas relaciones 5 y 6 que se definen a continuación:

$$\text{Relación 5: } o = [\frac{i-k}{s}] + 1 \quad \text{Relación 6: } o = [\frac{i+2p-k}{s}] + 1 \tag{3.9}$$

donde i define la dimensión de la estructura de la capa de entrada, k es el tamaño del filtro aplicado, que en la figura previa aparece como kxk; p determina el *padding* y s el desplazamiento o *stride*. Así, por ejemplo, fijándose en la operación de convolución conv2, la estructura de entrada tiene dimensión 27, esto es i = 27, con un núcleo de dimensión 5x5 por lo que k

= 5, p = 2 y s = 1, por lo que aplicando la Relación 6, se obtiene la salida o = 27, que determina las dimensiones alto y ancho de la estructura de la siguiente capa, mientras que la profundidad de la misma lo determina K, en este caso K = 256.

Por tanto, bajo el planteamiento del modelo ResNet quedan diseñados dos tramos, figura 3.7. En efecto, el primero que enlaza los bloques 1, 2, 3, 4 y 5 junto con el cortocircuito A y el segundo que une los mismos bloques junto con el circuito B; obviamente los dos no conviven en la misma arquitectura, es decir, que se opta por uno u otro pero no ambos a la vez.
Si bien, en ambos casos durante el entrenamiento se ajustan los pesos que conforman las uniones con el objetivo indicado de que mediante los cortocircuitos la idea consiste en conseguir, a través de esos ajustes, que la función residual $\boldsymbol{F}$ sea cero en los cortocircuitos.

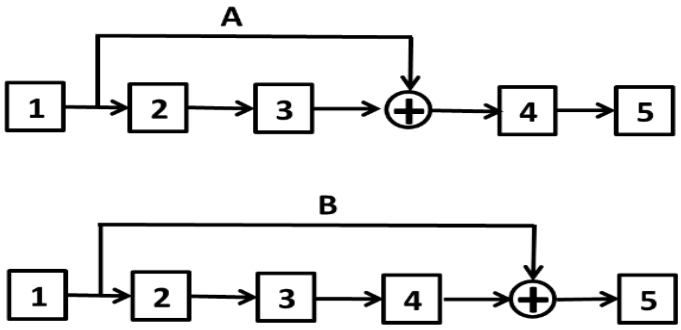
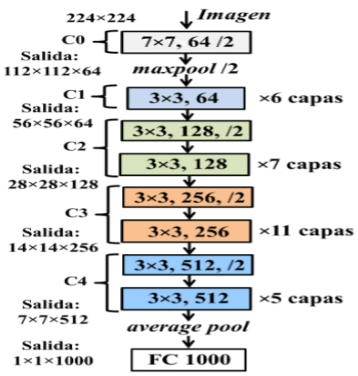
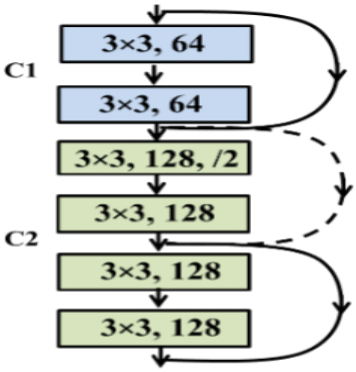


Figura 3.7: Tramos ResNet

En el trabajo de He y col.[14] se realiza un estudio comparativo a partir de una red base con 34 capas, inspirada en el modelo VGG-19 con una ResNet, con resultados favorables para esta. Más concretamente, inspirándose en la arquitectura VGG-19 se crea la red base de 34 capas tal y como aparece en la figura 3.8(a). Obsérvese que consta de cinco módulos C0 (1 capa), C1 (6 capas), C2 (8 capas), C3 (12 capas) y C4 (6 capas), haciendo un total de 33 que junto con la capa FC (*fully connected*) suman las 34. En estas representaciones, la nomenclatura viene dada por (M, *kxk*, K) donde M expresa la dimensión del mapa de características de la capa dada, kxk el tamaño del núcleo de convolución y K el número de filtros aplicados en la capa dada, que genera esa misma dimensión en el mapa de características de la capa de



(a) Modelo de 34 capas inspirado en la arquitectura VGG-19.



(b) Modelo de 50 capas.

Figura 3.8: ResNet.

salida; *maxpool* se refiere a la operación de *pooling* con la operación máximo, el símbolo /2 indica la operación desplazamiento (stride, con s igual a 2), FC 1000 (*fully connected*) con clasificación para 1000 categorías de clasificación y *pooling* promediado (*average pool*).
A partir de la red de 34 capas se crea la ResNet-50, que como su nombre indica, consta de 50 capas, obtenidas reemplazando cada bloque de dos capas en la red de 34 con el bloque de 3 capas con una arquitectura similar a la del bloque mostrado en la figura 3.8 si bien variando los tamaños de los núcleos de convolución y el número de filtros.
Suponiendo una imagen de entrada de dimensión 224x224, mediante la aplicación de la convolución con desplazamiento (*stride*) s = 2, se obtiene un mapa de características de salida de dimensión 112x112x64, de forma que a su vez tras la operación *maxpool* con s = 2, la dimensión del mapa de características es 56x56x64, que se propaga hasta la salida del bloque C1.
En el bloque C2 se aplican de nuevo las convoluciones indicadas, incluyendo un desplazamiento (s = 2), generando un mapa de características de 28x28x128. En los bloques C3 y C4 se generan las salidas de mapas 14x14x256 y 7x7x512 respectivamente, tras las convoluciones indicadas y considerando los dos desplazamientos, de nuevo con s = 2. Se llega así, a la capa FC con 1000 unidades de salida.

Capítulo 4

Diseño de la aplicación

4.1 Introducción

En los dos capítulos precedentes se han establecido las bases del proyecto, además de los fundamentos teóricos relativos al Aprendizaje Profundo que lo sustentan. En este capítulo se está en disposición de abordar el diseño de la aplicación en su conjunto. El punto de partida lo constituye el modelo de aprendizaje utilizado, que se describe en la sección 4.2, y que por otra parte constituye el núcleo central de la propia aplicación. En la sección 4.3 se describen las herramientas y recursos utilizados para el desarrollo de la aplicación, incluyendo el lenguaje de programación y los *frameworks* tanto para el desarrollo de la interfaz de usuario como de la base de datos necesaria. La sección 4.4 aborda la arquitectura de la aplicación bajo la perspectiva del Modelo-Vista-Modelo. Finalmente, en la sección 4.5 se muestra la estructura relativa a la secuencia de captura de datos y predicción de la acción como objetivo final de la aplicación.

4.2 Modelo

La secuencia de figuras 4.1 a 4.5 desglosadas a su vez en nueve partes, muestran los diferentes componentes que constituyen la red neuronal en su conjunto, cuya visualización se ha realizado con la aplicación Netron[15], específicamente diseñada para tal propósito en el ámbito del aprendizaje automático en general y las redes neuronales en particular, incluyendo naturalmente las situadas bajo el paradigma de Aprendizaje profundo, como es el caso. El mo-

delo recibe como entradas secuencias de poses humanas y proporciona como salida el tiempo de acción clasificada según la naturaleza de esta. Para ello, utiliza un array multidimensional cuya primera dimensión se corresponde con el índice para iterar sobre los 60 frames requeridos por el modelo. En el capítulo siguiente se justifica la necesidad de utilizar una ventana de 60 frames para poder llevar a cabo una predicción de la acción. La segunda dimensión indexa las dimensiones horizontales (x) y verticales (y) de cada imagen, junto con los puntos característicos son: ojos, oídos, nariz, cuello, hombros, codos, muñecas, caderas, rodillas y tobillos la probabilidad de las localizaciones de los puntos clave del cuerpo.

En las estructuras mostradas se identifican las distintas unidades convolucionales que lo conforman, así como las unidades de activación ReLU, pooling, dropout, normalización o softmax aplicadas.

El modelo en su conjunto es el que se muestra en las figuras 4.1 a 4.5. En cada una de ellas se muestra una parte del mismo de forma que la unión de todas ellas conforma el modelo global. Las distintas partes se unen en el orden establecido de forma que en la parte I es la que recibe la entrada de datos a través de la unidad textitposes. Cada parte se une con la siguiente hasta llegar a la parte IX que contiene la salida, de forma que para una entrada dada se obtiene la etiqueta correspondiente a la actividad reconocida, junto con las probabilidades asignadas a cada acción. Las partes I y II son estrictamente secuenciales, de forma que la información fluye entre las distintas unidades sin ningún tipo de puente o cortocircuito. En la parte I se distinguen las siguientes unidades y por este orden:

Transpose: Es una capa que se encarga de reordenar las dimensiones y los datos de una entrada. La entrada es de N-dimensiones así como la salida. Ejemplo: Si la entrada es [1, 2, 3, 4], producirá una salida [4, 5, 3, 1].
ReshapeStaticLayer: Se encarga de aplicar el formato especificado en el parámetro que recibe como entrada *targetShape*.
ExpandDims: Una capa que aumenta el rango del tensor de entrada agregando dimensiones unitarias.
Batchnorm: Esta capa aplica una normalización de tipo *batch*. Esta operación se realiza en el eje -3 y se repetie a través del resto de ejes si están presentes.
Squeeze: Reduce el rango del tensor de entrada eliminando dimensiones unitarias. El rango de salida es uno menos que el rango de entrada. La única excepción es que, si el rango de entrada es 1, el rango de salida también es 1.

SplitND: Una capa que se divide uniformemente a lo largo del eje = -3 para producir un número específico de salidas.
ConcatND: Una capa que se concatena a lo largo del eje = -3 o -5.

Aparte de algunas unidades descritas anteriormente, la parte II consta de
Padding: Especifica la cantidad de margen espacial que se debe rellenar o recortar.

Por el contrario, el resto de las partes contienen cortocircuitos de forma similar a las estructuras ResNet descritas en la sección 3.2. Sin embargo, a diferencia del módulo y la estructura descritos en dicha sección donde los cortocircuitos son la identidad (*identity shortcut connection*), en estos módulos se añaden unidades extra identificando tres tipos de módulos, a saber, A, B y C. Los módulos de tipo A, mostrados se caracterizan porque la rama principal consta de las siguientes unidades:

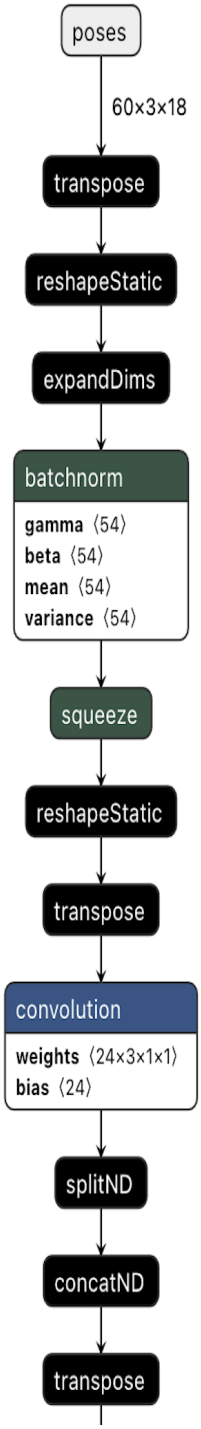
SplitND, ConcatND, Transpose, Padding, batchnorm han sido comentadas anteriormente: **Convolution:** Una capa que realiza una convolución o deconvolución espacial. Como entrada recibe un set de dimensión mayor o igual a 4. Este set representa [*Batch*, *channels*, *height*, *width*]. Para conjuntos de dimensiones mayores que 4, las dimensiones mayores que 4 (se cuenta a partir de 0), son tratadas todas como *batch*

En la tabla 4.1 se muestran las dimensiones de los estructuras o tensores de las unidades de convolución que contienen los pesos que se ajustan durante el entrenamiento. En aquellas unidades de convolución donde aparece el parámetro bias, correspondiente al sesgo las dimensiones coinciden con la primera dimensión del tensor correspondiente de convolución. Las unidades de normalización poseen valores para cada uno de los parámetros gamma, beta, media y varianza. Cada uno de estos módulos se inicia con una unidad de activación tipo ReLU

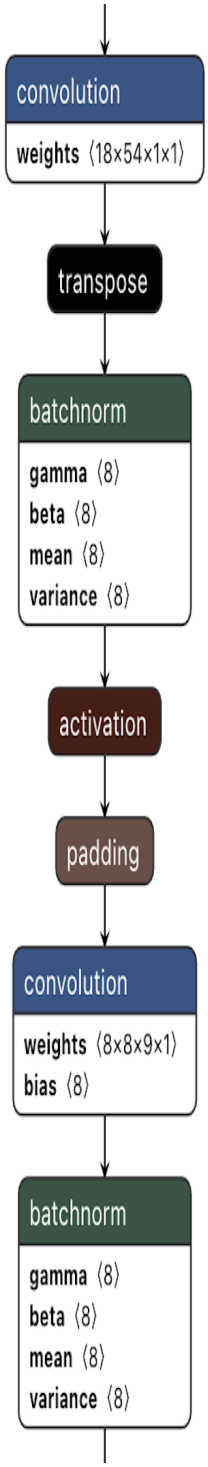
	Rama principal	Cortocircuito
Parte III	48x8x1x1 18x54x1x1 16x16x9x1	16x8x1x1
Parte IV	96x16x1x1 18x54x1x1 32x32x9x1	32x16x1x1
Parte V	192x32x1x1 18x54x1x1 64x64x9x1	64x32x1x1
Parte VII	384x64x1x1 18x54x1x1 128x128x9x1	128x64x1x1

Cuadro 4.1: Dimensiones de las unidades de convolución

El módulo de tipo B de la parte VI, mostrado en la figura 4.3 es un ResNet puro, esto es, con el cortocircuito definido como la identidad, de forma que la rama principal consta de las mismas unidades que la rama principal de los módulos tipo A. El módulo de tipo C de la parte IX, mostrado en la figura 4.5, contiene también un cortocircuito que es la identidad.

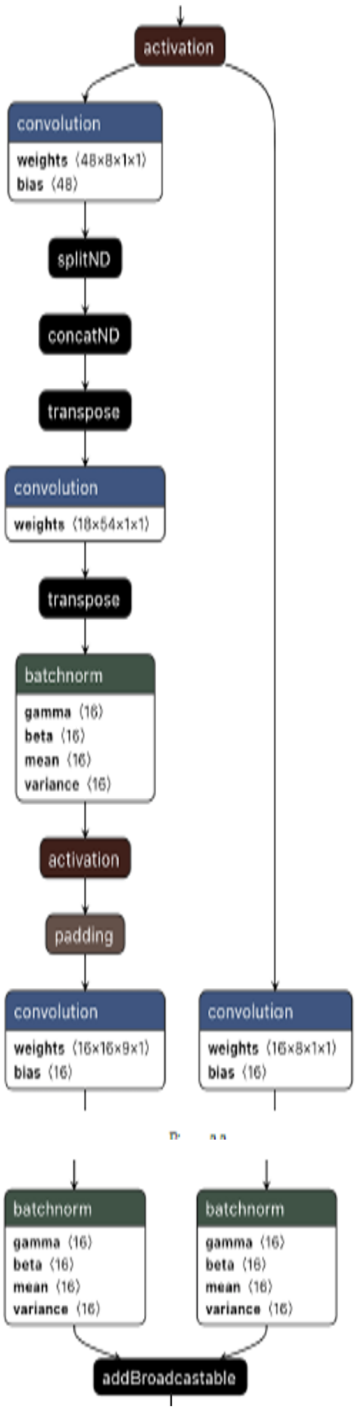


(a) Parte I: secuencia

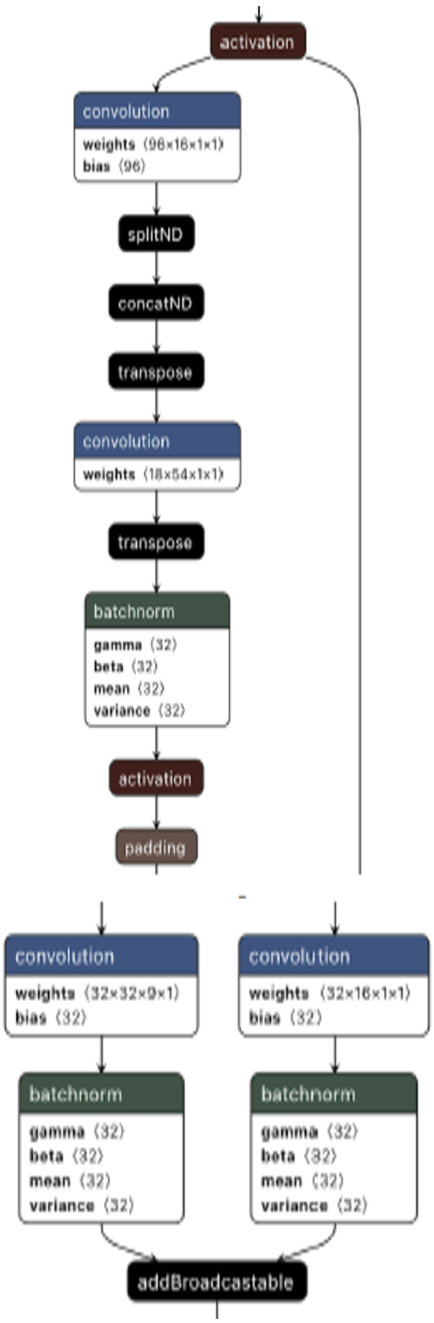


(b) Parte II: secuencia

Figura 4.1: Partes I y II

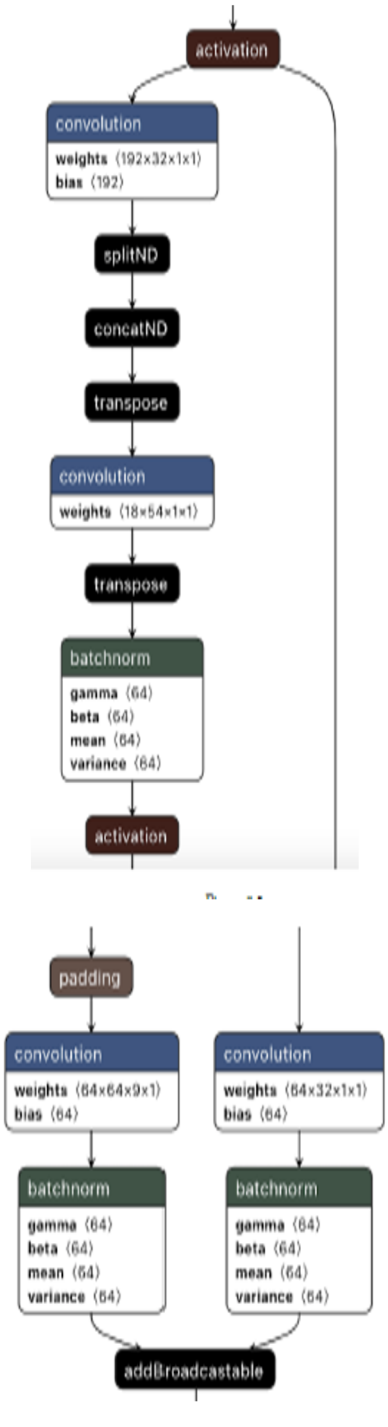


(a) Parte III: módulo ResNet tipo A

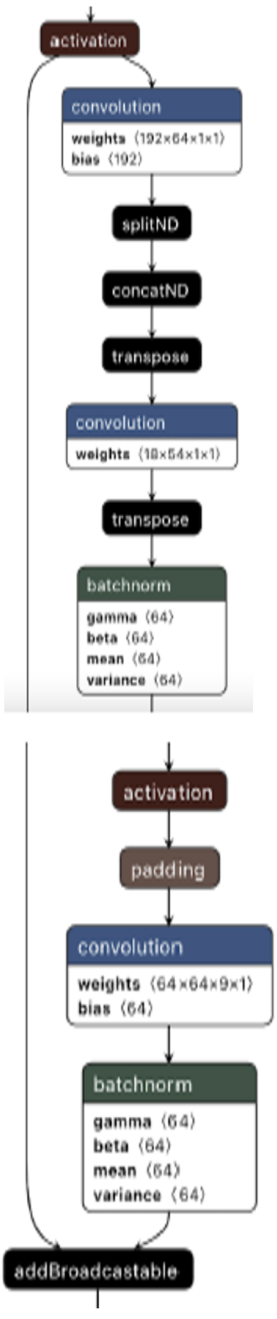


(b) Parte IV: módulo ResNet tipo A

Figura 4.2: Partes III y IV

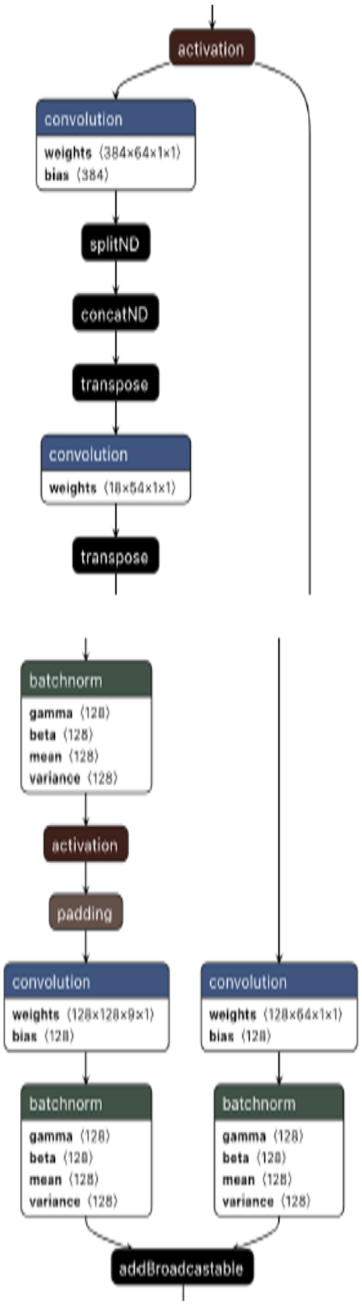


(a) Parte V: módulo ResNet tipo A

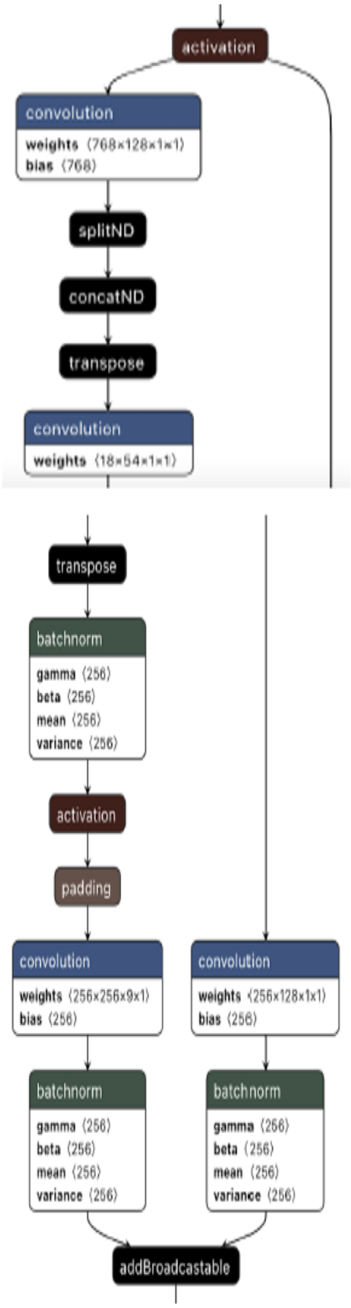


(b) Parte VI: módulo ResNet tipo B

Figura 4.3: Partes V y VI



(a) Parte VII: módulo ResNet tipo A



(b) Parte VIII: módulo ResNet tipo A

Figura 4.4: Partes VII y VIII

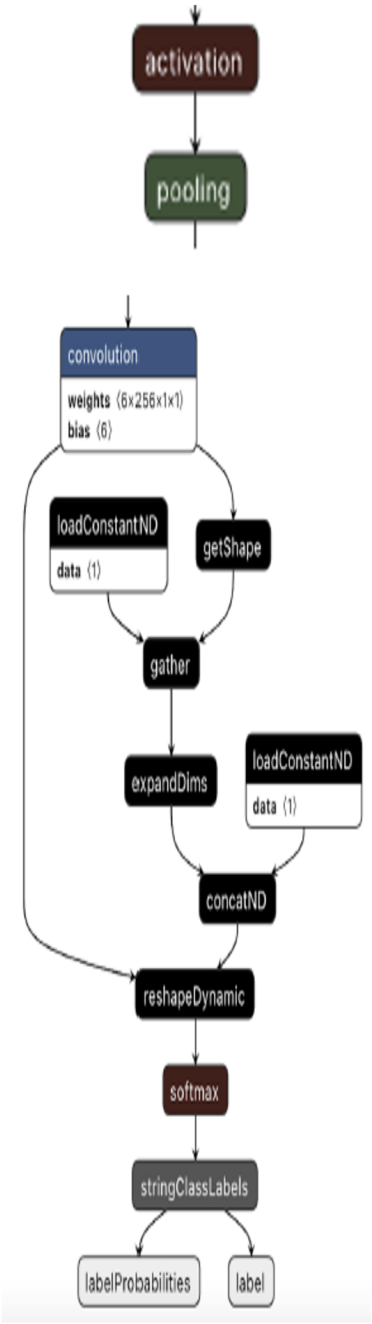


Figura 4.5: Partes IX: módulo ResNet tipo C

4.3 Herramientas y recursos

4.3.1 Swift

El lenguaje utilizado para desarrollar la aplicación ha sido Swift[16]. Swift es un lenguage de programación funcional desarrollado en sus inicios por Chris Lattner[17] y presentado en 2014 y enfocado para el desarrollo de aplicaciones para iOS y macOS.

4.3.2 SwiftUI

Es un *framework* que nos permite desarrollar la interfaz de usuario de una aplicación de forma declarativa. SwiftUI junto con Xcode permite visualizar de forma rápida la evolución en el desarrollo de la interfaz de usuario al mostrar una previsualización de la misma conforme añades elementos[18].

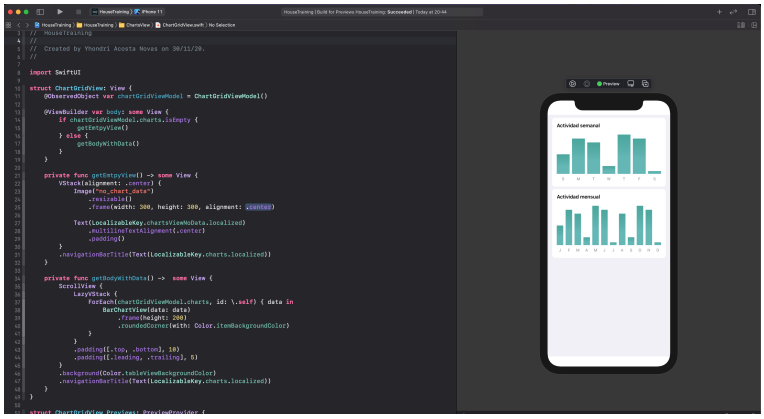


Figura 4.6: Previsualización de la interfaz en Xcode

En la figura 4.6 se muestra una imagen ilustrativa donde en la parte izquierda aparece la interfaz escrita en SwiftUI y en la derecha la interfaz tal y como se verá al ejecutar la aplicación.

4.3.3 UIKit

UIKit[19] es otro *framework* de Apple orientado al desarrollo de la interfaz de usuario de una aplicación. Es completamente compatible con SwiftUI y ha

sido necesaria su utilización, ya que SwiftUI aún no integra todos los componentes que contiene UIKit. La decisión de utilizar SwiftUI, aún no teniendo todos los componentes necesarios se debe a que en general, el desarrollo de la interfaz de usuario con SwiftUI es mucho más rápido y finalmente UIKit sólo ha sido necesario para desarrollar un par de pantallas.

4.3.4 Core Data

Core Data[20] es el *framework* de base datos desarrollado por Apple para el ecosistema de sus dispositivos iOS y macOS. Core Data provee entre otras soluciones generales la persistencia de datos, así como su gestión de forma gráfica. Este ha sido el *framework* seleccionado para la base de datos de la aplicación ya que se encuentra perfectamente integrado con SwiftUI entre otras razones porque gestiona automáticamente las actualizaciones de las entidades de base de datos y además se encarga de actualizar las vistas cuando ocurre algún cambio.

En la figura 4.7 se puede observar el modelo de base de datos de la aplicación. El panel lateral que se ilustra en la figura, sirve para gestionar las relaciones entre las entidades.

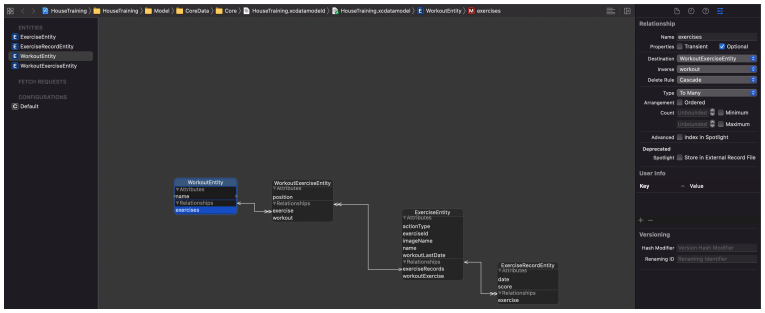


Figura 4.7: Esquema de las entidades que conforman la base de datos de la aplicación.

4.4 Arquitectura de la aplicación

La arquitectura seleccionada para el desarrollo de la aplicación ha sido, Modelo-Vista-Modelo de vista (en inglés *Model-View-ViewModel* MVVM). Esta arquitectura fue desarrollada por un grupo de trabajo en Microsoft y

presentada por John Gorssman en 2005[21]. Se ha utilizado esta arquitectura porque es la recomendada para utilizar con SwiftUI ya que el mismo framework provee de elementos para gestionar la comunicación entra la vistas y los modelos. Como podemos ver en el esquema MVVM se compone de 3 capas.

- **Modelo:** Esta capa contiene los objetos y la lógica de negocio. Es la única capa que tiene conocimiento respecto donde se almacenan y desde dónde se recuperan los datos (base de datos, servidor, etc...) y notifica a la vista cuando los datos han cambiado.
- **Modelo de la vista:** Guarda el estado de la vista y contiene los métodos para gestionar las acciones del usuario. Se comunica con la capa Modelo para guardar y recuperar los datos.
- **Vista:** Muestra la interfaz del usuario. No contiene ninguna lógica de usuario, a cambio delega toda esta gestión al Modelo de la vista.

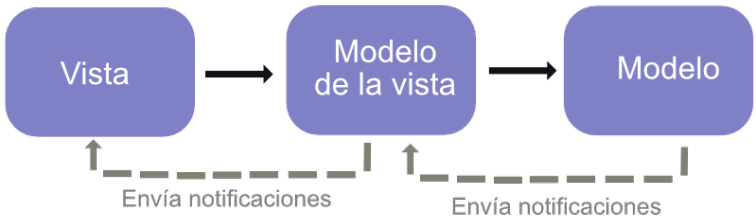


Figura 4.8: Esquema de la arquitectura MVVM

4.5 Secuencia captura de datos y predicción de la acción

En esta sección se describe el flujo de datos a través de las distintas capas de la aplicación para conseguir una predicción de la acción que está realizando el usuario ante el dispositivo móvil.

La figura 4.9 muestra el esquema total de la secuencia que se describe a continuación. La captura de datos se lleva a cabo en el ViewController que implementa el Delegate AVCaptureVideoDataOutputSampleBufferDelegate.

De este delegado se implementa el método captureOutput(output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection: AVCaptureConnection). Los delegados reciben este mensaje cada vez que la salida captura un nuevo frame de video.

El Objeto *sampleBuffer*, es el *frame* capturado y se envía inmediatamente al ViewModel para su procesamiento. El ViewModel en primer lugar crea un objeto de tipo *VNImageRequestHandler* que permite realizar operaciones de *machine learning* sobre una imagen pasada como un *buffer*.
A continuación sobre el objeto *VNImageRequestHandler* se ejecuta el método *perform(VNDetectHumanBodyPoseRequest)*, que detecta los diferentes cuerpos humanos en la imagen y devuelve como resultado una colección de *VNHumanBodyPoseObservation* que contiene los diferentes puntos claves de cada cuerpo detectado.

El objeto *VNHumanBodyPoseObservation* se almacena a continuación en un objeto de tipo *PlayStats*. Cada 5 segundos se comprueba si se ha alcanzado el número mínimo de 60 frames detectados para poder ejecutar una predicción, esta operación se lleva a cabo en este tiempo ya que Apple advierte que no es necesario realizar predicciones constantemente. Una vez ha transcurrido el tiempo mínimo entre predicciones y se han alcanzado 60 frames, el ViewModel solicita al objeto PlayStats que genere una predicción llamando al método getAction(). En este método primero se crea un objeto *PlayerActionClassifier*, este objeto toma el fichero del Modelo (*HouseTrainingActionClassifier.mlmodel*) de *Machine Learning* que contiene las acciones que la apliación puede detectar.

A continuación se preparan las observaciones almacenadas, esto es que por cada observación, se extraen las coordenadas de los puntos del cuerpo humano y los convierte en un formato compatible con Core ML. Estas coordenadas son almacenadas en un Multiarray y se insertan en el objeto *PlayerActionClassifier* anteriormente creado. Una vez se tienen preparados los datos, *PlayerStats*, solicita al *PlayerActionClassifer* una predicción pasándole como parámetro el multiArray anteriormente creado. Esta predicción devuelve como resultado un objeto *MLBatchProvider*; de este objeto se extraen las características necesarias para realizar una observación y se normalizan en un objeto *PlayerActionClassifierOutput*. Esta normalización hace que luego sea más fácil tratar los datos de probabilidades de la acción detectada respecto

a las acciones que pueden detectarse a través del modelo.

Finalmente, esta predicción se devuelve al *PlayerStats*, este extrae el *label* o nombre de la acción detectada (la que ha obtenido un mayor porcentaje de probabilidad) y su probabilidad, devolviendo estos datos al ViewModel como un objeto tipo Action. A continuación el *ActionType* pasa al *ViewController* el cual actualiza esta información para mostrársela al usuario.

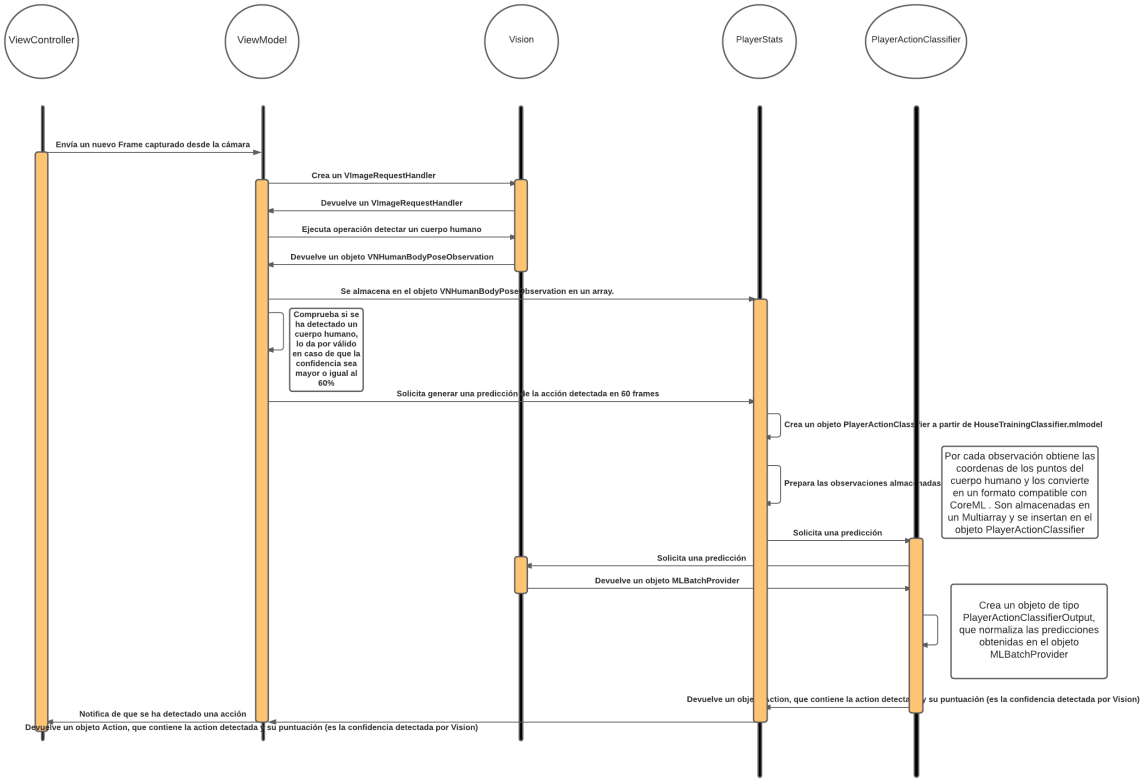


Figura 4.9: Diagrama de secuencia que ilustra el flujo de datos desde su captura hasta su procesamiento

Capítulo 5

Resultados

5.1 Introducción

Este capítulo se centra en la integración de los módulos y técnicas que conforman la aplicación a la vez que se muestran los resultados en la diferentes niveles de la propia aplicación. Se han descrito los métodos y técnicas que utiliza la aplicación a bajo nivel para poder predecir las acciones que está realizando el usuario delante del dispositivo.

En la figura 5.1 se muestra un esquema simplificado del flujo de datos en su tránsito por los módulos de la aplicación.

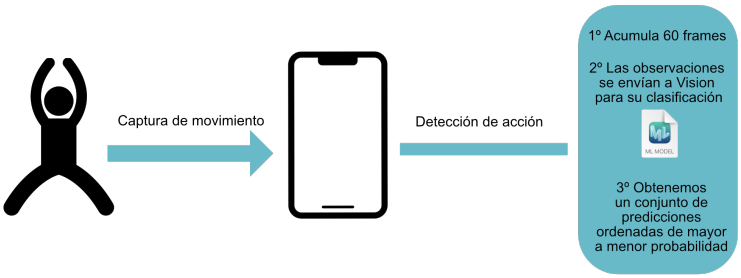
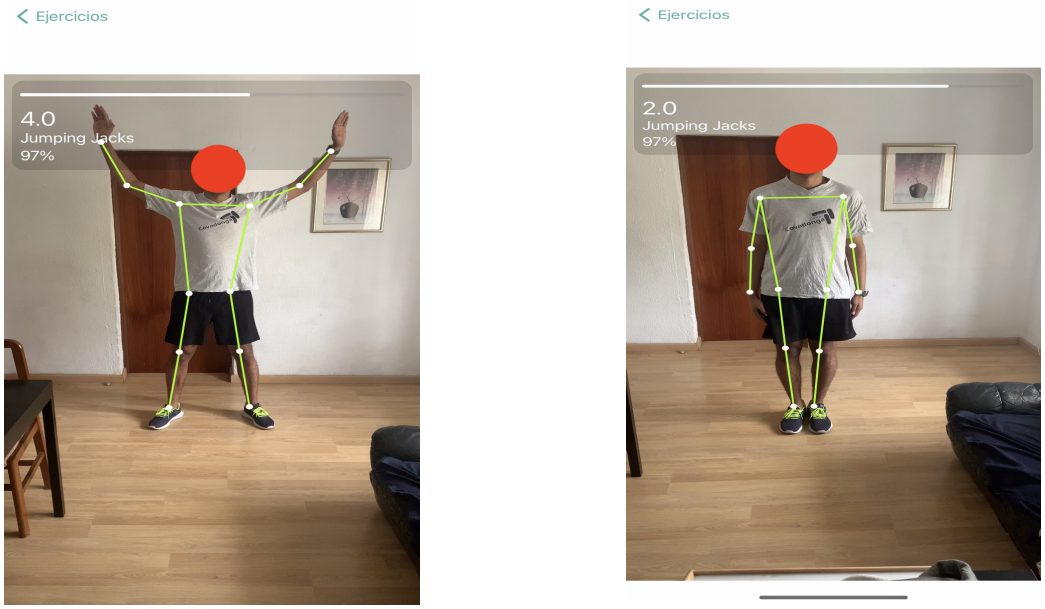


Figura 5.1: Esquema simplificado del funcionamiento de la aplicación

En la figura 5.2 se visualiza una secuencia de la actividad *Jumping Jacks* que se compone de 2 movimientos. En la secuencia se aprecian los datos que arroja la aplicación al analizar la imagen. Los puntos blancos representan los puntos de unión que se extraen de cada *frame* capturado por la aplicación.

Estos puntos característicos, como se ha indicado previamente, son: ojos, oídos, nariz, cuello, hombros, codos, muñecas, caderas, rodillas y tobillos.

En la figura 4.9 se describe cómo se obtienen estos puntos a partir de un objeto *VNHumanBodyPoseObservation*. Las líneas verdes son uniones entre las coordenadas arrojadas por cada punto. Esta figura se procesa por cada frame detectado y es independiente del procesamiento de detección de la actividad que se está realizando.
Además de las uniones, la aplicación muestra un contador que va disminuyendo a cada segundo y que indica el tiempo restante para acabar la actividad, la actividad detectada (en este caso *Jumping Jacks*) y la probabilidad o confidencia de la actividad.
De este último valor se extrae su media de las distintas observaciones que se hacen para mostrar más tarde como puntuación en la vista de resumen.



(a) Jumping Jack movimiento 1. (b) Jumping Jack movimiento 2.

Figura 5.2: Secuencia de la actividad Jumping Jack.

En la tabla 5.1 se visualiza un resumen de las pruebas llevadas a cabo con la aplicación para probar el modelo. Las pruebas han consistido en 30 sesiones de 30 segundos por cada actividad a lo largo de 5 días. Cada sección estaba enfocada en probar el modelo en base a unas condiciones. Condición

ideal, corresponde a la primera columna y consiste en realizar los ejercicios frente al dispositvo colocado en un lugar estable y con buena iluminación. En la segunda columna se llevaron a cabo las actividades pero con el dispositivo colocado en una posición lateral de la misma, con esta prueba se ha intentado comprobar qué tal ha sido la función de generación de datos artificiales ya que el modelo se ha entrenado con vídeos mayoritariamente captados de forma frontal. Por último tenemos las condiciones menos ideales, las pruebas consisiteron en captar al usuario pero realizando a la vez pequeños movimientos sobre el dispositivo para desestabilizar las imágenes.

Los resultados han sido bastante satisfactorios teniendo en cuenta que se trata de un modelo entrenado con lo básico, es decir, con el númeor de interacciones mínimas (80) y con vídeos de calidad regular. Las pruebas realizadas en condiciones optimas, arrojaron un resultado del 90 % de precisión en todas las actividades salvo en la actividad de Plank, cuyo campo de visión está limitado si se captura de forma frontal. Para esta actividad se obtienen mejores resultados desde una vista lateral.

Respecto a las pruebas realizadas con poca estabilidad, es notable la disminución de la precisión en aquellas actividades donde se realizan movimientos. Esto se debe entre otros inconvenientes a que las imágenes se están capturando cada vez en un ángulo diferente (el dispositivo se movia de lado a lado), llegando en ocasiones a difuminar la imagen capturada.

En actividades como *Plank* o *Wall Sit* el decremento de precisión ha sido menos acusado ya que es más fácil para el modelo lidiar con los inconvenientes de estabilidad al mantener el usuario la posición fija durante toda la captura de vídeo a diferencia de otras actividades como *Jumping Jacks* o *Sumo Squad* donde el usuario varía su posición durante todo el entrenamiento.

Actividad	Porcentaje de acierto entrenamiento frontal	Porcentaje de acierto entrenamiento lateral	Porcentaje de acierto entrenamiento baja estabilidad
Jumping Jacks	8/10	9/10	7/10
High Knees Run In Place	9/10	9/10	7/10
Plank	8/10	9/10	8/10
Sumo Squad	9/10	9/10	7/10
Wall Sit	9/10	9/10	7/10

Cuadro 5.1: Resultados de pruebas

5.2 Modelo

Modelo: creación y entrenamiento

Create ML[4] es un *framework* de *Machine Learning* desarrollado por Apple para la creación de modelos de *Machine Learning*. Create ML permite a los desarolladores crear y entrenar modelos de Machine Learning para reconocer imágenes, detectar objetos, clasificar textos, sonidos, actividades y acciones entre otros. Crear un modelo de *Machine Learning* es fácil gracias a que Apple provee una aplicación bajo el nombre de Create ML que permite crear y entrenar los modelos.

En la figura 5.3 se muestran los distintos modelos que se pueden crear.

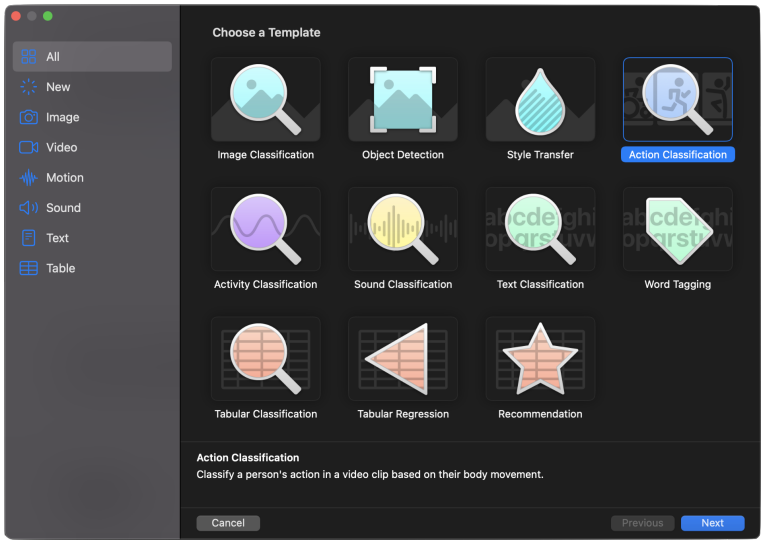


Figura 5.3: Vista de inicio de Create ML.

5.3 Generación de datos artificiales - Data augmentation

Es una técnica que permite incrementar la cantidad de datos para el entrenamiento del modelo a partir de la aplicación de transormaciones sobre

los datos existentes[22], entre las distintas técnicas existentes el proyecto ha aplicado las siguientes técnicas:

1. **Dar la vuelta**: girar la imagen horizontal o verticalmente.
2. **Rotación**: Al igual que la técnica anterior se gira la imagen en el eje horizontal o vertical, pero se diferencia en que la transformación preserva las dimensiones originales de la imagen.
3. **Escalado**: Esta técnica cambia el tamaño de la imagen original, convirtiéndola en una imagen de mayor tamaño.
4. **Corte**: Consiste en tomar una sección escogida aleatoriamente de la imagen.
5. **Traslado**: Mueve la imagen a lo largo de los ejes X e Y. De esta forma se consigue que durante el entrenamiento las redes neuronales observen todo el espacio de la imagen.
6. **Ruido Gaussiano**: Esta técnica distorsiona las características de alta frecuencia de una imagen. De esta forma se intenta que las redes neuronales aprendan de otras características y así evitar un sobreajuste durante el aprendizaje de determinadas características.

5.4 Modelo: entrenamiento, validación y salida

5.4.1 Creación del Modelo

El entrenamiento de un modelo de Machine Learning es bastante simple, basta con reunir los datos para entrenar y testear el modelo, en nuestro caso los datos son vídeos.
CreateML necesita cincuenta o más vídeos por cada acción a detectar para entrenar un modelo. Además es recomendable, tal y como indican los creadores de Create ML, incluir una última categoría de acciones que no espera que haga el usuario para que en caso de no poder conseguir detectar la acción que está realizando el usuario, seleccione esta. Por ejemplo, la app se encarga de detectar ejercicios, como *Jumping Jacks*, *Plank*, etc...
El usuario puede, durante el entrenamiento, quedarse sin energías o surgirle la necesidad de parar, por ello se ha incluido una categoría extra donde

aparecen personas, caminando, descansando, de pie, etc... De esta forma el modelo utilizará esta categoría como una acción de respaldo, donde si no consigue detectar alguna de las acciones esperadas, el modelo devolverá esta acción.

Para la creación del modelo de la aplicación se utiliza un modelo de plantilla denominado, *Action Classification*. Esta plantilla contiene un modelo preentrenado que extrae determinadas características de los cuerpos que detecta. Con el entrenamiento que aplica Create ML, al pasarle los vídeos, lo que hace es especializarlo en la detección de actividades concretas a través de una metodología, transferencia de aprendizaje[23].

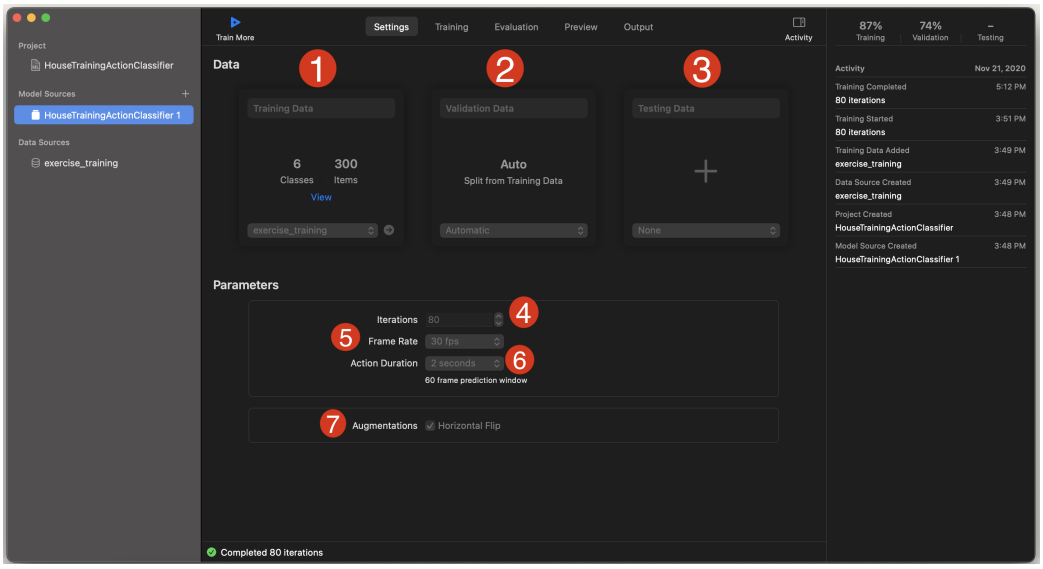


Figura 5.4: Vista ajustes de entrenamiento en Create ML.

Una vez se han introducido los ajustes iniciales del modelo (nombre del modelo, autor, etc...), la siguiente pantalla, mostrada en la figura 4.4, que se visualiza, corresponde a ajustes del Modelo.

1. ***Training data:*** Donde se introducen los datos para entrenar el modelo. En nuestro caso al tratarse de vídeos tiene que ser una carpeta que a su vez contenga las subcarpetas necesarias, cada una con los vídeos de las acciones a detectar. Create ML utilizará el nombre de cada una de las subcarpetas como el nombre de las categorías de las acciones.

2. ***Validation data***: En esta sección se pueden introducir más datos para ajustar el modelo creado a partir de los datos de entrenamiento. Es útil para ajustar los parámetros como por ejemplo elegir las unidades ocultas en una red neuronal. Si no se dispone de más datos, Create ML separa automáticamente el 5 % de los datos del apartado de Training data para utilizarlos como Validation data.

3. ***Test data***: Datos que CreateML utiliza únicamente para evaluar el rendimiento del modelo y que no pertenecen a ninguno de los subconjuntos anteriores.

4. ***Iterations***: Ajuste de iteraciones, es decir, el número de veces que queremos que Creat ML utiliza para entrenar el modelo.

5. ***Frame Rate***: Especificación de la frecuencia de grabación de los vídeos, es decir el número de *frames* por segundo.

6. ***Action Duration***: Se indica el tiempo medio que dura cada acción del modelo. A partir de este parámetro Create ML posteriormente realiza una estimación del número de *frames* que necesita para proporcionar una predicción.

7. ***Augmentations***: Selecciona si se quiere aplicar la técnica de generación de datos artificiales (explicada en el punto 5.3).

5.4.2 Vista entrenamiento

Una vez Create ML ha terminado de entrenar el modelo, en la pestaña de entrenamiento (*Training*), se puede visualizar la evolución de la precisión del modelo a lo largo de las iteracciones.

La gráfica se separa en 2 partes, *Training Accuracy* y *Validation Accuracy*. Ambas describen la calidad del modelo a la hora de hacer las predicciones, pero es la medida obtenida en Validation Accuracy la más interesante ya que esta nos indica cómo han ido las predicciones del modelo basadas en datos que no había visto antes.

Tal y como se muestra en la figura 5.5 esta fase de entrenamiento se ha realizado con 80 iteraciones observándose cómo tanto las curvas de Training

accuracy como la Validation accuracy evolucionan hacia la tendencia correcta, esto es, hacia el 100 %, si bien los resultados finales han sido del 87.1 % y 73.8 % respectivamente. En este caso no se ha conseguido el resultado deseable.

Esto es, con casi absoluta seguridad, debido al número de muestras utilizadas para el entrenamiento. Es muy probable que el incremento de muestras llevaría a una mejora de los resultados, si bien de cara al presente trabajo esto no constituye una cuestión prioritaria, ya que el objetivo fundamental es la integración de la integración de todas las funcionalidades para conseguir la integración propuesta que da lugar a la aplicación desarrollada"

Figura 5.5: Vista de entrenamiento

5.4.3 Evaluación

En la figura 5.6 se muestra una vista relativa a la evaluación donde se puede observar de forma resumida las clases o acciones del modelo propuesto, así como las métricas de precisión y recuperación[24]. Estas métricas juntas describen la compensación entre aplicar incorrectamente una etiqueta con demasiada libertad y falta de ejemplos de esa etiqueta.

- La precisión describe la eficacia del modelo para aplicar una etiqueta solo cuando es apropiado para una categoría determinada (pocos falsos positivos).

- Recall describe cuán efectivo fue el modelo para encontrar todos los ejemplos relevantes de una categoría (pocos falsos negativos).

En la figura 5.6, se observan las clases generadas a partir de las subcarpetas que contenían los vídeos con las actividades a observar. La precisión para las categorías high_knees_run_in_place y sumo_squat han tenido un resultado bastante mejorable, lo cual indica que se necesitan más vídeos para mejorar el entrenamiento o que los vídeos que se han utilizado no son de buena calidad. Sin embargo el modelo arroja mejores resultados en los falsos negativos, ya que las actividades rondan entre el 90-100 %.

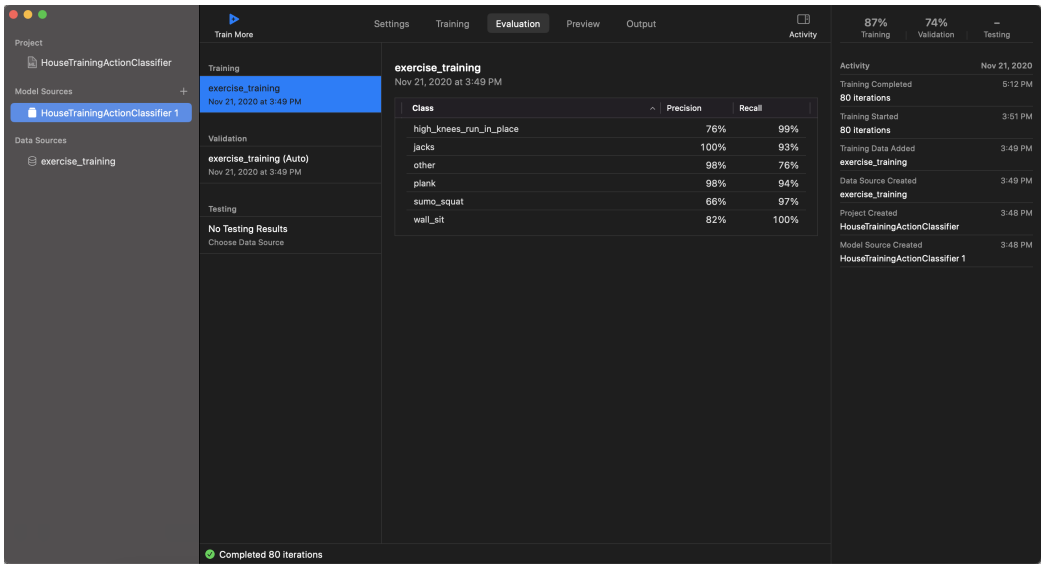


Figura 5.6: Vista de Evaluación

5.4.4 Output

En esta vista se puede ver un resumen del modelo creado. Los detalles que se muestran son, el tipo de modelo, el tamaño, los sistemas operativos para los que se encuentra disponible, las acciones capaces de detectar, la ventana

de predicción necesaria y el número de iteracciones realizadas para entrenar el modelo.

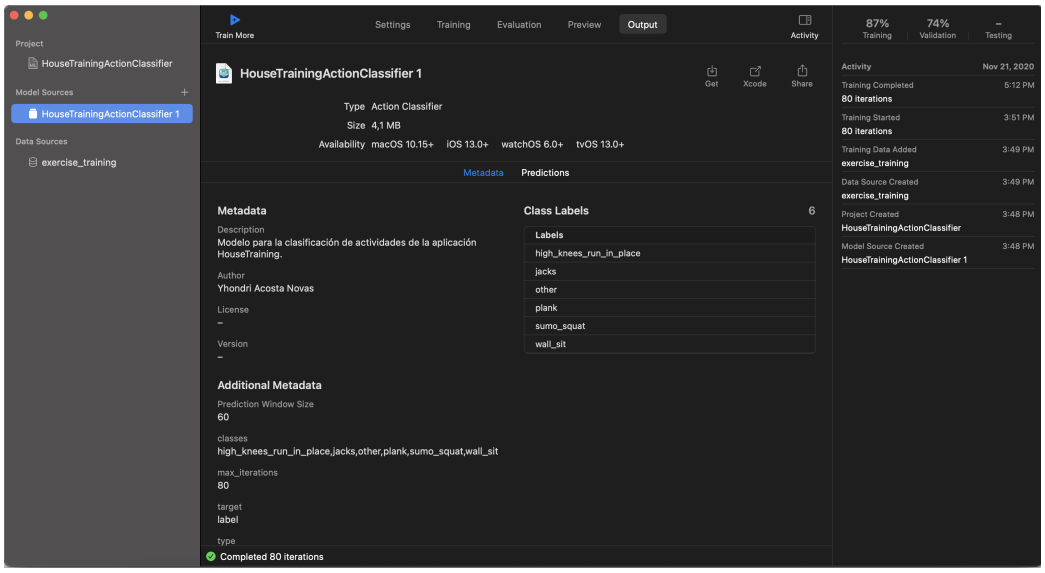


Figura 5.7: Vista de Salida

5.5 Interfaz de la aplicación

A continuación se describen los diferentes módulos de la interfaz de la aplicación. Concretamente la parte gráfica, el entrenamiento, la creación de la rutina, el resumen de la actividad y otras consideraciones de interés.

5.5.1 Gráficos

Al abrir la aplicación si existe información, ésta se visualiza en dos gráficas, una de ellas son los datos respecto al tiempo dedicado a realizar ejercicio con la aplicación durante la semana actual y la otra durante los meses del año actual.

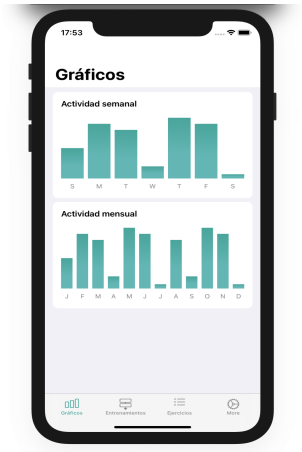


Figura 5.8: Gráficos

5.5.2 Entrenamientos

Si el usuario tienen entrenamientos creados, es decir, una lista de ejercicios/actividades a realizar, estas se muestran en este módulo ordenadas por el nombre asignado durante su creación.

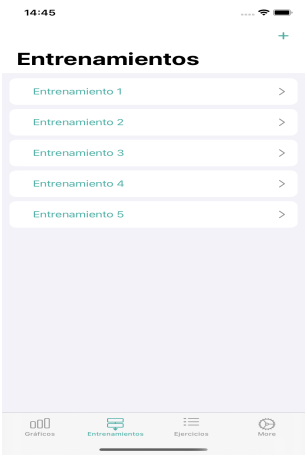


Figura 5.9: Entrenamientos

En la figura 5.9 se muestra el listado de entrenamientos con varios entrenamientos creados. En la parte superior, se pueden crear nuevos entrenamientos, como se muestra en la sección a continuación

5.5.3 Crear rutina

En la vista entrenamiento, al pulsar el botón en la parte superior -", se abrirá el módulo de Crear rutina, que primero permite elegir los ejercicios que se quieran añadir, luego da la oportunidad de cambiar el orden en el que el usuario quiera realizar los ejercicios y por último pedirá dar un nombre a la rutina.

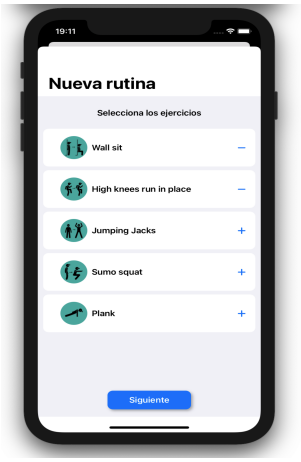


Figura 5.10: Vista selección de ejercicios para la creación de una nueva rutina

5.5.4 Resumen

Tras realizar un entrenamiento, se muestra un resumen del entrenamiento. En esta vista pueden ver estadísticas del tiempo empleado y del desempeño del entrenamiento desglosado por cada ejercicio realizado.

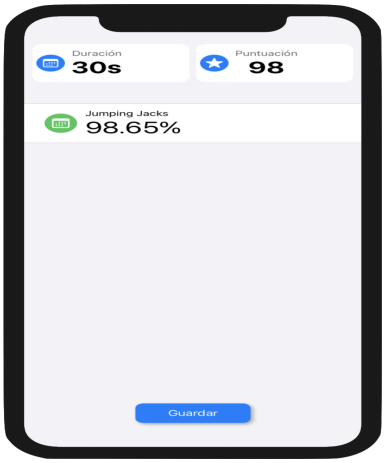
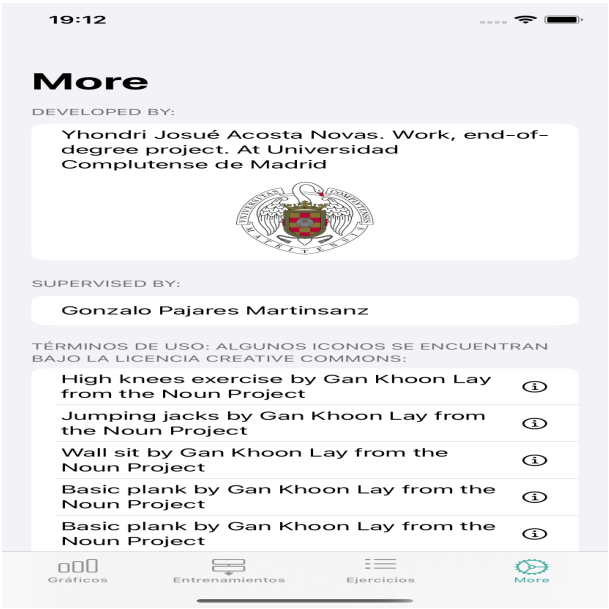


Figura 5.11: Vista de resumen tras realizar un entrenamiento

5.5.5 Más

En esta vista, se muestran las credenciales del proyecto. Aquí también se listan y enlazan la fuente de los recursos como iconoes imágenes creados por diseñadores y utilizados bajo la licencia *Creative Commons*.



Capítulo 6

Conclusiones y trabajo futuro

6.1 Conclusiones

En este proyecto se ha desarrollado una aplicación que detecta acciones humanas a partir de un vídeo en directo utilizando técnicas específicas de aprendizaje profundo. Este tipo de tecnologías hasta hace poco tiempo estaban relegadas a dispositivos de gran tamaño que utilizaban además un dispositivo específico para poder realizar el trabajo de predicción de acciones. Con este proyecto se ha podido comprobar que esto ya es posible llevar a cabo en dispositivos móviles gracias al avance en la investigación de Aprendizaje Automático, Redes neuronales y el desarrollo de chips específicos como el Neural Engine[25] para llevar a cabo tareas de Aprendizaje Automático.

Para poder desarrollar el reconocimiento de acciones se han aplicado técnicas de Aprendizaje Automático, utilizando el paradigma ResNet. Este modelo se ha creado a partir de un modelo preentrenado de Core ML uno de los *frameworks* de iOS y macOS que ha sido entrenada para detectar las acciones concretas que requería la aplicación.

Esta metodología se conoce como transferencia de aprendizaje[23], y consiste en partir de un modelo ya entrenado y especializarlo utilizando datos de entrenamientos limitados. En el caso del proyecto ha sido utilizado para crear un modelo que es capaz detectar 5 acciones: *High knees run in place, Jumping Jacks, Plank, Sumo Squat y Wall Sit.*

Todas estas acciones se detectan directamente en el dispositivo sin nece-

sidad de enviar los datos a un servidor externo, lo que permite a la aplicación mantener la privacidad del usuario al no salir las grabaciones de vídeo del dispositivo.

6.2 Trabajo futuro

De cara al futuro se plantea la implementación de una serie de mejoras, algunas de las identificadas en el momento actual son las que se listan a continuación:

1. Aumentar la precisión de detección de acciones. Para ello se plantea reentrenar el modelo modificando los parámetros de entrenamientos como número de iteraccciones, tiempo estimado de realización de las acciones a detectar, etc... Para llevar a cabo esta tarea, también será necesario obtener mejores datos de entrenamiento, es decir, vídeos en mejor calidad y más variedad.

2. Aumentar el número de acciones a detectar. Para poder realizar esta propuesta será necesario obtener una mayor cantidad de vídeos de otras acciones aún no soportadas por el modelo entrenado.

3. Dar soporte a la orientación horizontal. Permitir que el usuario pueda cambiar la orientación en la que realiza un entrenamiento, además de notificarle cuál es la orientación recomendada para cada ejercicio.

4. Entrenamiento en el dispositivo. Permitir que los usuarios puedan crear sus propios entrenamientos, de esta forma la aplicación será capaz de aprender de nuevas acciones sin tener que recrear el modelo y compilar la aplicación de nuevo. Core ML permite el entrenamiento directamente en el dispositivo[26]. Para ello habrá que crear un módulo que guíe al usuario de como crear nuevos ejercicios y entrenar el modelo.

5. Consejos a la hora de hacer ejercicio: Es necesario contar con expertos en los ejercicios implementados que indiquen la forma correcta de realizar un ejercicio, como por ejemplo indicaciones del ángulo en el que colocar un brazo o las piernas. Core ML/Vision proporciona las coordenas de cada uno de los puntos destacados del cuerpo humano detectado,

por lo que una posible solución sería comprobar que se están colocando correctamente en los ángulos correctos los brazos, piernas y otras partes del cuerpo.

Capítulo 7

Conclusions and future work

7.1 Conclusions

In this project, an application has been developed that detects human actions from a live video using specific deep learning techniques. In the past, this type of technology was relegated to large devices that also used a specific device to perform the job of predicting actions. With this project it has been possible to verify that this is already possible to be carried out on mobile devices thanks to the advance in the investigation of Machine Learning, Neural Networks and the development of specific chips where Machine Learning tasks can be carried out such as the Neural Engine [25].

In order to develop action recognition tasks, Machine Learning techniques have been applied, using the ResNet paradigm. This model has been created from a pre-trained Core ML model, one of the iOS and macOS frameworks that has been trained to detect the specific actions required by the application.

This methodology is known as Transfer Learning [23], and consists of starting from an already trained model and specializing it using limited training data. In the case of the project, it has been used to create a model that is capable of detecting 5 actions: High knees run in place, Jumping Jacks, Plank, Sumo Squat y Wall Sit.

All these actions are detected directly on the device without the need to send the data to an external server, which allows the application to maintain

user privacy by not leaving the video recordings from the device.

7.2 Future work

Looking ahead, the implementation of several improvements is proposed, some of those identified at the present time are those listed below:

1. Increase the accuracy of action detection. To do this, it is proposed to retrain the model by modifying the training parameters such as number of iterations, estimated time for performing the actions to be detected, etc ... To carry out this task, it will also be necessary to obtain better training data, that is, videos in better quality and more variety.

2. Increase the number of actions to detect. In order to carry out this proposal, it will be necessary to obtain a greater number of videos of other actions not yet supported by the trained model.

3. Implement horizontal orientation. Allow the user to change the orientation in which they perform a workout, as well as notify them of the recommended orientation for each exercise.

4. Training on the device. Allow users to create their own workout, in this way the application will be able to learn new actions without having to recreate the model and compile the application again. Core ML enables training directly on the device [26]. To do this, a module will have to be created that guides the user on how to create new exercises and train the model.

5. Tips. It is necessary to have experts in the implemented exercises that indicate the correct way to perform an exercise, such as indications of the angle at which to place an arm or legs. Core ML / Vision provides the coordinates for each of the detected human body highlights, so a possible solution would be to check that the arms, legs, and other body parts are being positioned correctly at the correct angles.

Bibliografía

[1] Apple. *World Wide Developers*. URL: https://developer.apple.com/wwdc/.

[2] Asadi-Aghbolaghi M., Clapés A., Bellantonio M., Escalante H.J., Ponce-López V., Baró X., Guyon I., Kasaei S. y Escalera S. *Deep Learning for Action and Gesture Recognition in Image Sequences: A Survey. Gesture Recognition. pp. 539-578.* http://www.deeplearningbook.org. Springer Verlag, 2017.

[3] Apple. *Apple Developer Documentation*. URL: https://developer.apple.com/documentation/.

[4] Apple. *CreateML*. URL: https://developer.apple.com/documentation/createml.

[5] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Numerentur. *Redes Neuronales Artificiales*. URL: http://numerentur.org/redes-neuronales-artificiales/.

[7] Pajares G., Herrera P.J. y Besada E. *Red Neuronal Convolucional*. RC-Libros, 2021.

[8] Wikipedia. *ReLu*. URL: https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[9] Azel Daniel. *Pool Overlapping*. URL: https://towardsdatascience.com/understanding-alexnet-a-detailed-walkthrough-20cd68a490aa.

[10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever y R. R Salakhutdinov. *Improving neural networks arXiv 1207.0580*. preventing co-adaption of feature detectors., 2012.

[11] Ioffe S. y Szegedy C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167v3.* URL: https://en.wikipedia.org/wiki/Recurrent_neural_network.

[12] Chris Bishop. *Pattern Recognition and Machine Learning.* C.M., 2016.

[13] Sebastian Ruder. "An overview of gradient descent optimization algorithms". En: (sep. de 2016).

[14] K. He, X. Zhang, S. Ren y J. Sun. "Deep Residual Learning for Image Recognition". En: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016, págs. 770-778. DOI: 10.1109/CVPR.2016.90.

[15] Lutz Roeder. *Netron.* URL: https://github.com/lutzroeder/netron.

[16] Apple. *Swift.* URL: https://developer.apple.com/swift/.

[17] Chris Lattner y Apple. *Swift.* URL: https://swiftbycoding.dev/el-lenguaje-de-programacion-swift/.

[18] Apple. *Swift.* URL: https://developer.apple.com/swift/ui.

[19] Apple. *UIKit.* URL: https://developer.apple.com/swift/uikit.

[20] Apple. *Core Data.* URL: https://developer.apple.com/documentation/coredata.

[21] Carlo Russo. *KnockoutJS Blueprints.* 2015.

[22] Arun Gandhi. *Generación de datos artificiales.* URL: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

[23] Jacopo Mangiavacchi. *Transferencia de aprendizaje.* URL: https://medium.com/@JMangia/coreml-on-device-training-with-transfer-learning-from-swift-for-tensorflow-models-b66004eb3068.

[24] Apple. *Precision y Recall.* URL: https://developer.apple.com/documentation/createml/mlclassifiermetrics/3005453-precisionrecalln.

[25] Hollance. *Neural Engine.* URL: https://github.com/hollance/neural-engine.

[26] Apple. *Entrenamiento en el dispositivo.* URL: https://developer.apple.com/videos/play/wwdc2019/704/.

[27] Apple. *TestFlight.* URL: https://developer.apple.com/testflight/.

Capítulo 8

Anexo

8.0.1 Instalación de la aplicación:

A continuación se explican dos formas de probar la aplicación en un dispositivo físico. Primero la instalación a través de TestFLight y después la instalación manual vía Xcode:

Instalación vía TestFlight

TestFlight[27] es una aplicación desarrollada por Apple que permite a los desarrolladores publicar versiones de prueba de sus aplicaciones antes de ponerla a disposición de todos los usuarios en la App Store.

1. Descarga de TestFlight: Para poder instalar la aplicación de este proyecto, es necesario dirigirse a la App Store desde tu dispositivo iOS (disponible por el momento sólo para iPhone) y descargarse la aplicación TestFlight que se puede encontrar en el siguiente enlace: https://apps.apple.com/es/app/testflight/id899247664

2. Una vez descargada la aplicación TestFlight, en el mismo dispositivo abre el siguiente enlace https://testflight.apple.com/join/2aS0Nv8P que dará acceso a la aplicación a través de TestFlight.

3. Accede a TestFlight y a continuación a la sección de la aplicación, por último pulsa el botón instalar.

4. Cuando haya terminado la instalación el botón Descargar habrá cambiado a Abrir. Pulsando sobre la opción Abrir, cambiará Testflight por la aplicación instalada.

Con estos sencillos pasos se instala la aplicación. TestFlight además permite enviar comentarios e imágenes a los desarrolladores, lo que ayuda a la resolución de errores o la mejora de la aplicación gracias a las sugerencias de los usuarios.

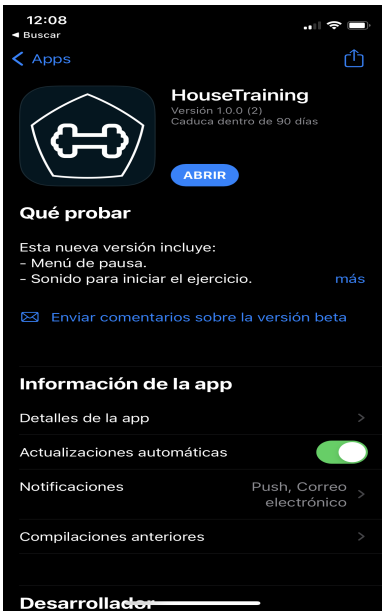


Figura 8.1: La aplicación descargada y lista para abrir desde TestFlight

Instalación vía Xcode

Para instalar la aplicación es necesario un Mac capaz de ejecutar Xcode 12 (o versiones superiores). Para poder acceder a un entrenamiento es necesario disponer de un dispositivo iOS (móvil) físico capaz de ejecutar iOS 14+, ya que los simuladores que vienen con Xcode no pueden ejecutar vistas que acceden a la cámara. Además es posible que necesites una cuenta de Apple para poder firmar la aplicación e instalarla en un dispositivo físico.

1. Descarga de Xcode: Se puede descargar e instalar Xcode desde la aplicación Mac App Store. La Mac App Store con Xcode se puede abrir

directamente desde el siguiente enlace:
`https://apps.apple.com/es/app/xcode/id497799835?l=en&mt=12`

2. Descarga del proyecto: La descarga del proyecto se puede realizar utilizando git desde Terminar o directamente desde Github en el siguiente enlace:
`https://github.com/yhondri/HouseTraining`

3. Apertura del proyecto: Una vez descargado el proyecto, accedemos al directorio donde lo hemos descargado y descomprimido. Hacemoz doble clic en el archivo HouseTraining.xcodeproj

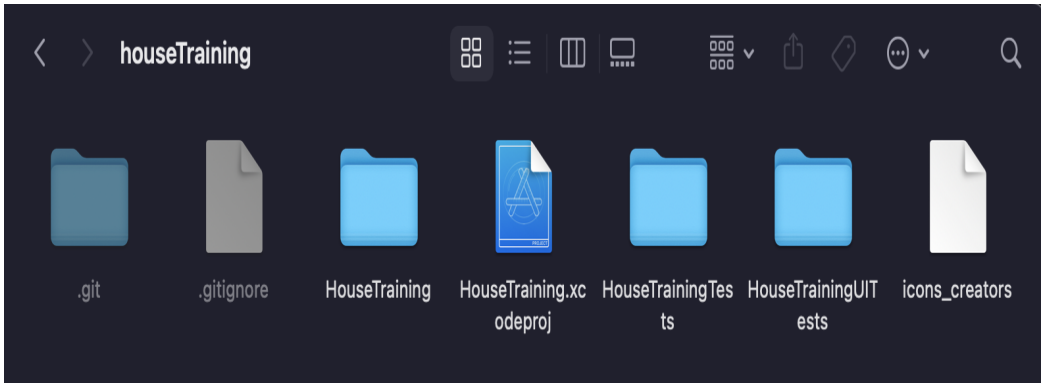


Figura 8.2

4. Ejecución de la aplicación: 1º Elegimos el dispositivo o simulador en el que queremos ejecutar la aplicación. 2º Hacemos clic sobre el botón de ejecución o mediante un atajo pulsando las teclas CMD (command) + R.

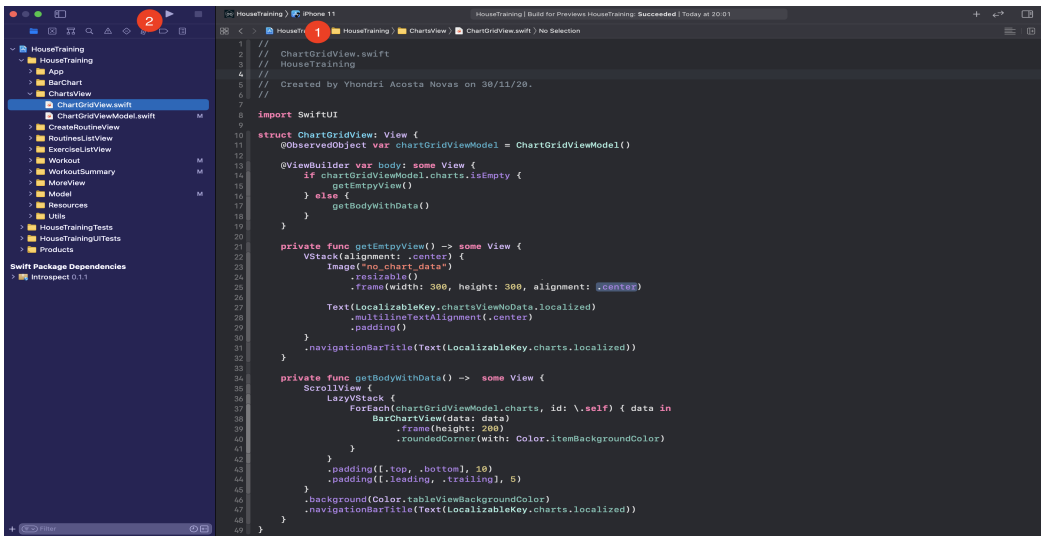


Figura 8.3

5. Problemas para ejecutar en un dispositivo físico: Para este paso necesitarás tener una cuenta de desarrollador de Apple y deberás haber registrado la cuenta en Xcode. A continuación selecciona el Workspace (paso 1) y por último selecciona tu cuenta de desarrollador de Apple (paso 2).

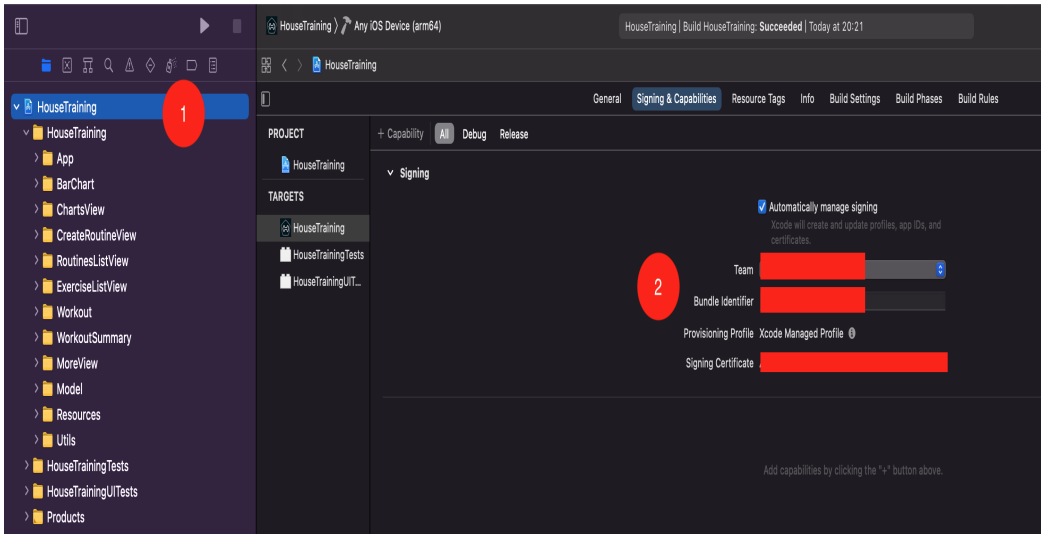


Figura 8.4

Tras la primera ejecución (en este caso se ilustra desde un simulador), visualizaremos la aplicación en el dispositivo seleccionado.

Figura 8.5

8.0.2 Manual de usuario

Módulo gráficos

En este módulo podemos observar el tiempo dedicado a entrenar. Si pulsamos sobre las barritas de alguno de los gráficos nos indicará numéricamente el tiempo dedicado ese día o mes.

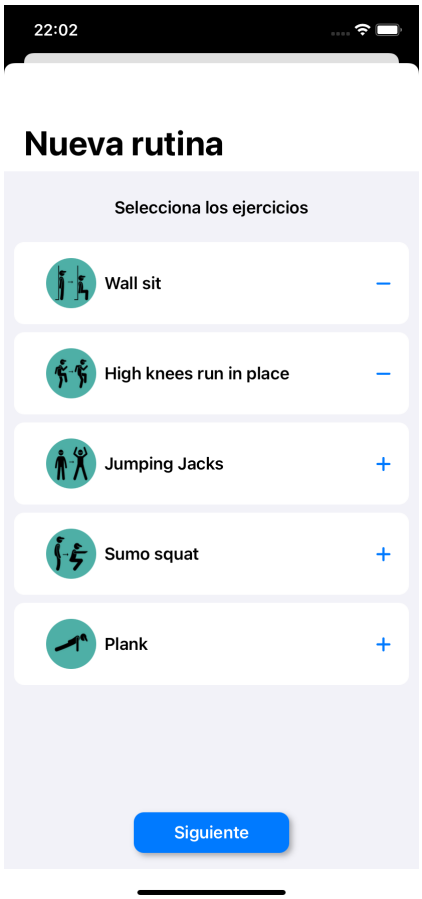
(a) Primera apertura - Sin datos.

(b) Tras realizar algún entrenamiento.

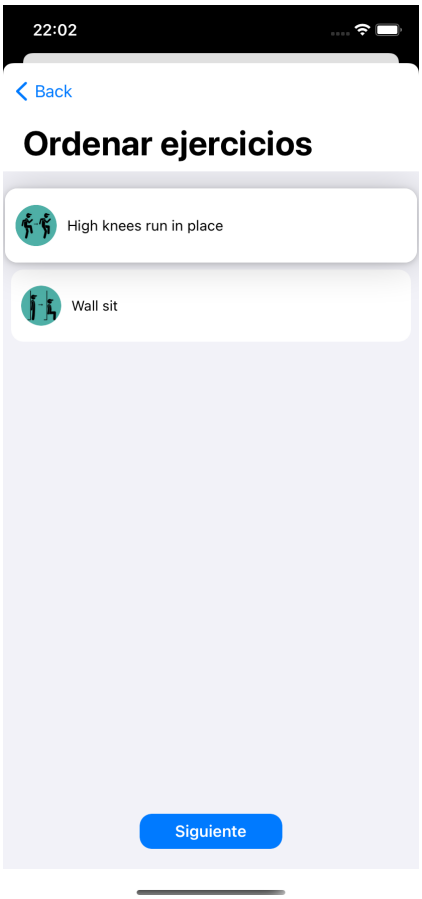
Figura 8.6: Módulo gráficos.

Crear rutina

Para crear una rutina es necesario ir a la vista de entrenamiento y pulsar en el botón -"de la esquina superior derecha. En la pantalla de ordenación, debes seleccionar durante 1 segundo el ejercicio que quieres ordenar y a continuación cuando flote moverlo a la posición deseada.



(a) Añade -.º quita ejercicios de la rutina ".

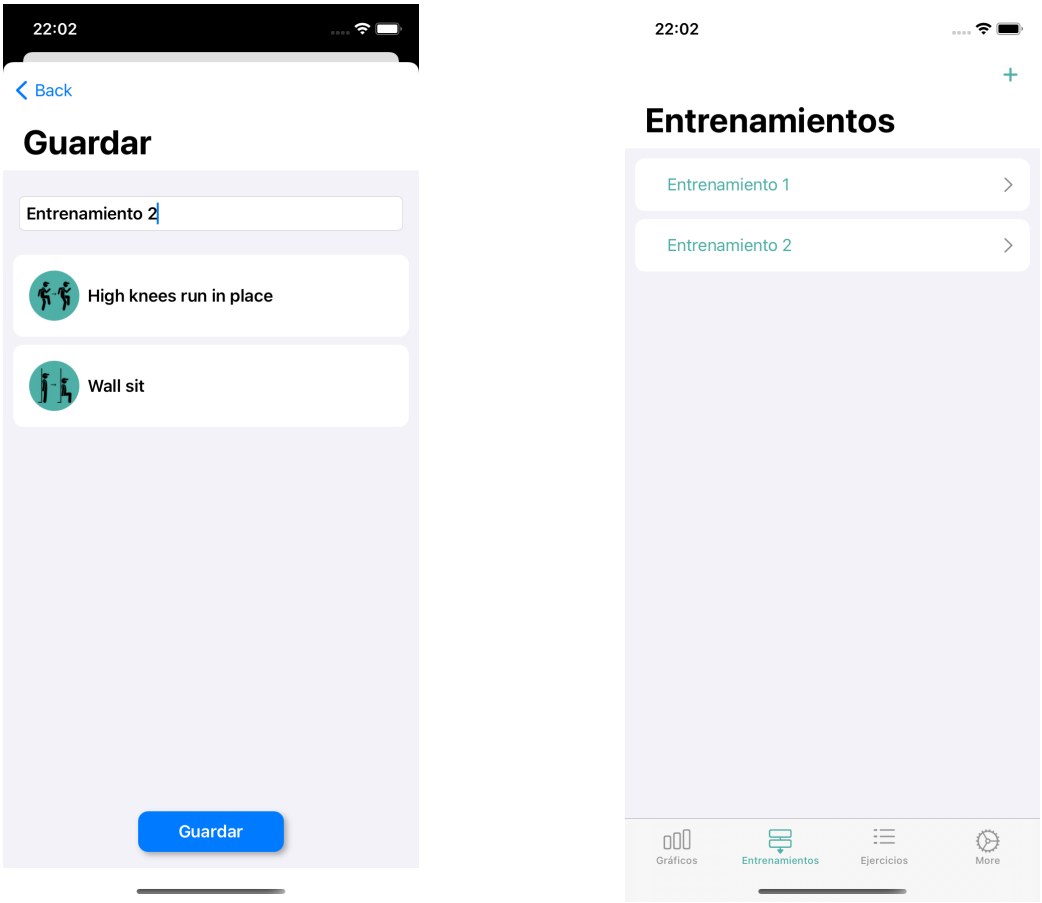


(b) Reordena los ejercicios mediante pulsación larga.

Figura 8.7: Creación de rutina.

A continuación tienes que dar un nombre a la rutina y tras esto se habilitará el botón guardar. Pulsa el botón guardar y se cerrará el módulo de

Crear rutina tras lo cual visualizarás la rutina creada.



(a) Dar nombre a la rutina y guardar.

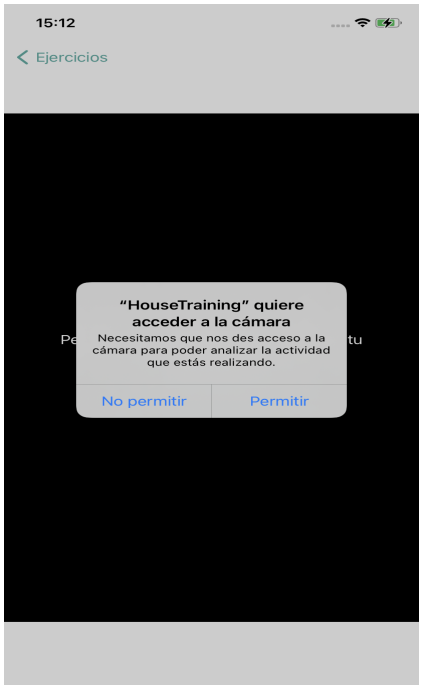
(b) Visualizar rutina creada.

Figura 8.8: Creación de rutina.

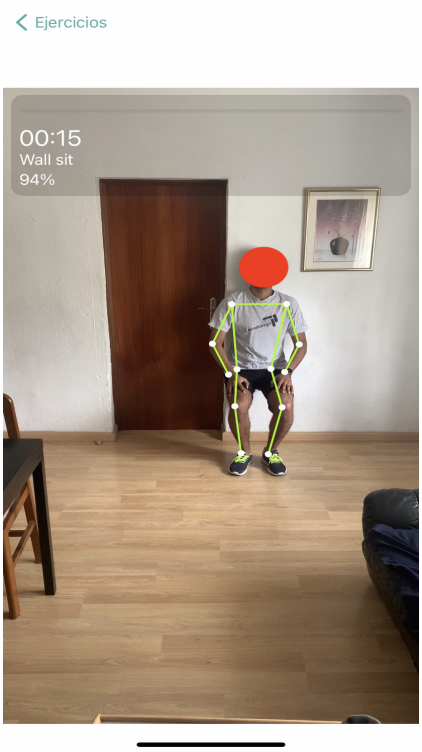
Realizar entrenamiento

Desde la vista Entrenamientos, selecciona el entrenamiento a realizar, si es la primera vez que realizas un entrenamiento, debes dar permiso de acceso a la cámara. A continuación si se detecta un cuerpo delante de la cámara, se señalarán los puntos claves que utiliza la aplicación para detección. Para iniciar un entrenamiento es necesario tocar la pantalla. Si se desea pausar el entrenamiento toca otra vez la pantalla. Para reanudar, vuelve a tocar la pantalla.

En la aplicación del proyecto se ha creado el modelo para detectar 5 acciones que son High knees, run in place, Jumping Jacks, Plank y Sumo Squat.



(a) Concede permiso para acceder a la cámara.



(b) Realiza los ejercicios de la rutina.

Figura 8.9: Realización de entrenamiento.

Resumen

Tras terminar el entrenamiento, se muestra de forma resumida qué tal ha ido el entrenamiento. Al pulsar el botón Guardar, se cerrará el módulo de entrenamiento y visualizaremos el litado de entrenamientos. Si tras esto te diriges al módulo de gráficos, verás datos.

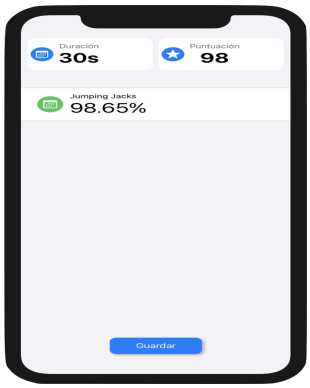


Figura 8.10: Vista de resumen tras realizar un entrenamiento

8.0.3 Más

Pulsa sobre cualquier celda con el icono ï"para visualizar la página de donde se ha obtenido el icono mencionado.

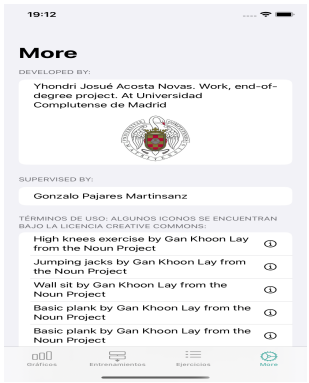


Figura 8.11