TRABAJO FIN DE GRADO

Grado en Ingeniería Informática


UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA


# Reconocimiento de caídas con Visión Artificial


Curso 2020/2021


**Alumno/a:**
Raquel Gutiérrez García

**Director/es:**
Pilar Martínez Ortigosa
Savíns Puertas Martín

Universidad de Almería

# Reconocimiento de caídas con Visión Artificial


Raquel Gutiérrez García

Grado en Ingeniería Informática

Promoción 2017-2021


Mayo 2021


Tutora

Pilar Martínez Ortigosa

Cotutor

Savíns Puertas Martín

# Agradecimientos

Gracias a todos mis seres queridos por apoyarme en cada decisión y por confiar en mí, pues ha sido gracias a su paciencia y su apoyo que cada día estoy un paso más cerca de cumplir mis sueños.

En especial, gracias a mis padres por orientarme y hacerme saber que soy merecedora de sus esfuerzos y expectativas y por hacerme sentir segura de mí misma, a mis hermanas por traer esa radiante y tranquilizadora luz a mi vida capaz de sanar cualquier herida, a mi pareja, por demostrarme que a su lado todo es posible y a mi amigo, por ser la mejor compañía que podría haber tenido durante mis años de estudio.

Gracias también a todos los profesores que han formado las bases necesarias para poder desarrollar este TFG, y, sobre todo, a mis directores de proyecto.

# Siglas

| | |
|---|---|
| **FC** | Totalmente conectado |
| **FN** | Falso negativo |
| **FP** | Falso positivo |
| **FPS** | Fotogramas por segundo |
| **HSC** | Clasificador de Estado Humano |
| **IA** | Inteligencia Artificial |
| **ID** | Identificación |
| **IDE** | Entorno de Desarrollo Integrado |
| **KNN** | K-Vecinos más cercanos |
| **LHMM** | Modelo de Markov oculto por capas |
| **MLP** | Perceptrón Multicapa |
| **MP** | Megapíxeles |
| **OMS** | Organización Mundial de la Salud |
| **RAM** | Memoria de acceso aleatorio |
| **RVM** | Máquina de vectores de relevancia |
| **SSL** | Capa de conexión segura |
| **TIC** | Tecnologías de la información y comunicación |
| **VP** | Verdadero positivo |
| **VPN** | Red privada virtual |

# Listado de ilustraciones

# Listado de tablas



# Listado de formulas

# ÍNDICE

Capítulo 1

# Introducción

Se quiere desarrollar un detector de caídas usando visión artificial y aprendizaje computacional, haciendo uso de las cámaras y una unidad de procesamiento del Smart Home de la Universidad de Almería. Es importante conseguir que el sistema funcione con el menor retardo posible.

## 1. Justificación

**Las caídas son la segunda causa mundial de muertes** por lesiones accidentales o no intencionales. La OMS calcula que anualmente mueren en todo el mundo unas 646.000 personas debido a las caídas. Son los mayores de 65 años quienes más sufren caídas mortales. Al año hay 37,3 millones de caídas cuya gravedad requiere atención médica. [1].

Cuando estamos en casa, nos sentimos seguros, ya que es nuestra zona de confort, pero a veces ocurren accidentes que perturban la tranquilidad de nuestro hogar, como tropiezos accidentales, desmayos, ataques al corazón, u otros problemas que pueden dar lugar a una peligrosa caída. Accidentes que ocurren con frecuencia en personas con movilidad reducida, con problemas de salud o de edad avanzada. Por ello, se quiere desarrollar un sistema inteligente que vele por su seguridad, percatándose de los accidentes y tomando medidas al respecto, avisando a un familiar, persona encargada o atención sanitaria en caso de emergencia.

Gracias a la rápida actuación, se podrán tratar las personas afectadas con mayor velocidad evitando que los problemas se agraven, **pudiendo incluso, salvar vidas**.

## 2. Objetivos

El objetivo principal es tener un sistema de Inteligencia Artificial (IA) **capaz de identificar la caída de una persona y que actúe en consecuencia.** Se divide en los siguientes subobjetivos:

1) Estudio y selección de lenguajes, bibliotecas e IDE a usar.
2) Reducción del peso de las imágenes obtenidas para aumentar el rendimiento.
3) Estudio y selección del método de aprendizaje para identificar humanos.

4) Entrenar la IA para que sea capaz de identificar humanos.
5) Reducción del peso de las imágenes tras identificar humanos.
    a. Selección de datos estrictamente necesarios del resultado.
6) Entrenamiento/desarrollo del sistema de IA capaz de identificar caídas.
    a. Investigación de cómo entrenar el sistema IA para que sea capaz de recibir un trozo de vídeo.
    b. Selección de un algoritmo de aprendizaje.
7) Realizar bot de la aplicación de móvil Telegram para que envíe las alertas.

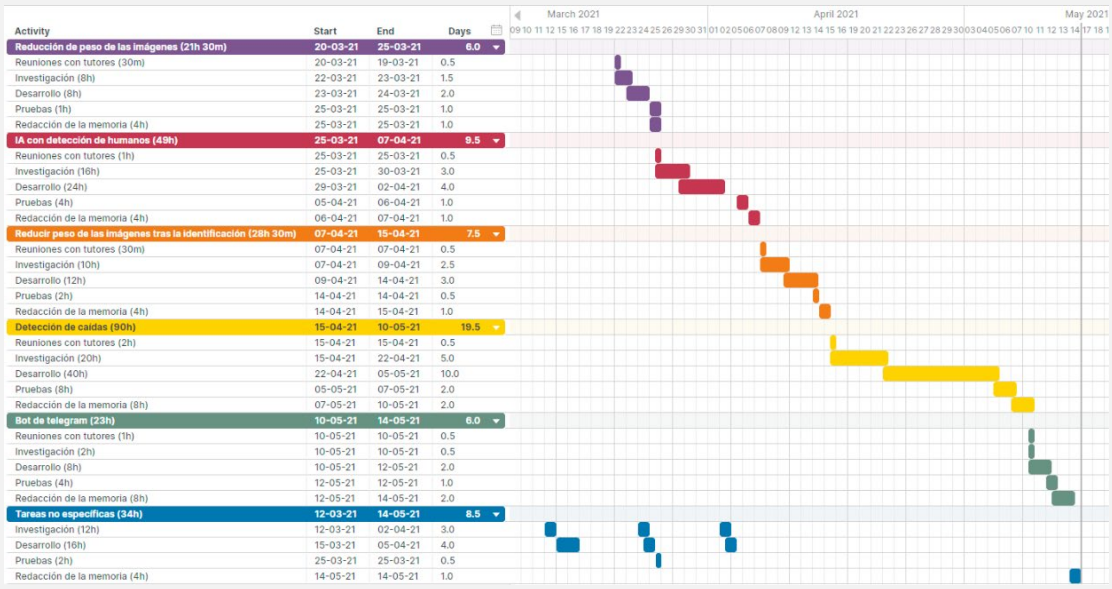
Como visión general del sistema se propone:

- En la Smart Home existen cámaras repartidas por las habitaciones, las cuales, están conectadas en la misma red que la unidad de procesamiento (es necesario que la unidad de procesamiento tenga conexión a Internet). La unidad de procesamiento obtiene las imágenes de las cámaras y comprueba si la secuencia de imágenes se puede identificar como caída. De ser así, la unidad de procesamiento envía una alerta con el vídeo de la caída a un grupo o contacto de Telegram.


*Ilustración Capítulo 1.2.1. Visión general del sistema.*

# 3. Fases de desarrollo

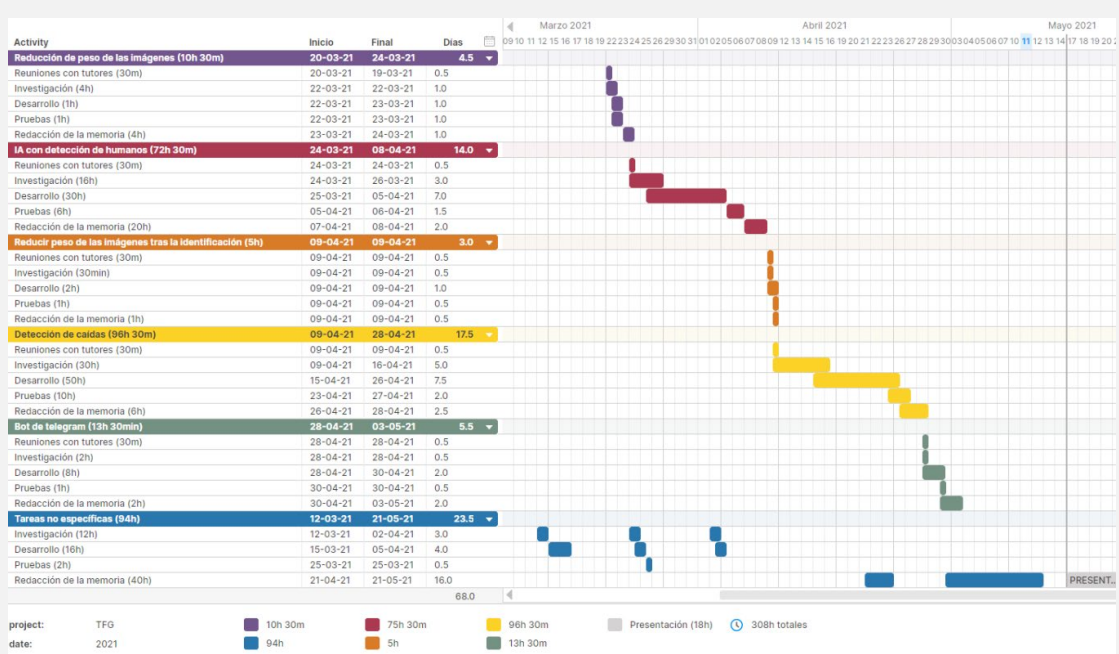
Con el objetivo de repartir las horas de forma eficiente, se estimó un total de 246 horas repartidas como se muestra en la *Ilustración Capítulo 1.3.1*.




| Activity | Start | End | Days |
|---|---|---|---|
| Reducción de peso de las imágenes (21h 30m) | 20-03-21 | 25-03-21 | 6.0 |
| Reuniones con tutores (30m) | 20-03-21 | 19-03-21 | 0.5 |
| Investigación (8h) | 22-03-21 | 23-03-21 | 1.5 |
| Desarrollo (8h) | 23-03-21 | 24-03-21 | 2.0 |
| Pruebas (1h) | 25-03-21 | 25-03-21 | 1.0 |
| Redacción de la memoria (4h) | 25-03-21 | 25-03-21 | 1.0 |
| IA con detección de humanos (49h) | 25-03-21 | 07-04-21 | 9.5 |
| Reuniones con tutores (1h) | 25-03-21 | 25-03-21 | 0.5 |
| Investigación (16h) | 25-03-21 | 30-03-21 | 3.0 |
| Desarrollo (24h) | 29-03-21 | 02-04-21 | 4.0 |
| Pruebas (4h) | 05-04-21 | 06-04-21 | 1.0 |
| Redacción de la memoria (4h) | 06-04-21 | 07-04-21 | 1.0 |
| Reducir peso de las imágenes tras la identificación (28h 30m) | 07-04-21 | 15-04-21 | 7.5 |
| Reuniones con tutores (30m) | 07-04-21 | 07-04-21 | 0.5 |
| Investigación (10h) | 07-04-21 | 09-04-21 | 2.5 |
| Desarrollo (12h) | 09-04-21 | 14-04-21 | 3.0 |
| Pruebas (2h) | 14-04-21 | 14-04-21 | 0.5 |
| Redacción de la memoria (4h) | 14-04-21 | 15-04-21 | 1.0 |
| Detección de caídas (90h) | 15-04-21 | 10-05-21 | 19.5 |
| Reuniones con tutores (2h) | 15-04-21 | 15-04-21 | 0.5 |
| Investigación (20h) | 15-04-21 | 22-04-21 | 5.0 |
| Desarrollo (40h) | 22-04-21 | 05-05-21 | 10.0 |
| Pruebas (8h) | 05-05-21 | 07-05-21 | 2.0 |
| Redacción de la memoria (8h) | 07-05-21 | 10-05-21 | 2.0 |
| Bot de telegram (23h) | 10-05-21 | 14-05-21 | 6.0 |
| Reuniones con tutores (1h) | 10-05-21 | 10-05-21 | 0.5 |
| Investigación (2h) | 10-05-21 | 10-05-21 | 0.5 |
| Desarrollo (8h) | 10-05-21 | 12-05-21 | 2.0 |
| Pruebas (4h) | 12-05-21 | 12-05-21 | 1.0 |
| Redacción de la memoria (8h) | 12-05-21 | 14-05-21 | 2.0 |
| Tareas no específicas (34h) | 12-03-21 | 14-05-21 | 8.5 |
| Investigación (12h) | 12-03-21 | 02-04-21 | 3.0 |
| Desarrollo (16h) | 15-03-21 | 05-04-21 | 4.0 |
| Pruebas (2h) | 25-03-21 | 25-03-21 | 0.5 |
| Redacción de la memoria (4h) | 14-05-21 | 14-05-21 | 1.0 |

***Ilustración Capítulo 1.3.1.*** *Diagrama de Gantt con la planificación estimada del proyecto.*

En la práctica, han sido 290 horas repartidas como se muestra en la *Ilustración Capítulo 1.3.2*, adicionalmente, se estiman 18 horas más en la preparación de la presentación. El cambio ha sido motivado principalmente por una disminución en el tiempo estimado para la realización de las tareas primera y tercera, ya que se ha encontrado una forma más sencilla de realizar dichas tareas. Gracias a la reducción de esas 2 tareas, ha sido posible dedicar más tiempo al resto.




| Activity | Inicio | Final | Dias |
|---|---|---|---|
| Reducción de peso de las imágenes (10h 30m) | 20-03-21 | 24-03-21 | 4.5 |
| Reuniones con tutores (30m) | 20-03-21 | 19-03-21 | 0.5 |
| Investigación (4h) | 22-03-21 | 22-03-21 | 1.0 |
| Desarrollo (1h) | 22-03-21 | 23-03-21 | 1.0 |
| Pruebas (1h) | 22-03-21 | 23-03-21 | 1.0 |
| Redacción de la memoria (4h) | 23-03-21 | 24-03-21 | 1.0 |
| IA con detección de humanos (72h 30m) | 24-03-21 | 08-04-21 | 14.0 |
| Reuniones con tutores (30m) | 24-03-21 | 24-03-21 | 0.5 |
| Investigación (16h) | 24-03-21 | 26-03-21 | 3.0 |
| Desarrollo (30h) | 25-03-21 | 05-04-21 | 7.0 |
| Pruebas (6h) | 05-04-21 | 06-04-21 | 1.5 |
| Redacción de la memoria (20h) | 07-04-21 | 08-04-21 | 2.0 |
| Reducir peso de las imágenes tras la identificación (5h) | 09-04-21 | 09-04-21 | 3.0 |
| Reuniones con tutores (30m) | 09-04-21 | 09-04-21 | 0.5 |
| Investigación (30min) | 09-04-21 | 09-04-21 | 0.5 |
| Desarrollo (2h) | 09-04-21 | 09-04-21 | 1.0 |
| Pruebas (1h) | 09-04-21 | 09-04-21 | 0.5 |
| Redacción de la memoria (1h) | 09-04-21 | 09-04-21 | 0.5 |
| Detección de caídas (96h 30m) | 09-04-21 | 28-04-21 | 17.5 |
| Reuniones con tutores (30m) | 09-04-21 | 09-04-21 | 0.5 |
| Investigación (30h) | 09-04-21 | 16-04-21 | 5.0 |
| Desarrollo (50h) | 15-04-21 | 26-04-21 | 7.5 |
| Pruebas (10h) | 23-04-21 | 27-04-21 | 2.0 |
| Redacción de la memoria (6h) | 26-04-21 | 28-04-21 | 2.5 |
| Bot de telegram (13h 30min) | 28-04-21 | 03-05-21 | 5.5 |
| Reuniones con tutores (30m) | 28-04-21 | 28-04-21 | 0.5 |
| Investigación (2h) | 28-04-21 | 28-04-21 | 0.5 |
| Desarrollo (8h) | 28-04-21 | 30-04-21 | 2.0 |
| Pruebas (1h) | 30-04-21 | 30-04-21 | 0.5 |
| Redacción de la memoria (2h) | 30-04-21 | 03-05-21 | 2.0 |
| Tareas no específicas (94h) | 12-03-21 | 21-05-21 | 23.5 |
| Investigación (12h) | 12-03-21 | 02-04-21 | 3.0 |
| Desarrollo (16h) | 15-03-21 | 05-04-21 | 4.0 |
| Pruebas (2h) | 25-03-21 | 25-03-21 | 0.5 |
| Redacción de la memoria (40h) | 21-04-21 | 21-05-21 | 16.0 |
| | | | 68.0 |

***Ilustración Capítulo 1.3.2.*** *Diagrama de Gantt con la planificación real del proyecto. Para verlo en grande ir a* https://plan.tomsplanner.com/public/tfg-rgg242

## 3.1. Fases

Se han definido 6 fases, las cuales coinciden con los diferentes objetivos del proyecto.

### Reducción del peso de las imágenes capturadas por las cámaras

Se quiere trabajar con imágenes del menor tamaño posible para conseguir un mejor rendimiento en cuanto a tiempo y menor carga de recursos.

### IA con detección de humanos

La lógica es que, para detectar una caída, en primer lugar, se debe de detectar la existencia de una persona en la imagen [2], más concretamente, se busca obtener la forma del cuerpo.

### Reducción del peso de las imágenes tras la identificación

Una vez detectada la forma del cuerpo se puede eliminar todo aquello que no lo sea, dando como resultado una drástica reducción de peso de la imagen.

### Detección de caídas

Una vez extraída la información necesaria sobre la forma del cuerpo, se ha de entrenar un sistema de IA, que, a partir de un trozo de vídeo reducido, sea capaz de determinar si se ha producido una caída.

### Bot de Telegram

La finalidad del *Bot de Telegram* es que alerte a un familiar, amigo o persona encargada cuando se detecte una caída.

### Tareas no específicas

Las tareas no específicas son tareas que no entran en ninguna fase en particular, pero son imprescindibles para la realización del proyecto, por ejemplo, la realización de este capítulo.

## 3.2. Subtareas de cada fase

Cada fase del proyecto tiene las siguientes 5 subtareas.

### Reuniones con tutores

Gracias a las reuniones se ha podido definir de manera precisa el estado objetivo del proyecto y también se ha ido informando de los avances y como medio para aclarar dudas.

### Investigación

Con el objetivo de ampliar u obtener nuevos conocimientos para encontrar la mejor solución en cada fase, se usan diferentes artículos, páginas webs y libros para la obtención de información.

### Desarrollo
Una vez se obtienen conocimientos sobre el problema a solucionar, en primer lugar, se realiza un planteamiento teórico y se lleva a cabo en un entorno reducido, una vez funciona correctamente, se completa su desarrollo.

### Pruebas
Tiene como objetivo arreglar posibles fallos que tenga el código o planteamiento. Se prueban datos que se consideran propensos a fallar. Se aprovecha esta subtarea para eliminar código redundante para así embellecer el resultado final.

### Redacción de memoria
Se documenta la información obtenida gracias a realizar las subtareas anteriores. Con la ayuda de ilustraciones y ejemplos se quiere conseguir un documento claro y fácil de comprender.

# 4. Estructura del documento

En el **capítulo 1** se busca transmitir la motivación por la cual se quiere realizar este proyecto, especifica sus fases y su objetivo final.

En el **capítulo 2** se investigan diferentes soluciones al mismo problema planteado.

En el **capítulo 3** se da a conocer los diferentes métodos y herramientas que serán necesarias para llevar a cabo el proyecto.

En el **capítulo 4** se explica de forma detallada el desarrollo realizado hasta obtener una solución completamente funcional.

En el **capítulo 5** se comprueba el rendimiento del programa.

En el **capítulo 6** se habla de los resultados obtenidos, la conclusión y las posibles mejoras.

Por último, se encuentra la **bibliografía** en la que se basa el proyecto.

# 5. Competencias utilizadas

Para el desarrollo de este proyecto han sido necesarias la utilización las competencias básicas y de una serie de competencias transversales.

El uso de diferentes métodos para buscar información tanto en español como en inglés, la capacidad de trabajar de forma autónoma y el compromiso para llevar a cabo los objetivos del proyecto respetando a las personas y al medio ambiente se encuentran relacionados con las siguientes competencias transversales:

- **UAL1**: Conocimientos básicos de la profesión.
- **UAL2**: Habilidad en el uso de las TIC.
- **UAL3**: Capacidad para resolver problemas.

- **UAL4**: Comunicación oral y escrita en la propia lengua.
- **UAL5**: Capacidad crítica y autocrítica.
- **UAL7**: Aprendizaje en una lengua extranjera.
- **UAL8**: Compromiso ético.
- **UAL9**: Capacidad para aprender a trabajar de forma autónoma.
- **UAL10**: Competencia social y ciudadanía global.

También se ha hecho uso de diferentes competencias básicas del título asociadas a la hora de diseñar y llevar a cabo el proyecto de la forma más eficaz posible aplicando la legislación necesaria y generando documentación útil de forma autónoma:

- **CT1**: Capacidad para concebir, redactar, organizar, planificar, desarrollar y firmar proyectos en el ámbito de la ingeniería informática.
- **CT2**: Capacidad para dirigir las actividades objeto de los proyectos del ámbito de la informática.
- **CT7**: Capacidad para conocer, comprender y aplicar la legislación necesaria durante el desarrollo de la profesión de ingeniero informático.
- **CT9**: Capacidad para resolver problemas con iniciativa, toma de decisiones, autonomía y creatividad.

Por último, se ha desarrollado el proyecto eligiendo las formas más eficientes, estructuras de datos adecuados y evaluando el consumo de recursos disponibles, para esto han sido necesarias las siguientes competencias específicas:

- CC01, CC02, CC04, CC05, CC06, CC07, CC08, CC09, CC10, CC11, CC13, CT3, CT4, CT5, CT10, CT11, CB01, CB03, CB04, CC14, CC15, CC16.

  La definición de cada competencia está recogida en el documento ual.es/application/files/2115/8202/2495/competencias-esp-informatica4015.pdf.

Las asignaturas que han sido un pilar en este proyecto han sido las siguientes:

- **Estructuras de datos y algoritmos I, II, Lógica y Algorítmica e Ingeniería del software**: Ha sido necesario en el desarrollo a la hora de elegir estructuras de datos. También para realizar el recorrido de los datos de forma correcta.
- **Algebra lineal y matemática discreta**: Ha sido necesaria tanto como para obtener una adecuada comprensión de las matrices, como para poder escribir correctamente y con la notación necesaria las diferentes operaciones matriciales.
- **Cálculo y estadística**: A la hora de realizar operaciones para la obtención o corrección de datos y para comparar los resultados obtenidos siguiendo diferentes métodos.

- **Planificación y gestión de proyectos informáticos**: Se han establecido fechas límite para cada fase, de esta manera es más fácil acabar el proyecto en tiempo y forma.
- **Multiprocesadores y Sistemas de tiempo real:** En este proyecto se ha buscado hacer un sistema que funcione en tiempo real, para ello ha sido necesario el uso de varios hilos.
- **Tratamiento digital de imágenes:** La comprensión de cómo se guarda la información de una imagen y como tratar dicha información ha sido muy importante para este proyecto.
- **Tecnologías Multimedia:** En esta asignatura se enseña el uso de un software de control de versiones y el lenguaje Python. También se desarrolla un software de comunicación en tiempo real iterativo.

Capítulo 2

# Estado del arte

Una vez planteado el problema, se investigan las diferentes soluciones a este problema o uno similar para comparar las diferentes alternativas. Gracias a esta investigación se obtendrá un punto de vista más amplio a la hora de empezar a desarrollar el proyecto.

Existen artículos relacionados con el reconocimiento de caídas usando cámaras, entre ellos:

- **An eight-camera fall detection system using human fall pattern recognition via machine learning by a low-cost android box** [2]**.** Este método usa aprendizaje computacional KNN y modelos matemáticos. En primer lugar, entrenan un sistema de IA para reconocer humanos usando SpeedyAI, INC [3] con la técnica Relevance Vector Machine (RVM) [4] utilizando el Histograma de Gradientes Orientados [5].

- **Home Camera-Based Fall Detection System for the Elderly** [6]**.** Combina varios algoritmos (sustracción de fondo [7], filtrado Kalman [8] y flujo óptico [9]). El ratio de detección es superior al 96%.

- **Fall Detection With Multiple Cameras: An Occlusion-Resistant Method Based on 3-D Silhouette Vertical Distribution** [10]**.** Método basado en una red de cámaras que reconstruye la forma de las personas, detecta la caída analizando la distribución del volumen a lo largo del eje vertical, activando la alarma cuando la mayor parte del volumen está anormalmente cerca del suelo durante un periodo de tiempo. El ratio de detección del 99.7%.

- **3D depth image analysis for indoor fall detection of elderly people** [11]**.** Método basado en el análisis de la profundidad de la forma 3D captadas por un sensor Kinect. Las imágenes son pre-procesadas mediante un filtro de paso medio. La silueta de las imágenes de profundidad en movimiento se consigue mediante un método de sustracción de los fotogramas del fondo.

- **A vision-based approach for fall detection using multiple cameras and convolutional neural networks: A case study using the UP-Fall detection dataset** [12]. Detector de caídas basado en un método de inferencia con Redes Neuronales convolucionales (CNN) 2D y en múltiples cámaras. Analiza las imágenes en momentos de tiempo fijos y extrae la información sobre el estado de las personas usando flujo óptico. Se ha probado en su conjunto de datos públicos (UP-Fall Detection dataset), alcanzando una precisión del 95,64%.

- **Robust Video Surveillance for Fall Detection Based on Human Shape Deformation** [13]. Se utiliza una técnica de coincidencia de formas para seguir la silueta de la persona en la secuencia de vídeo. Se cuantifica la deformación de la forma a partir de estas siluetas basándose en métodos de análisis de forma. Se detectan las caídas a partir de las actividades normales utilizando un modelo de mezcla gaussiana. Este trabajo se ha llevado a cabo con un conjunto de datos realistas de actividades diarias y caídas simuladas, y ofrece muy buenos resultados (hasta un 0% de error con una configuración de varias cámaras).

- **A Real-Time, Multiview Fall Detection System: A LHMM-Based Approach** [14]. En este método, el movimiento se modela mediante un modelo de Markov oculto en capas (LHMM) [15] y es capaz de detectar cambios muy bruscos en la secuencia de vídeo.

- **Human fall detection in surveillance video based on PCANet** [16]**.** Detector de caídas basado en métodos de aprendizaje automático de características. Los fotogramas extraídos, incluidos los humanos, de secuencias de vídeo de diferentes vistas, forman el conjunto de entrenamiento. Se entrena un modelo PCANet utilizando todas las muestras para predecir cada fotograma. Dado que el comportamiento de una caída está contenido en muchos fotogramas continuos, la detección fiable de una caída no debe analizar sólo un fotograma, sino también una secuencia de vídeo.

Capítulo 3

# Herramientas

En este capítulo se describirán las diferentes herramientas usadas para realizar el proyecto.

## 1. Comunicación

En este apartado se mencionan los medios que se usan para la comunicación.

### 1.1. Reuniones

Se ha usado **Google Meet** (meet.google.com) para plantear el problema y comentar la solución a lo largo del proyecto con mis tutores.

### 1.2. Correo electrónico

Se ha usado el correo electrónico para dudas o temas sencillos con mis tutores.

### 1.3. Telegram

Telegram (telegram.org) es una plataforma de mensajería instantánea. Adicionalmente, en Telegram se pueden crear bots (abreviación de robots) y actúan de manera autónoma dependiendo del comando que reciban. En este proyecto, se usa un bot de Telegram para enviar alertas al detectar una caída o recuperación.

## 2. Documentación

En este apartado se mencionan las herramientas que se han usado para generar el documento.

### 2.1. Microsoft Word

Word es un editor de texto (microsoft.com/microsoft-365/word) que ofrece alojamiento en la nube y un entorno colaborativo, de esta manera, los tutores pueden revisar el estado del documento en cualquier momento.

## 2.2. Tom's planner

Tom's Planner (tomsplanner.es) es un proveedor de servicios de aplicaciones y herramientas basado en la web para la planificación, gestión y colaboración de proyectos. Se ha usado para crear y compartir el diagrama de Gantt.

## 2.3. Adobe Illustrator

Adobe Illustrator (adobe.com/illustrator) es un editor de gráficos vectoriales. Se ha usado para crear elementos del diseño e ilustraciones del documento.

# 3. Desarrollo

En este apartado se mencionan las herramientas y recursos usados para la elaboración de una solución.

## 3.1. Lenguaje de programación Python

El lenguaje de programación Python fue creado a finales de los ochenta por Guido van Rossum. Python se ha convertido en uno de los lenguajes de programación más usados actualmente.

Su filosofía hace hincapié en la legibilidad de su código. Es un lenguaje multiparadigma ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

## 3.2. Bibliotecas

- **Tensorflow** [tensorflow.org]
  Es una biblioteca de código abierto usado para desarrollar y entrenar modelos con Machine Learning.

- **Modelo PoseNet** [tensorflow.org/lite/examples/pose_estimation/overview]
  Es un modelo ligero desarrollado por Google usando Deep Learning en Tensorflow que estima la pose humana.

- **OpenCV-Python** [pypi.org/project/opencv-python]
  Cv2 es una biblioteca multiplataforma libre de visión artificial enfocado al procesamiento de imágenes.

- **Numpy** [numpy.org]
  Es una biblioteca usada para trabajar con vectores y matrices grandes multidimensionales. También tiene funciones para trabajar en dominios de algebra lineal, transformadas de Fourier y matrices.

- **Pandas** [pypi.org/project/pandas]
  Es una biblioteca escrita como extensión de Numpy para la manipulación y el análisis de datos.

- **Sklearn** [pypi.org/project/scikit-learn]
  Es una biblioteca de software libre usada para el aprendizaje automático. Incluye varios algoritmos de aprendizaje. También soporta librerías de numéricas tipo Numpy y SciPy.

- **Pickle** [docs.python.org/3/library/pickle]
  Este módulo implementa protocolos binarios para serializar y deserializar una estructura de objetos.

- **Telegram-send** [pypi.org/project/telegram-send]
  Es una herramienta que permite que un bot de Telegram enviar mensajes y archivos por Telegram a una cuenta, grupo o canal.

- **Argparse** [docs.python.org/3/library/argparse]
  Es una herramienta que facilita el uso del programa por línea de comandos.

## 3.3. PyCharm

Pycharm (jetbrains.com/pycharm) es un IDE (Entorno de Desarrollo Integrado) desarrollado por la empresa JetBrains y enfocado al lenguaje de programación Python.

## 3.4. GitHub

GitHub (github.com) es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

## 3.5. OpenVPN

OpenVPN (openvpn.net) es una herramienta de conectividad basada en software libre: SSL, VPN Virtual Private Network. OpenVPN ofrece conectividad punto-a-punto con validación jerárquica de usuarios y host conectados remotamente. Se usa para conectarse a un host dentro de la Universidad de Almería.

# 4. Periféricos

En la Smart Home de la Universidad de Almería, se cuentan con diferentes dispositivos para la captura de imágenes.

## 4.1. Foscam R2

Las dos cámaras de este modelo se encuentran en zonas altas, una en el distribuidor y otra en el salón.

**Características**

- 2 megapíxeles (MP) (1920 × 1080 píxeles).
- 30 fotogramas por segundo (FPS).
- 300º de movimiento horizontal.
- 100º de movimiento vertical.
- Visión nocturna.
- Encriptación WiFi.

*Ilustración Capítulo 3.4.1.*
*Cámara Foscam R2*

## 4.2. Foscam FI9800P

Las dos cámaras de este modelo se encuentran en zonas altas, una en la habitación y otra en la cocina.

**Características**

- 1 MP (1280 × 720 píxeles).
- 15 FPS.
- 180º de movimiento horizontal.
- 180º de movimiento vertical.
- Visión nocturna.
- Encriptación WiFi.

*Ilustración Capítulo 3.4.1.*
*Cámara Foscam FI9800P*

## 4.3. Xiaomi Redmi Note 9 Pro

El dispositivo móvil se ha usado para grabar los vídeos de entrenamiento.

**Cámara principal**

- Resolución: 64 MP (1/1,72'')
- Apertura: f/1.89.

**Cámara dual**

- Resolución: 8 MP, ultra gran angular
- Apertura: f/2.2.
- *Campo de visión*: 119º.

*Ilustración Capítulo 3.4.3.*
*Xiaomi Redmi Note 9 Pro*

# Capítulo 4

# Desarrollo

El objetivo de este capítulo es conseguir una solución funcional para detectar caídas a partir de varias cámaras de forma simultánea. Para conseguir esto, se obtienen una serie de imágenes de las cuales se extraen las poses de una persona gracias a un modelo entrenado. A partir de la secuencia de poses, se entrena un nuevo modelo con el objetivo de detectar caídas y recuperaciones. Con el nuevo modelo se extrae el resultado y en caso de detectar una caída o recuperación, se envía una alerta a partir del servicio de mensajería Telegram.

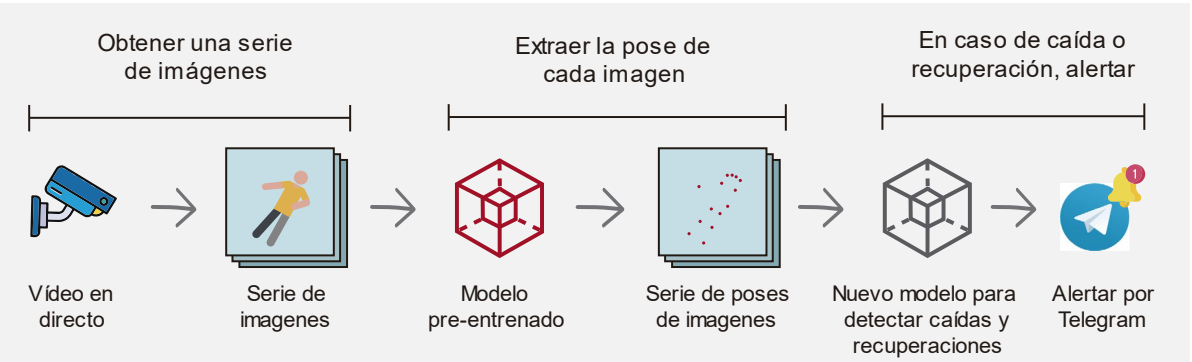



***Ilustración Capítulo 4.1**. Funcionamiento del programa a realizar.*

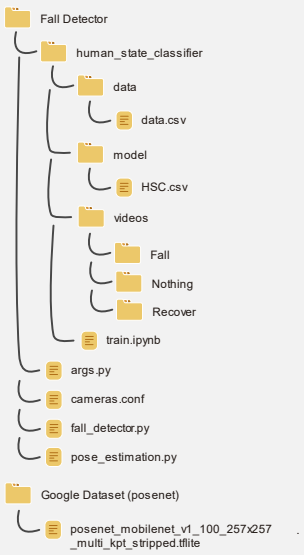



***Ilustración Capítulo 4.2**. Organización de carpetas del proyecto.*

Se explica de manera detallada el desarrollo de una solución útil para resolver el problema planteado, el proyecto se puede encontrar en github.com/RaquelGG/Fall-Detector.

La organización de carpetas es como se muestra en la *Ilustración Capítulo 4.2.* Por motivos de privacidad, los vídeos no han sido subidos, pero los datos y el modelo obtenidos de ellos se pueden encontrar en el repositorio GitHub.

Solo el contenido de la carpeta `Fall Detector` está disponible en GitHub.

# 1. Estimación de pose corporal

Para estimar las posiciones de las diferentes partes del cuerpo, se ha decidido usar PoseNet con TensorFlow. PoseNet es un modelo que ha sido optimizado para poder funcionar a tiempo real incluso, en dispositivos móviles. También es capaz de detectar las poses de múltiples personas, aunque para este proyecto se usará únicamente la detección de pose de una persona. El nombre del modelo PoseNet es posenet_mobilenet_v1_100_257x257_multi_kpt_stripped.tflite.

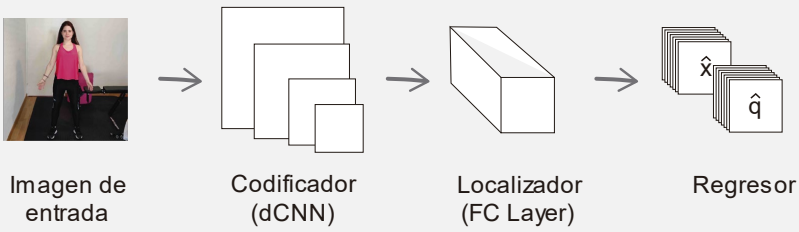



***Ilustración Capítulo 4.1.1**. Esquematización de la arquitectura de PoseNet. El codificador con arquitectura dCNN genera vectores de características a partir de la imagen de entrada. Mediante una capa FC, la codificación visual de la imagen se asigna a un vector de características de localización y, por último, dos capas separadas, pero conectadas se usan para hacer la regresión y obtener la pose estimada $= (\hat{x}, \hat{q})$ [20].*

Se puede encontrar todo el código desarrollado para obtener las partes del cuerpo en archivo `pose_estimation.py`.

## 1.1. Preprocesamiento

Para poder usar el modelo, las imágenes de entrada deben ser del mismo tamaño que las imágenes con las que se entrenó el modelo. Esto podemos saberlo gracias al intérprete de TensorFlow, con el que se pueden extraer detalles del modelo. Este modelo en particular requiere que el tamaño de las imágenes sea de 256x256 píxeles.

Como las imágenes de entrada provienen de diferentes cámaras con diferentes dimensiones, es necesario redimensionar las imágenes sin cambiar sus proporciones o perder píxeles que podrían contener información fundamental. Se ha desarrollado un método llamado `make_square`, que obtiene el mayor de los lados (*lado_mayor*) de la imagen original (*img*), crea un array de numpy (*img_cuadrada*) lleno de ceros de tamaño *lado_mayor*, calcula la posición del centro de *img_cuadrada* para ahí pegar *img*, y, por último, se redimensiona la *img_cuadrada* al tamaño deseado, tal como se muestra en la *Ilustración Capítulo 4.1.2*.

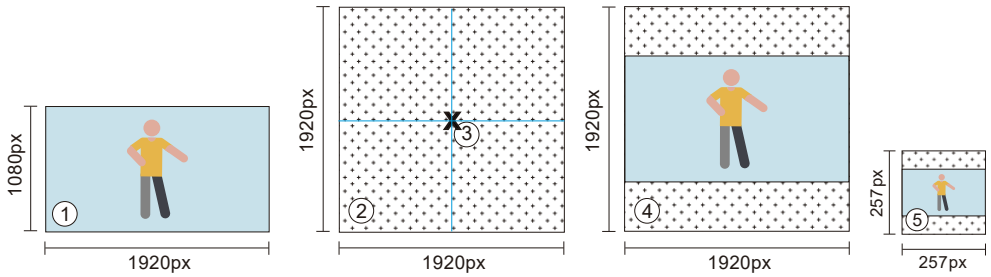



***Ilustración Capítulo 4.1.2**. Método make_square: Redimensiona a una imagen con dimensiones cuadradas.*
*(1) Imagen original capturada.*
*(2) Nueva imagen vacía con el tamaño del mayor de los lados de la imagen original.*
*(3) Obtención del centro de la nueva imagen vacía.*
*(4) Colocación de la imagen original en el centro obtenido.*
*(5) Redimensión de la imagen resultante.*

La imagen resultante de tamaño 257x257 píxeles será la entrada del modelo.

## 1.2. Obtención de las posiciones del cuerpo

Gracias al modelo, se pueden obtener 17 posiciones del cuerpo, llamados *keypoints* o *puntos clave*.


***Ilustración Capítulo 4.2.3**. Diecisiete puntos clave (los puntos rojos) detectados por PoseNet.*

Aunque para obtener el resultado de la *Ilustración Capítulo 4.2.2*, es necesario interpretar los datos de salida del modelo PoseNet.

El modelo tiene como salida un Heatmap y un Offset Vector [17], ambos son tensores 3D, un tensor es una estructura de datos multidimensional.

> *In the general case, an array of numbers arranged on a regular grid with a variable number of axes is known as a tensor.*
>
> Pág 31, Deep Learning, 2016 [18]

Así como un vector es una estructura de una dimensión sería un tensor de primer orden, una matriz es una estructura de dos dimensiones sería un tensor de segundo orden.

El tamaño de ambos tensores dependerá de la resolución, que se puede calcular usando la *Fórmula Capítulo 4.2.1*. Por lo tanto, sabiendo que el *Tamaño de imagen* es 257 píxeles y que *OutputStride* en el modelo usado es 32, la resolución será de 9.

$$Resolución = \frac{Tamaño\ de\ imagen - 1}{OutputStride} + 1$$

***Fórmula Capítulo 4.2.1.*** *Obtención de la resolución. Internamente, el parámetro OutputStride afecta a la altura y anchura de las capas de la red neuronal.*

## Heatmap

Es un tensor 3D de tamaño **resolución x resolución x keypoints**, es decir, 9x9x17, donde cada elemento del heatmap tiene una puntuación de coincidencia, que es la probabilidad de que esa parte del cuerpo esté realmente ahí.

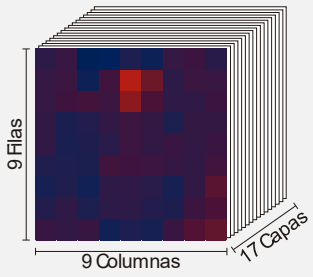



***Ilustración Capítulo 4.2.4****. Heatmap obtenido a partir del modelo PoseNet.*

## Offset Vector

Es un tensor 3D de tamaño **resolución x resolución x (keypointsx2)**, es decir, 9x9x34. A diferencia del Heatmap, el Offset Vector se usa para obtener la posición exacta de los keypoints. Las 17 primeras capas contienen la coordenada $x$ y las 17 últimas la coordenada $y$.

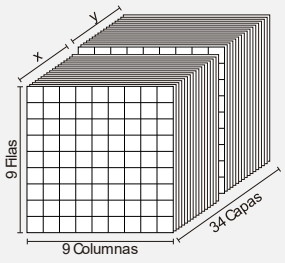



***Ilustración Capítulo 4.2.5****. Offset Vector obtenido a partir del modelo PoseNet.*

## Interpretar salida del modelo

Cuando una imagen es procesada, lo que se obtiene del modelo es un Heatmap (mapa de calor) y un Offset Vector (vectores de desplazamiento), este último puede ser decodificado para obtener áreas de alta confianza en la imagen, que se corresponden a las coordenadas de los keypoints. *Véase la Ilustración Capítulo 4.2.5*.

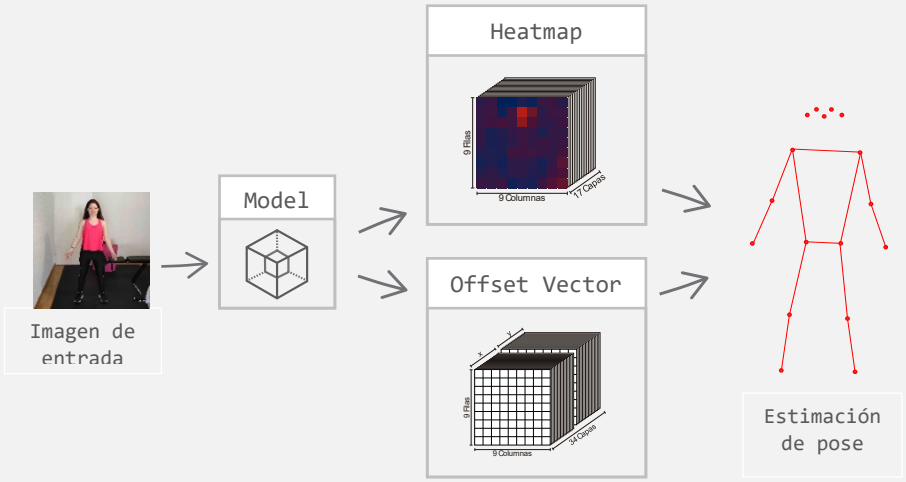



***Ilustración Capítulo 4.2.6**. Diagrama de obtención de puntos clave usando PoseNet.*

A partir del modelo, se pueden extraer los siguientes 17 *puntos clave*:

| ID | Parte |
|---|---|
| 0 | Nariz |
| 3 | Oreja Izquierda |
| 6 | Hombro Derecho |
| 9 | Muñeca Izquierda |
| 12 | Cadera Derecha |
| 15 | Tobillo Izquierdo |

| ID | Parte |
|---|---|
| 1 | Ojo Izquierdo |
| 4 | Oreja Derecha |
| 7 | Codo Izquierdo |
| 10 | Muñeca Derecha |
| 13 | Rodilla Izquierda |
| 16 | Tobillo Derecho |

| ID | Parte |
|---|---|
| 2 | Ojo Derecho |
| 5 | Hombro Izquierdo |
| 8 | Codo Derecho |
| 11 | Cadera Izquierda |
| 14 | Rodilla Derecha |

***Tabla Capítulo 4.2.1.** Nombre e ID de los 17 puntos clave obtenidos por PoseNet.*

Cada una de las 17 capas del Heatmap corresponde a un *punto clave* específico, los índices de cada posición de la capa corresponden a la posición aproximada y cada posición contiene la probabilidad de que la parte del cuerpo esté realmente ahí. Por ejemplo, la primera capa del Heatmap corresponde con la posición aproximada de la nariz y la última corresponde con la posición aproximada del tobillo derecho.

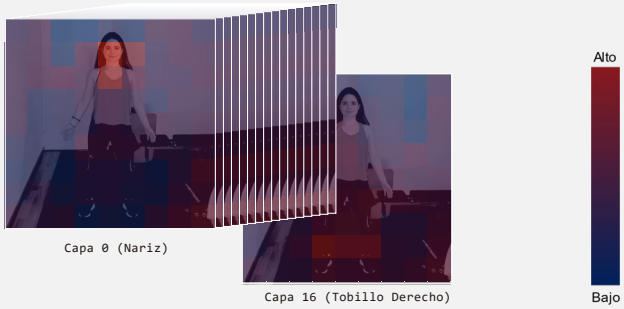



***Ilustración Capítulo 4.2.7**. Contenido del tensor Heatmap (los colores indican la coincidencia).*

Gracias a las posiciones aproximadas que se obtienen de Heatmap se pueden calcular los índices de Offset Vector que contienen las coordenadas exactas de cada punto clave.

El método *get_keypoints_positions* devuelve un array de tamaño **nº_puntos_clave x (posición x, posición y)**, es decir, 17x2. Los índices de este array corresponderán con los IDs de los *puntos clave* de la *Tabla Capítulo 4.2.1,* y su contenido serán las posiciones $x$, $y$ de cada punto clave. Este array se denominará *keypoints_positions*.

Para guardar en *keypoints_positions* las posiciones exactas, se recorrerán una a una las capas de Heatmap.

Para cada capa, se buscará el valor máximo (el de mayor coincidencia) para obtener los índices que lo contienen y a la tupla ($x$, $y$) se le llamará *approx_position*.

Como cada Offset Vector tiene una resolución de 9x9, es necesario calcular la ubicación para establecer una correspondencia con las posiciones de la imagen de 257x257 píxeles. Se quiere que las posiciones tomen valores acordes al tamaño de la imagen de entrada:

- Los valores que puede tomar los índices x e y en la imagen de entrada son de 0 a 256.
- Los valores que puede tomar los índices x e y en las capas de Offset Vector son de 0 a 8.

Se aplica la *Fórmula Capítulo 4.2.2.* a la tupla *approx_position*, la tupla resultante se denominará *relocation*.

$$relocation_i = \frac{approx\_position_i}{\mathrm{n}} \cdot \mathrm{s} \quad (\mathrm{i} = \mathrm{x}, \mathrm{y}, \qquad \mathrm{n} = 8, \qquad \mathrm{s} = 256)$$

***Fórmula Capítulo 4.2.2.*** *Reubicación para corresponder a una posición acorde al tamaño de imagen.*

Por último, se obtienen las posiciones exactas de Offset Vector, se le suma el valor *relocation* y se almacenan en *keypoints_positions*.

- Para $x$, se usan los mismos índices obtenidos de Heatmap, *approx_position* y la capa correspondiente
- Para $y$, se usan los mismos índices obtenidos de Heatmap, *approx_position* y la capa correspondiente + 17, ya que los valores y se encuentran en la segunda mitad de capas del tensor.

Para entender mejor lo explicado anteriormente, el pseudocódigo correspondiente sería el siguiente:

```
get_keypoints_positions(heatmap, offsets):
  keypoints_detected  = 17 (Num de capas de heatmap)
  keypoints_positions = ndarray de tamaño 17x2

  for i in keypoints_detected:
     1. Obtener capa actual de heatmap a partir del ID (i)
        layer = capa de heatmap i.

     2. Obtener los índices que contienen el valor máximo de layer
        approx_position = get_index(layer, max(layer))

     3. Calcular reubicación para la tupla
        relocation = approx_position/8*256

     4. Almacenar posiciones exactas en keypoints_positions:
        x:
        keypoints_positions[i, 0] = relocation[0] + offsets[approx_position[0],
                                                            approx_position[1],
                                                            i]
        y:
        keypoints_positions[i, 1] = relocation[1] + offsets[approx_position[0],
                                                            approx_position[1],
                                                            i+keypoints_detected]

     5. Devolver las posiciones
        return keypoints_positions
```

***Pseudocódigo Capítulo 4.2.1.*** *Método get_keypoints_positions de la clase PoseEstimation.*

# 2. Clasificador de estado humano

Para conseguir el objetivo de este proyecto, que es detectar caídas, se entrena un nuevo modelo que se llamará HSC (Human State Classifier) para que sea capaz de determinar el estado de un humano; en este proyecto existen 3 estados: Nada, Caída, Recuperación.

- Nada: No se ha detectado una persona o la persona está actuando con normalidad.

***Ilustración Capítulo 4.2.1****. Secuencia de vídeo del estado "Nada".*

- Caída: La persona se ha caído, se pedirá ayuda.

***Ilustración Capítulo 4.2.2****. Secuencia de vídeo del estado "Caída".*

- Recuperación: La persona se ha recuperado de su caída, se notificará el cambio.

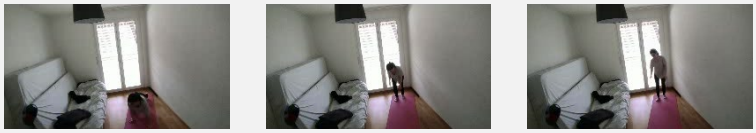


*Ilustración Capítulo 4.2.3. Secuencia de vídeo del estado "Recuperación".*

Aunque en este proyecto solo existan 3 estados, el algoritmo es fácilmente escalable. Solo será necesario añadir una carpeta con el nombre del estado en `/human_state_classifier/videos` y en su interior añadir los vídeos de 3 segundos para entrenar al modelo. El código para entrenar el modelo se puede encontrar en `/human_state_classifier/train.ipynb`.

## 2.1. Preparación de datos de entrada para el modelo

En vez de entrenar un modelo con toda la información que tiene una imagen, la información la proporciona el algoritmo explicado en la sección 1 del capítulo 4.

Se quiere detectar una caída a partir de vídeo en vivo. Se ha estimado que una caída se lleva a cabo en un intervalo de 3 segundos y por motivos de rendimiento se usan 5 fotogramas por segundo, es decir, **los trozos de vídeo serán de 15 fotogramas**.

En primer lugar, los nombres de los estados se obtienen del nombre de las carpetas situadas en `/human_state_classifier/videos`.

Se obtienen exactamente 15 fotogramas de cada vídeo independientemente de los fotogramas por segundo que tenga el vídeo original.

Se extraen los puntos clave de la pose de cada fotograma y se normalizan.

Para normalizar la posición, se aplican las *Fórmulas Capítulo 4.3.1* tanto para las coordenadas de x como para las de y, los valores obtenidos se restan a cada elemento de la matriz (*promedio_x* se resta a cada coordenada de x, *promedio_y* a cada coordenada de y), de esta manera, la idea es que sea irrelevante en qué parte de la imagen se detecte la caída, ya que el centro de masa de los puntos estará siempre centrado en las coordenadas (0, 0). En la *Ilustración Capítulo 4.3.2* se puede ver de manera gráfica.

$$p_{ij} = p_{ij} - \frac{\sum_{k=1}^{m} p_{kj}}{m} \quad (1, \dots, m, \qquad j = x, y)$$

Siendo $P_{m,n} = (p_{ij})$ la matriz que contiene los *puntos clave* con $m = 17$ y $n = 2$.

***Fórmulas Capítulo 4.3.1.** Normalizar posición de los puntos clave obtenidos del modelo PoseNet.*

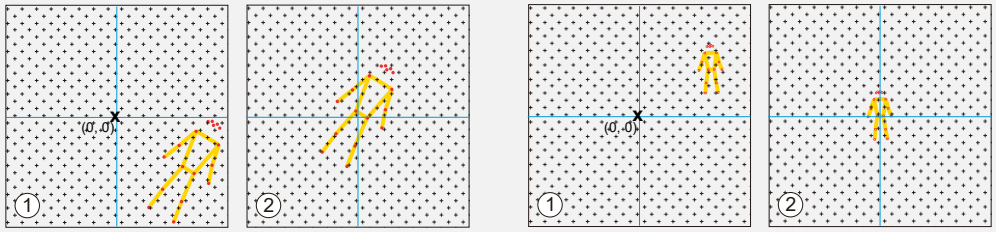



*Ilustración Capítulo 4.3.4. Normalizar posición. (1) puntos clave detectados. (2) Posición normalizada.*

Además, de normalizar la posición, también se normaliza la escala, es decir, que el rango de valores (x, y) sea de -1 a 1, se aplica la *Fórmula Capítulo 4.3.2*, que obtiene el valor máximo de la matriz de puntos clave y lo divide entre cada elemento de la matriz. En la *Ilustración Capítulo 4.3.3* se puede ver de manera gráfica.

$$p_{ij} = \frac{p_{ij}}{\max_{k=1,\dots,m}(|p_{kj}|)} \quad (\mathrm{i} = 1, \dots, m, \qquad j = x, y)$$

Siendo $P_{m,n} = (p_{ij})$ la matriz que contiene los *puntos clave* con $m = 17$ y $n = 2$.

*Fórmula Capítulo 4.3.2. Normalizar escala de los puntos clave tras normalizar la posición.*

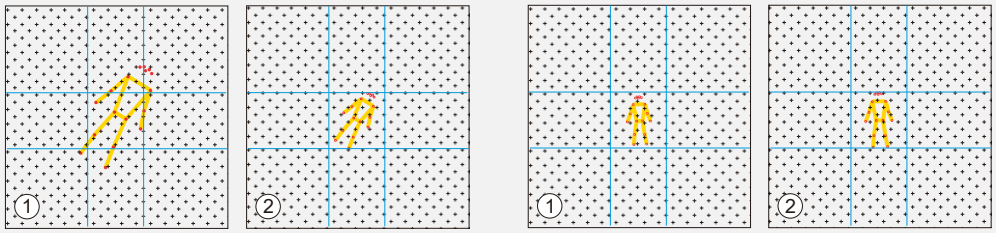



*Ilustración Capítulo 4.3.5. Normalizar escala. (1) puntos clave detectados. (2) Escala normalizada.*

Gracias a esto es irrelevante la estatura de la persona y si está lejos o cerca.

Por último, se serializan los datos de cada vídeo de la siguiente manera:

- `Frame0_nose_X,Frame0_nose_Y,…,Frame[i]_[keypoint_j]_X, Frame[i]_[keypoint_j]_Y,…,[resultado].`

  Siendo `[i]` el número de fotograma, `[keypoint_j]` la parte del cuerpo y `[resultado]` el estado al que corresponden los datos.

Cada vídeo se corresponderá con una línea de entrada para el modelo, se pueden ver los datos exportados en `/human_state_classifier/data`.

## 2.2. Selección de algoritmo de aprendizaje

En este proyecto, el factor más importante a la hora de elegir un algoritmo de aprendizaje para la clasificación del estado es la precisión. Para conseguir una buena precisión en el modelo, se han probado diferentes algoritmos de aprendizaje con diferentes parámetros de entrada.

Para grabar los vídeos, se han usado las dos cámaras del dispositivo móvil Xiaomi Redmi Note 9 Pro, El dispositivo se ha colocado en posiciones altas y en diferentes ubicaciones. Se han grabado un total de 121 vídeos; 42 vídeos son de "Caídas", 39 vídeos son de "Nada" y 40 vídeos son de "Recuperaciones". Se obtienen las secuencias de puntos clave de los vídeos y se usan como conjunto de datos (dataset) para entrenar al modelo. El 80% de datos se usan para entrenar y el 20% para pruebas.

Se han probado los siguientes algoritmos de aprendizaje; Random Forest, MLP y Gradient Boosting Tree.

### Random Forest

Un Random Forest (en castellano Bosque Aleatorio) es una colección de árboles predictores $h(x;\theta_k), k = 1, \dots, K$ donde $x$ representa el vector de entrada (covariable) de longitud $p$ con el vector aleatorio asociado $X$ y los $\theta_k$ son vectores aleatorios independientes e idénticamente distribuidos (iid) [18]. Este algoritmo es muy popular por su adaptabilidad y precisión.

Para mantener el error individual lo más bajo posible es importante dejar crecer el árbol hasta la profundidad máxima que pueda alcanzar.

Los hiperparámetros que se prueban son:

- Número de árboles en el bosque (n_estimators).
- La función para medir la calidad del corte (criterion): "gini" para la impureza de Gini y "entropy" para la ganancia de información.
- La profundidad máxima del árbol (max_depth). Sin especificar, los nodos se expanden hasta que todas las hojas sean puras o hasta que todas las hojas contengan menos muestras que min_samples_split.
- Número mínimo de muestras para dividir un nodo interno (min_samples_split).

| | | | n_estimators | | |
|---|---|---|---|---|---|
| criterion | min_samples_split | max_depth | 20 | 50 | 100 |
| Gini | 2 | Sin definir | 0.84 | 0.92 | 0.92 |
| | | 30 | 0.88 | 0.88 | 0.84 |
| | 4 | Sin definir | 0.88 | 0.84 | 0.88 |
| | | 30 | 0.88 | 0.88 | 0.84 |
| Entropy | 2 | Sin definir | 0.84 | 0.88 | 0.88 |
| | | 30 | 0.84 | 0.88 | 0.88 |
| | 4 | Sin definir | 0.80 | 0.88 | 0.88 |
| | | 30 | 0.88 | 0.88 | 0.84 |

***Tabla Capítulo 4.2.1.*** *Resultados del modelo usando diferentes parámetros para el algoritmo Random Forest.*

Como hay varias configuraciones que coinciden con la mayor precisión (0.92), se escoge el que menos n_estimators tenga, de esta manera el bosque es más simple y hay menos riesgo de que se produzca un sobreajuste. Por lo tanto, el mejor resultado ha sido con criterion: "gini", min_samples_split: 2, max_depth: Sin definir, n_estimators: 50. Para reproducir el resultado se ha usado el número 16 como semilla para generar números aleatorios. La precisión obtenida con 25 muestras de prueba ha sido del 92%.

| | Precisión | Recall | F1-score | Nº Muestras |
|---|---|---|---|---|
| Caída | 0.89 | 0,89 | 0.89 | 9 |
| Nada | 1.00 | 0.90 | 0.95 | 10 |
| Recuperación | 0.86 | 1.00 | 0.92 | 6 |
| Media ponderada | 0.93 | 0.92 | 0.92 | 25 |

***Tabla Capítulo 4.2.2.*** *Resultados del modelo aplicando el algoritmo Random Forest ajustado.*
*Precisión: Proporción de positivos correctos.* $VP/VP + FP$
*Recall: Proporción de verdaderos positivos correctos.* $VP/VP + FN$
*F1-score: Ofrece una única puntuación que equilibra Precisión y Recall.*
*Donde: VP = Verdadero Positivo, FP = Falso Positivo y FN = Falso Negativo.*

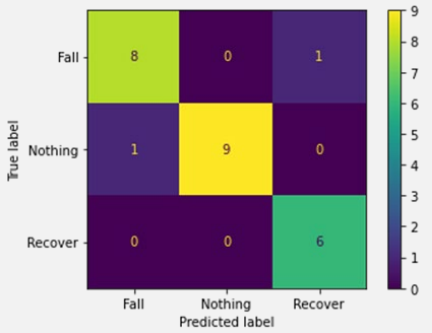



***Ilustración Capítulo 4.2.6.*** *Matriz de confusión del modelo enseñado con Random Forest.*

## MLP (Multilayer perceptron)

El perceptrón multicapa es una red neuronal que comprende múltiples capas de modelos de regresión logística (con no linealidades continuas). Una de sus mayores virtudes es que el modelo resultante puede ser mucho más compacto, y, por lo tanto, más rápido a la hora de evaluar, pero es más lento a la hora de entrenar el modelo [19].

Los hiperparámetros que se prueban son:

- Número de neuronas en la capa oculta (hidden_layer_sizes).
- Función de activación para la capa oculta (activation):
  - "identity": La función lineal, $f(x) = x$.
  - "logistic": La función logística sigmoid, $f(x) = \frac{1}{1+e^{-x}}$.
  - "tanh": La función tangente hiperbólica, $f(x) = \tanh(x)$.
  - "relu": La función lineal rectificada, $f(x) = \max(0, x)$.
- El solucionador para la optimización de peso (solver):
  - "lbfgs": Método de optimización quasi-Newton.

- “sgd”: Descenso de gradiente estocásico.
- “adam”: Optimizador basado en el gradiente estocásico propuesto por Kingma, Diederik y Jimmy Ba [20].

| | | hidden_layer_sizes | | |
|---|---|---|---|---|
| Solver | activation | 20 | 50 | 100 |
| lbfgs | identity | 0.80 | 0.80 | 0.80 |
| | logistic | 0.68 | 0.76 | 0.84 |
| | tanh | 0.76 | 0.76 | 0.80 |
| | relu | 0.80 | 0.80 | 0.76 |
| sgd | identity | 0.72 | 0.72 | 0.68 |
| | logistic | 0.33 | 0.33 | 0.41 |
| | tanh | 0.68 | 0.72 | 0.72 |
| | relu | 0.68 | 0.64 | 0.72 |
| adam | identity | 0.72 | 0.72 | 0.72 |
| | logistic | 0.72 | 0.76 | 0.80 |
| | tanh | 0.80 | 0.80 | 0.80 |
| | relu | 0.80 | 0.80 | 0.80 |

***Tabla Capítulo 4.2.3.*** *Resultados del modelo usando diferentes parámetros para el algoritmo MLP.*

Los parámetros que maximizan la precisión son; solver: “lbfgs”, activation: “logistic” y hidden_layer_sizes: 100, la precisión obtenida ha sido del 84%

| | Precisión | Recall | F1-score | Nº Muestras |
|---|---|---|---|---|
| Caída | 0.73 | 0,89 | 0.80 | 9 |
| Nada | 1.00 | 0.80 | 0.89 | 10 |
| Recuperación | 0.83 | 0.83 | 0.83 | 6 |
| Media ponderada | 0.86 | 0.84 | 0.84 | 25 |

***Tabla Capítulo 4.2.4.*** *Resultados del modelo aplicando el algoritmo MLP ajustado.*

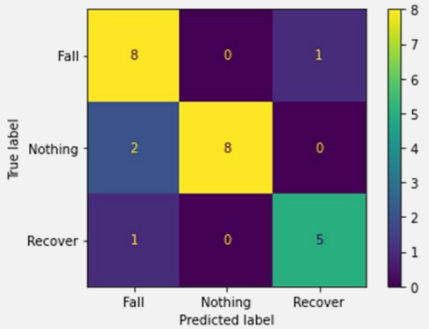



***Ilustración Capítulo 4.2.7****. Matriz de confusión del modelo enseñado con MLP.*

## Gradient Boosting Tree

El potenciador de gradientes, como el Random Forest entrena una secuencia de árboles de decisión [21].

Los hiperparámetros principales son:

- Número de etapas de mejora (n_estimators). Este modelo es robusto al sobreajuste, por lo que un número grande suele dar lugar a un mejor rendimiento [22].

- Reducción de contribución para cada árbol (learning_rate). Es una técnica para ralentizar el aprendizaje, es efectivo para reducir el sobreajuste.

Al disminuir el valor de learning_rate aumenta el mejor valor de n_estimators [23].

| | n_estimators | | | |
|---|---|---|---|---|
| learning_rate | 5 | 20 | 50 | 100 |
| 0.1 | 0.88 | 0.88 | 0.92 | 0.92 |
| 0.2 | 0.88 | 0.88 | 0.92 | 0.88 |
| 0.5 | 0.84 | 0.84 | 0.88 | 0.88 |
| 1 | 0.84 | 0.84 | 0.84 | 0.84 |

***Tabla Capítulo 4.2.5.*** *Resultados del modelo usando diferentes parámetros para el algoritmo Gradient Boosting Tree.*

Como hay varias configuraciones que coinciden con la mayor precisión (0.92), se escoge el que menos n_estimators y learning_rate tenga, de esta manera el hay menos riesgo de que se produzca un sobreajuste. Por lo tanto, el mejor resultado ha sido con learning_rate: 0.1, n_estimators: 50. Para reproducir el resultado se ha usado el número 20 como semilla para generar números aleatorios. La precisión obtenida con 25 muestras de prueba ha sido del 92%.

| | Precisión | Recall | F1-score | Nº Muestras |
|---|---|---|---|---|
| Caída | 0.89 | 0,89 | 0.89 | 9 |
| Nada | 1.00 | 0.90 | 0.95 | 10 |
| Recuperación | 0.86 | 1.00 | 0.92 | 6 |
| | | | | |
| Media ponderada | 0.93 | 0.92 | 0.92 | 25 |

***Tabla Capítulo 4.2.6.*** *Resultados del modelo aplicando el algoritmo Gradient Boosting Tree ajustado.*

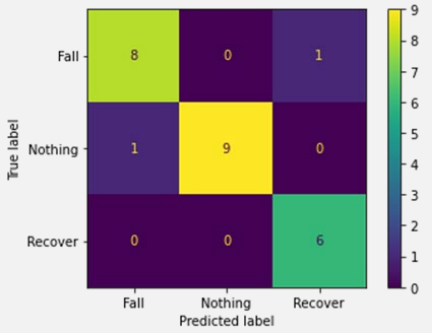



***Ilustración Capítulo 4.2.8****. Matriz de confusión del modelo enseñado con Gradient Boosting Tree.*

## Comparativa de resultados

Tras haber probado diferentes algoritmos, los resultados obtenidos varían poco con diferentes hiperparámetros y se parecen, incluso, usando diferentes algoritmos de aprendizaje. Esto es debido a que la cantidad de datos de entrenamiento y prueba son reducidos.

Se descarta el uso del algoritmo de aprendizaje MLP ya que da resultados significativamente peores, como se muestra en la *Tabla Capítulo 4.2.7*.

| | | Random Forest | MLP | Gradient Boosting |
|---|---|---|---|---|
| F1-score | Caída | 0.89 | 0.80 | 0.89 |
| | Nada | 0.95 | 0.89 | 0.95 |
| | Recuperación | 0.92 | 0.83 | 0.92 |
| | Media ponderada | 0.92 | 0.84 | 0.92 |

*Tabla Capítulo 4.2.7. Comparativa de resultados aplicando los algoritmos Random Forest, MLP y Gradient Boosting Tree.*

Para este conjunto de datos de prueba, Random Forest y Gradient Boosting ofrecen exactamente los mismos resultados, para salir de dudas, se vuelve a probar cada algoritmo con los mejores hiperparámetros pero sin especificar la semilla para generar números aleatorios. Como preferencia, se busca que de falsos positivos por encima de falsos negativos en caídas ya que las consecuencias si una persona se cae, pero el modelo da falso negativo pueden ser fatales.

Entonces, se realizan 100 pruebas y se elige el que menos falsos negativos tenga en total, como se muestra en la *Tabla Capítulo 4.2.8*.

| | Falsos Positivos | | Falsos Negativos | |
|---|---|---|---|---|
| Algoritmo | Casos | % | Casos | % |
| Random Forest | 203 | 8.12 | 100 | 4.00 |
| Gradient Boosting | 126 | 5.04 | 186 | 7.44 |

*Tabla Capítulo 4.2.8. Falsos positivos y Falsos negativos en caídas a partir de 100 pruebas con un conjunto de 25 series de datos cada una con Random Forest y Gradient Boosting.*

Se elige el algoritmo de aprendizaje Random Forest ya que, a pesar de que se obtendrán más falsos positivos que con Gradient Boosting, también se obtendrán menos falsos negativos, que es el objetivo.

Por último, una vez elegido el mejor algoritmo de aprendizaje, se vuelve a entrenar el modelo con todos los datos disponibles (tanto de entrenamiento como de validación) con el algoritmo elegido.

## 2.3. Aplicación del modelo HSC al programa

En primer lugar, se siguen los mismos pasos que en la sección 2 Capítulo 4, que es, obtener 5 fotogramas de vídeo por segundo, se extraen los *puntos clave* de cada fotograma, se normalizan los puntos y se serializan los datos.

Como lo que se quiere comprobar son trozos de 15 fotogramas, se definirá una estructura de datos de esa longitud. Para este proyecto, se ha decidido almacenar los puntos clave de cada fotograma a comprobar en una cola doblemente terminada (`collections.deque` en Python).

Para comprobar los trozos de vídeo, se han planteado tres opciones; fotograma a fotograma, cada 7 fotogramas o cada 15 fotogramas.

### Fotograma a fotograma

Es, sin duda, la opción más lenta y redundante, pero gracias a la abundancia de comprobaciones, a priori es menos probable que pase un falso negativo de caída, entonces, se haría una comprobación de 15 fotogramas por fotograma.

Tras hacer pruebas con este método, se decidió descartarlo ya que generaba demasiada información errónea.

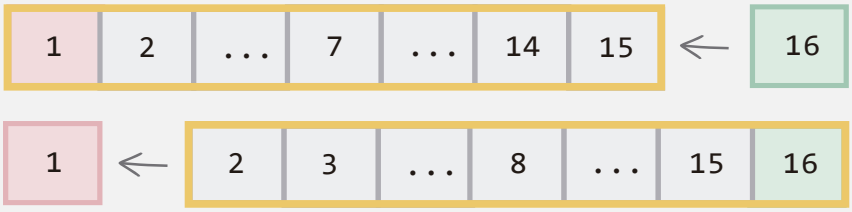



*Ilustración Capítulo 4.2.9. Sustitución de cada fotograma. Amarillo: cola, rojo: eliminar, verde: nuevo.*

### Cada 7 fotogramas

La mitad entera de la longitud de la cola, su objetivo es que si, por casualidad, el vídeo captado corte justo al empezar una caída y no se detecte, se elimina la primera mitad y se vuelve a comprobar la segunda mitad los 7 siguientes fotogramas. Como resultado, también genera información errónea.

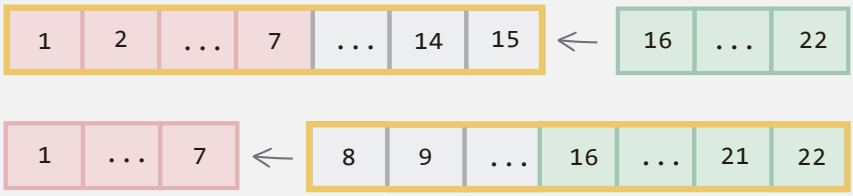



*Ilustración Capítulo 4.2.10. Sustitución cada 7 fotogramas. Amarillo: cola, rojo: eliminar, verde: nuevo.*

### Cada 15 fotogramas

La opción más rápida y la misma con la que se entrenó el modelo, es el que menos información errónea genera, con este método es más probable que se den falsos negativos de caída, pero gracias a la precisión del modelo, es la opción finalmente elegida.

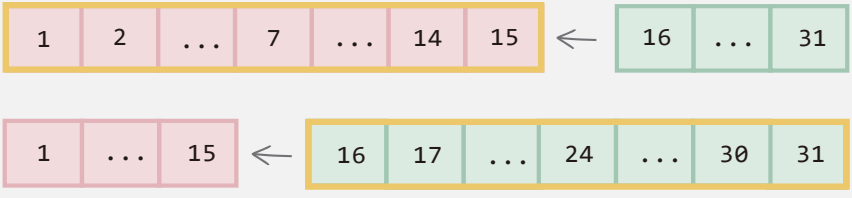



*Ilustración Capítulo 4.2.11. Sustitución cada 15 fotogramas. Amarillo: cola, rojo: eliminar, verde: nuevo.*

Una vez la cola esté completa, se serializa, y se usa en el modelo para determinar el estado.

# 3. Bot de Telegram

Con el objetivo de enviar un mensaje de forma autónoma cuando se detecta una caída o recuperación, se configura un Bot de Telegram, el proceso para crearlo es muy sencillo, se envía el mensaje `/newbot` al *Gestor de Bots* @botfather, es necesario guardar el *token* devuelto al crear el *Bot de Telegram*.

Con Telegram-send instalado, se introduce en la consola el comando: `telegram-send --configure`, se pone el *token y* nos facilita una contraseña que se envía al Bot (@detectordecaidas_bot), tras esto, el Bot está listo enviar mensajes y vídeos.

El método para enviar mensajes se ejecuta en un hilo aparte (un hilo en total para todas las cámaras) y únicamente el hilo está en ejecución si se detecta una caída o una recuperación. Se ha decidido implementarlo en otro hilo porque para enviar un vídeo por Telegram es necesario que el vídeo tenga formato, el proceso de dar formato a un vídeo es costoso y ralentiza el tiempo de ejecución, de esta manera, mientras se está dando formato y enviando el vídeo por Telegram, el resto del programa sigue ejecutándose con normalidad a tiempo real.

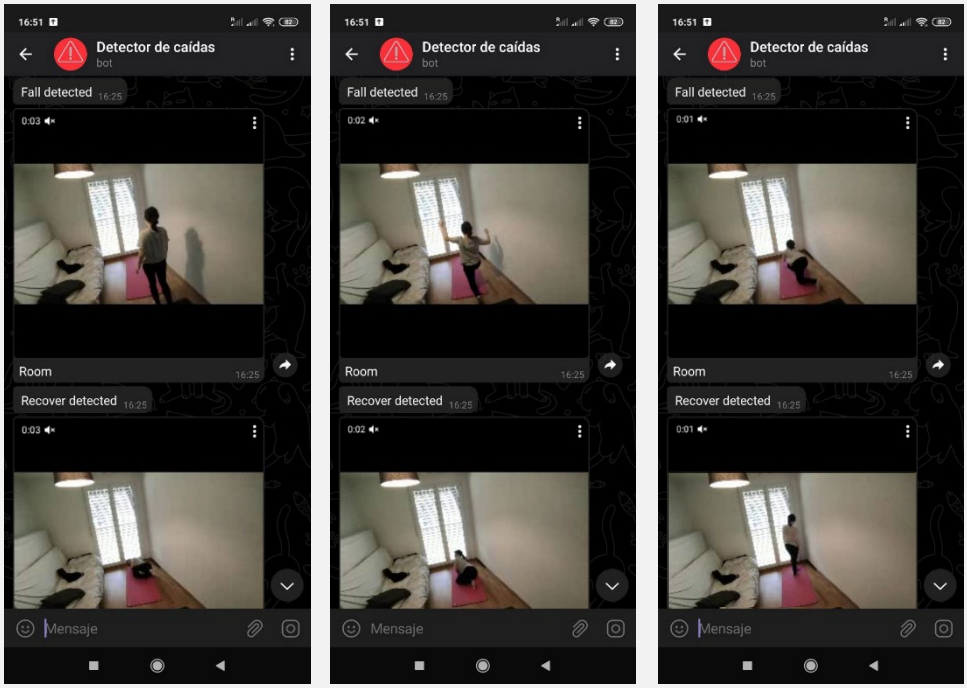



***Ilustración Capítulo 4.3.1**. Mensajes enviados por el bot. Previsualización de vídeos de izquierda a derecha.*

Lo primero que envía el *Bot de Telegram* es un mensaje de texto corto y conciso; *caída detectada: habitación*, con el objetivo de obtener ayuda lo más rápido posible, lo siguiente que se envía es el trozo de vídeo en el que se ha detectado la caída o la recuperación.

# 4. Programa principal

En esta sección se explica cómo se combinan todas las funcionalidades explicadas en este capítulo en un único programa. El archivo principal (main) es `fall_detector.py`.

En primer lugar, se ha creado el archivo `cameras.conf` que contiene el enlace a un video en directo o ruta a un vídeo almacenado y un nombre para identificarlo, cada línea del archivo es una cámara. El enlace o ruta se separa del nombre con un `", "` (coma espacio):

```
Enlace/ruta, Habitación
```

Para obtener el vídeo de las cámaras de la Smart Home, el enlace debe de tener la siguiente sintaxis:

```
rtsp://user:pass@ip:port/videoMain
```

El programa está preparado para recibir diferentes argumentos, como la ruta hacia `cameras.conf`, que, por defecto, está en la misma carpeta que el archivo principal o si recibir alertas en Telegram, por defecto a `True`.

Para vigilar diferentes cámaras, se ha decidido usar un hilo por cámara.

Los pasos que sigue el programa son:

- Obtener los fotogramas por segundos a los que trabaja cada cámara, se divide entre la cantidad de fotogramas que se quiere, y si el fotograma actual es múltiplo de 5, se redimensiona el fotograma y se almacena en la cola *`video_to_send`*.
- Obtener los *puntos clave* del fotograma gracias al modelo PoseNet y la clase PoseEstimation, se normaliza la escala de los puntos clave obtenidos y se almacenan en la cola *`video_keypoints`*.
- Si la cola *`video_keypoints`* contiene 15 elementos, entonces se comprueba con el modelo HSC si se trata de una caída o recuperación.
- De ser así, con otro hilo, se envía una alerta a Telegram con el vídeo almacenado en la cola *`video_to_send`*.
- Se repite el proceso.

Se puede ver el diagrama de flujo del programa en la *Ilustración Capítulo 4.4.1*.

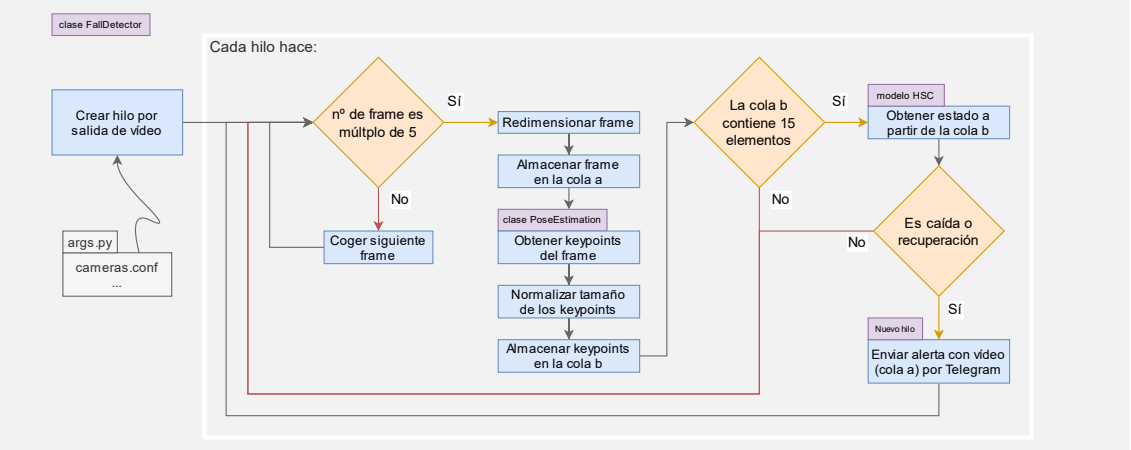



***Ilustración Capítulo 4.4.1**. Diagrama de flujo del proyecto "Detector de caídas".*

Capítulo 5

# Resultados obtenidos

Una vez se tiene la funcionalidad del programa, es importante asegurar que responde correctamente a tiempo real y con el menor retardo posible, en este capítulo se mostrará el rendimiento de las diferentes partes y en conjunto.

Las pruebas se realizan en un equipo que tiene 16Gb de RAM y procesador Intel Core i7-87450H.

## 1. Estimación de pose corporal

Para esta parte del programa, se mide el tiempo que tarda el algoritmo en redimensionar la imagen, obtener los puntos clave de la imagen y traducirlos a coordenadas.

El tiempo promedio de 387 muestras ha sido 0,05 segundos. Como se quieren obtener 5 fotogramas por segundo se estima que por cada segundo de vídeo tarde 0,25 segundos en realizar las operaciones. Esto quiere decir que **el algoritmo** no supone un retraso adicional y que **funciona perfectamente a tiempo real**, dejando 0,75 segundos para el resto de las operaciones.

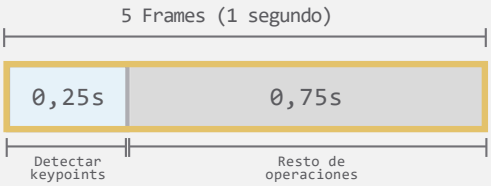



*Ilustración Capítulo 5.1.1. Tiempo usado para obtenerlas coordenadas de los puntos clave.*

## 2. Clasificador de estado humano (HSC)

En este apartado, se mide el tiempo que se tarda en normalizar los puntos y obtener la predicción.

El tiempo promedio de 387 muestras ha sido 0,01 segundos en el peor caso, es decir, cuando se usa el clasificador (una vez cada 3 segundos).

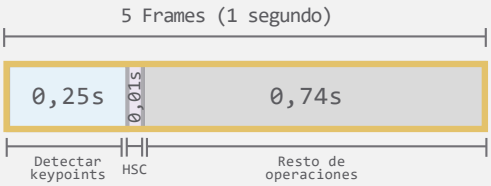



*Ilustración Capítulo 5.2.1. Tiempo usado para obtener el estado con el modelo HSC (peor caso).*

# 3. Bot de Telegram

En esta parte del programa, el tiempo usado depende sobre todo de la conexión a internet y el proceso de dar formato a un vídeo. El tiempo usado en el peor caso (cuando se detecta una caída o recuperación) es 0,57 segundos. Se estima que el resto del programa tarda adicionalmente 0,025 segundos (este tiempo ha sido obtenido midiendo el tiempo del programa completo y restándole las partes medidas anteriormente), con lo que se ajusta correctamente si se usa únicamente una cámara y se tiene buena conexión a Internet, ver la *Ilustración Capítulo 5.2.1*.

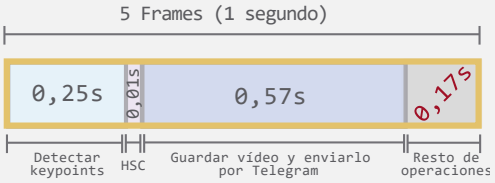



*Ilustración Capítulo 5.3.1. Tiempo usado para guardar el vídeo y enviarlo por Telegram (peor caso).*

En el caso de añadir más hilos (cámaras), que aumente la carga de la unidad de procesamiento o que haya mala conexión, es probable que se pase de los 0,17 segundos que tiene disponibles, dado este caso ya no sería a tiempo real, y es por eso que de dar formato al vídeo y enviar el vídeo por Telegram se hace en otro hilo, como se muestra en la *Ilustración Capítulo 5.2.2*.

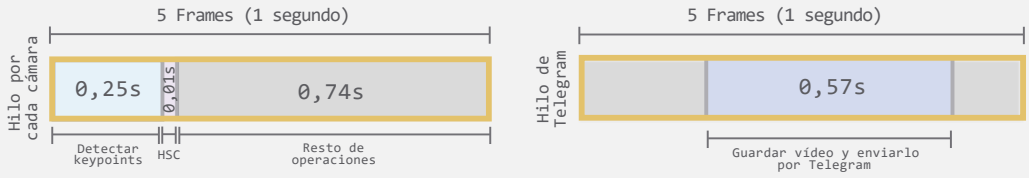



*Ilustración Capítulo 5.3.4. Tiempo estimado para cada hilo (peor caso).*

El hilo de Telegram solo está activo si alguna de las cámaras detecta una caída o recuperación. En el caso de que el hilo ya esté ocupado y otra cámara detecte una caída o recuperación, el segundo vídeo se queda en cola hasta que el hilo vuelva a estar libre.

# 4. Funcionalidad del programa

El programa es funcional y cumple su cometido a tiempo real, incluso, con varias cámaras a la vez.

Para conocer las diferentes configuraciones disponibles del programa, en la terminal, poner el comando: `py fall_detector.py --help`:

```
Real-Time Fall Detector

optional arguments:
  -h, --help            show this help message and exit
  -s CHUNK_SECONDS, --chunk_seconds CHUNK_SECONDS
                        Seconds for each chunk. (default: 3)
  -f FRAME_RATE, --frame_rate FRAME_RATE
                        Frames per second. (default: 5)
  -t TELEGRAM_ALERT, --telegram_alert TELEGRAM_ALERT
                        Sends an alert when a fall is detected. (default: True)
  -p POSENET_MODEL_PATH, --posenet_model_path POSENET_MODEL_PATH
                        Path to the PoseNet model. (default: ../Google Dataset (posenet)/posenet_mobilenet_v1_100_257x257_multi_kpt_stripped.tflite)
  -m FALL_MODEL_PATH, --fall_model_path FALL_MODEL_PATH
                        Path to the human state classifier model. (default: human_state_classifier/model/HSC.sav)
  -x PATH_MODEL_HEADER, --path_model_header PATH_MODEL_HEADER
                        Path to the human state classifier model header. (default: human_state_classifier/model/header.txt)
  -c PATH_CAMERAS, --path_cameras PATH_CAMERAS
                        Path to the configuration file containing the camera urls (default: cameras.conf)
  -d DISPLAY_VIDEO, --display_video DISPLAY_VIDEO
                        Display videos from cameras (default: False)
```

***Ilustración Capítulo 5.4.1**. Parámetros del programa Fall Detector.*

El parámetro `--display_video` permite visualizar las cámaras y el estado predicho por el modelo HSC en tiempo real, aunque está desactivado de forma predeterminada ya que ralentiza el proceso, pueden verse ejemplos en ejecución en la *Ilustración Capítulo 5.4.2*.

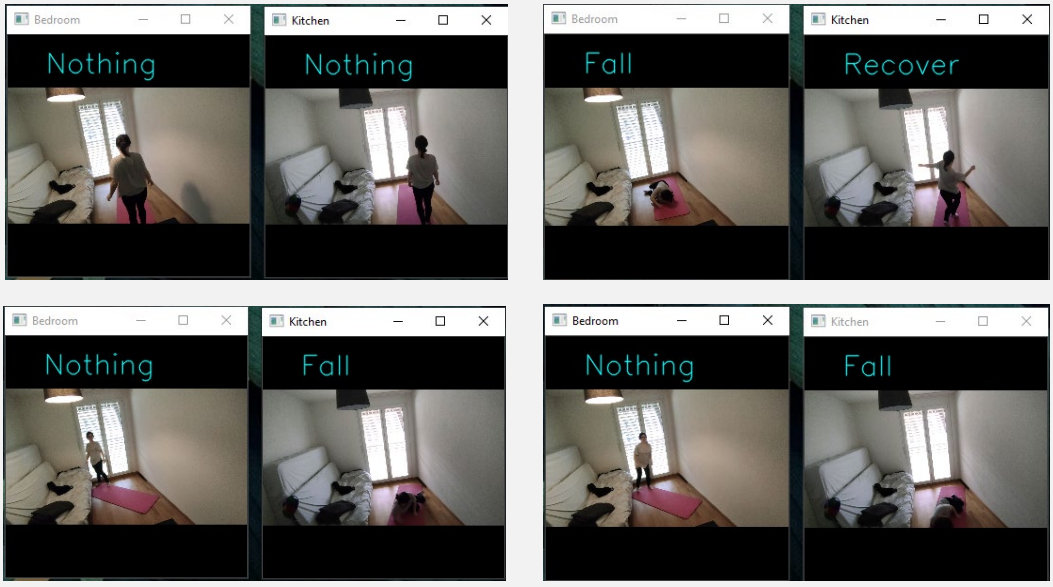



***Ilustración Capítulo 5.4.4**. Diferentes capturas del programa en ejecución con `--display_video True`.*

Por cada caída o recuperación detectada, el programa ha enviado un mensaje a través de Telegram como se mostró anteriormente en la *Ilustración Capitulo 4.3.1*.

Por último, en el caso que se quiera configurar los segundos por cada trozo o los fotogramas por segundo será necesario volver a entrenar el modelo HSC con parámetros por los que se quieren cambiar.

Capítulo 6

# Conclusión y mejoras

Para concluir, el modelo HSC funciona de manera muy acertada, pero existe riesgo de que haya sobreajuste por el bajo número de vídeos para entrenar el modelo y porque en todos los vídeos sale la misma persona. La solución para este problema es grabar muchos más videos con diferentes personas, es bueno tener en cuenta al grabar lo siguiente:

- Grabar con diferentes cámaras, esto es porque no todas las cámaras tienen la misma apertura, la posición de los puntos clave puede variar.
- Grabar desde muchos ángulos diferentes.
- Grabar a diferentes personas:
    - De diferente sexo, ya que las proporciones corporales son diferentes.
    - De diferente edad, porque, por ejemplo, las personas mayores tienen otro ritmo.

La razón por la que no se ha ampliado el conjunto de datos es debido a que todo el proyecto se ha realizado fuera de la Universidad de Almería y no se ha podido contar con la colaboración de otros voluntarios por la situación de pandemia.

También se puede mejorar la forma de ajustar los hiperparámetros en los algoritmos de aprendizaje, por ejemplo, eligiendo un rango de valores que tenga sentido para cada hiperparámetro y automatizando la comparación, así se pueden encontrar de manera mucho más eficiente y fiable.

Aparte de ajustar los hiperparámetros de forma eficiente, se pueden probar otros tipos de algoritmos neuronales más complejos que un perceptrón multicapa.

Otro factor para tener en cuenta es la aplicación de este programa en dispositivos menos potentes Para ello sería necesario estudiar mejoras de rendimiento como:

- Obtener directamente de las cámaras los fotogramas con la dimensión y los FPS deseados.
- Comenzar a almacenar fotogramas y obtener sus puntos clave únicamente si se detecta a un humano o movimiento.
- Enviar alerta de recuperación únicamente si se ha detectado una caída en alguna cámara previamente.

# Bibliografía


[1] Word Health Organization, "Falls," 26 April 2021. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/falls.

[2] F. Shu and J. Shu, "An eight-camera fall detection system using human fall pattern recognition via machine learning by a low-cost android box," *in Scientific reports,* vol. 11, p. 2471, Jan 2021, doi: 10.1038/s41598-021-81115-9.

[3] SpeedyAI, "On-device mobile AI computing," [Online]. Available: speedyai.com.

[4] W. Wang, D. Sun, P. Shao, H. Kuang and C. Sui, "Fully Bayesian Analysis of the Relevance Vector Machine Classification for Imbalanced Data," *in IEEE Access,* Jul 2020, doi: 10.1109/ACCESS.2021.3052935.

[5] Wikipedia, "Histogram of oriented gradients," Feb 2021. [Online]. Available: https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients.

[6] K. De Miguel, A. Brunete, M. Hernando and E. Gambao, "Home Camera-Based Fall Detection System for the Elderly," *in Sensors,* vol. 17, no. 12, p. 2864, Dec 2017, doi: 10.3390/s17122864.

[7] ScienceDirect, "Background Subtraction," 2018. [Online]. Available: https://www.sciencedirect.com/topics/engineering/background-subtraction.

[8] Y. Kim and H. Bang, "Introduction to Kalman Filter and Its Applications," *InTechOpen,* 2018.

[9] ScienceDirect, "Optical Flow," 2017. [Online]. Available: https://www.sciencedirect.com/topics/engineering/optical-flow.

[10] E. Auvinet, F. Multon, A. Saint-Arnaud, J. Rousseau and J. Meunier, "Fall Detection With Multiple Cameras: An Occlusion-Resistant Method Based on 3-D Silhouette Vertical Distribution," *in IEEE Transactions on Information Technology in Biomedicine,* vol. 15, no. 2, pp. 290-230, March 2011, doi: 10.1109/TITB.2010.2087385.

[11] L. Yang, R. Yanyun and W. Zhang, "3D depth image analysis for indoor fall detection of elderly people," *in Digital Communications and Networks,* vol. 2, no. 1, pp. 24-34, Feb 2016, doi: 10.1016/j.dcan.2015.12.001.

[12] R. Espinosa, H. Ponce, S. Gutiérrez, L. Martínez-Villaseñor, J. Brieva and E. Moya-Albor, "A vision-based approach for fall detection using multiple cameras and convolutional neural networks: A

case study using the UP-Fall detection dataset," *in Computers in Biology and Medicine,* vol. 115, Dec 2019. doi: 10.1016/j.compbiomed.2019.103520.

[13] C. Rougier, J. Meunier, A. St-Arnaud and J. Rousseau, "Robust Video Surveillance for Fall Detection Based on Human Shape Deformation," *in IEEE Transactions on Circuits and Systems for Video Technology,* vol. 21, no. 5, pp. 611-622, May 2011, doi: 10.1109/TCSVT.2011.2129370.

[14] N. Thome, S. Miguet and S. Ambellouis, "A Real-Time, Multiview Fall Detection System: A LHMM-Based Approach," *in IEEE Transactions on Circuits and Systems for Video Technology,* vol. 18, no. 11, pp. 1522 - 1532, Nov. 2008, doi: 10.1109/TCSVT.2008.2005606.

[15] Wikipedia, "Layered hidden Markov model," Oct 2018. [Online]. Available: https://en.wikipedia.org/wiki/Layered_hidden_Markov_model.

[16] S. Wang, L. Chen, Z. Zhou, X. Sun and J. Dong, "Human fall detection in surveillance video based on PCANet," *in Multimedia Tools and Appl,* vol. 75, no. 19, p. 11603–11613, 2015, doi: 10.1007/s11042-015-2698-y.

[17] G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler y K. Murphy, «Towards Accurate Multi-person Pose Estimation in the Wild,» *in Computer Vision and Pattern Recognition,* pp. 4-5, Jul. 2017, doi: abs/1701.01779.

[18] M. R. Segal, "Machine Learning Benchmarks and Random Forest Regression," *in UCSF: Center for Bioinformatics and Molecular Biostatistics,* vol. 18, no. 03, pp. 4-5, 2004.

[19] C. Bishop, Pattern Recognition and Machine Learning, Springer, Jan. 2006, pp. 225-241.

[20] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *Published as a conference paper at ICLR 2015,* Jan. 2017, doi: arXiv:1412.6980.

[21] L. Qinbin, W. Zeyi and H. Bingsheng, "Practical Federated Gradient Boosting Decision Trees," *in AAAI Technical Track: Machine Learning,* vol. 34, no. 4, Apr. 2020, doi: 10.1609/aaai.v34i04.5895.

[22] Scikit Learn, «sklearn.ensemble.GradientBoostingClassifier» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

[23] J. H. Friedman, «Greedy Function Aprpoximation: A Gradient Boosting Machine,» *in Technical Report, Department of Statistics,* p. 13, 1999.

[24] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Massachusetts: The MIT Press, 2016.

[25] TensorFlow, "Real-time Human Pose Estimation in the Browser with TensorFlow.js," 7 May 2018. [Online]. Available: https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5.

[26] Y. Shavit and R. Ferens, "Introduction to Camera Pose Estimation with Deep Learning," Jul. 2019, doi: 10.13140/RG.2.2.32079.23208.

UNIVERSIDAD DE ALMERÍA


## Resumen/Abstract

Las caídas son la segunda causa de muerte por lesiones no intencionadas en todo el mundo. Por este motivo es importante proporcionar ayuda a tiempo a las personas que han sufrido lesiones a causa de una caída, especialmente si son personas dependientes. Para solucionar este problema se ha decidido desarrollar un detector de caídas a tiempo real haciéndo uso de cámaras de vídeo que envía alertas por Telegram cuando se produce una caída. Existen otros detectores de caídas basados en imágenes, pero no son publicos o son de pago. Este sistema puede ser integrado en smart homes y otros lugares como residencias de ancianos. El detector transforma los fotogramas, incluso si son de baja resolución, en un modelo del cuerpo de la persona que aparece en el vídeo mediante el estimador de pose PoseNet. En base a los modelos de las personas en sucesivos instantes de tiempo, un algoritmo de aprendizaje computacional es capaz de detectar tanto las caídas que se producen como las posibles recuperaciones posteriores, si la persona es capaz de levantarse por sí misma. Con una precisión del 92%, este proyecto puede suponer una importante mejora en el tiempo que tarda una persona en recibir ayuda tras caerse.

Falls are the second leading cause of unintentional injury death worldwide. For this reason it is important to provide timely help to people who have suffered injuries from a fall, especially if they are dependent. To solve this problem, it has been decided to develop a real-time fall detector using video cameras that sends alerts via Telegram when a fall occurs. There are other image-based fall detectors, but they are either not public or paid. This system can be integrated into smart homes and other places such as nursing homes. The detector transforms the frames, even if they are low resolution, into a model of the person's body using the PoseNet pose estimator. A machine learning algorithm detects both falls and possible subsequent recoveries based on people's models in successive moments of time. With an accuracy of 92%, this project can be a significant improvement in the time it takes for a person to receive help after falling.