Escuela
Politécnica
Superior

Detección de caídas para dispositivos de bajo coste

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Sami Hadj Djilani
Tutor/es:
Miguel Angel Cazorla Quevedo
Francisco Gómez Donoso

Julio 2022

Universitat d'Alacant
Universidad de Alicante

Detección de caídas para dispositivos de bajo coste

Autor
Sami Hadj Djilani

Tutor/es
Miguel Angel Cazorla Quevedo
Ciencia de la Computación e Inteligencia Artificial
Francisco Gómez Donoso
Ciencia de la Computación e Inteligencia Artificial

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2022

Preámbulo

Con el aumento de la calidad de vida de las personas, las políticas sociales y las mejoras en medicina, cada vez las personas viven más tiempo. Por otro lado, el ritmo de vida actual muchas veces es incompatible con la responsabilidad de cuidar a nuestros familiares, que es común que vivan solos aún siendo muy mayores. Esto genera situaciones domésticas potencialmente dañinas para ellos. Por ejemplo, un tipo de accidente muy común en las casas de las personas mayores son las caídas. En este trabajo, se propone un sistema de detección de caídas capaz de correr en hardware de bajo coste con el objetivo de avisar a los familiares y minimizar el impacto de la posible caída.

Agradecimientos

Primero me gustaría agradecer a mis tutores Miguel y Fran toda la ayuda que me han proporcionado durante todo el curso para realizar este trabajo. Gracias por haber estado ahí en todo momento.

También me gustaría agradecer a mis padres el haber creído tanto en mí hasta cuando ni yo lo hacia. Asimismo le doy gracias a mi abuelo por su apoyo incondicional y por haber sido siempre un ejemplo a seguir. Además me gustaría agradecer a mi hermano el haberse llevado un golpe ayudándome en este trabajo.

Por último quiero dar las gracias a todos los compañeros y profesores que he conocido y que me han apoyado tanto a lo largo de estos 4 años de carrera.

Mucha gente pequeña,
en lugares pequeños,
haciendo cosas pequeñas,
puede cambiar el mundo

Eduardo Galeano.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

Como se ha comentado en el preámbulo este proyecto nace de la necesidad de proteger a nuestros mayores de un problema que se esta convirtiendo en un problema de salud pública, las caídas. Las caídas son la segunda causa mundial de muerte por traumatismos involuntarios, después de los accidentes de tráfico. Se estima que anualmente fallecen en todo el mundo aproximadamente **684.000** personas debido a las caídas. La mayoría de los casos graves/letales lo sufren los mayores de 60 años. Además, cada año se producen **37,3 millones** de caídas cuya gravedad requiere atención médica. Además el número muertes por caídas evitables va creciendo año tras año, como podemos ver en la Figura 1.1. [1]

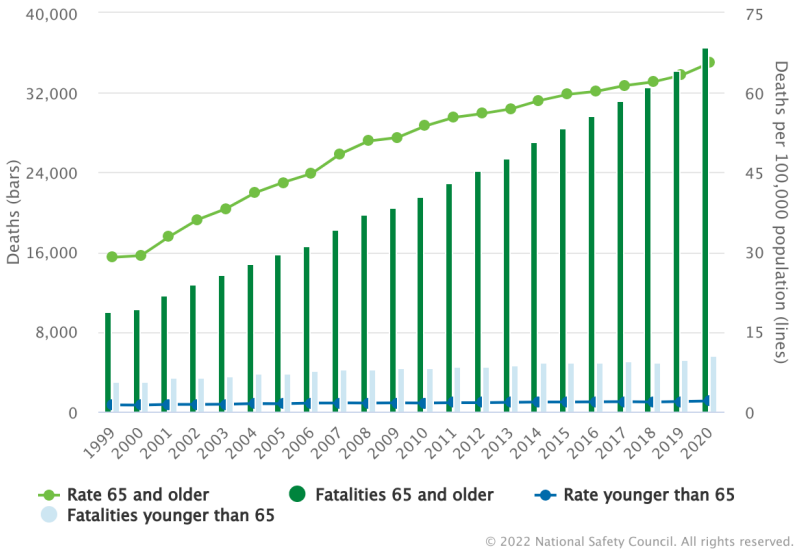

Figura 1.1: Muertes por caídas evitables 1999-2020

En España, aproximadamente una de cada 3 personas mayores de 65 años sufre una caída cada año, y esta cifra puede llegar al 50% en el caso de los mayores de 80 años [2].

Debido a todo lo anterior, se ha propuesto desarrollar un sistema de detección de caídas capaz de correr en hardware de bajo coste con el objetivo de avisar a los familiares o a los servicios de emergencia para poder actuar con la mayor brevedad posible y poder minimizar el impacto de la posible caída.

Para el desarrollo de este proyecto se ha propuesto un sistema de monitoreo basado en visión por computador y técnicas de Deep Learning.

En esta memoria documentaremos el proceso de trabajo llevado a cabo durante el desarrollo del proyecto, y está formado por:

- Objetivos, que consiste en una lista de objetivos que se pretenden conseguir con este trabajo.
- Marco teórico, donde introduciremos algunos de los conceptos más importantes y conocimientos previos que se deben tener para entender/desarrollar este proyecto.
- Estado del arte, donde hablaremos de las aproximaciones existentes que realizan una tarea similar a la nuestra.
- Metodología, donde describiremos las herramientas que se usaremos para desarrollar nuestro proyecto (frameworks, librerías y datasets).
- Desarrollo, donde hablaremos de todas las etapas que hemos seguido para el desarrollo de nuestro proyecto.
- Experimentación y resultados, donde describiremos todos los experimentos que hemos realizado y analizaremos los resultados que nos han dado. También hablaremos de las métricas que hemos usado para evaluar los resultados de nuestro sistema.
- Conclusión y trabajos futuros, donde comentaremos los resultados que hemos conseguido en el desarrollo de este proyecto, además de posibles mejores.

2 Objetivos

El objetivo de este proyecto es proponer un método de detección de caídas mediante tecnologías de deep learning analizando secuencias de vídeo, para ello nos hemos marcado los siguientes objetivos:

- Realizar un estudio sobre las tecnologías actualmente usadas para la detección de caídas.
- Proponer un método para la detección de caídas basado en Deep Learning.
- Evaluar los resultados obtenidos del método propuesto, mediante diferentes métodos de evaluación.
- Implementar una programa para usar el modelo propuesto.

3 Marco teórico

En este apartado vamos a introducir algunos de los conceptos más importantes y conocimientos previos que se deben tener para entender/desarrollar este proyecto.

3.1 Deep Learning

El **aprendizaje profundo** es un subcampo del aprendizaje automático que hace uso de los algoritmos inspirados en la estructura y el funcionamiento del cerebro llamados redes neuronales artificiales. Podemos ver el posivionamiento del Deep Learning en la inteligencia artificial en la Figura 3.1

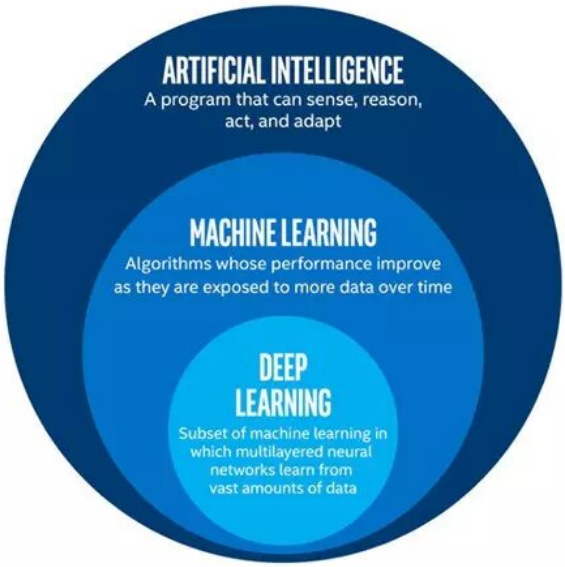

Figura 3.1: Diferencia entre IA, Machine Learning y Deep Learning

El deep learning permite resolver problemas de percepción, como el reconocimiento de imágenes, del habla... Los métodos de aprendizaje profundo, como las redes neuronales artificiales profundas, utilizan múltiples capas de procesamiento para descubrir patrones y estructuras en conjuntos de datos muy grandes. Cada capa aprende un concepto de los datos sobre el que se basan las capas siguientes; cuanto más alto es el nivel, más abstractos son los conceptos que se aprenden. [3]

Un ejemplo sencillo seria una red neuronal profunda encargada de interpretar formas, la primera capa aprendería a reconocer bordes simples y luego añadiría el reconocimiento de las

formas más complejas compuestas por esos bordes en las siguientes capas.

3.2 Redes neuronales

Las redes neuronales son actualmente la forma más usada para construir soluciones de aprendizaje profundo. Como hemos dicho antes, el concepto de aprendizaje profundo imita el funcionamiento del cerebro humano mediante la creación de una red artificial de neuronas, que consta de muchas capas de neuronas para procesar con precisión una gran cantidad de información. Nuestro cerebro está formado por neuronas, que trabajan juntas para formar nuestro cerebro. El cerebro de un adulto medio está formado por 100.000 millones de neuronas, que trabajan de forma unificada para controlar el cuerpo y la mente humanos. Las redes neuronales artificiales emulan esta estructura del cerebro humano en su forma de creación, sustituyendo así las neuronas orgánicas por funciones matemáticas y al igual que el cerebro, en una red neuronal artificial hay muchas neuronas, cada una con una función específica que clasifica los datos dados al programa. Estas neuronas artificiales están estructuradas en capas, y los datos pasan por cada capa para ser interpretados por la máquina. Una red neuronal artificial suele estar formada por muchas capas de neuronas, también conocidas como nodos. Estos nodos están programados con ”pesos”, que actúan como un filtro para los datos que pasan por los nodos. Normalmente, estos pesos dictan cómo deben procesar los datos los nodos de la red, y cada nodo amplifica o suprime la importancia de la entrada. La ponderación exacta de las redes neuronales da lugar a un resultado preciso, ya que los nodos acabarán aprendiendo a qué datos merece la pena darles importancia y a cuáles no. A medida que pasa el tiempo, las entradas suprimidas no pasarán por las primeras capas de la red, ya que cada capa filtra un único campo de los datos. Las primeras capas de nodos de una red neuronal se conocen como capa de entrada, y las últimas se denominan capa de salida, tal y como se puede apreciar en la Figura 3.2.

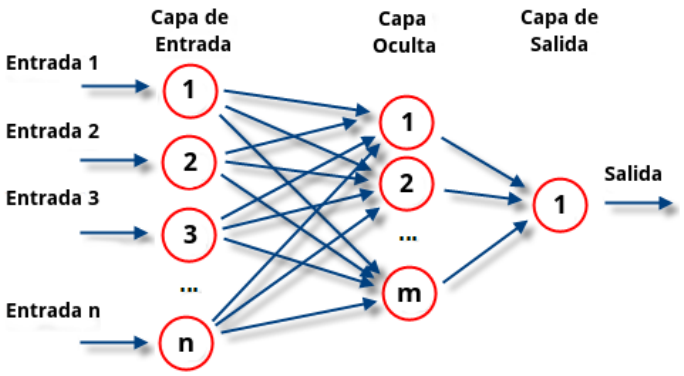

Figura 3.2: Estructura de una red neuronal artificial

Entre estas dos capas, hay una capa oculta. Esta capa se llama así porque la entrada o la salida de esta capa no se pueden ver. Las redes neuronales son excelentes para encontrar patrones en los datos, reconocer la relación entre variables, agrupar y clasificar.[4]

3.2.1 Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal que se basa en la estructura del cerebro animal, más concretamente en la corteza visual, ya que el patrón de conexión entre los nodos de la red es similar al de las neuronas del córtex visual. Se conocen como redes "convolucionales", ya que emplean una operación matemática conocida como convolución (Figura 3.3). [5]

En el contexto de las CNN, la convolución es una operación lineal que implica la multiplicación de un conjunto de pesos con las imágenes de entrada representadas por métricas similares a las redes neuronales tradicionales. En este caso, un conjunto de pesos se denomina filtro o kernel. Los strides definen el movimiento del filtro; si se establece stride=1, que es el valor por defecto, el núcleo da un paso cada vez.

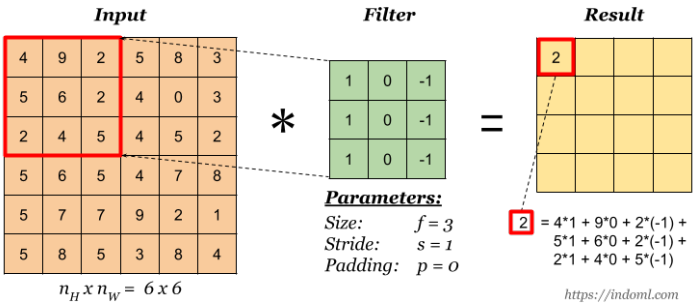

Figura 3.3: Operación de convolución

Normalmente, el tamaño del filtro es menor que el de los datos de entrada, y la multiplicación de tipo que se aplica entre el filtro y la muestra del tamaño del filtro de los datos de entrada es el producto punto. Un producto de puntos es una multiplicación por elementos entre los pesos del filtro y la muestra del tamaño del filtro de los datos de entrada, sumados en un único valor.

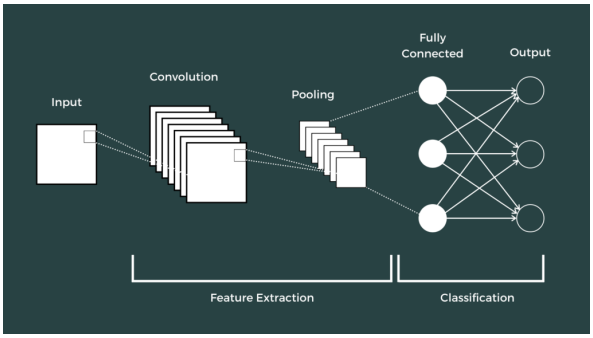

Figura 3.4: Diagrama de la arquitectura de una CNN.

La CNN es suele estar compuesta por tres tipos de capas, una capa de convolución, pooling y capas totalmente conectadas, tal y como se muestra en la Figura 3.4. Las dos primeras de convolución y pooling, realizan la extracción de características, mientras que la tercera, una capa totalmente conectada, convierte las características extraídas en un resultado final.

3.2.2 Redes neuronales recurrentes (RNN)

Una red neuronal recurrente es una red neuronal que contiene unidades ocultas capaces de analizar flujos de datos, se puede ver la arquitectura de una RNN en la Figura 3.5. Esto es importante en varias aplicaciones en las que la salida depende de los cálculos anteriores, como el análisis de texto, habla, secuencias de ADN, entre otras cosas. La RNN suele alimentarse con muestras de entrenamiento que tienen fuertes interdependencias y una representación significativa para mantener la información sobre lo que ocurrió en todos los pasos temporales anteriores, es decir, el resultado obtenido por la red en el tiempo **t-1** afecta a la elección en el tiempo **t**. De este modo, las RNN explotan dos fuentes de entrada, el presente y el pasado reciente, para proporcionar la salida de los nuevos datos y es por esta razón que se suele decir que las RNN tienen memoria [6].

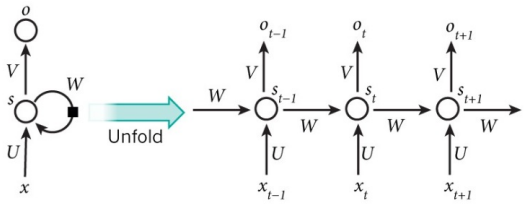

Figura 3.5: Arquitectura de una RNN

Una variación de las RNNs son las Long short-term memory (LSTMs), podemos ver su arquitectura en la Figura 3.6. Las LSTM son especialmente adecuadas para aplicaciones en las que existen desfases temporales muy largos de tamaños desconocidos entre eventos importantes. Para ello, las LSTM explotan nuevas fuentes de información, de modo que los datos pueden almacenarse, escribirse o leerse de un nodo en cada paso. Durante el entrenamiento, la red aprende qué almacenar y cuándo permitir la lectura/escritura para minimizar los errores de clasificación.

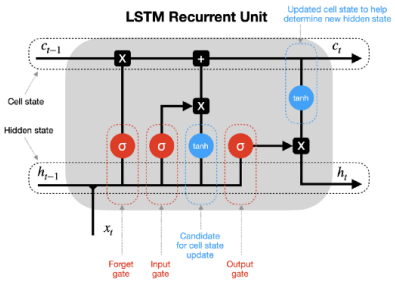

Figura 3.6: Arquitectura de una Long Short-Term Memory

A diferencia de otros tipos de redes neuronales profundas que utilizan pesos diferentes en cada capa, una RNN o una LSTM comparten los mismos pesos en todos los pasos. Esto reduce en gran medida el número total de parámetros que la red necesita aprender.

3.2.3 Transfer Learning

Entrenar una red neuronal profunda desde cero no suele ser factible por varias razones, una de ellas es que se necesita un conjunto de datos bastante grande y alcanzar la convergencia puede llevar demasiado tiempo para que los experimentos merezcan la pena. El ajuste de los pesos de una red preentrenada al continuar con el proceso de entrenamiento es uno de los principales escenarios del aprendizaje por transferencia. Tenemos un ejemplo en la Figura 3.7

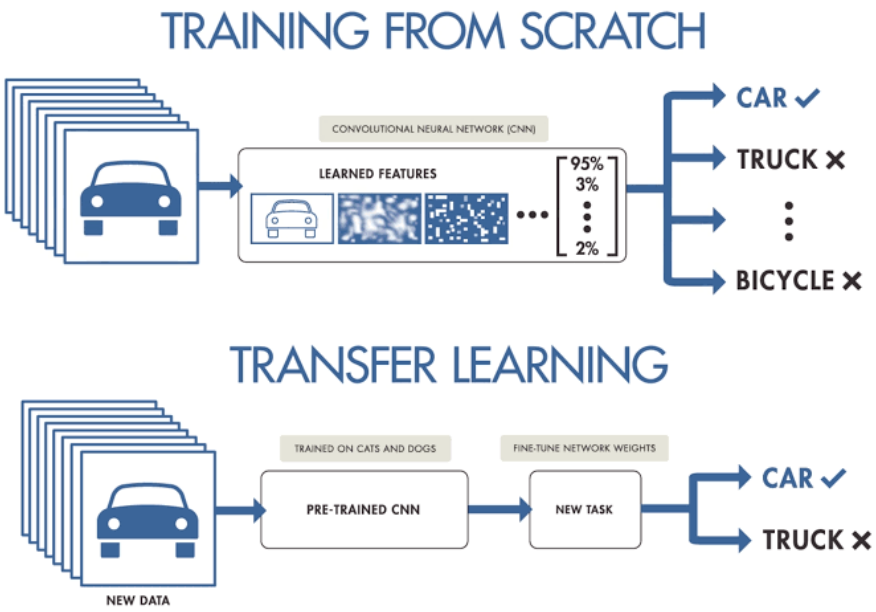

Figura 3.7: Diferencia entre entrenar desde 0 y transfer learning.

La aplicación de esta técnica de aprendizaje por transferencia no es del todo sencilla. Por un lado, existen restricciones arquitectónicas que deben cumplirse para utilizar una red preentrenada. Sin embargo, dado que no es habitual idear una arquitectura completamente nueva, es habitual reutilizar arquitecturas (o componentes) de red ya existentes, lo que permite el aprendizaje por transferencia. Por otro lado, el proceso de entrenamiento difiere ligeramente cuando se realiza un ”fine-tuning” en lugar de un entrenamiento desde cero. Es importante elegir adecuadamente las capas que se van a ajustar, normalmente la parte más alta de la red, ya que la más baja suele contener características más genéricas, también hay elegir una política adecuada para la tasa de aprendizaje, que suele ser menor debido a que se espera que los pesos preentrenados sean relativamente buenos, por lo que no es necesario cambiarlos drásticamente. [7]

Un modelo preentrenado se suele entrenar en un enorme conjunto de datos. Si se quiere usar un modelo preentrenado para la clasificación de imagenes se suele usar un modelo preentrenado con **ImageNet**, un dataset que contiene mas de 14 millones de imagenes organizadas en 20.000 clases [8]. En nuestro caso, al tratarse de un problema de clasificación de videos, disponemos de otros datasets como los son **Kinetics-400-600-700** y **UCF101**.

1. **VGG**: Visual Geometry Group (VGG) es un modelo de CNN introducido por el Visual Geometry Group (VGG) de la Universidad de Oxford.

   Propusieron varios modelos y configuraciones de CNN profundas, uno de los cuales se presentó al ImageNet Large Scale Visual Recognition Challenge (ILSVRC)-2013. Ese modelo, también conocido como VGG-16 debido a que está compuesto por 16 capas de peso, se hizo popular gracias a que alcanzó el 92,7% de precisión en la prueba TOP-5.

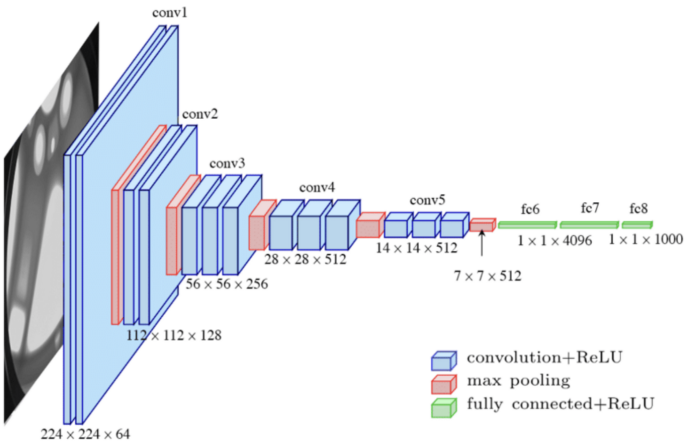

Figura 3.8: Arquitectura de la VGG16.

   La principal diferencia entre VGG-16 y sus predecesores es el uso de una pila de capas de convolución con pequeños campos receptivos en las primeras capas en lugar de pocas capas con grandes campos receptivos, como se muestra en la Figura 3.8. Esto hace que haya menos parámetros y más no linealidades entre ellos, lo que hace que la función de decisión sea más discriminativa y que el modelo sea más fácil de entrenar. [9]

2. **Resnet50**: Es una red neuronal convolucional de 50 capas de profundidad como podemos ver en la Figura 3.9. ResNet, abreviatura de Residual Networks, es una red neuronal clásica que se utiliza como columna vertebral en muchas tareas de visión por ordenador. El avance fundamental de ResNet fue que nos permitió entrenar redes neuronales extremadamente profundas con más de 150 capas.

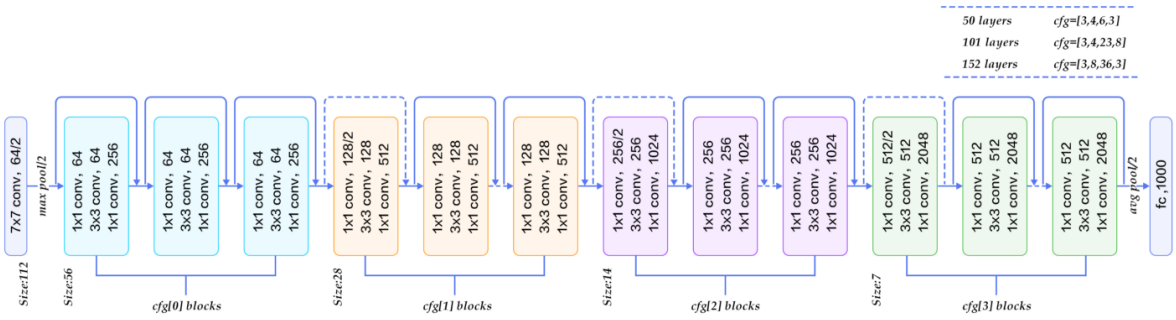

Figura 3.9: Arquitectura de la Resnet50.

Las redes residuales, o ResNets, aprenden funciones residuales con referencia a las entradas de las capas, en lugar de aprender funciones sin referencia. En lugar de esperar que cada una de las capas apiladas se ajuste directamente a un mapeo subyacente deseado, las redes residuales dejan que estas capas se ajusten a un mapeo residual. Apilan los bloques residuales uno encima de otro para formar la red. [10]

3. **Inception**: Es una red neuronal convolucional para creada para ayudar en el análisis de imágenes y la detección de objetos. Comenzó como un módulo para Googlenet. Actualmente existen 4 versiones de esta red, la mas usada es la tercera edición, presentada originalmente durante el reto de reconocimiento de ImageNet.

   Inception v3 es un modelo de reconocimiento de imágenes ampliamente utilizado que ha demostrado alcanzar una precisión superior al 78,1% en el conjunto de datos ImageNet y alrededor del 93,9% de precisión en los 5 primeros resultados. El modelo es la culminación de muchas ideas introducidas por múltiples investigadores en los últimos años. Su diseño estaba pensado para permitir redes más profundas (mirese Figura 3.10) y, al mismo tiempo, evitar que el número de parámetros creciera demasiado, ya que tiene menos de 25 millones de parámetros, frente a los 60 millones de AlexNet.

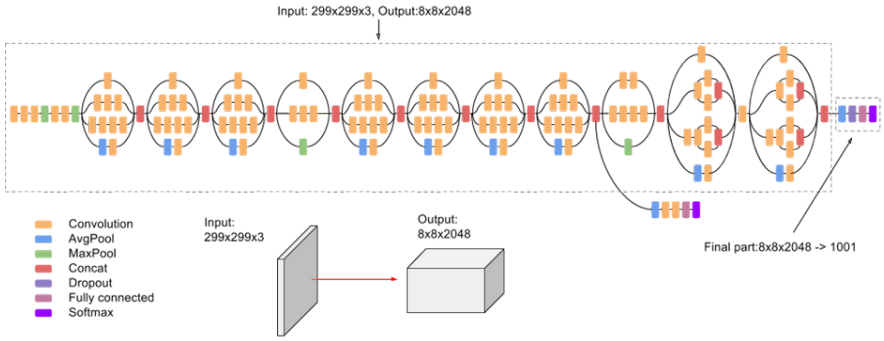

Figura 3.10: Arquitectura de la red Inception.

   La arquitectura de Inceptionv3 se ha reutilizado en muchas aplicaciones diferentes, a menudo utilizadas ”preentrenadas” a partir de ImageNet. Uno de estos usos es en las ciencias de la salud, donde ayuda en la investigación de la leucemia.

4. **EfficientNet**: Este modelo fue propuesto por Mingxing Tan y Quoc V. Le, del equipo Google Research, Brain, en la **International Conference on Machine Learning**, 2019. Estos investigadores estudiaron el escalamiento del modelo e identificaron que equilibrar cuidadosamente la profundidad, el ancho y la resolución de la red puede conducir a un mejor rendimiento y basándose en esta observación, propusieron un nuevo método de escalado que escala uniformemente todas las dimensiones de profundidad, anchura y resolución de la red. Podemos observar la arquitectura de EfficientNet B0 en la Figura 3.11.

   Utilizaron la búsqueda de arquitecturas neuronales para diseñar una nueva red de referencia y la escalaron para obtener una familia de modelos de aprendizaje profundo,

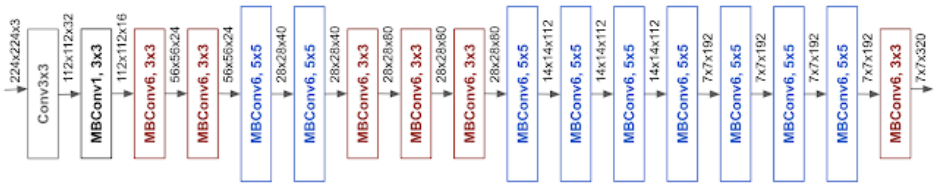

Figura 3.11: Arquitectura de la EfficientNet B0.

denominada EfficientNets, que consigue una precisión y una eficiencia mucho mejores en comparación con las anteriores redes neuronales convolucionales.

3.3 Flujo óptico

El flujo óptico es el movimiento de los objetos entre fotogramas consecutivos de la secuencia, causado por el movimiento relativo entre el objeto y la cámara. Ejemplo en la Figura 3.12

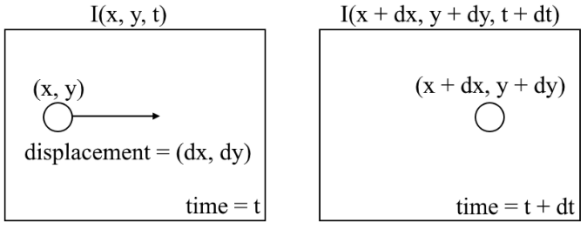

Figura 3.12: Concepto del flujo óptico.

Existen 2 tipos de flujo óptico que podemos observar en la figura 3.13, **disperso**, que proporciona los vectores de flujo de algunas características interesantes, por ejemplo, unos pocos píxeles que representan los bordes o las esquinas de un objeto dentro del fotograma, mientras que el flujo óptico **denso** proporciona los vectores de flujo de todo el fotograma (todos los píxeles), hasta un vector de flujo por píxel. Como es de suponer, el flujo óptico denso tiene una mayor precisión a costa de ser lento y costoso a nivel computacional.

Figura 3.13: Ejemplo de flujo óptico disperso (izquierda) y flujo óptico denso (derecha).

4 Estado del Arte

El problema de la detección de caídas es un problema que ha sido ampliamente estudiado. Podemos catalogarlo dentro del reconocimiento de actividad humana que ha su vez podemos catalogarlo como un problema de clasificación de vídeos (Como podemos ver en la Figura 4.1). Para la solución de este problema se han presentado una gran cantidad de aproximaciones, por ejemplo, los sistemas basados en sensores, basados en visión y multimodales.

Figura 4.1: Esquema del posicionamiento de la detección de caídas.

4.1 Sistemas de detección de caídas basados en sensores

Los enfoques basados en sensores hacen uso del ambiente, dispositivos inteligentes y sensores portátiles para proporcionar información importante, como aceleración (Ejemplo en la Figura 4.2), ausencia/presencia de personas... Tenemos el ejemplo de la **Universidad de Grenoble** que diseñó un sensor autónomo colocado debajo de la axila, que detecta cuando la velocidad supera un umbral específico, la secuencia de una postura vertical a la postura acostada y la ausencia de movimientos después de la caída. El dispositivo logró una sensibilidad y especificidad cercana al 85% a partir de 15 escenarios de caída realizados 5 veces por 5 personas [11].

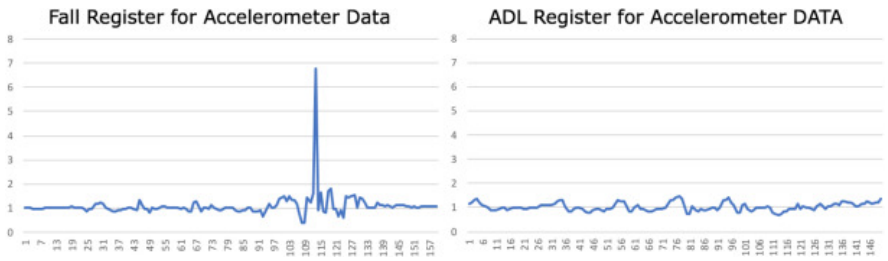

Figura 4.2: Comparación de 2 lecturas de un acelerómetro, una caída(izquierda) y una actividad cotidiana(derecha).

4.2 Sistemas de detección de caídas basados en la visión

Las estrategias basadas en visión usan imágenes, como reconstrucciones 3D del entorno, secuencias de vídeo RGB 2D simples con una o varias cámaras, o imágenes de profundidad adquiridas de sensores de profundidad 3D, como entrada principal. La mayoría de los enfoques basados en la visión utilizan cámaras RGB simples, cámaras web, sistemas de cámara de movimiento o incluso Kinects. El uso de un dispositivo Kinect para la detección de caídas ha aumentado porque puede obtener información en 3D, como poses humanas o posiciones de las extremidades.

La mayoría de sistemas de detección de caídas basadas en la visión constan de cuatro fases, adquisición de datos de secuencias de vídeo, extracción de características de imágenes, selección de características y aprendizaje. Un ejemplo del flujo de trabajo es la Figura 4.3.

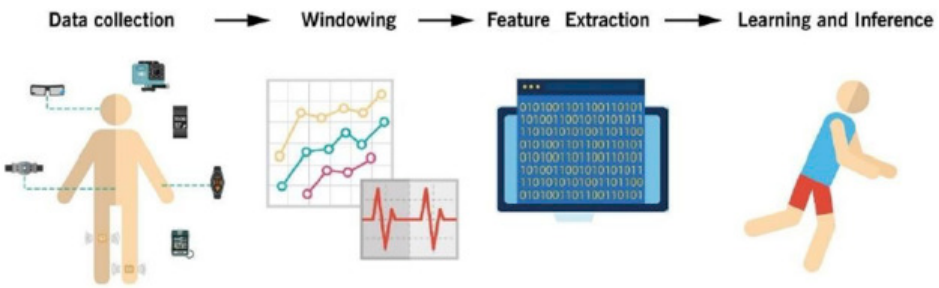

Figura 4.3: Flujo de trabajo tradicional para sistemas de detección de caídas basado en visión.

Las técnicas de aprendizaje profundo actualmente están cambiando y mejorando los métodos utilizados para abordar los problemas de visión por computadora. Las CNN aprenden automáticamente las características de los datos de entrenamiento, creando así un método factible de extracción automática de características para las imágenes. Por ejemplo los autores utilizan el aprendizaje profundo para detectar accidentes usando el flujo óptico como método de extracción de características y luego prueban este método usando vídeos reales.Se entrenó una sola CNN usando imágenes para clasificar y detectar directamente los accidentes, y se logró un área bajo la curva (AUC) de 0,96%. Y en la detección de caídas, las CNN se han utilizado con un enfoque basado en sensores, se ha logrado una precisión de hasta el 92,3% y con un enfoque portátil con un AUC de 0,75.

Existen diferentes técnicas de detección de caídas al usar sistemas de detección de caídas basados en visión, por ejemplo, un equipo canadiense propuso un sistema de detección de caídas basado en las variaciones de la silueta humana en el control de la visión y máquinas de vectores de soporte (SVM) para identificar posturas . Luego, estos autores utilizaron un modelo ocultos de Markov (HMM) para clasificar los datos en eventos de caída y no caída [12]. Su aproximación fue rastrear la silueta de la persona junto con las secuencias de vídeo, a continuación, cuantificar la deformación de la forma a partir de estas siluetas basándose en métodos de análisis de forma y finalmente, se detectaron caídas de las actividades diarias utilizando modelos de mezcla gaussiana (GMM). Con esta aproximación los autores lograron un 98% de precisión en laboratorio.

También tenemos el experimento publicado en el **IEEE** [13] donde los autores implementaron una CNN para un enfoque basado en la visión, en el que los autores utilizaron una CNN 3D con vídeos de entrada de la cinemática de las personas del dataset **Sports-1M** y en el conjunto de datos de caída de múltiples cámaras, se pudo obtener una precisión del 100%, que es el mejor resultado jamás informado.

4.3 Sistemas de detección de caídas basados en multimodales

Los enfoques multimodales recopilan toda la información posible de cámaras, micrófonos, sensores portátiles, sensores ambientales y dispositivos inteligentes, entre otros, y combinan toda esta información para mejorar la detección de caídas y la clasificación resulta de manera práctica, ya que los "wearables" no son capaces de distinguir una gran cantidad de actividades humanas detalladas y complejas, ya que tienen dificultad para diferenciar entre actividades similares, por lo que se necesitan sensores ambientales para la conciencia del contexto. En este sentido, los enfoques multimodales pueden combinar más de una fuente de datos para obtener información tanto sobre el entorno como sobre el usuario.

Estos enfoques hacen factible la detección y clasificación de caídas al aprovechar diferentes modos de detección de una amplia gama de fuentes. Aún así existen algunas debilidades, ya que muchos tipos de información requieren la aplicación de técnicas sólidas de extracción y selección de características y consideraciones sobre enfoques de aprendizaje automático para diferentes tipos de datos de entrada, lo que hace que la detección de caídas sea computacionalmente costosa y difícil de realizar. Además poniéndolo en contexto, como estos sistemas de detección de caídas están pensadas generalmente para gente mayor, el uso de múltiples sensores con una colocación compleja en el cuerpo y en el medio ambiente podría generar dificultades prácticas y molestias para las personas mayores, además de altos costos.

5 Metodología

En este capitulo hablaremos de las diferentes herramientas, librerías, frameworks... que se han usado para el desarrollo de este proyecto.

5.1 Librerías

El lenguaje que hemos usado para el desarrollo de este proyecto ha sido Python, ya que dispone de una amplia gama de herramientas que facilitan el trabajo de desarrollar soluciones de Deep Learning.

5.1.1 Tensorflow y Keras

TensorFlow es una biblioteca de código abierto desarrollada por Google principalmente para aplicaciones de aprendizaje profundo. También es compatible con el aprendizaje automático tradicional. Se desarrolló originalmente para grandes cálculos numéricos sin tener en cuenta el aprendizaje profundo, sin embargo, ha demostrado ser muy útil para el desarrollo del aprendizaje profundo y, por lo tanto, Google lo ha hecho de código abierto. TensorFlow acepta datos en forma de matrices multidimensionales de dimensiones superiores llamadas tensores. Las matrices multidimensionales son muy útiles para manejar grandes cantidades de datos.

Keras es una biblioteca de código abierto de Python potente y fácil de usar para desarrollar y evaluar redes neuronales con facilidad. Forma parte de la biblioteca TensorFlow (Figura 5.1) y permite definir y entrenar modelos de redes neuronales en tan solo unas líneas de código.

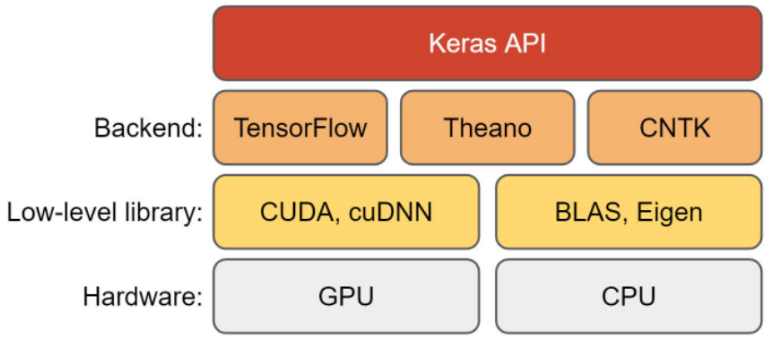

Figura 5.1: La pila de software y hardware de aprendizaje profundo con keras.

5.1.2 NumPy

NumPy es una biblioteca de Python utilizada para trabajar con arrays multidimensionales. NumPy fue creado en 2005 por Travis Oliphant. Es un proyecto de código abierto y se puede utilizar libremente. En Python tenemos listas que sirven para el mismo propósito, pero son mucho mas lentas de procesar. NumPy pretende proporcionar un objeto array que sea hasta 50 veces más rápido que las listas tradicionales de Python. El objeto array en NumPy se llama ndarray, proporciona un montón de funciones de apoyo que hacen que trabajar con ndarray sea muy fácil y como los arrays se utilizan con mucha frecuencia en deep learning, donde la velocidad y los recursos son muy importantes.

5.1.3 OpenCV

OpenCV es la gran biblioteca de código abierto para la visión por ordenado y el procesamiento de imágenes, actualmente desempeña un papel fundamental en el funcionamiento en tiempo real, que es muy importante en los sistemas actuales. Mediante su uso, se pueden procesar imágenes y vídeos para identificar objetos, rostros o incluso la escritura de un ser humano. tiene más de 2.500 algoritmos optimizados, como puede ser el algoritmo que optiene el Optical Flow de 2 imagenes, algoritmo que usaremos posteriormente en el desarrollo. Cuando creamos aplicaciones de visión por ordenador que no queremos construir desde cero, podemos utilizar esta biblioteca para empezar a centrarnos en los problemas del mundo real.

5.1.4 KerasTuner

Los algoritmos de optimización suelen tener algunos hiperparametros que debemos ajustar para que funcionen mejor. Para ajustar estos valores vamos a usar KerasTuner, un marco de optimización de hiperparámetros escalable y fácil de usar que resuelve los puntos débiles de la búsqueda de hiperparámetros y busca fácilmente las configuraciones óptimas para el modelo ANN. Además KerasTuner tambien es una buena herramienta si quisiésemos ajustar el modelo en si, como por ejemplo determinar el numero de capas.

5.2 Datasets:

Para el desarrollo de este proyecto hemos usado 4 datasets, 3 para entrenar y probar nuestras redes (conjuntos de train y test) y otro para validar:

5.2.1 Fall detection dataset ImViA:

Se trata de un Dataset de la Universidad de Bourgogne creado en abril del 2020. Contiene un conjunto de datos en un entorno realista de videovigilancia utilizando una sola cámara, podemos ver una muestra en la Figura 5.2. La frecuencia de imágenes es de 25 frames/s y la resolución es de 320×240 píxeles. Contienen una iluminación variable y dificultades típicas como oclusiones o un fondo desordenado. Los actores realizaban diversas actividades cotidianas normales y caídas. El conjunto de datos contiene 191 vídeos.

Figura 5.2: Muestra de 2 caídas del dataset en diferentes representaciones.

5.2.2 UR Fall Detection Dataset:

Es conjunto de datos del Centro Interdisciplinario de Modelización Computacional de la **Universidad de Rzeszow** y contiene 70 vídeos (30 caídas + 40 actividades de la vida cotidiana). Los eventos de caídas se registran con 2 cámaras Microsoft Kinect y los correspondientes datos del acelerómetro, podemos ver una muestra en la Figura 5.3. Los eventos de AVD se registran con un solo dispositivo (cámara 0) y acelerómetro. Los datos de los sensores se recogieron con dispositivos PS Move (60Hz) y x-IMU (256Hz). Este dataset se encuentra bajo una licencia **Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License** y está destinada a un uso académico no comercial. Aunque si se quiere usar para fines comerciales se puede contactar con ellos.

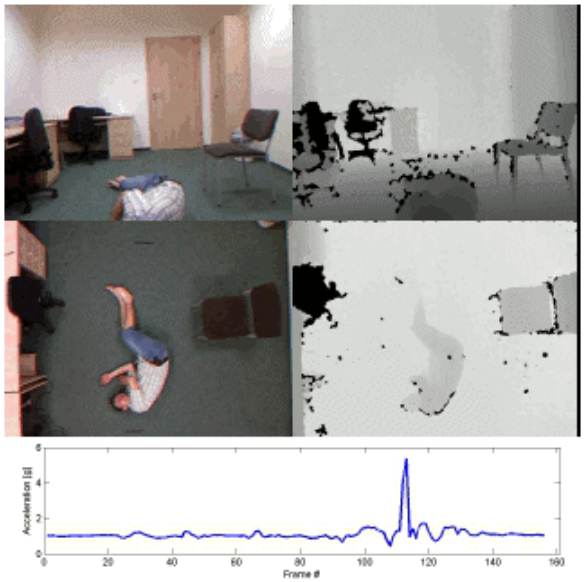

Figura 5.3: Ejemplo de caída capturada desde 2 puntos de vista.

5.2.3 Multiple cameras fall data set:

Este conjunto de datos de la **Universidad de Montréal** publicado en Julio 2010. Contiene 24 vídeos grabados con 8 cámaras de vídeo IP con la configuración que se muestra en la figura 5.4. Los primeros 22 escenarios contienen una caída y actividades de la vida diaria, los últimos 2 contienen sólo actividades de la vida cotidiana. Podemos ver una muestra en la Figura 5.5

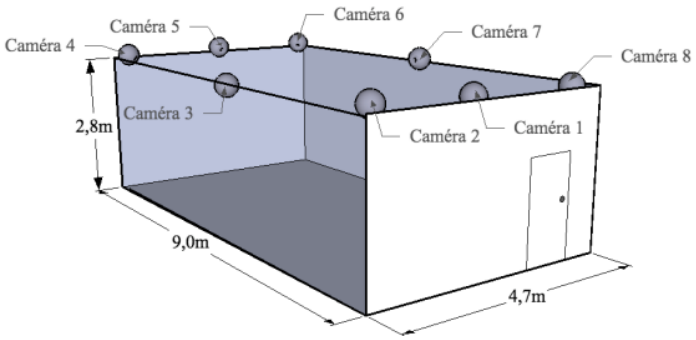

Figura 5.4: Configuración de las cámaras del laboratorio donde se capturaron los datos.

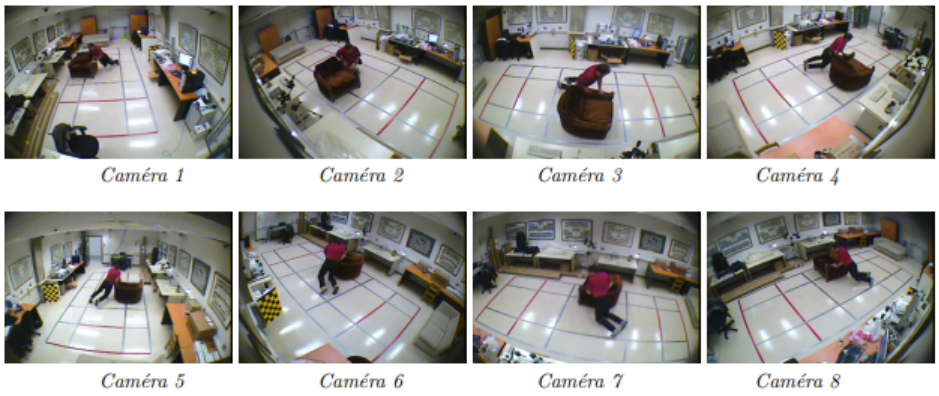

Figura 5.5: Ejemplo de caída capturada desde las 8 cámaras.

5.2.4 Validation dataset

El dataset que hemos usado para validar nuestros modelos consiste en **36 vídeos** de diferentes fuentes y diferentes ambientes. Ya que para validar el funcionamiento del nuestro modelo, necesitamos que se enfrente a un conjunto de datos con los que no haya entrenado (ni si quiera parecidos). El conjunto esta compuesto por:

1. 16 vídeos de youtube.
2. 12 vídeos de la Universidad de Málaga.
3. 8 vídeos propios.

5.2.5 Distribución final del dataset

Al final hemos podido construir un recopilar un dataset que de 156 vídeos, donde 90 serán para entrenar, 30 (25% del conjunto total) para testear y 36 para validar. Aunque después de aplicar técnicas de aumentado de datos el número asciende a 360 vídeos para entrenamiento, 120 vídeos para test y 36 vídeos para validación. (Véase la Figura 5.6).

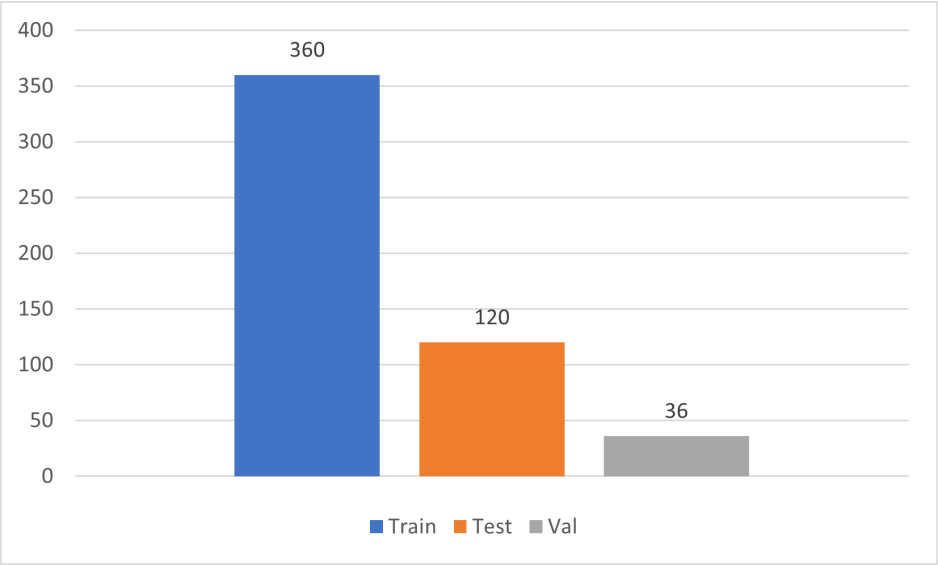

Figura 5.6: Distribución del dataset en conjuntos de train ,test y validación.

6 Desarrollo

En este apartado vamos a hablar sobre nuestra aproximación que consiste en un sistema de detección basado en deep learning y visión por computador, donde iremos cogiendo imagenes de una cámara, las pasaremos por un filtro de preprocesado y el resultado lo introduciremos en una red neuronal previamente entrenada y nos dará el resultado de caída o no caída. En la Figura 6.1 podemos ver el pipeline que vamos a seguir.

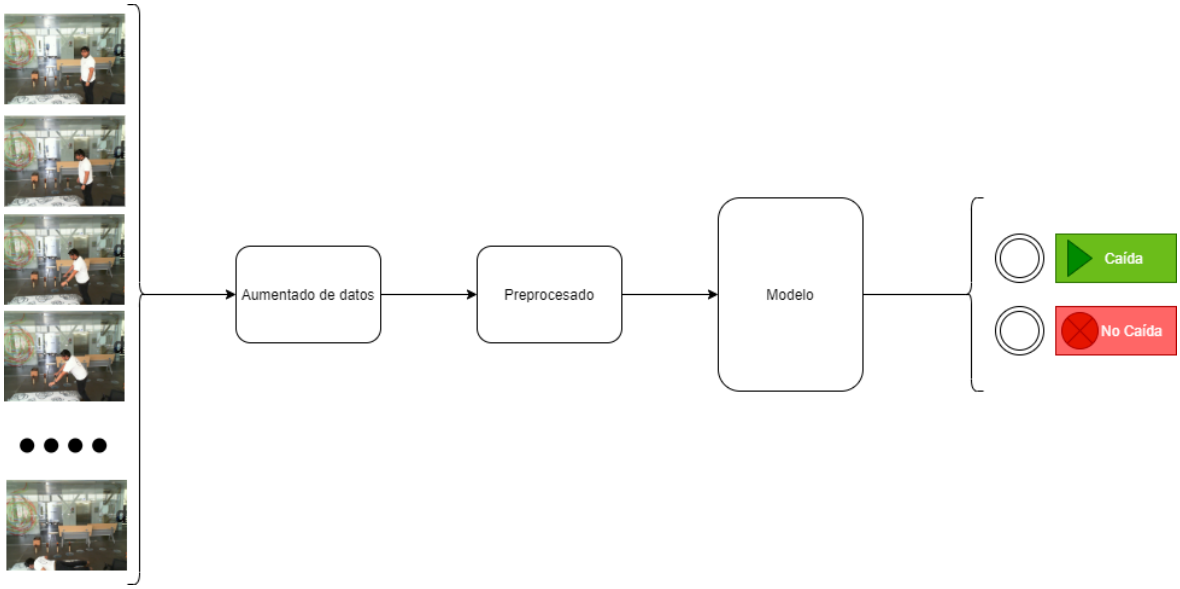

Figura 6.1: Ejemplo de un flip horizontal de un vídeo del UR Fall Detection Dataset.

6.1 Preparar los datos

Como ya hemos comentado en el apartado anterior para el desarrollo de este proyecto hemos usado 3 datasets para entrenar el modelo y uno para validarlo. La primera fase de este proyecto ha sido preparar los datos, para ello hemos obtenido los vídeos y los hemos dividido en **30 frames**, que es lo que dura más o menos una caída (± 1 segundo) y en frames de **128x128 píxeles**.

6.1.1 Aumentado de datos

Debido a la poca cantidad de vídeos de la que disponemos, decidimos aplicar algunas técnicas de aumentado de datos. Hemos intentado hacer un aumentado de datos algo realista, nada de girar el vídeo, ya que la persona caería hacia arriba (giro 180º), cosa que no nos serviría ya que este caso nunca se daría en la vida real. Las 2 técnicas que hemos usado las hemos usado ya que son casos que se pueden dar en un escenario real:

1. **Flip horizontal**: Hemos realizado un flip horizontal en cada vídeo, es decir, si una persona se caía de derecha a izquierda, caerá de izquierda a derecha, podemos ver un ejemplo en la Figura 6.2. Y así duplicamos el número de vídeos que teníamos, de **120 vídeos** a **240 vídeos**.

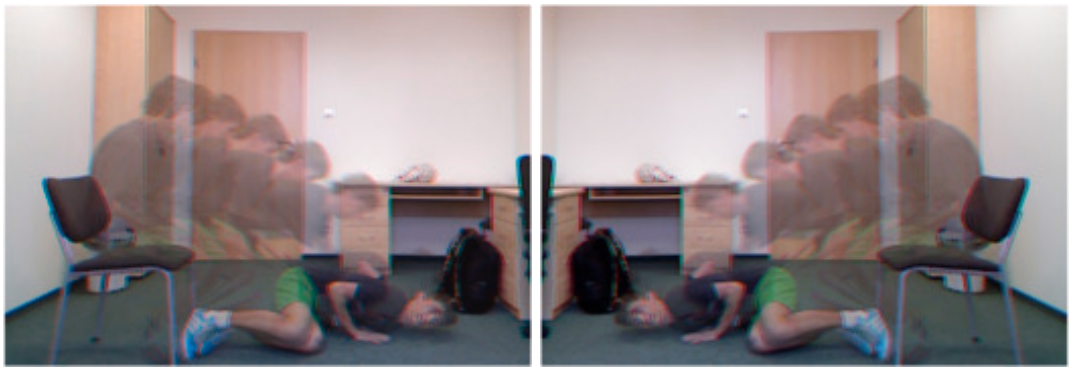

Figura 6.2: Ejemplo de un flip horizontal de un vídeo del UR Fall Detection Dataset.

2. **Difuminado**: Hemos aplicado un filtro de difuminado a todos los frames de los vídeos que teníamos, ya que pensamos que seria un caso que se podría dar debido al desenfoque que se puede producir al usar una cámara, podemos ver un ejemplo en la Figura 6.3. Y así logramos duplicar el número de vídeos que teníamos, de **240 vídeos** a **480 vídeos**.

Figura 6.3: Ejemplo de aplicar un difuminado a un vídeo del UR Fall Detection Dataset.

6.1.2 Preprocesado de datos

Para el preprocesado de datos hemos usado el flujo óptico para saber que objetos se han movido entre frame y frame. Este método lo implementa la librería de **OpenCV** en sus 2 versiones, flujo óptico denso y flujo óptico disperso. Podemos ver un ejemplo en la Figura 6.4

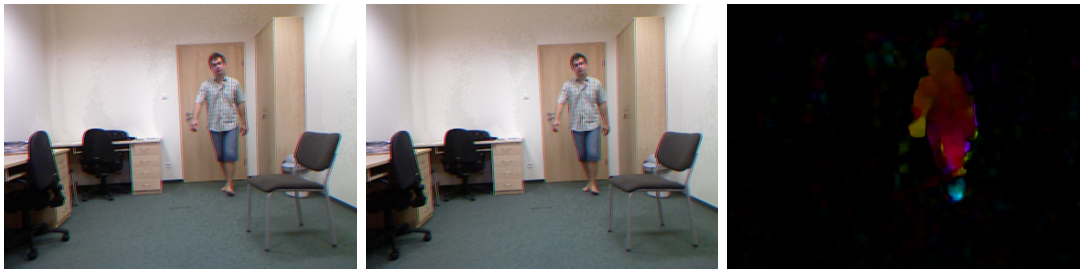

Figura 6.4: Ejemplo del flujo óptico denso de 2 imagenes del UR Fall Detection Dataset.

6.2 Métodos/Aproximaciones

Debido a que hemos decido basar nuestra solución en Deep Learning, la siguiente etapa del proyecto es preparar un modelo capaz de detectar las caídas de un vídeo.

6.2.1 Time-distributed CNN + LSTM

La primera solución que se nos ocurrió fue usar una red que tuviese un bloque convolucional básico para la extracción de características de los datos de entrada, con una capa **Time-distributed** para poder procesar los **n frames** y por último un bloque LSTM procesar la información espacio-temporal. Tenemos la arquitectura del modelo en la Figura 6.4.

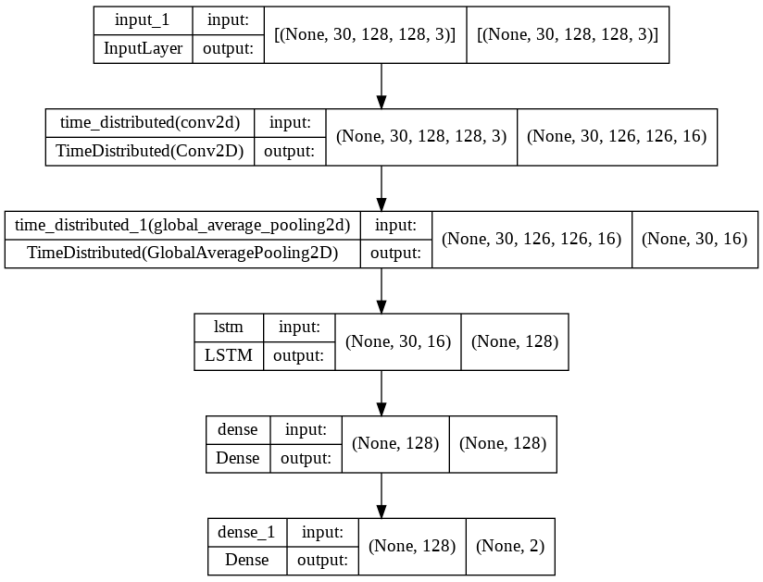

Figura 6.5: Arquitectura del modelo Time-distributed CNN LSTM.

6.2.2 CNN pre-entrenada + LSTM

La siguiente solución que se nos ocurrió fue sustituir el bloque convolucional básico por una red pre-entrenada en **ImageNet**, manteniendo la capa **Time-distributed** y el bloque LSTM. Para el desarrollo de estos modelos vamos a usar 4 redes pre-entrenadas, Resnet50, VGG19 , InceptionV3 y EfficientNetB7. [14]

6.2.3 Red convolucional 3D

La última solución que usamos fue usar una red convolucional 3D. La arquitectura de una red convolucional 3D es una variación de la arquitectura 2D tradicional. Utiliza todas las características de la arquitectura CNN estándar pero adopta una tercera dimensión, que en este caso es la secuencia de frames. La operación de la capa de convolución 3D se aplicara con un núcleo tridimensional aprendible que considera las características espacio-temporales de la secuencia de frames, que son la altura x anchura × número de frame. Podemos ver la arquitectura del modelo en la Figura 6.4.

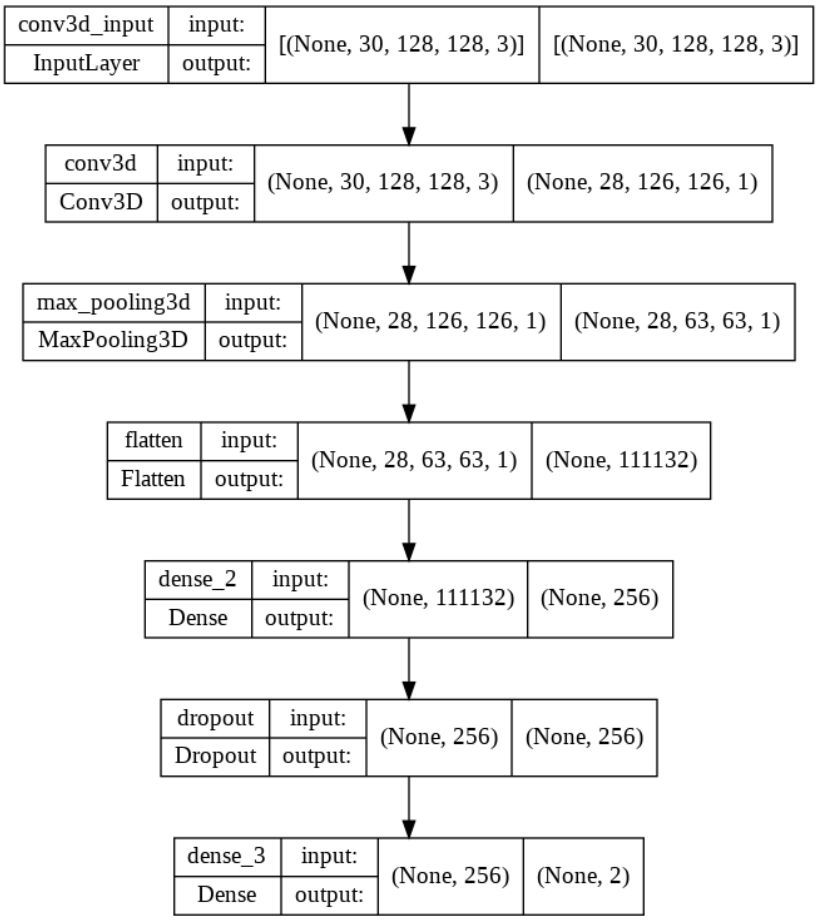

Figura 6.6: Arquitectura del modelo Conv3DNet.

6.3 Desarrollo del programa para usar los modelos

Después de desarrollar la red capaz de detectar caídas, tuvimos que desarrollar un programa para poder el modelo. Este programa tiene un funcionamiento muy sencillo, tal y como podemos ver en su diagrama de flujo en la Figura 6.7.

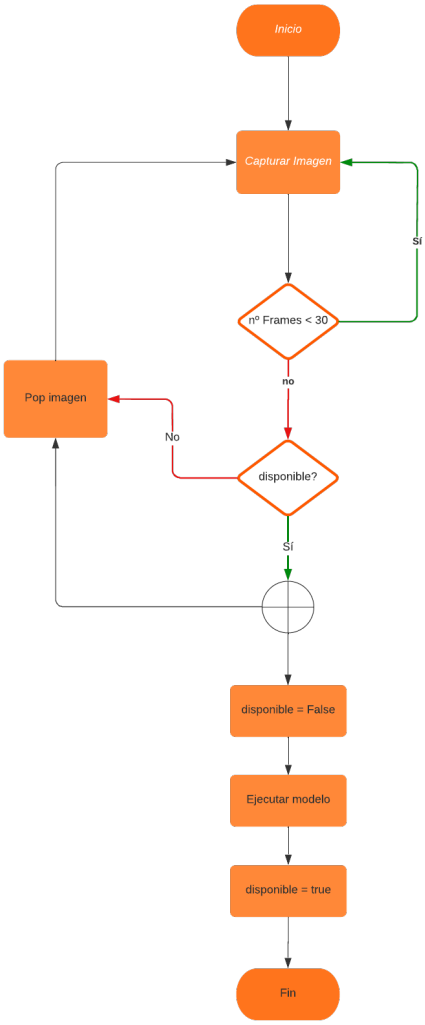

Figura 6.7: Diagrama de flujo del programa desarrollado.

El programa irá recopilando imagenes desde la entrada de vídeo hasta llegar a las 30 imagenes (frames) después calculara el flujo óptico de la secuencia de imagenes y ejecutará el modelo en paralelo. Una vez ejecute el modelo, debido a que esta ejecución consume tiempo, el programa seguirá recopilando imagenes, borrando la primera imagen e introduciendo una nueva a la secuencia (pop-push) hasta que el modelo vuelva a estar disponible, una vez el modelo vuelva a estar disponible, este volverá a ejecutar el modelo en paralelo. La sincronización entre los 2 procesos se ha hecho mediante mediante un semáforo.

7 Experimentación y Resultados

En este apartado vamos a mostrar los resultados de las pruebas que hemos realizado para el desarrollo de este proyecto. Usaremos los modelos mencionados en el apartado 6. Para realizar las pruebas temporales usaremos un dispositivo con la siguientes características:

Procesador	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
RAM	16GB
GPU	NVIDIA GeForce GTX 1650 with Max-Q Design

Para referirnos a los modelos lo haremos de la siguiente forma:

Modelo TD-RN50-LSTM	TimeDistributed Resnet50 + LSTM
Modelo TD-VGG-LSTM	TimeDistributed VGG19 + LSTM
Modelo TD-INCV3-LSTM	TimeDistributed InceptionV3 + LSTM
Modelo TD-EfB7-LSTM	TimeDistributed EfficientNetB7 + LSTM
Modelo TD-CNN-LSTM	TimeDistributed Simple CNN + LSTM
Modelo CONV3D	Red convolucional 3D

7.1 Métricas para la evaluación de resultados

Al tratarse de un problema de clasificación disponemos un conjunto de métricas que podemos usar para evaluar nuestros modelos, como el accuracy, precision, recall, F1-score y AUC. En los siguientes apartados veremos mas en detalle cada una de estas métricas.

7.1.1 Confusion Matrix

Una matriz de confusión es una tabla que se utiliza para definir el rendimiento de un algoritmo de clasificación, podemos ver un diagrama de la matriz de confusión en la Figura 7.1. Una matriz de confusión visualiza y resume el rendimiento de un algoritmo de clasificación.

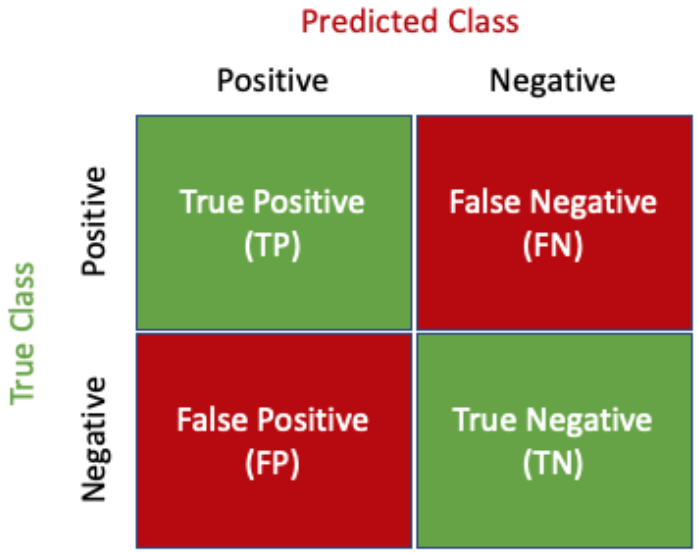

Figura 7.1: Diagrama de matriz de confusión.

La matriz de confusión consta de cuatro partes que se utilizan para definir las métricas de medición del clasificador (Véase la Figura 7.1). Supongamos un algoritmo que clasifica entre 2 clases, C1 y C2, las partes de la tabla serian:

1. TP (True Positive), el numero de elementos de la C1 correctamente clasificados como C1.
2. TN (True Negative), el numero de elementos de la C2 correctamente clasificados como C2.
3. FP (False Positive), el numero de elementos de la C1 clasificados como C2.
4. FN (False Positive), el numero de elementos de la C2 clasificados como C1.

7.1.2 Accuracy

Es la métrica más usada en la clasificación. Indica el número de elementos clasificados correctamente en comparación con el número total de elementos, podemos ver su fórmula matemática en la Figura 7.2. Usando esta métrica suele ser suficiente para evaluar el modelo en caso de que los datos estén bien equilibrados.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Figura 7.2: Fórmula de la métrica **accuracy**.

Hay muchos casos en los que el accuracy no es un buen indicador del rendimiento del modelo. Uno de estos escenarios es cuando su distribución de clases está desequilibrada. En este caso, incluso si predice todas las muestras como la clase más frecuente, obtendría una alta tasa de precisión, lo que no tiene ningún sentido (porque su modelo no está aprendiendo nada y solo está prediciendo todo como la clase superior) Así que se suele usar otras métricas (estas pueden ser complementarias).

7.1.3 Precision

La precisión se calcula como la relación entre el número de muestras Positivas clasificadas correctamente y el número total de muestras clasificadas como Positivas (ya sea correcta o incorrectamente), podemos ver su fórmula matemática en la Figura 7.3. La precisión mide la exactitud del modelo a la hora de clasificar una muestra como positiva.

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

Figura 7.3: Fórmula de la métrica **precision**.

La precisión es una buena métrica cuando los costos de falsos positivos son altos. Por ejemplo, en la detección de spam de correo electrónico, un falso positivo significa que un correo electrónico que no es spam (negativo real) ha sido identificado como spam (spam predicho). El usuario de correo electrónico puede perder correos electrónicos importantes si la precisión no es alta para el modelo de detección de spam.

7.1.4 Recall

El recall se calcula como la relación entre el número de muestras positivas correctamente clasificadas como positivas y el número total de muestras positivas. La recall mide la capacidad del modelo para detectar muestras positivas, podemos ver su fórmula matemática en la Figura 7.4. Cuanto mayor sea la recuperación, mayor será el número de muestras positivas detectadas.

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

Figura 7.4: Fórmula de la métrica **recall**.

Recall es una buena métrica para cuando hay un alto costo asociado con el Falso Negativo. Por ejemplo, en detección de fraude. Si una transacción fraudulenta (Positivo real) se pronostica como no fraudulenta (Negativo previsto), la consecuencia puede ser muy mala para el banco.

7.1.5 F1 Score

La puntuación F1 combina la precisión y el recall de un clasificador en una sola métrica tomando su media armónica, podemos ver su fórmula matemática en la Figura 7.5. Se utiliza principalmente para comparar el rendimiento de dos clasificadores. Supongamos que el clasificador A tiene un mayor recall y el clasificador B tiene una mayor precisión, en este caso, las puntuaciones F1 de ambos clasificadores pueden utilizarse para determinar cuál produce mejores resultados.

$$\text{F1 Score} = \frac{2 \text{ x (Precision x Recall)}}{\text{Precision + Recall}}$$

Figura 7.5: Fórmula de la métrica **F1 score**.

7.1.6 Area Under the Curve

La curva ROC es una medida de rendimiento para los problemas de clasificación en varios ajustes de umbral. La ROC es una curva de probabilidad y la AUC representa el grado o la medida de separabilidad. Indica en qué medida el modelo es capaz de distinguir entre clases. Cuanto más alto sea el AUC, mejor será el modelo para predecir las clases C0 como C0 y las clases C1 como C1. Podemos ver un diagrama que lo refresenta en la Figura 7.6.

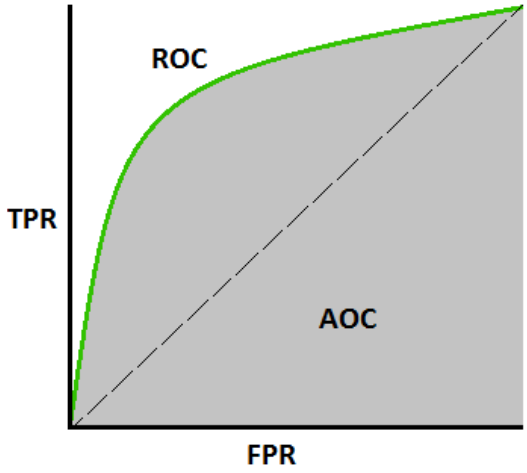

Figura 7.6: AUC-ROC Curva.

7.2 Validación cruzada

La validación cruzada (cross-validation) es una técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición elegida (entrenamiento o prueba). K-Fold CV proporciona un modelo con menos sesgo en comparación con otros métodos. En K-Fold CV, tenemos un parámetro k, donde este decide en cuántas particiones se dividirá el conjunto de datos, podemos un ejemplo de un 10-CV en la Figura 7.7. Cada partición tiene la oportunidad de aparecer en el conjunto de entrenamiento (k-1) veces, lo que a su vez garantiza que cada observación del conjunto de datos aparezca en el conjunto de datos, lo que permite que el modelo aprenda mejor la distribución de datos.

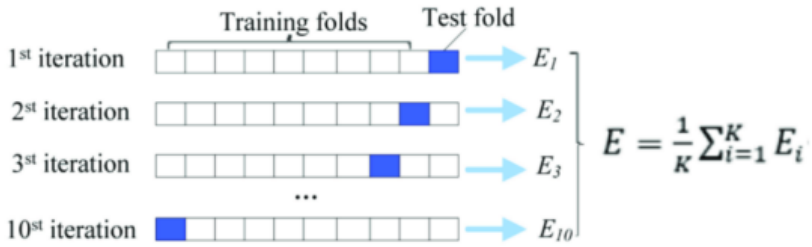

Figura 7.7: Ejemplo de validación cruzada 10-CV.

Para lograrlo en este desafío, hemos utilizado la clase StratifiedKFold de la librería clase StratifiedKFold de la biblioteca sklearn para dividir el conjunto de datos de entrenamiento en 10 particiones, donde el algoritmo intenta equilibrar el número de instancias de cada clase en cada partición. El ejemplo crea y evalúa 10 modelos utilizando las 10 divisiones de los datos y recopila todas las puntuaciones. El rendimiento se imprime para cada modelo y se almacena. La desviación promedio y estándar del rendimiento del modelo se imprime al final de la ejecución para proporcionar una estimación sólida de la precisión del modelo.

7.3 Sin optical flow ni aumentado de datos

En este apartado realizaremos pruebas sobre nuestros modelos pero sin aplicar un preprocesado a nuestros datos ni técnicas de aumentado de datos.

En este apartado apartado no se han conseguido datos dignos de mención ya que ningún modelo ha obtenido buenos resultados, el mejor resultado ha sido del Modelo TD-RN50-LSTM con un 53% de accuracy en los datos de entrenamiento y 50% de accuracy en validación.

7.3.1 Modelo TD-RN50-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.534	0.31	0.534	0.387	53,43% (+/- 10,29%)
Test	0.503	0.296	0.533	0.368	53,33% (+/- 10,00%)
Val	0.498	0.249	0.497	0.329	49,75% (+/- 0,75%)

Cuadro 7.1: Resultados prueba 19.

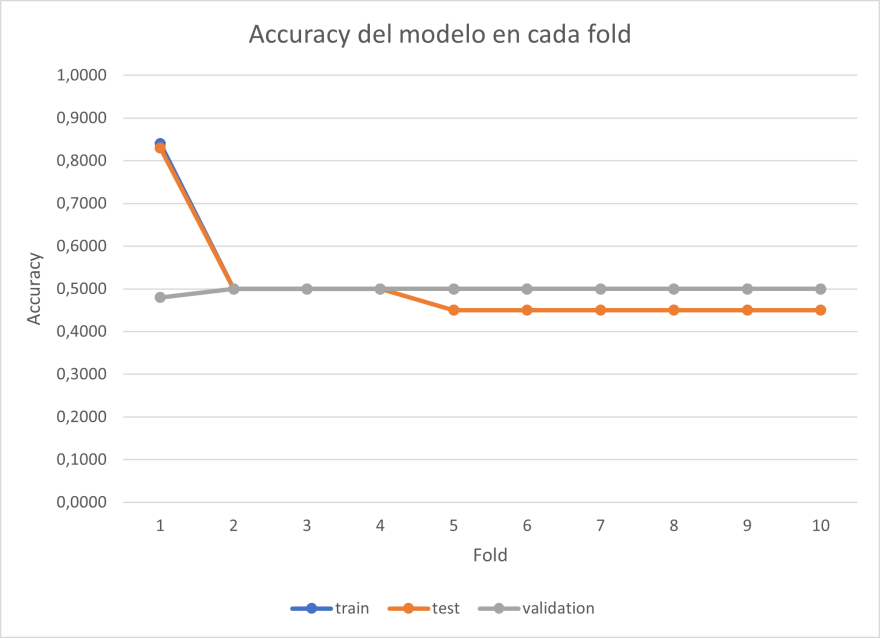

Figura 7.8: Accuracy del Modelo TD-RN50-LSTM sin optical flow ni aumentado de datos en cada fold.

7.3.2 Modelo TD-VGG-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.506	0.282	0.506	0.353	50,59% (+/- 1,76%)
Test	0.49	0.271	0.5	0.342	50,00% (+/- 0,00%)
Val	0.506	0.312	0.506	0.358	50,50% (+/- 1,00%)

Cuadro 7.2: Resultados prueba 20.

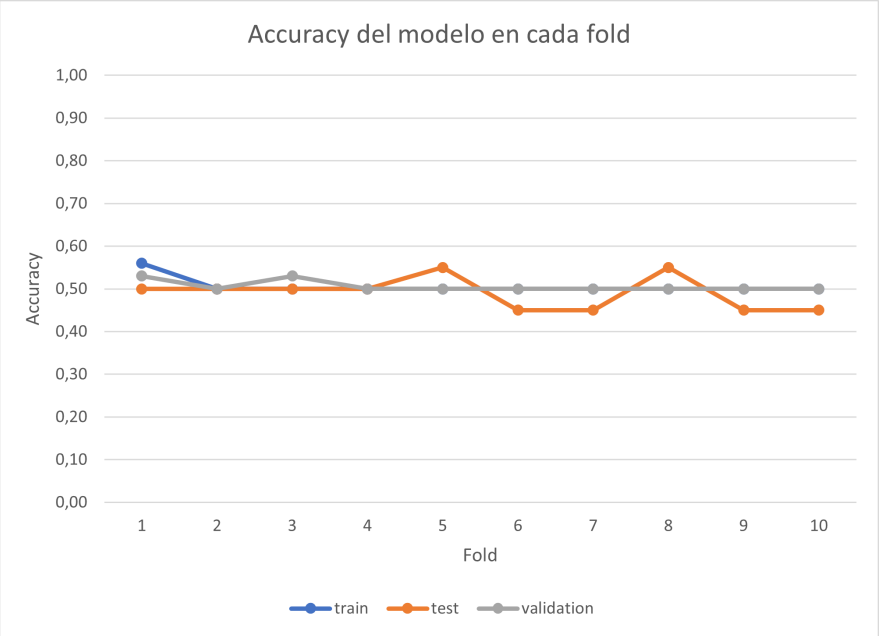

Figura 7.9: Accuracy del Modelo TD-VGG-LSTM sin optical flow ni aumentado de datos en cada fold.

7.3.3 Modelo TD-INCV3-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.509	0.303	0.509	0.353	50,88% (+/- 2,65%)
Test	0.518	0.304	0.525	0.367	52,50% (+/- 7,50%)
Val	0.489	0.448	0.448	0.413	48,75% (+/- 7,35%)

Cuadro 7.3: Resultados prueba 21.

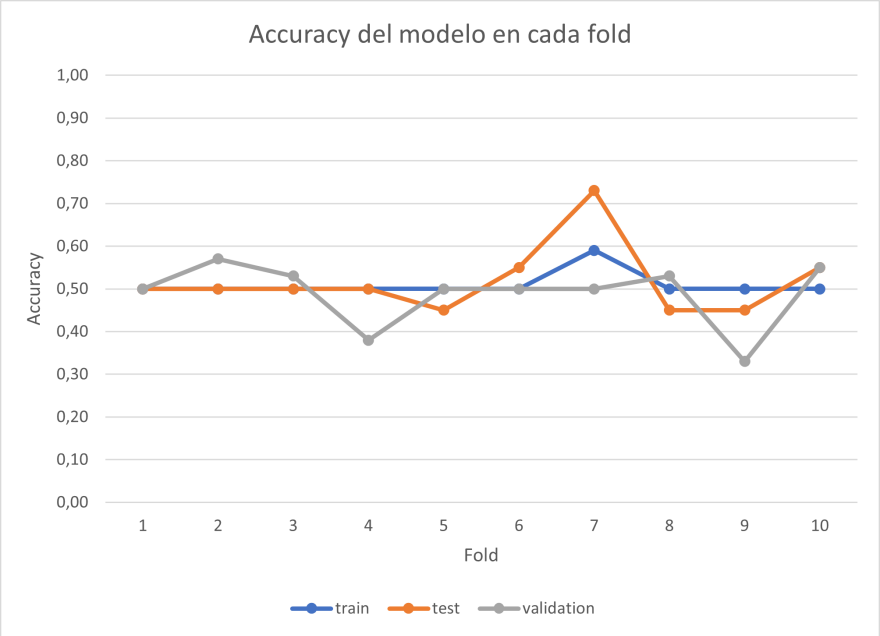

Figura 7.10: Accuracy del Modelo TD-INCV3-LSTM sin optical flow ni aumentado de datos en cada fold.

7.3.4 Modelo TD-EfB7-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.506	0.282	0.506	0.353	50,59% (+/- 1,76%)
Test	0.49	0.271	0.5	0.342	50,00% (+/- 0,00%)
Val	0.506	0.312	0.506	0.358	50,50% (+/- 1,00%)

Cuadro 7.4: Resultados prueba 22.

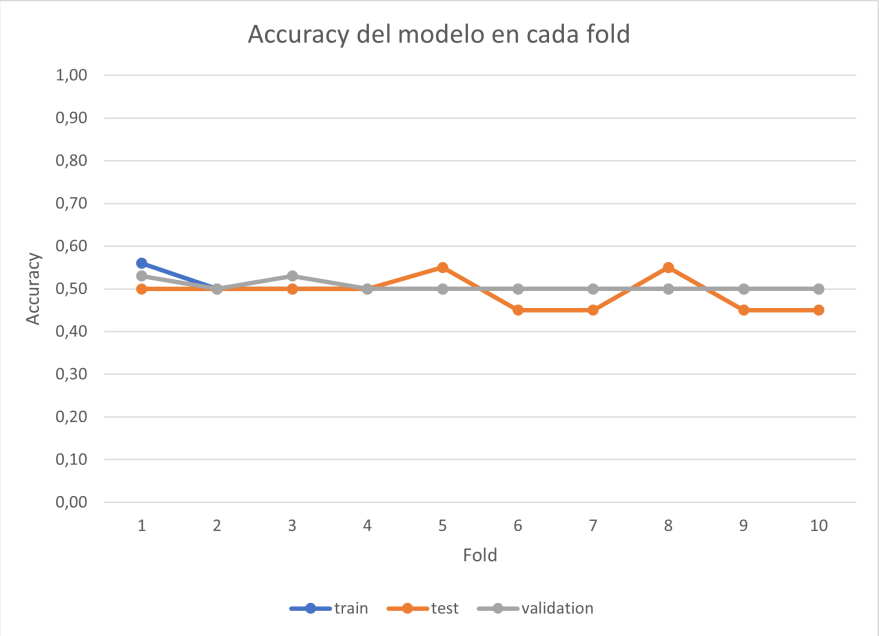

Figura 7.11: Accuracy del Modelo TD-VGG-LSTM sin optical flow ni aumentado de datos en cada fold.

7.3.5 Modelo TD-CNN-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.5	0.25	0.5	0.334	50,00% (+/- 0,00%)
Test	0.47	0.238	0.5	0.318	50,00% (+/- 0,00%)
Val	0.513	0.429	0.514	0.435	51,25% (+/- 8,00%)

Cuadro 7.5: Resultados prueba 23.

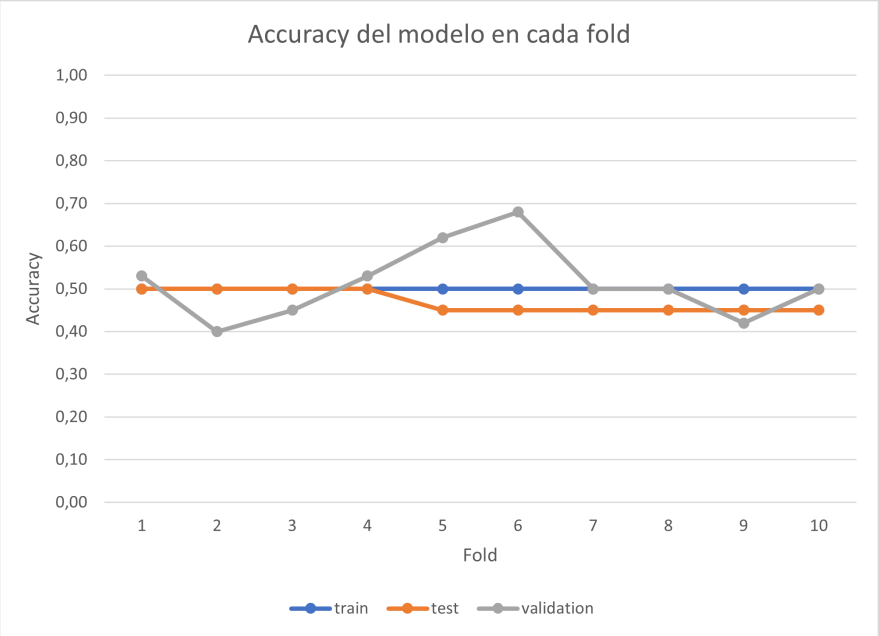

Figura 7.12: Accuracy del Modelo TD-CNN-LSTM sin optical flow ni aumentado de datos en cada fold.

7.3.6 Modelo CONV3D

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.5	0.25	0.5	0.331	50,00% (+/- 0,00%)
Test	0.495	0.25	0.5	0.33	50,00% (+/- 0,00%)
Val	0.505	0.402	0.505	0.358	50,50% (+/- 3,67%)

Cuadro 7.6: Resultados prueba 24.

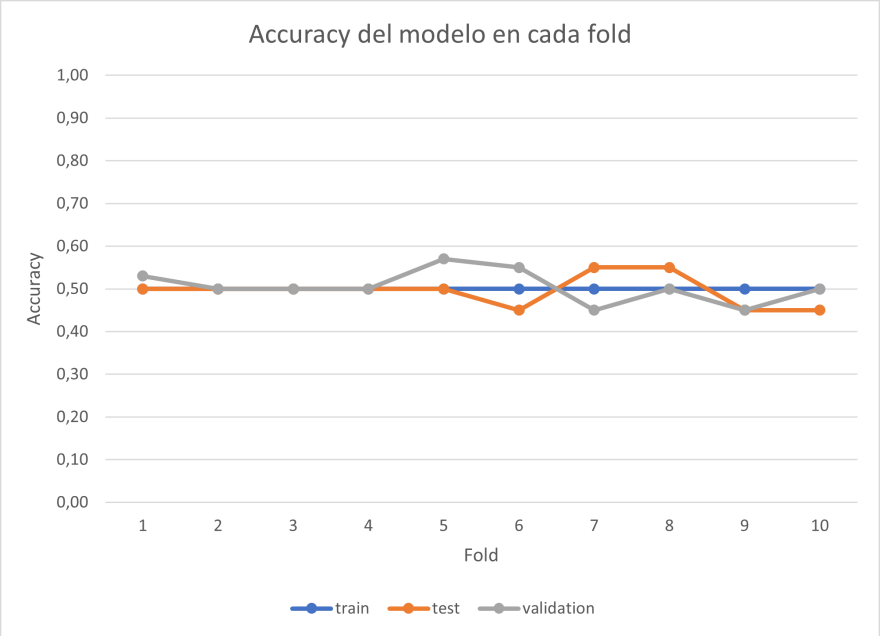

Figura 7.13: Accuracy del Modelo CONV3D sin optical flow ni aumentado de datos en cada fold.

7.4 Sin optical flow con aumentado de datos

En este apartado realizaremos pruebas sobre nuestros modelos pero esta vez sin aplicar un preprocesado a nuestros datos, pero si que usaremos técnicas de aumentado de datos.

En este apartado tampoco hay resultados dignos de mención, ningún modelo ha obtenido buenos resultados, el mejor resultado ha sido del Modelo TD-EfB7-LSTM con un 61% de accuracy en los datos de entrenamiento pero 50% de accuracy en validación.

7.4.1 Modelo TD-RN50-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.524	0.299	0.524	0.370	52,37% (+/- 7,10%)
Test	0.517	0.293	0.517	0.366	51,74% (+/- 5,22%)
Val	0.522	0.366	0.522	0.392	52,25% (+/- 6,56%)

Cuadro 7.7: Resultados prueba 13.

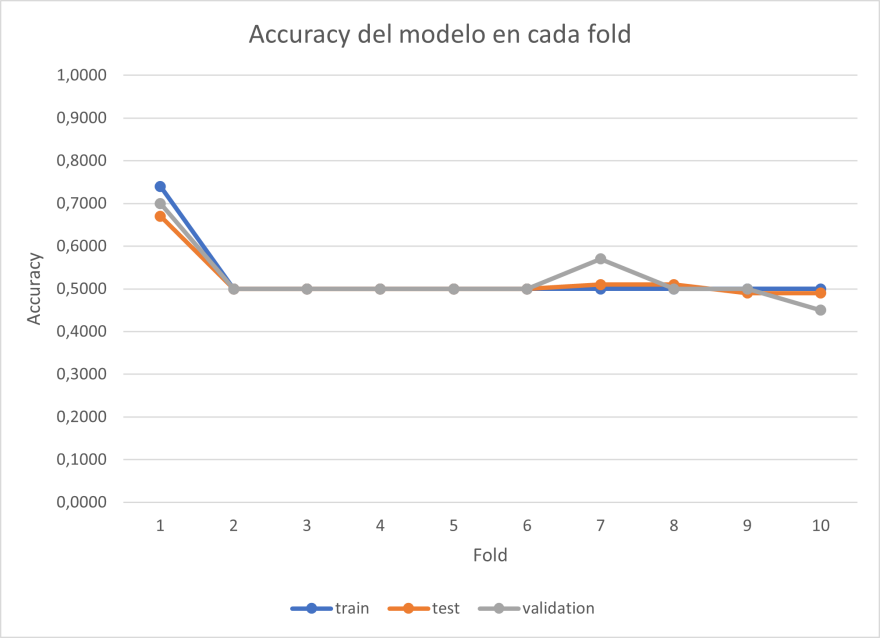

Figura 7.14: Accuracy del Modelo TD-RN50-LSTM sin optical flow y con aumentado de datos en cada fold.

7.4.2 Modelo TD-VGG-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.524	0.299	0.524	0.370	52,37% (+/- 7,10%)
Test	0.517	0.293	0.517	0.366	51,74% (+/- 5,22%)
Val	0.522	0.366	0.522	0.392	52,25% (+/- 6,56%)

Cuadro 7.8: Resultados prueba 14.

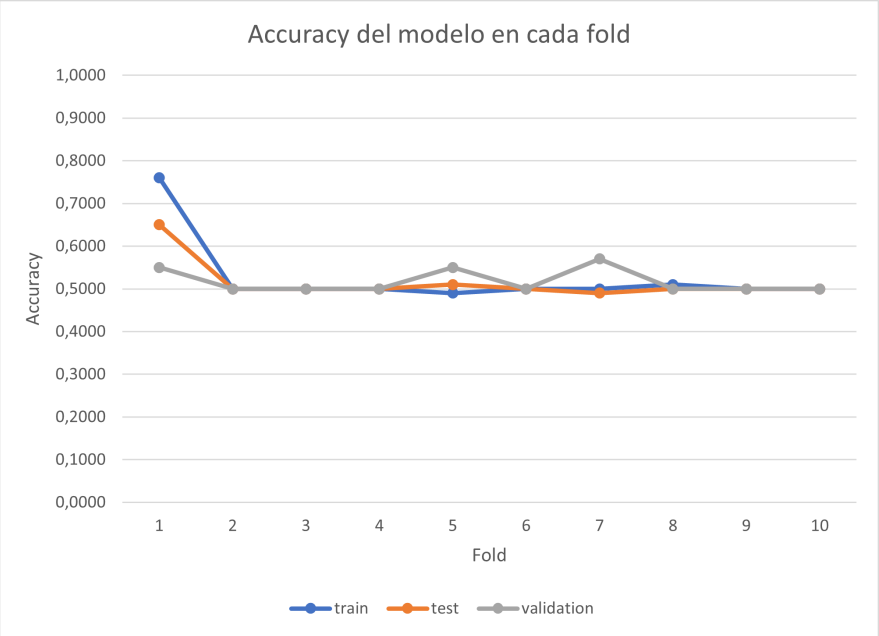

Figura 7.15: Accuracy del Modelo TD-VGG-LSTM sin optical flow y con aumentado de datos en cada fold.

7.4.3 Modelo TD-INCV3-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.5	0.25	0.5	0.33	50,00% (+/- 0,00%)
Test	0.498	0.248	0.5	0.331	50,00% (+/- 0,00%)
Val	0.455	0.365	0.452	0.381	45,25% (+/- 6,27%)

Cuadro 7.9: Resultados prueba 15.

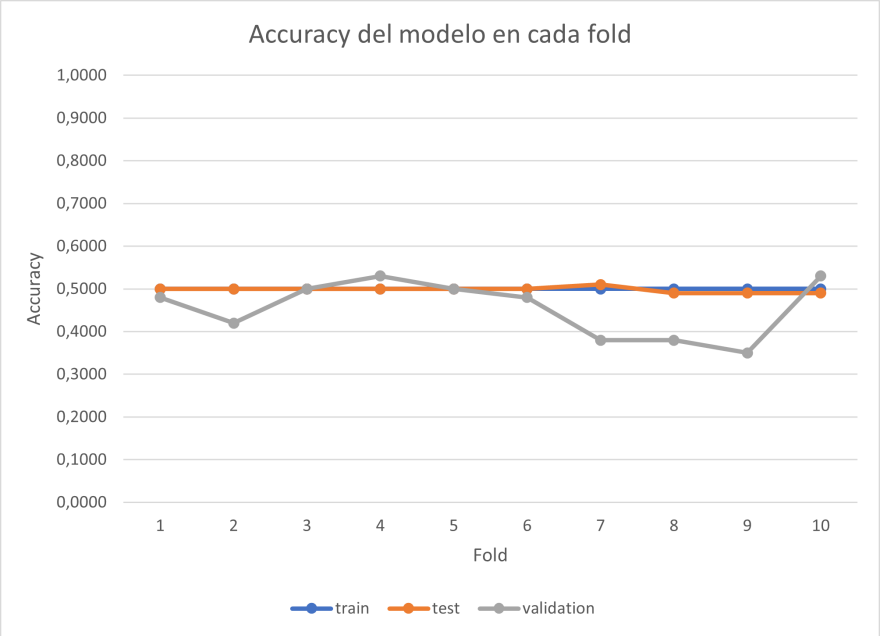

Figura 7.16: Accuracy del Modelo TD-INCV3-LSTM sin optical flow y con aumentado de datos en cada fold.

7.4.4 Modelo TD-EfB7-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.613	0.573	0.613	0.547	61,30% (+/- 8,63%)
Test	0.584	0.547	0.584	0.512	58,39% (+/- 8,65%)
Val	0.499	0.351	0.497	0.380	49,75% (+/- 1,35%)

Cuadro 7.10: Resultados prueba 16.

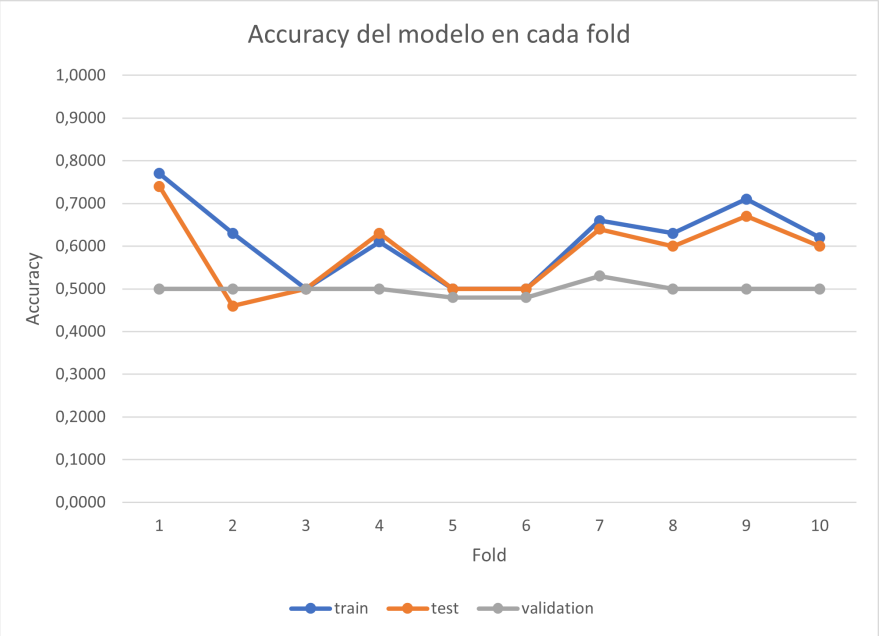

Figura 7.17: Accuracy del Modelo TD-EfB7-LSTM sin optical flow y con aumentado de datos en cada fold.

7.4.5 Modelo TD-CNN-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.5	0.25	0.5	0.33	50,00% (+/- 0,00%)
Test	0.496	0.246	0.5	0.33	50,00% (+/- 0,00%)
Val	0.493	0.468	0.49	0.419	49,25% (+/- 4,34%)

Cuadro 7.11: Resultados prueba 17.

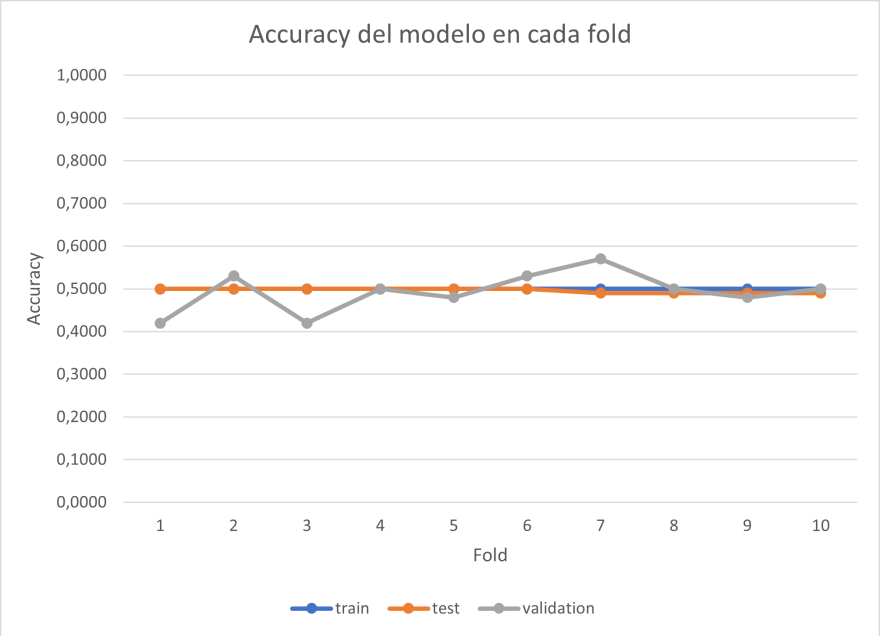

Figura 7.18: Accuracy del Modelo TD-CNN-LSTM sin optical flow y con aumentado de datos en cada fold.

7.4.6 Modelo CONV3D

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.523	0.308	0.523	0.368	52,34% (+/- 7,02%)
Test	0.509	0.287	0.507	0.352	50,65% (+/- 1,96%)
Val	0.514	0.515	0.514	0.399	51,25% (+/- 2,80%)

Cuadro 7.12: Resultados prueba 18.

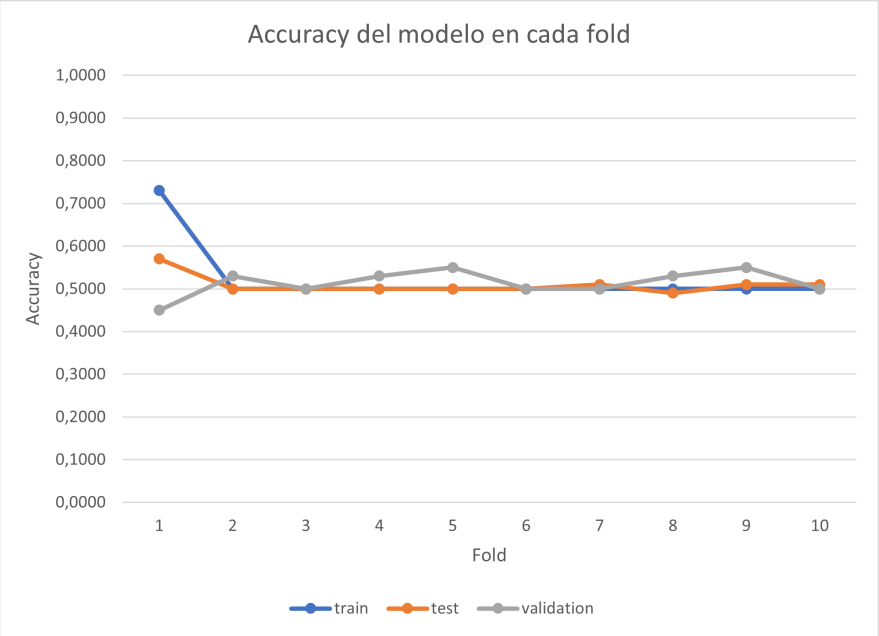

Figura 7.19: Accuracy del Modelo CONV3D sin optical flow y con aumentado de datos en cada fold.

7.5 Optical flow sin aumentado de datos

En este apartado realizaremos pruebas sobre nuestros modelos pero a diferencia del apartado anterior, esta vez si que aplicaremos un preprocesado a nuestros datos (Optical flow) pero no usaremos técnicas de aumentado de datos.

7.5.1 Modelo TD-RN50-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.616	0.443	0.614	0.49	61,40% (+/- 17,48%)
Test	0.58	0.431	0.608	0.475	60,83% (+/- 17,50%)
Val	0.576	0.409	0.574	0.456	57,50% (+/- 12,25%)

Cuadro 7.13: Resultados prueba 7.

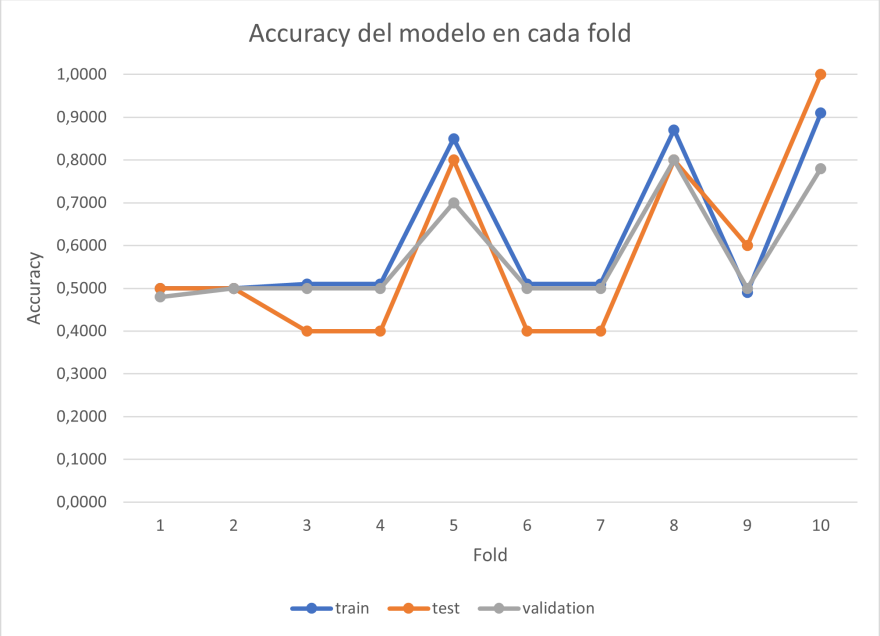

Figura 7.20: Accuracy del Modelo TD-RN50-LSTM con optical flow y sin aumentado de datos en cada fold.

Observando los resultados en el Cuadro 7.13 podemos ver que la eficiencia del modelo ha subido bastante en comparación con los experimentos anteriores, donde podemos ver que el accuracy en el conjunto de entrenamiento ha llegado a mas de un 61%, en el conjunto de text un 58% y en validación un 57%.

7.5.2 Modelo TD-VGG-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.508	0.258	0.5	0.338	50,00% (+/- 0,00%)
Test	0.42	0.21	0.5	0.298	50,00% (+/- 0,00%)
Val	0.51	0.303	0.51	0.349	51,00% (+/- 3,00%)

Cuadro 7.14: Resultados prueba 8.

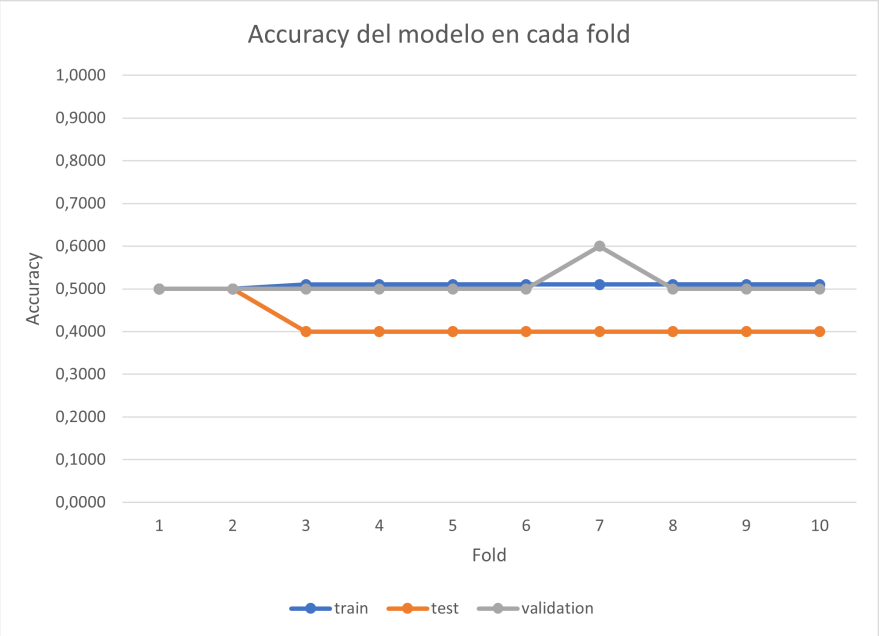

Figura 7.21: Accuracy del Modelo TD-VGG-LSTM con optical flow y sin aumentado de datos en cada fold.

Observando los resultados de los experimentos anteriores y de este experimento, podemos ver que dan resultados muy similares, donde el modelo no logró ajustarse ni a los datos de entrenamiento.

7.5.3 Modelo TD-INCV3-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.645	0.748	0.644	0.562	64,38% (+/- 15,39%)
Test	0.617	0.543	0.626	0.519	62,50% (+/- 15,48%)
Val	0.55	0.481	0.549	0.435	55,00% (+/- 9,62%)

Cuadro 7.15: Resultados prueba 9.

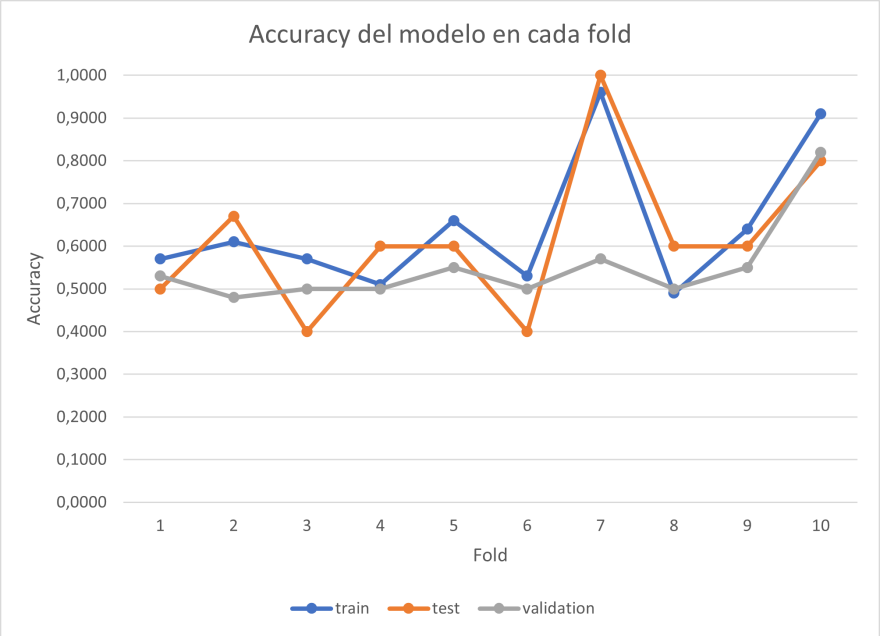

Figura 7.22: Accuracy del Modelo TD-INCV3-LSTM con optical flow y sin aumentado de datos en cada fold.

Al igual que el modelo TD-RN-50, este modelo si que ha conseguido dar buenos resultados, donde si que ha llegado a ajustarse a los datos de entrenamiento con un 64% de accuracy, en el conjunto de test con un 62% aunque no haya logrado generalizar del todo bien sobre el conjunto de validación, donde ha dado un 55% de accuracy.

7.5.4 Modelo TD-EfB7-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.508	0.258	0.5	0.338	50,00% (+/- 0,00%)
Test	0.42	0.21	0.5	0.298	50,00% (+/- 0,00%)
Val	0.51	0.303	0.51	0.349	51,00% (+/- 3,00%)

Cuadro 7.16: Resultados prueba 8.

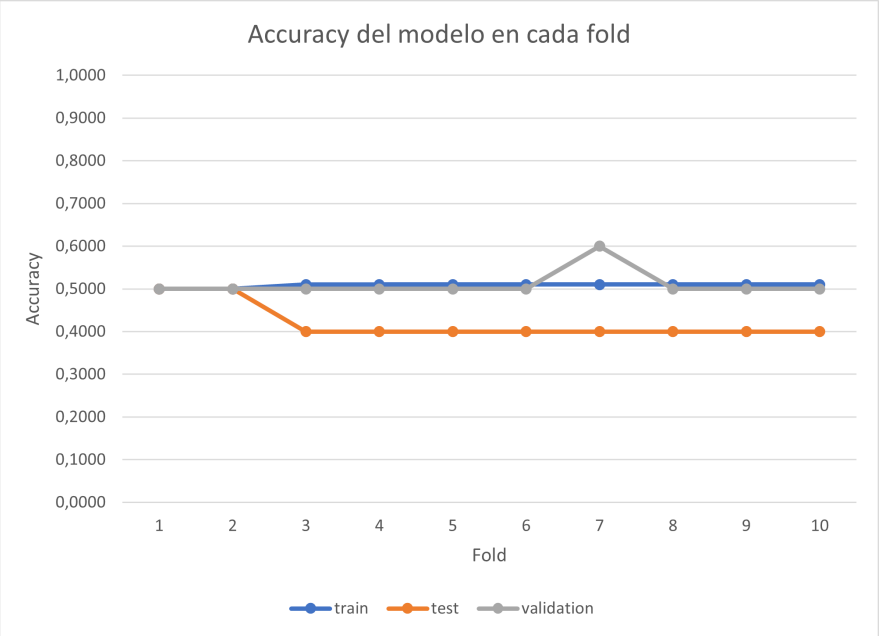

Figura 7.23: Accuracy del Modelo TD-EfB7-LSTM con optical flow y sin aumentado de datos en cada fold.

Al igual que el apartado anterior, este modelo sigue sin aprender.

7.5.5 Modelo TD-CNN-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	1.0	1.0	1.0	1.0	100,00% (+/- 0,00%)
Test	0.766	0.756	0.748	0.724	75,00% (+/- 18,26%)
Val	0.652	0.683	0.652	0.637	65,25% (+/- 3,94%)

Cuadro 7.17: Resultados prueba 11.

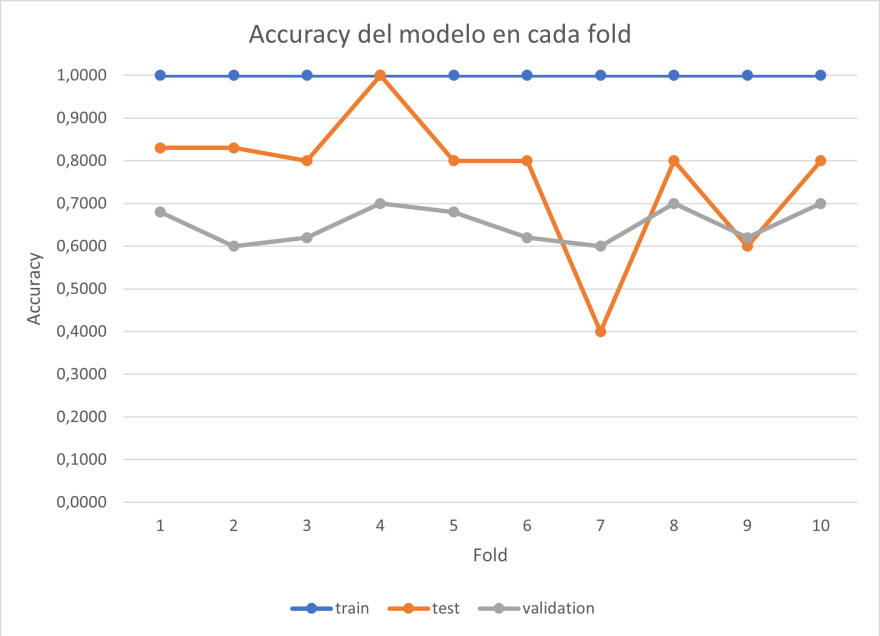

Figura 7.24: Accuracy del Modelo TD-CNN-LSTM con optical flow y sin aumentado de datos en cada fold.

Observando los datos del Cuadro 7.17 podemos ver que el modelo se ha ajustado muy bien a los datos de entrenamiento, aunque no haya generalizado tan bien como al usar aumentado de datos, pero aún así son buenos resultados (Los mejores sin aumentado de datos).

7.5.6 Modelo CONV3D

	Accuracy	Precision	Recall	F1-Score	AUC
Train	1.0	1.0	1.0	1.0	100,00% (+/- 0,00%)
Test	0.714	0.756	0.709	0.695	70,83% (+/- 17,18%)
Val	0.543	0.543	0.543	0.530	54,25% (+/- 8,74%)

Cuadro 7.18: Resultados prueba 12.

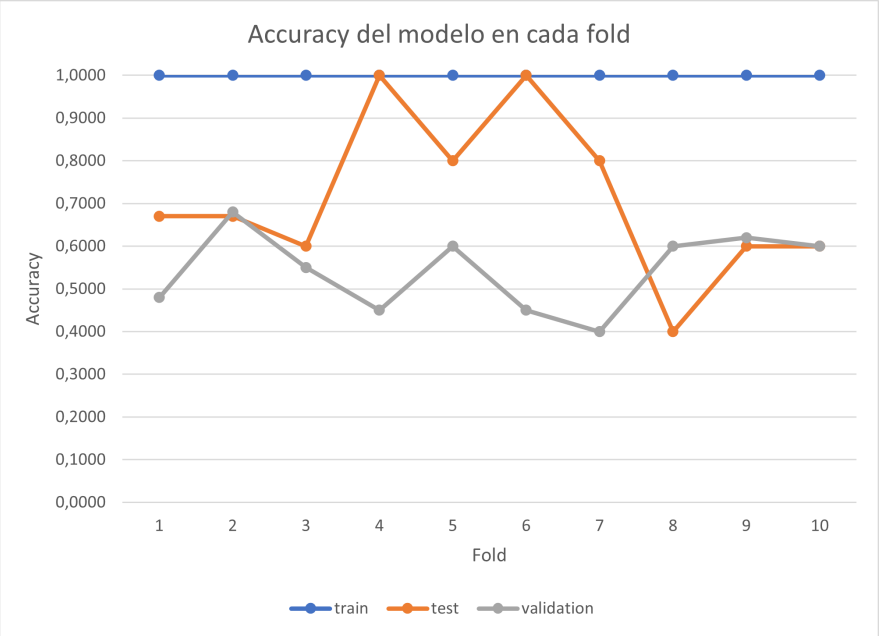

Figura 7.25: Accuracy del Modelo CONV3D con optical flow y sin aumentado de datos en cada fold.

Al igual que el Modelo TD-CNN-LSTM, al observar los resultados del Cuadro 7.18 podemos ver que el modelo se ha ajustado muy bien a los datos de entrenamiento, aunque no haya generalizado bien como el Modelo TD-CNN-LSTM. Además otra diferencia en comparación con el Modelo TD-CNN-LSTM, es que el Modelo TD-CNN-LSTM tiene 91.458 parámetros y este modelo tiene 28.450.644 cosa que pasará factura mas adelante en la eficiencia temporal.

7.6 Optical flow con aumentado de datos

En este apartado realizaremos pruebas sobre nuestros modelos aplicando un preprocesado a nuestros datos (Optical flow) y técnicas de aumentado de datos.

7.6.1 Modelo TD-RN50-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.9967	0.999	0.999	0.9990	99.95% (+/- 0.16%)
Test	0.9902	0.991	0.990	0.9900	99.05% (+/- 1.91%)
Val	0.8040	0.8070	0.8060	0.8030	80.50% (+/- 4.15%)

Cuadro 7.19: Resultados de la prueba 1.

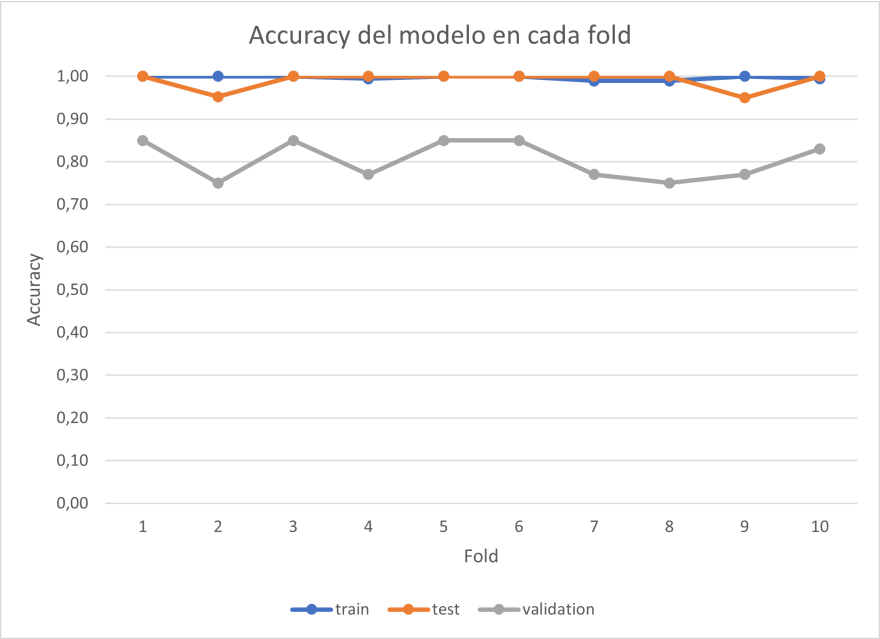

Figura 7.26: Accuracy del Modelo TD-RN50-LSTM con optical flow y aumentado de datos en cada fold.

Como podemos observar en el Cuadro 7.19, este modelo se ha adaptado bastante bien a los datos de entrenamiento y test consiguiendo un accuracy de ±99%. También podemos observar que el modelo a generalizado bastante bien teniendo en cuenta que el conjunto de validación es muy distinto al conjunto con el que hemos entrenado el modelo.

7.6.2 Modelo TD-VGG-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.4957	0.25	0.50	0.33	50.00% (+/- 0.00%)
Test	0.5095	0.249	0.50	0.329	50.00% (+/- 0.00%)
Val	0.50	0.25	0.50	0.33	50.00% (+/- 0.00%)

Cuadro 7.20: Resultados prueba 2.

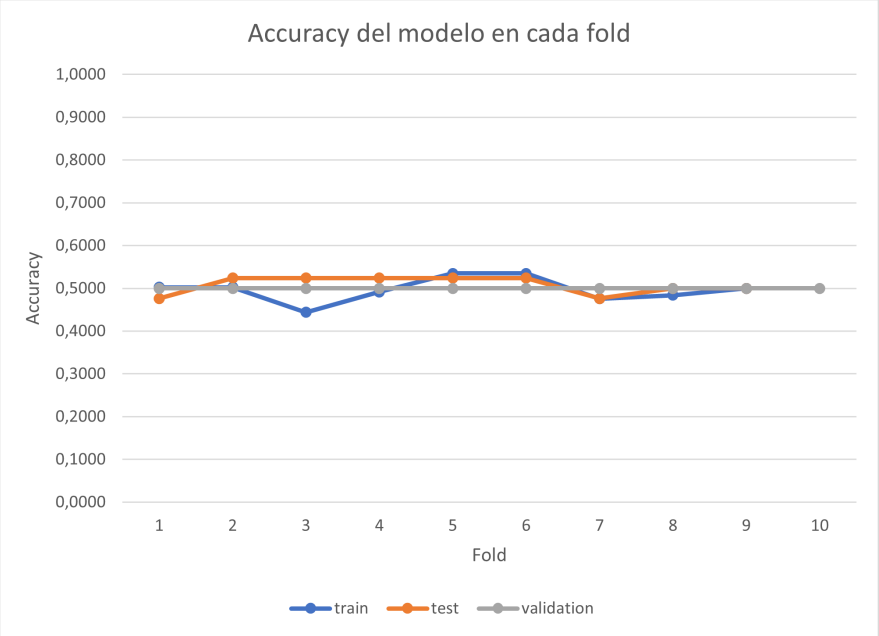

Figura 7.27: Accuracy del Modelo TD-VGG-LSTM con optical flow y aumentado de datos en cada fold.

Si observamos este los resultados del Cuadro 7.20 podemos llegar a la conclusión de que el modelo no esta aprendiendo nada, ni si quiera es capaz de ajustarse a los datos de entrenamiento, donde el AUC es de un 50% (Función rand).

7.6.3 Modelo TD-INCV3-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.7759	0.694	0.762	0.711	76.23% (+/- 17.91%)
Test	0.7857	0.7857	0.764	0.710	76.45% (+/- 18.56%)
Val	0.6249	0.553	0.625	0.549	62.50% (+/- 15.53%)

Cuadro 7.21: Resultados prueba 3.

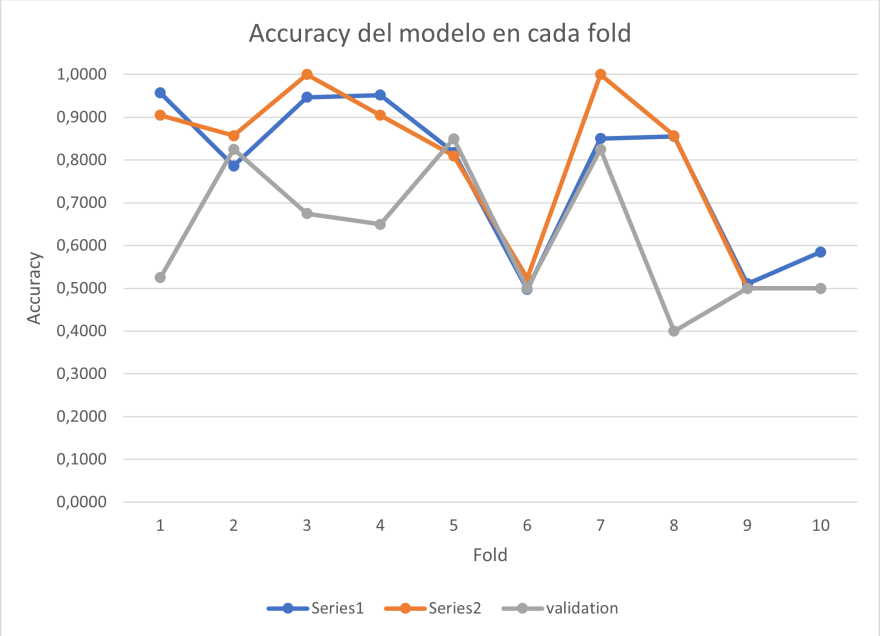

Figura 7.28: Accuracy del Modelo TD-INCV3-LSTM con optical flow y aumentado de datos en cada fold.

A partir del Cuadro 7.21 podemos observar que este modelo ha obtenido resultados aceptables en el entrenamiento aunque no haya conseguido generalizar bien sobre el conjunto de validación. Los resultados son algo flojos si los comparamos con los del Modelo TD-RN50-LSTM, pero son mucho mejores que los del Modelo TD-VGG-LSTM.

7.6.4 Modelo TD-EfB7-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.4957	0.25	0.50	0.33	50.00% (+/- 0.00%)
Test	0.5095	0.249	0.50	0.329	50.00% (+/- 0.00%)
Val	0.50	0.25	0.50	0.33	50.00% (+/- 0.00%)

Cuadro 7.22: Resultados prueba 4.

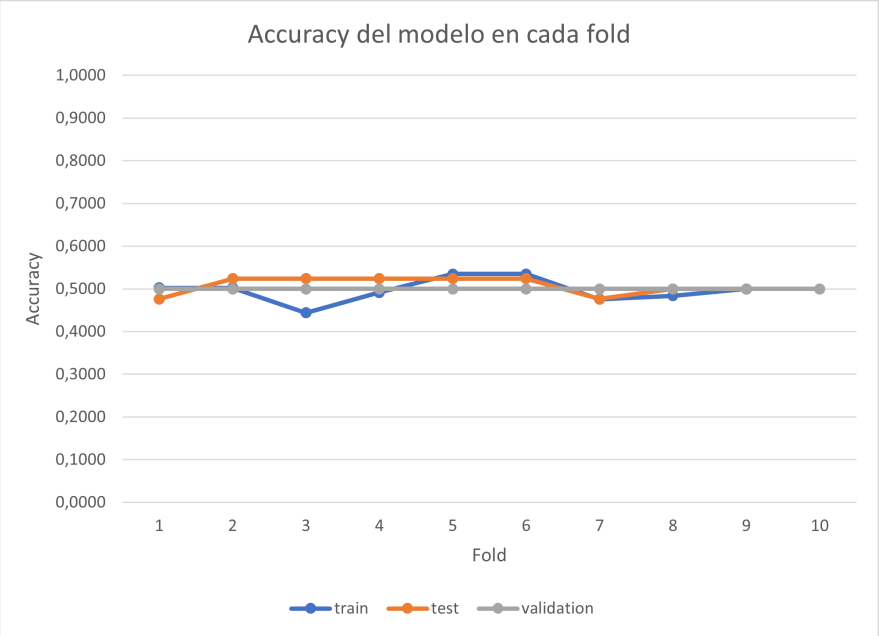

Figura 7.29: Accuracy del Modelo TD-EfB7-LSTM con optical flow y aumentado de datos en cada fold.

Al igual que el Modelo TD-VGG-LSTM, viendo los resultados del Cuadro 7.22 y de los experimentos anteriores, podemos ver que el modelo no aprende.

7.6.5 Modelo TD-CNN-LSTM

	Accuracy	Precision	Recall	F1-Score	AUC
Train	0.997	0.997	0.997	0.997	99,73% (+/- 0,65%)
Test	0.873	0.877	0.875	0,873	87,59% (+/- 4,64%)
Val	0,749	0,784	0,752	0,744	74,75% (+/- 5,41%)

Cuadro 7.23: Resultados prueba 5.

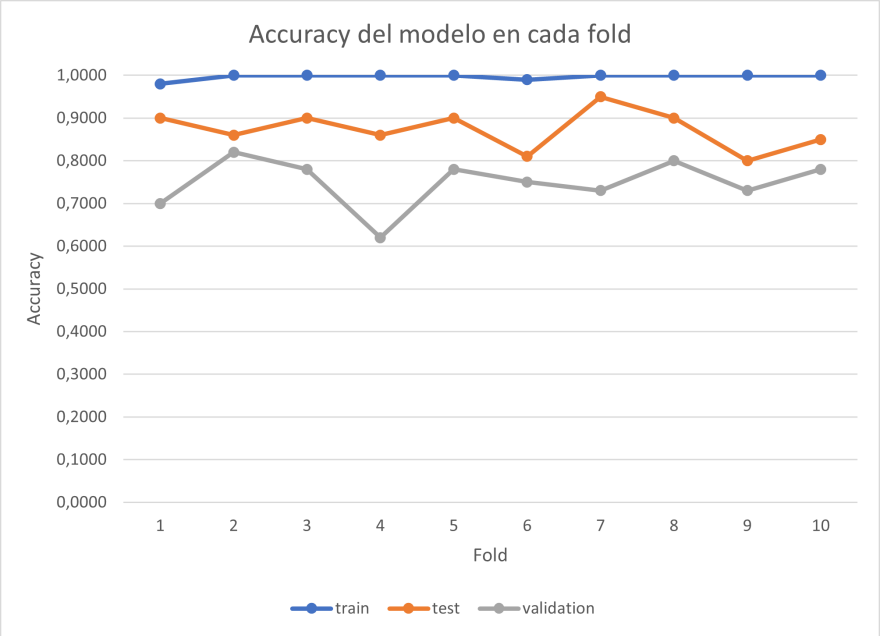

Figura 7.30: Accuracy del Modelo TD-CNN-LSTM con optical flow y aumentado de datos en cada fold.

Observando los resultados del Cuadro 7.23 podemos observar que este modelo ha conseguido aprender bien de los datos de entrenamiento, además de generalizar bien sobre los datos de validación. La diferencia entre este modelo y el Modelo TD-RN50-LSTM es que este modelo es mucho mas simple y con muchos menos parámetros que el Modelo TD-RN50-LSTM por lo que tendrá una eficiencia temporal mejor que la del Modelo TD-RN50-LSTM, cosa que veremos más adelante.

7.6.6 Modelo CONV3D

	Accuracy	Precision	Recall	F1-Score	AUC
Train	1.0	1.0	1.0	1.0	100,00% (+/- 0,00%)
Test	0.673	0.707	0.672	0.656	67,36% (+/- 11,16%)
Val	0.529	0.539	0.529	0.519	53,00% (+/- 9,00%)

Cuadro 7.24: Resultados prueba 6.

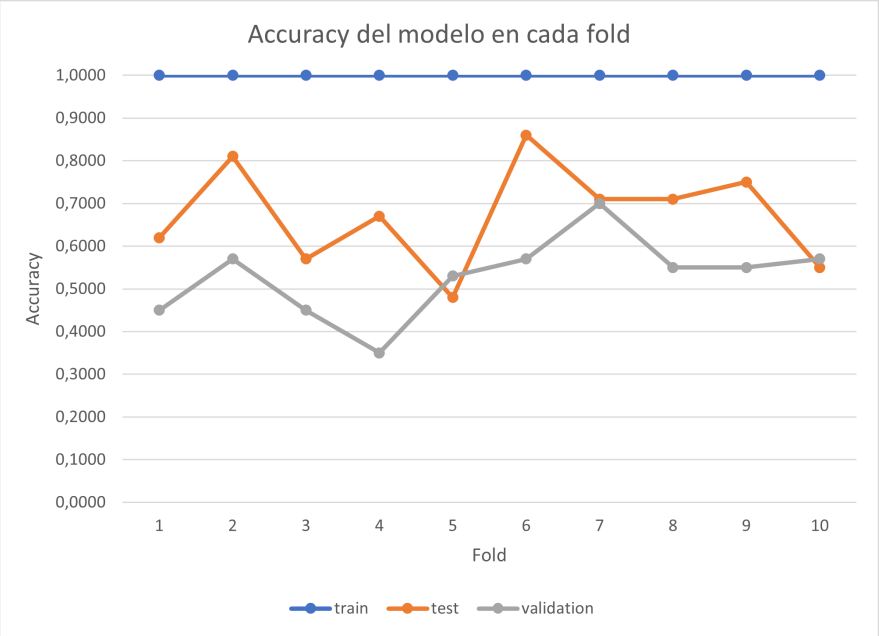

Figura 7.31: Accuracy del Modelo CONV3D con optical flow y aumentado de datos en cada fold.

Como podemos ver en el Cuadro 7.24 modelo ha conseguido ajustarse a los datos de entrenamiento (hasta sobreajustarse) pero no ha conseguido generalizar bien, ni para los datos de test ni para los datos de validación.

7.7 Tiempos de ejecución

Después de evaluar la fiabilidad de nuestros modelos, vamos a comprobar cuanto tardan las predicciones de cada modelo. La eficiencia temporal es un aspecto muy importante ya que si queremos que nuestro modelo sea capaz de detectar caídas que se producen en margen de 1 segundo, el sistema tiene que ser muy rápido para que no se nos escape ninguna caída.

Modelo	Trainable params	Non-trainable params	Tiempo de predicción (s)
Modelo TD-RN50-LSTM	2625538	23587712	3.1s
Modelo TD-VGG-LSTM	1052674	20024384	3.5s
Modelo TD-INCV3-LSTM	2625538	21802784	1.75s
Modelo TD-EfB7-LSTM	3149826	64097687	4.35
Modelo TD-CNN-LSTM	91458	———	1.45s
Modelo CONV3D	28450644	———	5.2s

Cuadro 7.25: Comparativa de los tiempos de ejecución y parámetros de los modelos

Como podemos observar, el modelo que mejor tiempo nos da es el Modelo TD-CNN-LSTM con una velocidad de 1.45s por predicción. Aunque no es el modelo con mejor precisión.

Podemos concluir este apartado de experimentación destacando lo que acabamos de conseguir. Los mejores modelos que hemos conseguido son los siguientes:

Modelo	Validation Accuracy	Tiempo de predicción (s)
Modelo TD-RN50-LSTM	0.80	3.1s
Modelo TD-CNN-LSTM	0.749	1.45s

Cuadro 7.26: Mejores modelos obtenidos

Ambos modelos entrenados con datos preprocesados y aumentados.

7.8 Resultados cualitativos del modelo TD-RN50-LSTM

Después de obtener el mejor modelo, que nos ha dado un accuracy de 0.8055 (Figura), vamos a mas en detalle en casos esta funcionando nuestro modelo y en que casos está fallando.

	nº vídeos	Aciertos
Youtube	16	4
Vídeos própios	8	0
Universidad de Málaga	12	3

Cuadro 7.27: Desglose de aciertos en el conjunto de validación

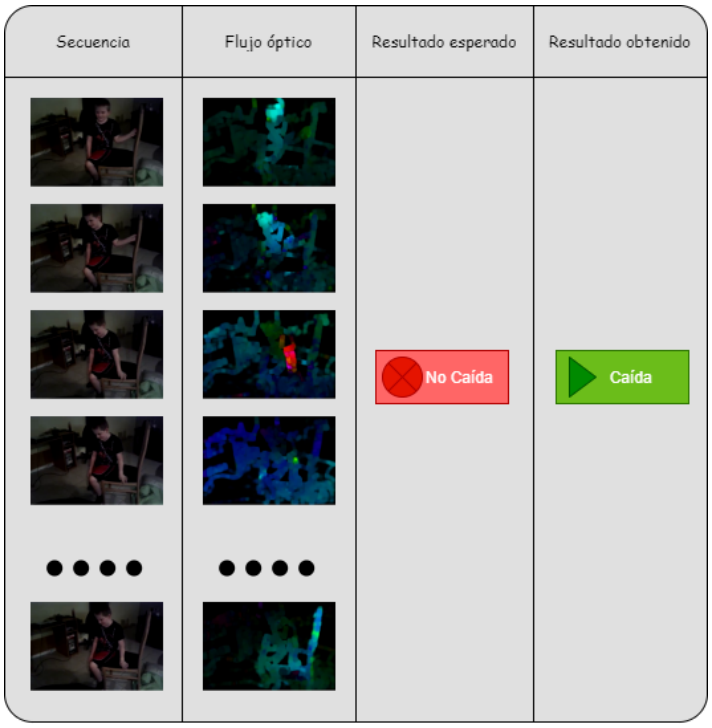

Figura 7.32: Resultado experimento vídeo de youtube

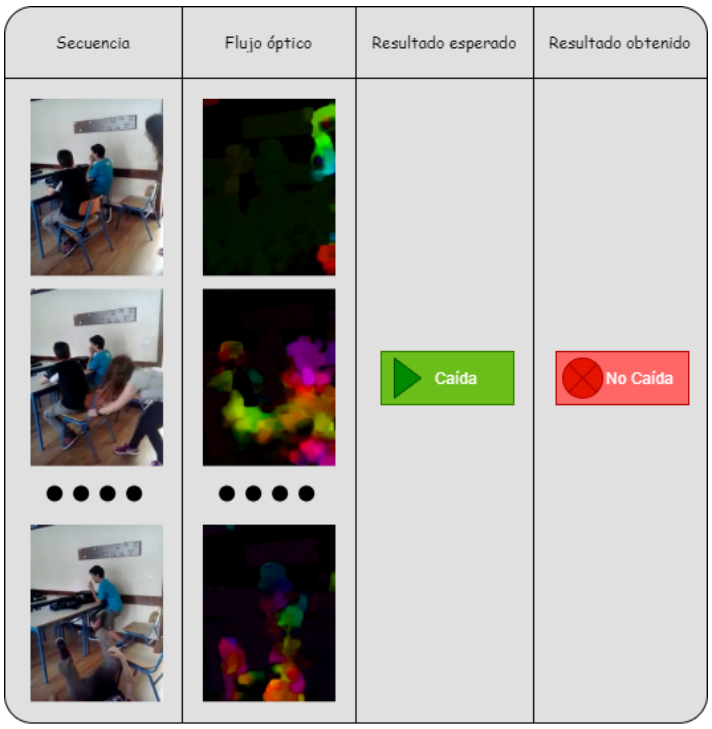

Figura 7.33: Resultado experimento vídeo de youtube

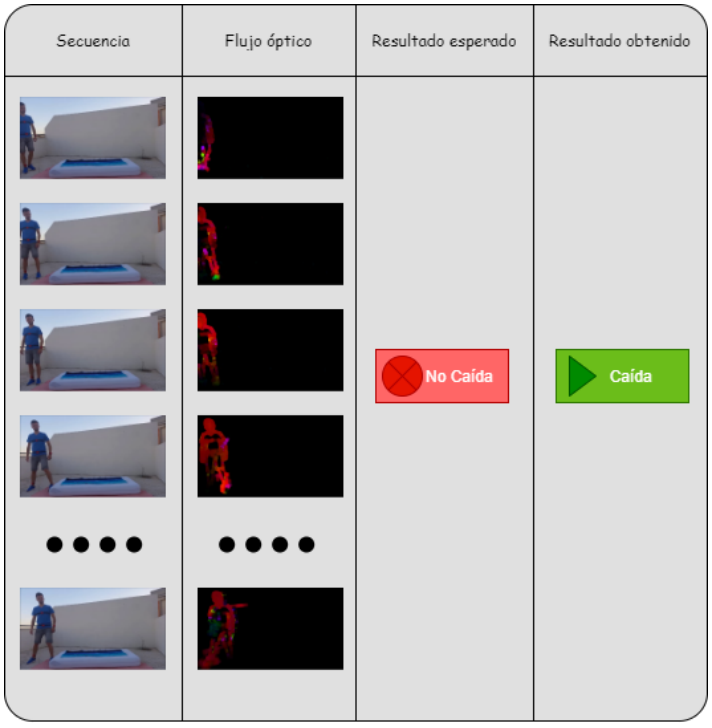

Figura 7.34: Resultado experimento vídeo de la Universidad de Málaga

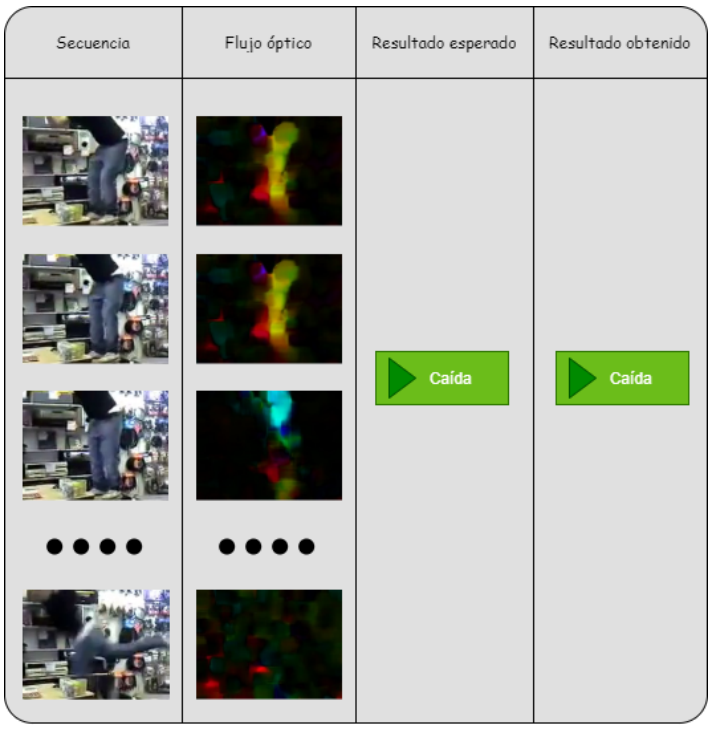

Figura 7.35: Resultado experimento vídeo de youtube

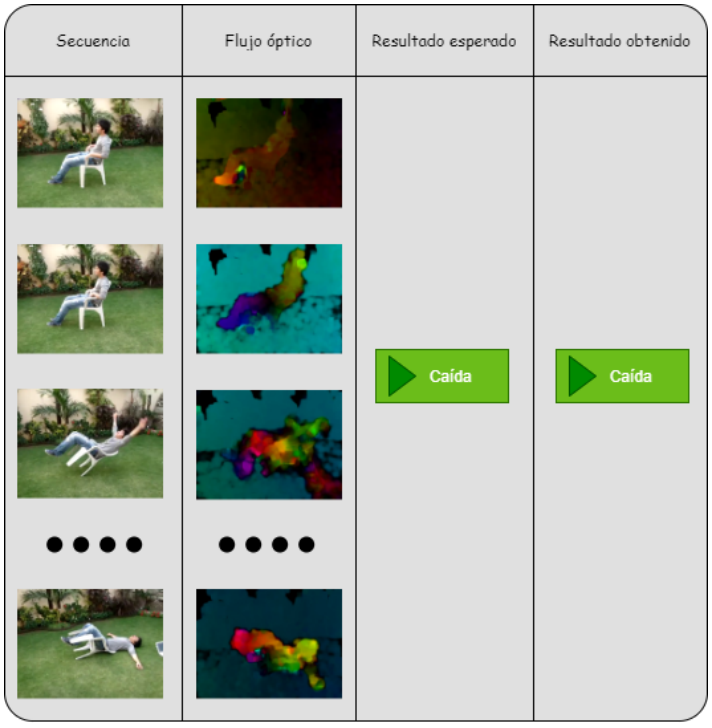

Figura 7.36: Resultado experimento vídeo de youtube

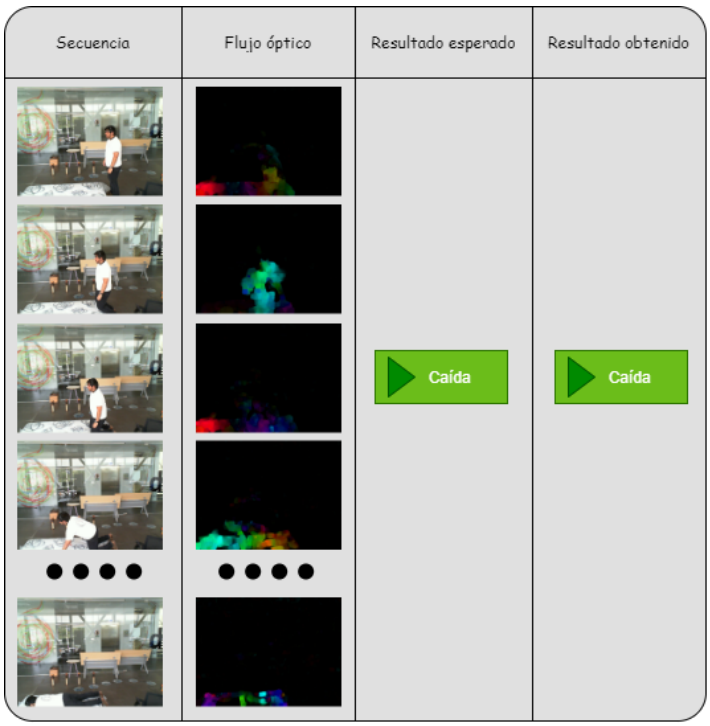

Figura 7.37: Resultado experimento vídeo del Fall detection dataset ImViA

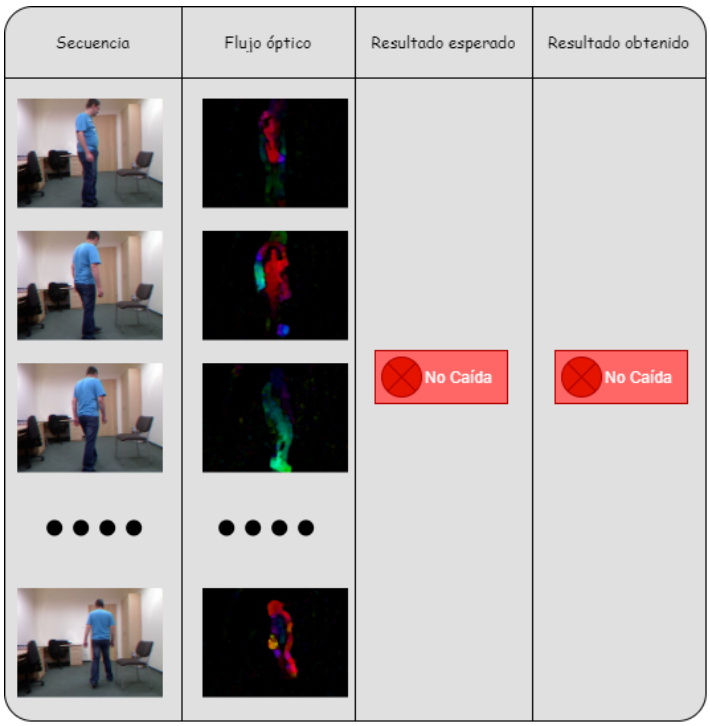

Figura 7.38: Resultado experimento vídeo del UR Fall Detection Dataset

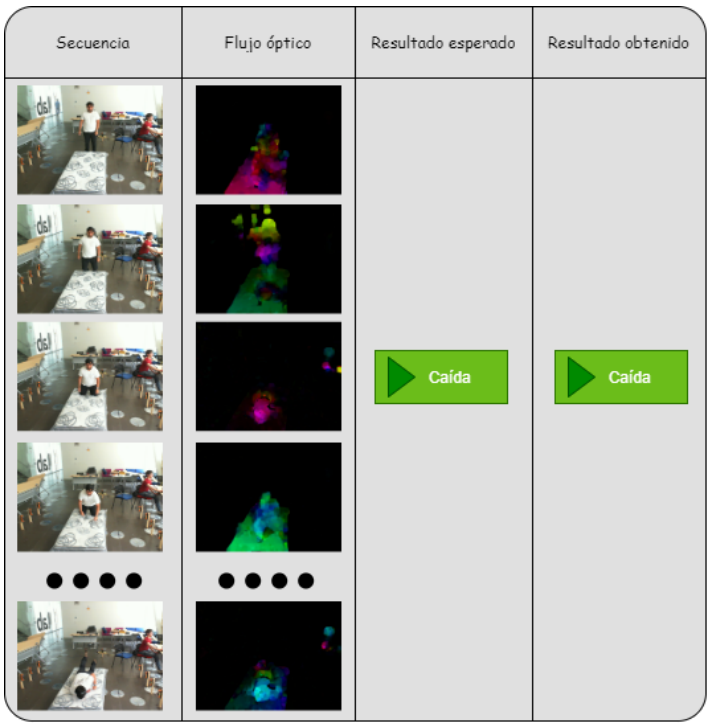

Figura 7.39: Resultado experimento vídeo del Fall detection dataset ImViA

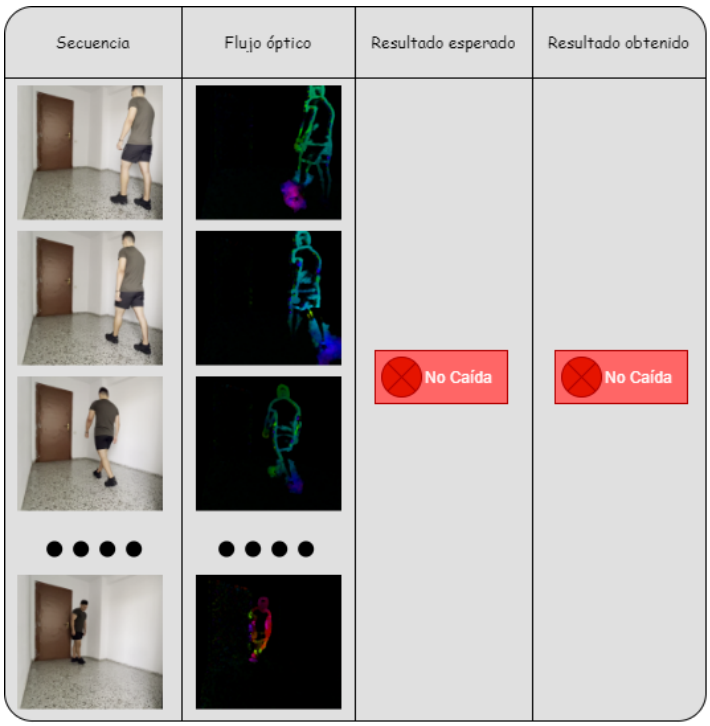

Figura 7.40: Resultado experimento con vídeo grabado en casa

8 Conclusión y Trabajos Futuros

A lo largo del desarrollo de este proyecto hemos podido realizar un estudio sobre las técnicas actuales de los sistemas detección de caídas. A modo de conclusión nos gustaría hacer una visión global del proyecto que hemos realizado así como de los objetivos y aprendizajes alcanzados.

Para poder entender el alcance que tiene nuestra propuesta, hay que destacar la principal limitación que nos hemos encontrado durante el desarrollo, la cantidad de datos tan limitada que disponíamos (aun habiendo aplicado técnicas de aumentado de datos). Estos eran una proporción lo suficientemente pequeña como para no poder alcanzar y poner a prueba un sistema que diese un resultado con las características necesarias como para poder llevarlo a un producto realista, ya que para el problema que nos enfrentamos son muchos los casos posibles que nos podemos encontrar y, por tanto, es necesario un gran conjunto de datos que pueda contemplar cada uno de ellos. Sin embargo, no resta a que se han podido lograr los objetivos propuestos en este trabajo, con los cuales además hemos conseguido un buen punto de partida para, en un futuro, si se desea y disponiendo de un mejor conjunto de datos, poder adaptar nuestra propuesta a un producto competitivo que ofrezca solución a un problema que hasta ahora no se ha planteado desde una perspectiva como la nuestra, mediante el uso de cámaras. Ya que hemos podido desarrollar un sistema capaz de detectar caídas mediante visión por computador con unos resultados bastante aceptables, pues nuestro sistema ha sido capaz de adaptarse bien a la muestra con la que le hemos entrenado y ha sido capaz de generalizar bastante bien ($\pm$80% precisión) sobre un conjunto de vídeos variados con poca relación entre ellos que hemos usado a modo de validación de nuestro sistema. De manera que, como hemos comentado, el modelo ha demostrado que ha aprendido bien, y a partir de las pruebas que hemos realizado consideramos que estos buenos resultados se extrapolarían tras un respectivo aumento del conjunto de datos, permitiendo que en un futuro se mejore en gran medida la red.

Como posibles mejoras que consideramos importante a mencionar, se podría construir una app donde usar nuestro sistema para notificar de las caídas a los contactos de emergencias de los usuarios. Además, aun estando este trabajo basado al 100% en visión por computador y deep learning, creemos que podría aportar una mejora significativa combinar la información visual con la de algún tipo de sensor extra (acelerómetro, sensor de presión...), puede que mejore los resultados (aunque para asegurarlo tendríamos que probarlo).

Tras haber analizado los diferentes resultados que hemos alcanzado, puedo concluir que ha sido un proyecto en el que personalmente he podido aprender mucho, tanto en el ámbito de visión por computador y deep learning como uno más académico, ya que he podido aprender a realizar cada una de las etapas de un trabajo de investigación, desde el estudio del estado del arte, al desarrollo y propuesta de una solución y su respectiva evaluación.

En el siguiente vídeo se puede observar funcionamiento del sistema propuesto Detección de caídas para dispositivos de bajo coste

Bibliografía

[1] "Caídas," *World Health Organization*, Apr 2021.

[2] A. Rodríguez-Molinero, L. Narvaiza, C. Gálvez-Barrón, J. J. de la Cruz, J. Ruíz, N. Gonzalo, E. Valldosera, and A. Yuste, "Caídas en la población anciana española: incidencia, consecuencias y factores de riesgo," *Revista Española de Geriatría y Gerontología*, vol. 50, no. 6, pp. 274–280, 2015.

[3] H. M, "Deep learning o aprendizaje profundo : ¿qué es?," *DataScientest.com*, Apr 2022.

[4] O. G.-O. Olivera, "Redes neuronales artificiales: Qué son y cómo se entrenan," *Xeridia*, Oct 2021.

[5] B. AI, "Redes neuronales convolucionales," *Medium*, Nov 2019.

[6] J. TORRES, "Redes neuronales recurrentes," *Medium*, Jan 2021.

[7] M. A, "¿qué es el transfer learning?," *DataScientest.com*, Mar 2022.

[8] S. University and P. University, Mar 2021.

[9] R. R, *VGG: ¿Qué es este modelo? ¡Daniel Te Lo cuenta todo!*, Jun 2022.

[10] D. S. Team, "Redes neuronales residuales - lo que necesitas saber (resnet)," *DATA SCIENCE*, Dec 2020.

[11] N. Noury, "Monitoring behavior in home using a smart fall sensor and position sensors," *IEEE-EMBS Microtechnologies in Medicine Biology*, pp. 607–610, 2000.

[12] C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau, "Robust video surveillance for fall detection based on human shape deformation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 21, pp. 611 – 622, 06 2011.

[13] N. Lu, Y. Wu, L. Feng, and J. Song, "Deep learning for fall detection: Three-dimensional cnn combined with lstm on video kinematic data," *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 1, pp. 314–323, 2019.

[14] D. Lerch, *Fine-tuning en reconocimiento de Imágenes Mediante Deep Learning*, Nov 2021.

[15] L. Martínez-Villaseñor, H. Ponce, J. Brieva, E. Moya-Albor, J. Nuñez-Martínez, and C. Peñafort-Asturiano, "Up - fall detection," 2020.

[16] U. de Bourgogne, "Fall detection dataset," Apr 2020.

[17] M. Kępski, "Ur fall detection dataset."

[18] E. Auvinet, C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau, "Multiple cameras fall dataset," Jul 2010.

[19] R. Valencia, *Classifying Spatiotemporal Inputs with CNNs, RNNs, and MLPs*.

[20] P. Ferlet, "How to work with time distributed data in a neural network," *Medium*, Nov 2019.

[21] J. Brownlee, *How to use the timedistributed layer in Keras*, Aug 2019.