UNIVERSIDAD PONTIFICIA COMILLAS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO INDUSTRIAL

TRABAJO FIN DE MÁSTER:

IDENTIFICACIÓN DE ACCIONES EN FORMATO VÍDEO CON UN MODELO DE REDES CONVOLUCIONALES 3D (3DCNN)

AUTOR: Alberto Castillo Rodríguez

DIRECTORES: Álvaro López López, Lucía Güitta López

MADRID, 21 de julio de 2021

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título "IDENTIFICACIÓN DE ACCIONES EN FORMATO VÍDEO CON UN MODELO DE REDES CONVOLUCIONALES 3D (3DCNN)"

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2020/2021 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Alberto Castillo Rodríguez Fecha: 20/07/2021

Autorizada la entrega del proyecto

LOS DIRECTORES DEL PROYECTO

Fdo.: Lucía Güitta López Fecha: 21/07/2021

Fdo.: Álvaro Jesús López López Fecha: 21/07/2021

UNIVERSIDAD PONTIFICIA COMILLAS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO INDUSTRIAL

TRABAJO FIN DE MÁSTER:

IDENTIFICACIÓN DE ACCIONES EN FORMATO VÍDEO CON UN MODELO DE REDES CONVOLUCIONALES 3D (3DCNN)

AUTOR: Alberto Castillo Rodríguez

DIRECTORES: Álvaro López López, Lucía Güitta López

MADRID, 21 de julio de 2021

IDENTIFICACIÓN DE ACCIONES EN FORMATO VÍDEO CON UN MODELO DE REDES CONVOLUCIONALES 3D (3DCNN)

Autor: Alberto Castillo Rodríguez.

Directores: Álvaro López López, Lucía Güitta López.
Entidad Colaboradora: Instituto de Investigación Tecnológica (IIT).

RESUMEN DEL PROYECTO

Introducción

El proyecto surge a raíz del gran avance que está teniendo la inteligencia artificial estas últimas décadas y del gran impacto que este puede tener en el mundo empresarial y en la sociedad. Estos avances en los inicios del machine learning se produjeron en el campo de la información numérica estructurada, pero con el desarrollo y abaratamiento del poder de computación el campo de la visión artificial ha ido en aumento.

Las posibilidades de conseguir identificar acciones en vídeo son enormes ya que revolucionaría sectores como el de la seguridad donde se podría mejorar la detección de robos o de acciones violentas y donde se requiere que una sola persona vigile un gran número de cámaras al mismo tiempo. Este trabajo quiere dar respuesta a esta necesidad de innovar en el campo de la visión artificial y de donde diferentes empresas pueden encontrar ventajas competitivas si logran este objetivo.

Este avance no solo tiene que verse reflejado en los resultados de precisión, sino que también tiene que hacerse teniendo en cuenta las limitaciones de recursos, ya sea tanto en la posibilidad de conseguir vídeos como de capacidad de procesamiento para entrenar y para ejecutar la red. Si el modelo necesita de muchos recursos las imágenes se tienen que enviar a un servidor externo, conocido como Cloud Computing, frente a procesarlo localmente, conocido como Edge Computing.

Es por todo esto que se ha elegido la estructura 3 Dimensional Convolutional Neural Network (3D CNN) como base para probar diferentes arquitecturas que se utilicen de alguna forma en ésta. Una ventaja de utilizar redes convolucionales es que permiten aumentar la velocidad de procesamiento con él uso de la GPU, al estas estar optimizadas para trabajar con matrices que es justamente la base las operaciones convolucionales, y de esta forma poder tanto aligerar la carga al entrenar y al aplicarlas. Además, frente a las redes recurrentes son más eficientes por el menor número de pesos necesarios.

Objetivos

El objetivo es desarrollar un algoritmo capaz de detectar acciones en vídeos de cámaras de seguridad. Para ello se construirá la base de datos y se desarrollará el algoritmo.

Obtención de la base de datos

Se creará la base de datos con la que se probaran los distintos algoritmos y métodos de preprocesamiento. Ésta debe cumplir los siguientes criterios:

- Debe estar balanceada
- Debe ser preferiblemente de acceso público
- Deben estar bien clasificados
- Deben estar recortados
- Deben de tener la máxima calidad posible
- Debe haber una cantidad considerable de datos

Desarrollo del algoritmo

Se desarrollará diferentes algoritmos que utilicen la estructura de 3D CNN parcialmente o en su totalidad atendiendo a los siguientes criterios:

- Que sea ligero
- Lenguaje de programación: que utilice el lenguaje de programación Python [1] y alguno de los dos Deep Learning frameworks más importantes como son Tensorflow [2] o PyTorch [3] para facilitar su integración en una aplicación final.
- Que tenga una precisión de al menos 70 % en validación.

Solución

Ética en la IA

A la hora de diseñar un sistema que implemente algoritmos de inteligencia artificial, es muy importante mencionar la ética. Estos tipos de sistemas eliminan la intervención de seres humanos en la toma de decisiones y, por lo tanto, se corre el riesgo de crear sistemas que intencionada o accidentalmente tomen decisiones basadas en raza, sexo u otras categorías.

Datos

Obtención de la base de datos

Se exploraron bases de datos públicas como UCF 101 [4] o Kinetics [5], buscando categorías de vídeos de cámaras de seguridad. Al no haber una cantidad razonable para conseguir una base de datos robusta, se realizó un script en Python para descargar vídeos de YouTube con las características ya mencionadas. El número total de vídeos extraídos fueron 167 con muy diversas características de resolución, fotogramas por segundo (FPS) y escenarios o entornos.

Analizando el bias de la base de datos, teniendo en cuenta el contexto del uso de vídeos de cámaras de seguridad, observamos un gran desequilibrio en la representación de distintos tonos de color de piel y estilos culturales. Por lo tanto, a pesar de que la base de datos se puede seguir usando para probar distintos tipos de algoritmos, los algoritmos entrenados a partir de la base de datos no pueden ser usados en entornos de producción.

Post-procesamiento de la base de datos

Una vez hemos obtenido los vídeos, es necesario recortarlos y clasificarlos correctamente como vídeos normales y vídeos en los que se comete la acción. Para ello, se generó en un archivo .txt una lista con los respectivos nombres de cada archivo y los segundos que queremos etiquetar junto al nombre de esta. Tras esta clasificación, un script de Python recortó automáticamente los vídeos y los clasificó según la etiqueta. El resultado final fue la creación de 204 vídeos con situaciones normales y 208 vídeos con la acción. Para mejorar la calidad de la base de datos se han tenido en cuenta las siguientes consideraciones.

Recorte de vídeos

Los vídeos recortados deben tener exclusivamente la acción a detectar, dando como resultado clips de duración muy variable en un rango de entre 2 y 10 segundos. Para evitar que

haya un bias por la duración de estos vídeos recortados, por cada uno que contenga una acción dentro de lo posible se ha intentado obtener otro sin ninguna acción con la misma duración.

Bias del fondo y las personas

Una de las formas en el que el algoritmo podría falsamente clasificar los vídeos podría ser detectando características propias de las personas que cometen la acción frente a las que no las cometen. Es por eso por lo que se intentó siempre que fuese posible obtener recortes de cada vídeo donde se muestre a la misma persona cometiendo la acción y en una situación normal.

Figura 1. *Ejemplo de cómo se puede determinar la acción sin ver a la persona que la comete. De izquierda a derecha: salto, golf, canoa y patinaje sobre hielo. [6]*

Esto ocurre también con el entorno de la grabación, como se refleja en el paper "Why Cant I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition"[6]. En este artículo se demuestra que tanto los algoritmos como los humanos hacemos uso del contexto para determinar, sin ver a la persona, la acción que se está cometiendo (figura 1). Eliminando este bias, es posible mejorar la generalización del algoritmo pero también dificulta la tarea al reducir la información disponible. Se ha eliminado el bias generando vídeos de situaciones normales y con la acción en el mismo entorno evitando la memorización de características de este para la clasificación.

Filtro de movimiento

Otra forma de resolver este problema definitivamente sería capturando solo el movimiento que se detecta en cada vídeo. El ángulo de visión y posición de las imágenes son constantes y por tanto son las personas las únicas que modifican los píxeles en el tiempo cuando se mueven. Calculando la diferencia absoluta entre fotogramas consecutivos en blanco y negro, obtenemos píxeles negros donde no ha habido movimiento y píxeles blancos donde sí. Para ser más resiliente al ruido, es preferible calcular la diferencia con una media móvil ponderada de los fotogramas anteriores aunque esto produce la aparición del efecto secundario de imagen fantasma como se puede ver en la figura 2. Esta operación se puede realizar con la librería cv2 de OpenCV [7].

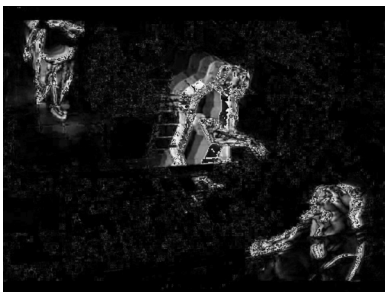

Figura 2. *Fotograma de uno de los vídeos al aplicar el filtro de movimiento.*

División de la base de datos

El último paso para preparar la base de datos para el proceso de aprendizaje es su división en tres grupos donde se ha intentado balancear las categorías lo máximo posible:

- **Train o entrenamiento:** este subgrupo será utilizado para entrenar el modelo automáticamente. Es por tanto el más grande, con un 70 % de los vídeos, 144 normales y 141 con la acción.

- **Validation o validación:** este subgrupo será utilizado para manual o automáticamente afinar los hiperparámetros del modelo. Contará con un 15 % de los vídeos, 31 normales y 35 con la acción.

- **Test o prueba:** este último subgrupo sirve para validar los resultados del modelo. Los hiperparámetros no deben ser modificados para mejorar el resultado en este subgrupo. Contará con un 15 % de los vídeos, 31 normales y 30 con la acción.

Data Augmentation

El objetivo de las técnicas es incrementar el número de datos disponible aplicando modificaciones a los datos ya existentes para así permitir que el algoritmo generalice mejor a datos un poco diferentes a los ya existentes. Las modificaciones pueden ser:

- **Espaciales:** los vídeos se reflejan horizontalmente para así conservar en la misma orientación el eje vertical.

- **Color:** modificar el espectro de colores, contraste y brillo entre otros parámetros permite al algoritmo ser más resiliente a estas variaciones entre vídeos.

- **Ruido:** introducir ruido en las imágenes permite al algoritmo trabajar con distintas calidades de vídeos y ser más robusto ante estas variaciones.

Pipeline

El trabajo del pipeline es gestionar de la forma más óptima posible el proceso de entrenamiento. Esto abarca desde realizar las transformaciones pertinentes a los datos, cargarlos en la RAM, CPU o la GPU, realizar las operaciones de entrenamiento, validación y prueba y finalmente guardar los resultados y el modelo.

Limitaciones del hardware

El primer aspecto a tener en cuenta al diseñar el pipeline son los recursos disponibles durante todo el proceso y los requerimientos de los algoritmos durante el entrenamiento. El objetivo es maximizar la velocidad de entrenamiento optimizando el uso del hardware del sistema. En este proyecto, hay dos recursos limitantes a la hora de minimizar el tiempo de entrenamiento: la RAM y la GPU.

Machine Learning Frameworks

Para facilitar y optimizar la definición de los algoritmos, el proceso de entrenamiento y la reproducibilidad de todo el proceso, se han desarrollado diversos frameworks. PyTorch [3] nos permite acceder fácilmente a los bucles de entrenamiento para poder modificarlos según nuestras necesidades, tanto para probar nuevas arquitecturas como para visualizar los datos.

Carga de datos

Una vez ya tenemos toda la base de datos generada, se debe configurar la interfaz que permita leer los datos, procesarlos y cargarlos en el dispositivo que realice el entrenamiento. Al no disponerse de suficiente RAM ni espacio en la memoria de la GPU para alojar toda la base de datos, la estrategia a seguir será cargar los datos poco a poco según se vaya necesitando durante el proceso de entrenamiento. Para ello se utilizaran dos clases: Dataset y DataLoader.

Optimizador

El proceso de entrenar un algoritmo de inteligencia artificial consiste en definir una función de pérdidas que represente como de bien realiza su cometido y mejorar este resultado a lo largo de las épocas de entrenamiento. El problema de minimizar esta función es que en general no son convexas y por lo tanto no tienen un único mínimo sino que tienen mínimos locales. La estrategia con la que se intenta encontrar el mínimo global es muy importante para evitar quedarse atascado en mínimos locales.

Los optimizadores son unas herramientas para resolver todos estos problemas. La mayoría de los optimizadores calculan el learning rate automáticamente y aplican el gradiente al modelo haciendo que este aprenda. Un buen optimizador es aquel que entrena rápido y a la vez evita que el modelo se quede atascado en un mínimo local. El optimizador Adam [9] es el que mejor se desenvuelve en la mayoría de escenarios [10] y es por eso por es el que finalmente se ha aplicado.

Función de perdidas

A la hora de evaluar como de bien un algoritmo ha realizado una tarea se debe definir una métrica para cuantificarlo matemáticamente. Se debe elegir una que represente la función de pérdidas. Esta función será la que se minimice durante el proceso de entrenamiento y por lo tanto no debe solo representar como de bien ha realizado la clasificación, sino también su confianza tanto al acertar como al fallar. La función Cross-Entropy Loss es la seleccionada por:

- Usar la última capa Softmax nos permite comprobar fácilmente las probabilidades que el modelo otorga a lada categoría. De esta forma se puede visualizar intuitivamente lo que hace el modelo.

- Puede ser aplicada tanto a problemas de clasificación binarios como a multiclase. Esto nos permite no tener que modificar la estructura del modelo en caso de añadir más categorías.

- Es una función probada extensamente en la literatura con resultados comprobados.

Optimización de hiperparámetros

La optimización de hiperparámetros es una de las fases más importantes en el proceso de entrenamiento de un modelo. Si normalmente los parámetros son entrenados automáticamente, los hiperparámetros son aquello que describen o configuran el proceso de entrenamiento y por lo tanto están fijos durante el proceso. El impacto de cada uno es diferente y existen diversos procesos de optimización.

Algoritmos

Una vez hemos diseñado un pipeline que es capaz de proporcionar los datos y gestionar el proceso de entrenamiento, debemos enfrentarnos al diseño de un algoritmo capaz de resolver el problema de detección de acciones en vídeos de cámaras de seguridad.

Problemas intrínsecos del objetivo

Al resolver problemas con modelos de inteligencia artificial, se tiene que tener mucho cuidado al analizar las dificultades que los datos pueden presentar. Si se consideran todas las fuentes de bias, es posible dirigir el diseño del modelo para evitar estos problemas de raíz. Se han identificado los siguientes desafíos:

- Múltiples categorías en el mismo vídeo.
- Superposición de sujetos.
- Entorno no identificativo.

Modelo

Una vez hemos identificado las dificultades, se han propuesto varios modelos o combinaciones de ellos para superarlas. Además de estas consideraciones, se ha tenido en cuenta las limitaciones de la capacidad de computación y de la base de datos generada.

Estrategias de eliminación de problemas intrínsecos

El principal problema intrínseco a resolver que podría presentar las mayores dificultades es el de la aparición de múltiples personas con diferentes categorías. Para resolverlo, se proponen dos métodos distintos: uso del filtro de movimiento y el conjunto de modelos YOLO v4 con Deep Sort.

Algoritmo principal de clasificación

El objetivo de este trabajo es desarrollar un algoritmo basándose en la capa 3DCNN, la cual consiste en operaciones convolucionales de dos dimensiones espaciales y una dimensión temporal. Existen muchos modelos que utilizan esta estructura con complejidad variable pero debido al gran número de parámetros a los que se llega y la limitada capacidad de computación, se ha decidido explorar la arquitectura más simple que la aplica como prueba de concepto.

Esta arquitectura está compuesta por 5 capas convolucionales en serie seguidas de 2 capas fully connected o conectadas completamente con la operación softmax al final para calcular las probabilidades asignadas para cada clase (figura 3). Esta arquitectura se describe en el artículo "Multi-cue based 3D residual network for action recognition"[11] como modelo base de esta capa.

Esta arquitectura se ha modificado para resolver varios de los problemas clásicos que los modelos de Deep Learning tienen:

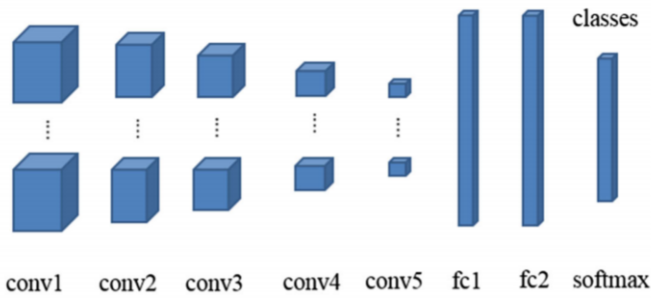

Figura 3. *Capas del modelo 3DCNN [11].*

- Overfitting
- Underfitting
- Carga computacional
- Internal covariate shift

Las capas utilizadas para combatir estos problemas son:

- **Max pooling:** Max pooling es un proceso de discretización cuyo objetivo es reducir el tamaño de su entrada al transformar la ventana de la operación en el valor máximo contenido en ella. Así reducimos el número de parámetros necesarios conservando las características de la región y reducimos el underfitting al proveer una forma abstracta de las subregiones a las que se les ha aplicado el filtro.
- **Dropout:** Esta técnica de regularización tiene como objetivo prevenir el overfitting. Se consigue ignorando un porcentaje de unidades durante el proceso de entrenamiento elegidas de forma aleatoria en cada época. Ignorándolas se permite no modificar su valor durante el proceso y eliminar co-dependencias entre los parámetros.
- **Batch normalization:** Esta capa se diseñó en el artículo "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"[12] para atajar el problema del internal covariate shift. Permite acelerar el entrenamiento reduciendo el número de épocas y suavizar la función de pérdidas.

Resultados

El desarrollo del modelo constaba de tres partes: construcción de la base de datos, desarrollo del pipeline y diseño y entrenamiento del modelo.

Base de datos

Se consiguieron 412 vídeos totales, 204 de ellos con la categoría normal y 208 de la acción a detectar. Estos se dividieron en tres particiones: train, validation y test con el 70 %, 15 % y 15 % correspondientemente. Se aseguró su balanceo en categorías en cada subdivisión. Muchos de ellos eran de calidad deficiente por lo que se decidió crear una segunda base de datos con solo aquellos que cumpliesen con unos requisitos mínimos de resolución y que se acogiesen a una definición más clara de la acción. Esta segunda base de datos contiene 210 vídeos, 105 de cada categoría, manteniendo porcentajes similares y estando balanceadas las categorías.

En cuanto a la calidad ética de la base de datos, no existe una gran diversidad étnica y sociocultural debido a las mayores restricciones que este contenido tiene en los países occidentales y a su mayor capacidad para aplicarlas. Es por eso que no es recomendable su uso para desarrollar modelos de producción.

Pipeline

Tras aplicar todas las técnicas mencionadas, se ha conseguido un pipeline eficiente con los recursos disponibles. Se ha comprobado la correcta carga y transmisión de los datos al algoritmo, así como la aplicación de las transformaciones pertinentes antes del entrenamiento. No se ha podido aplicar el algoritmo YOLO junto con Deep SORT debido a la falta de fiabilidad durante el recorte de las personas. Esto se debe a la baja calidad de los vídeos y a la excesiva superposición de múltiples personas. El optimizador Adam fue el usado finalmente debido a su fácil implementación y los buenos resultados que se obtienen en la mayoría de las aplicaciones de deep learning. En cuanto a la función de perdidas, la función aplicada fue Cross-Entropy Loss por su versatilidad en cuanto al número de clases con las que se puede aplicar, los buenos resultados en la literatura y permitirnos observar directamente las probabilidades que el modelo asigna. La optimización de hiperparámetros se realizó manualmente, tomando decisiones basadas en la información registrada en TensorBoard de entrenamientos pasados.

Modelo

Tras muchas iteraciones en la estructura del modelo y de los hiperparámetros, se ha conseguido obtener overfitting en los datos de entrenamiento, lo que implica que el modelo tiene la complejidad suficiente para aprenderse el problema. Esto se puede observar en la figura 4, donde la función de perdidas decrece a lo largo de las épocas en los datos de entrenamiento, pero aumenta en los datos de validación.

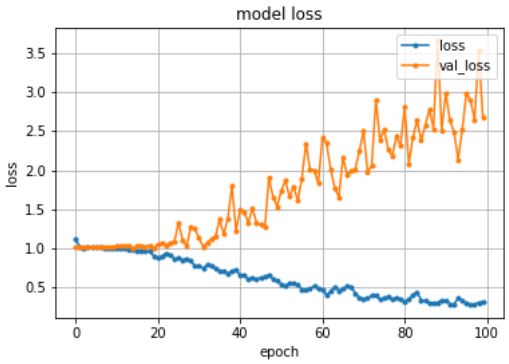

Figura 4. *Evolución de la función de perdidas.*

A pesar de ello, no se ha conseguido que el modelo obtenga mejores resultados en el proceso de clasificación que hacer predicciones aleatorias, dando lugar a una precisión de en torno el 50 % en validación como es el caso cuando tenemos una base de datos de dos categorías balanceada. Estas observaciones se reflejan en la figura 5, donde obtenemos una precisión en los datos de entrenamiento cercana al 100 % debido al overfitting pero en los datos de validación oscila en torno al 50 %.

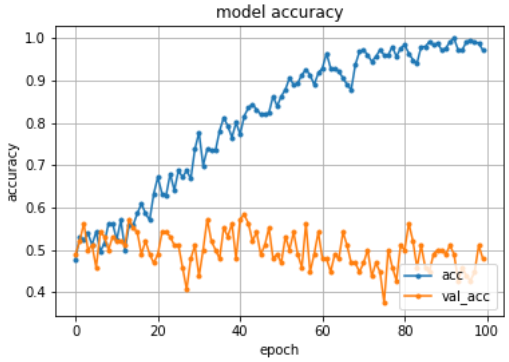

Figura 5. *Evolución de la precisión.*

Representando la matriz de confusión (figura 6), se observa que el algoritmo prefiere clasificar como la acción los casos normales dando lugar a falsos positivos. En este caso, es preferible a los falsos negativos los cuales además se encuentran en menor proporción.

Conclusiones

De los resultados obtenidos, se pueden extraer diferentes causas de porque no se ha obtenido un modelo capaz de realizar predicciones.

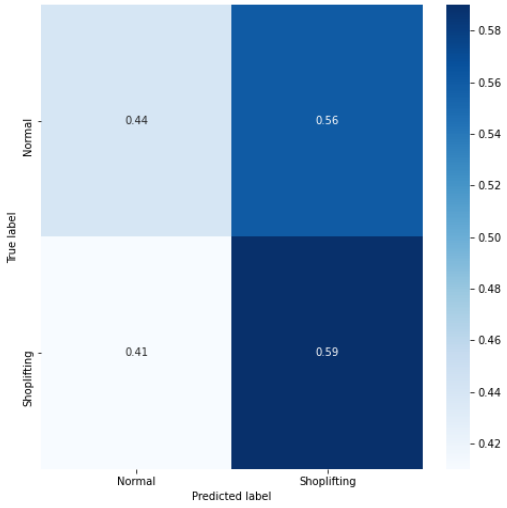

Figura 6. *Matriz de confusión.*

Base de datos

En cuanto a la base de datos, se ha cumplido los siguientes objetivos:

- Construir una base de datos balanceada.
- Usar datos de acceso público.
- La correcta clasificación de los vídeos.
- Recortados para solo representar el tramo donde aparece la acción.

No se han recortado finalmente a las personas individuales con YOLO junto con Deep SORT ya que no es muy fiable a la hora de realizar bien la operación en situaciones con muchas personas cercanas. Respecto a la calidad de los vídeos y al número encontrado, las grandes restricciones impiden encontrar vídeos de calidad y en cantidad, limitando la variedad disponible y con grandes deficiencias de cara al aspecto ético. Es muy importante, especialmente cuando se trata con vídeos de cámaras de seguridad, tener en cuenta la eliminación de bias por raza, sexo y aspectos socioculturales. Debido a esto, los algoritmos que se desarrollen con esta base de datos no pueden ser aplicados en producción.

Modelo

En cuanto al modelo, se a cumplido los siguientes objetivos:

- Utilizar la capa 3DCNN.

- Ser lo suficientemente ligero para ser entrenado con los recursos disponibles.
- El uso de python como lenguaje de programación y el framework PyTorch.

No se ha conseguido la precisión del 70 % debido a varios motivos. El primero es la falta de datos suficientes para entrenar un modelo que resuelva una tarea tan compleja como la detección de acciones. Segundo, no se disponían de suficientes recursos para entrenar modelos más complejos que son capaces de compensar el problema de la escasez de datos. Finalmente, la baja calidad de los vídeos limitaba la capacidad del algoritmo de apreciar el detalle de la acción, especialmente al aplicar transformaciones.

Recomendaciones para futuros estudios

Para resolver los problemas encontrados, se recomienda explorar los siguientes puntos.

- **Ampliar la base de datos:** Es necesario aumentar la calidad y cantidad de vídeos en la base de datos. Esto permitirá evitar bias, aumentar la calidad de imagen de los recortes de cada persona y se deberá tener en cuenta el aspecto ético. Para cumplir todo esto, se propone generar los vídeos mediante el uso de actores para asegurarse de cumplir con las obligaciones éticas y respetar la legislación vigente.
- **Transfer learning:** En caso de no poder aumentar la base de datos, se propone entrenar un modelo en otras bases de datos de acciones humanas con categorías más pobladas y con mejor calidad para posteriormente transferir lo aprendido a la base de datos.
- **Detección de personas:** Continuar explorando algoritmos de detección de personas como YOLO + Deep SORT para aislarlas. Si se aumentase su fiabilidad se podría entrenar el modelo solo con la representación de la persona. Esto minimizaría el bias de entorno o la aparición de personas de distintas categorías en la misma imagen.
- **Poder de computación:** El acceso a más poder de computación es necesario de cara al entrenamiento de modelos con vídeos. El aumento de las capacidades de computación permitiría reducir significativamente los tiempos de entrenamiento para poder probar nuevas arquitecturas y realizar una optimización de hiperparámetros más rigurosa y automatizada.

Referencias

[1] **Python**, *Python Software Foundation*. Last access: 06/06/2021
https://www.python.org/

[2] **Tensorflow**, *Open Source Machine Learning Framework for Everyone*. Last access: 06/06/2021
https://www.tensorflow.org/

[3] **PyTorch**, *An open source machine learning framework that accelerates the path from research prototyping to production deployment.*. Last access: 06/06/2021
https://pytorch.org/

[4] **Center for Research in Computer Vision (CRCV)**, *UCF101 - Action Recognition Data Set*. Last access: 05/06/2021
https://www.crcv.ucf.edu/data/UCF101.php

[5] **DeepMind**, *Kinetics*. Last access: 05/06/2021
https://deepmind.com/research/open-source/kinetics

[6] **Choi, J., Gao, C., Messou, J. C., & Huang, J. B.**, *Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition*. arXiv preprint 2019.
https://arxiv.org/abs/1912.05534

[7] **OpenCV**, *Open Source Computer Vision*. Last access: 15/06/2021
https://docs.opencv.org/4.5.2/

[8] **Algorithmia**, *Introduction to loss functions*. Last access: 05/07/2021
https://algorithmia.com/blog/introduction-to-loss-functions

[9] **Kingma, D.P. and Ba, J.**, *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980 2014.
https://arxiv.org/abs/1412.6980

[10] **Sanket Doshi**, *Various Optimization Algorithms For Training Neural Network*. Last access: 10/07/2021
https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[11] **Zong, M., Wang, R., Chen, Z., Wang, M., Wang, X., & Potgieter, J**, *Multi-cue based 3D residual network for action recognition*. Neural Computing and Applications, 33(10), 5167-5181, 2020.
https://doi.org/10.1007/s00521-020-05313-8

[12] **Ioffe, S., & Szegedy, C.**, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. International conference on machine learning (pp. 448-456), 2015.
http://proceedings.mlr.press/v37/ioffe15.html

ACTION RECOGNITION IN VIDEOS WITH A 3D CONVOLUTIONAL NEURAL NETWORK (3DCNN)

Author: Castillo Rodríguez, Alberto.

Directors: Álvaro López López, Lucía Güitta López.
Collaborationg Entity: Instituto de Investigación Tecnológica (IIT).

SUMMARY OF THE PROJECT

Introduction

The project arises as a result of the great progress that artificial intelligence has been making in recent decades and the great impact it can have on the business world and on society. These advances in the early days of machine learning were in the field of structured numerical information, but with the development and cheapening of computing power, the field of artificial vision has been growing.

The possibilities of being able to identify actions in video are enormous, as it would revolutionise sectors such as security, where the detection of robberies or violent actions could be improved and where a single person is required to monitor a large number of cameras at the same time. This work aims to respond to this need for innovation in the field of artificial vision and where different companies can find competitive advantages if they achieve this goal.

This progress not only has to be reflected in the accuracy results, but also has to be done taking into account the resource limitations, both in terms of the possibility to get videos and the processing capacity to train and run the network. If the model is resource intensive the images have to be sent to an external server, known as Cloud Computing, as opposed to processing it locally, known as Edge Computing.

This is why the 3 Dimensional Convolutional Neural Network (3D CNN) structure has been chosen as a basis for testing different architectures. One advantage of using convolutional

networks is that they allow to increase the processing speed with the use of the GPU, as they are optimised to work with matrices, which is precisely the basis of convolutional operations, and in this way to lighten the load when training and applying them. Moreover, compared to recurrent networks, they are more efficient due to the lower number of weights required.

Objectives

The objective is to develop an algorithm capable of detecting actions in security camera videos. To do so, a database will be built and a algorithm will be developed.

Obtaining the database

The database will be created on which the different algorithms and preprocessing methods will be tested. It must meet the following criteria:

- It must be balanced
- It should preferably be publicly accessible
- Must be well classified
- Must be trimmed
- They should be of the highest possible quality
- There should be a considerable amount of data

Algorithm development

Different algorithms using the 3D CNN structure partially or in its entirety will be developed according to the following criteria:

- Must be lightweight
- Programming language: it must use the Python programming language [1] and one of the two most important Deep Learning frameworks such as Tensorflow [2] or PyTorch [3] to facilitate its integration in a final application.
- That it has an accuracy of at least 70 % in validation.

Solution

AI ethics

When designing a system that implements artificial intelligence algorithms, it is very important to mention ethics. These types of systems eliminate human intervention in decision making and therefore run the risk of creating systems that intentionally or accidentally make decisions based on race, gender or other categories.

Data

Database generation

Public databases such as UCF 101 [4] or Kinetics [5] were explored, looking for categories of security camera videos. In the absence of a reasonable amount to get a robust database, a Python script was made to download YouTube videos with the aforementioned characteristics. The total number of videos extracted was 167 with very diverse characteristics of resolution, frames per second (FPS) and scenarios or environments.

Analysing the bias of the database, taking into account the context of the use of security camera videos, we observed a large imbalance in the representation of different skin colour tones and cultural styles. Therefore, although the database can still be used to test different types of algorithms, the algorithms trained from the database cannot be used in production environments.

Database post-processing

Once we have obtained the videos, it is necessary to trim them and classify them correctly as normal videos and videos in which the action is committed. To do this, a list was generated in a .txt file with the respective names of each file and the seconds we want to tag next to the name of the file. After this classification, a Python script automatically trimmed the videos and classified them according to the tag. The end result was the creation of 204 videos with normal situations and 208 videos with the action. To improve the quality of the database, the following considerations were taken into account.

Video clipping

The trimmed videos must have exclusively the action to be detected, resulting in clips of very variable length in the range of 2 to 10 seconds. To avoid a bias for the duration of these

trimmed videos, for each one that contains an action as far as possible we have tried to obtain another one without any action with the same duration.

Bias of background and people

One of the ways in which the algorithm could falsely classify the videos could be by detecting characteristics of the people who commit the action versus those who do not. That is why we tried whenever possible to obtain snippets of each video showing the same person committing the action and in a normal situation.

Figure 1. *Example of how the action can be determined without seeing the person committing it. From left to right: jumping, golf, canoeing and ice skating. [6]*

This is also true of the recording environment, as reflected in the paper "Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition" [6]. This paper demonstrates that both algorithms and humans make use of context to determine, without seeing the person, the action being committed (figure 1). By removing this bias, it is possible to improve the generalisation of the algorithm but it also makes the task more difficult by reducing the information available. The bias has been eliminated by generating videos of normal situations and with the action in the same environment avoiding the memorisation of features of the environment for classification.

Movement filter

Another way to solve this problem would be to capture only the motion that is detected in each video. The viewing angle and position of the images are constant and therefore it is only the people that modify the pixels over time as they move. By calculating the absolute difference between consecutive black and white frames, we get black pixels where there has been no movement and white pixels where there has. To be more resilient to noise, it is preferable to calculate the difference with a weighted moving average of the previous frames although this produces the appearance of the ghosting side effect as can be seen in the figure 2. This operation can be performed with the OpenCV cv2 library [7].

Division of the database

The last step to prepare the database for the learning process is its division into three groups where we have tried to balance the categories as much as possible:

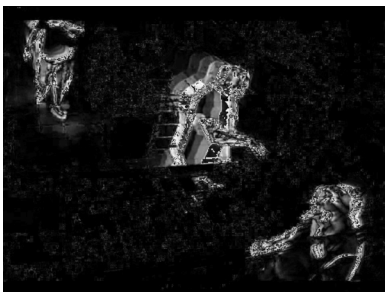

Figure 2. *Frame of one of the videos when applying the motion filter*

- **Train:** this subgroup will be used to train the model automatically. It is therefore the largest, with 70% of videos, 144 normal and 141 with action.

- **Validation:** this subgroup will be used to manually or automatically tune the hyperparameters of the model. It will contain 15% o of the videos, 31 normal and 35 with action.

- **Test:** this last subgroup is used to validate the results of the model. The hyperparameters should not be modified to improve the result in this subgroup. It will have 15% of the videos, 31 normal and 30 with the action.

Data Augmentation

The aim of this techniques is to increase the amount of data available by applying modifications to the existing data to allow the algorithm to better generalise to data that is slightly different from the existing ones. Modifications can be:

- **Spatial:** videos are mirrored horizontally to keep the vertical axis in the same orientation.

- **Colour:** modifying the colour spectrum, contrast and brightness among other parameters allows the algorithm to be more resilient to these variations between videos.

- **Noise:** introducing noise in the images allows the algorithm to work with different qualities of videos and be more robust to these variations.

Pipeline

The job of the pipeline is to manage the training process as optimally as possible. This ranges from performing the relevant transformations on the data, loading it into the RAM, CPU or GPU, performing the training, validation and testing operations and finally saving the results and the model.

Hardware limitations

The first aspect to consider when designing the pipeline is the available resources during the whole process and the requirements of the algorithms during training. The goal is to maximise the training speed by optimising the use of the system hardware. In this project, there are two limiting resources when it comes to minimising training time: RAM and GPU.

Machine Learning Frameworks

To facilitate and optimise the definition of the algorithms, the training process and the reproducibility of the whole process, several frameworks have been developed. PyTorch [3] allows us to easily access the training loops in order to modify them according to our needs, both to test new architectures and to visualise the data.

Data loading

Once we have the entire database generated, the interface must be configured to read the data, process it and load it into the training device. As there is not enough RAM or space in the GPU memory to hold the entire database, the strategy to follow will be to load the data little by little as it is needed during the training process. Two classes will be used for this purpose: Dataset and DataLoader.

Optimizador

The process of training an artificial intelligence algorithm consists of defining a loss function that represents how well it performs its task and improving this result over training epochs. The problem with minimising this function is that in general they are not convex and therefore do not have a single minimum but have local minima. The strategy with which one tries to find the global minimum is very important to avoid getting stuck in local minima.

Optimisers are tools to solve all these problems. Most optimisers calculate the learning rate automatically and apply the gradient to the model making it learn. A good optimiser is one that trains fast and at the same time prevents the model from getting stuck at a local minimum. Adam's optimiser Adam [9] is the one that performs best in most [10] scenarios and that is why it is the one that has finally been applied.

Loss function

When evaluating how well an algorithm has performed a task, a metric must be defined to quantify it mathematically. One must choose one that represents the loss function. This

function will be the one that is minimised during the training process and therefore should not only represent how well it has performed the classification, but also its confidence in both hit and miss. The Cross-Entropy Loss function is the one selected because:

- Using the last Softmax layer allows us to easily check the probabilities that the model gives to each category. This way you can intuitively visualise what the model does.
- It can be applied to both binary and multi-class classification problems. This allows us not to have to modify the structure of the model in case of adding more categories.
- It is a function extensively tested in the literature with proven results.

Hyperparameter tuning

Hyperparameter optimisation is one of the most important phases in the training process of a model. While parameters are normally trained automatically, hyperparameters are those that describe or shape the training process and are therefore fixed during the training process. The impact of each is different and there are different optimisation processes.

Algorithms

Once we have designed a pipeline that is able to provide the data and manage the training process, we must face the design of an algorithm able to solve the problem of action detection in security camera videos.

Intrinsic problems of the task

When solving problems with artificial intelligence models, great care must be taken when analysing the difficulties that the data may present. By considering all sources of bias, it is possible to target the model design to avoid these problems at their root. The following challenges have been identified:

- Multiple categories in the same video.
- Overlapping subjects.
- Non-identifying environment.

Model

Once we have identified the difficulties, several models or combinations of them have been proposed to overcome them. In addition to these considerations, the limitations of computing power and the generated database have been taken into account.

Intrinsic problem elimination strategies

The main intrinsic problem to be solved that could present the greatest difficulties is that of the appearance of multiple persons with different categories. To solve it, two different methods are proposed: use of the motion filter and the set of YOLO v4 models with Deep Sort.

Main classification algorithm

The aim of this work is to develop an algorithm based on the 3DCNN layer, which consists of convolutional operations of two spatial dimensions and one temporal dimension. There are many models using this structure with varying complexity but due to the large number of parameters involved and the limited computational capacity, it has been decided to explore the simplest architecture applying it as a proof of concept.

This architecture is composed of 5 convolutional layers in series followed by 2 fully connected layers with the softmax operation at the end to calculate the assigned probabilities for each class (figure 3). This architecture is described in the article "Multi-cue based 3D residual network for action recognition" [11] as the base model of this layer.

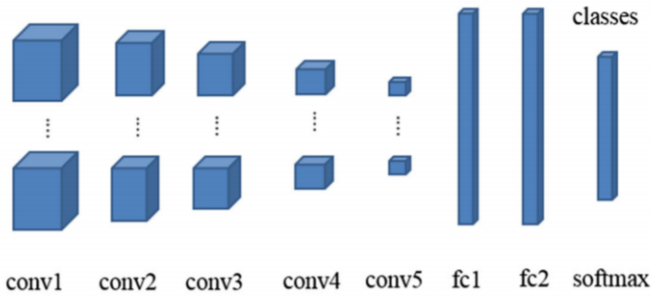

Figure 3. *Layers of the 3DCNNN model [11].*

This architecture has been modified to solve several of the classic problems that Deep Learning models have:

- Overfitting
- Underfitting
- Computational load
- Internal covariate shift

The layers used to combat these problems are:

- **Max pooling:** Max pooling is a discretisation process that aims to reduce the size of your input by transforming the window of the operation to the maximum value contained in it. This reduces the number of parameters required while preserving the characteristics of the region and reduces underfitting by providing an abstract form of the sub-regions to which the filter has been applied.

- **Dropout:** This regularisation technique aims to prevent overfitting. It is achieved by ignoring a percentage of units during the training process chosen randomly at each epoch. Ignoring them allows not modifying their value during the process and eliminating co-dependencies between parameters.

- **Batch normalization:** This layer was designed in the article "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" [12] to address the problem of internal covariate shift. It allows to speed up training by reducing the number of epochs and smoothing the loss function.

Results

The model development consisted of three parts: database construction, pipeline development and model design and training.

Database

A total of 412 videos were obtained, 204 of them with the normal category and 208 of the action to be detected. These were divided into three partitions: train, validation and test with 70%, 15% and 15% respectively. They were ensured to be balanced in categories in each subdivision. Many of them were of poor quality, so it was decided to create a second database with only those that met minimum resolution requirements and met a clearer definition of the action. This second database contains 210 videos, 105 from each category, with similar percentages and balanced categories.

In terms of the ethical quality of the database, there is no great ethnic and socio-cultural diversity due to the greater restrictions on this content in Western countries and their greater capacity to apply them. This is why it is not recommended for use in developing production models.

Pipeline

After applying all the techniques mentioned above, an efficient pipeline has been achieved with the available resources. The correct loading and transmission of data to the algorithm has been checked, as well as the application of the relevant transformations before training. It has not been possible to apply the YOLO algorithm together with Deep SORT due to unreliability during the clipping of individuals. This is due to the low quality of the videos and the excessive overlapping of multiple persons. The Adam optimiser was finally used due to its easy implementation and good results in most deep learning applications. As for the loss function, the function applied was Cross-Entropy Loss due to its versatility in terms of the number of classes with which it can be applied, the good results in the literature and allowing us to directly observe the probabilities that the model assigns. Hyperparameter optimisation was performed manually, making decisions based on information recorded in TensorBoard from past training.

Model

After many iterations on the model structure and hyperparameters, overfitting has been achieved on the training data, implying that the model has sufficient complexity to learn the problem. This can be seen in figure 4, where the loss function decreases over epochs in the training data, but increases in the validation data.

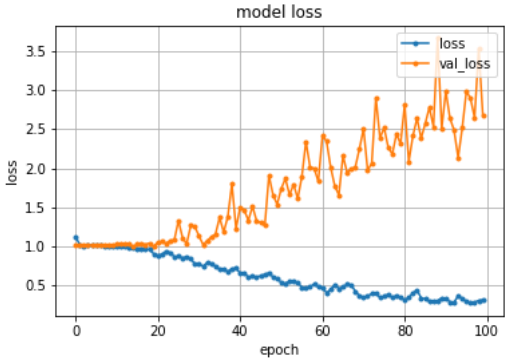

Figure 4. *Evolution of the loss function*

Despite this, the model has not been found to perform better in the classification process than making random predictions, resulting in an accuracy of around 50% in validation as is the case when we have a balanced two-category database. These observations are reflected in figure 5, where we obtain an accuracy in the training data close to 100 percent due to overfitting but in the validation data it oscillates around 50 percent.

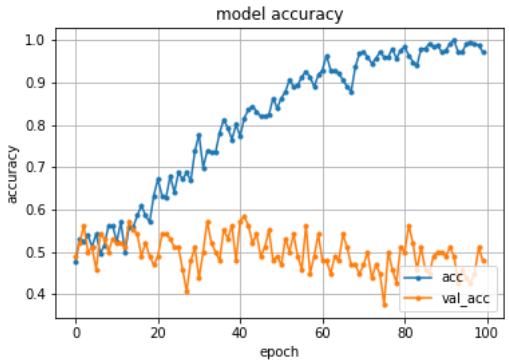

Figure 5. *Accuracy evolution*

Plotting the confusion matrix (figure 6), it is observed that the algorithm prefers to classify as the action the normal cases leading to false positives. In this case, it is preferable to false negatives which are also found in lower proportion.

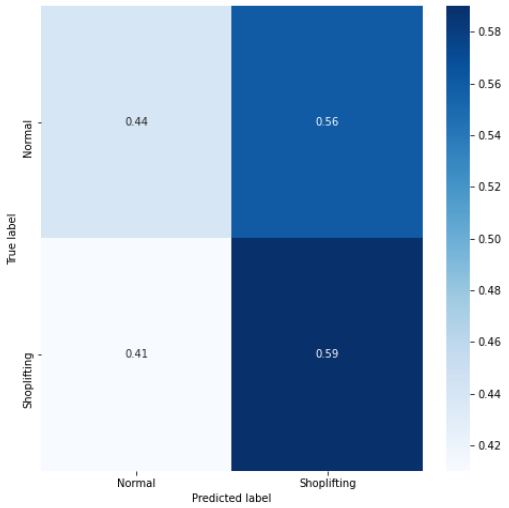

Figure 6. *Confusion matrix.*

Conclusions

From the results obtained, different causes can be extracted as to why a model capable of making predictions.

Database

Regarding the database, the following objectives have been met:

- Build a balanced database.
- Use publicly accessible data.
- Correct classification of videos.
- Cropped to only represent the section where the action appears.

Individual people have not been finally cropped with YOLO together with Deep SORT as it is not very reliable when performing the operation well in situations with many people in close proximity. Regarding the quality of the videos and the number of videos found, the severe restrictions prevent finding quality videos in quantity, limiting the variety available and with major shortcomings in terms of ethics. It is very important, especially when dealing with security camera videos, to take into account the elimination of bias by race, gender and socio-cultural aspects. Because of this, algorithms developed with this database cannot be applied in production.

Model

In terms of the model, the following objectives have been met:

- Use the 3DCNN layer.
- To be light enough to be trained with the available resources.
- The use of Python as programming language and the PyTorch framework.

The accuracy of 70% has not been achieved due to several reasons. The first is the lack of sufficient data to train a model to solve such a complex task as action detection. Second, there were not enough resources available to train more complex models that are able to compensate for the problem of data sparsity. Finally, the low quality of the videos limited the algorithm's ability to appreciate the detail of the action, especially when applying transformations.

Recommendations for future studies

To solve the problems encountered, it is recommended that the following points be explored.

- **Increase the size of the database:** It is necessary to increase the quality and quantity of videos in the database. This will allow to avoid bias, to increase the image quality of the clippings of each person and to take into account the ethical aspect. In order to fulfil

all this, it is proposed to generate the videos by using actors to make sure that the ethical obligations are met and that the current legislation is respected.

- **Transfer learning:** In case of not being able to increase the database, it is proposed to train a model on other databases of human actions with more populated categories and with better quality in order to transfer what has been learned to the database.

- **Human detection:** Continue to explore human detection algorithms such as YOLO + Deep SORT to isolate people. If their reliability could be increased, the model could be trained with only one person per video. This would minimise environment bias or the appearance of people from different categories in the same image.

- **Computational power:** Access to more computing power is necessary in order to train models with videos. Increased computational capabilities would allow significantly reduced training times for testing new architectures and more rigorous and automated hyperparameter optimisation.

Bibliography

[1] **Python**, *Python Software Foundation*. Last access: 06/06/2021
https://www.python.org/

[2] **Tensorflow**, *Open Source Machine Learning Framework for Everyone*. Last access: 06/06/2021
https://www.tensorflow.org/

[3] **PyTorch**, *An open source machine learning framework that accelerates the path from research prototyping to production deployment.*. Last access: 06/06/2021
https://pytorch.org/

[4] **Center for Research in Computer Vision (CRCV)**, *UCF101 - Action Recognition Data Set*. Last access: 05/06/2021
https://www.crcv.ucf.edu/data/UCF101.php

[5] **DeepMind**, *Kinetics*. Last access: 05/06/2021
https://deepmind.com/research/open-source/kinetics

[6] **Choi, J., Gao, C., Messou, J. C., & Huang, J. B.**, *Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition*. arXiv preprint 2019.
https://arxiv.org/abs/1912.05534

[7] **OpenCV**, *Open Source Computer Vision*. Last access: 15/06/2021
https://docs.opencv.org/4.5.2/

[8] **Algorithmia**, *Introduction to loss functions*. Last access: 05/07/2021
https://algorithmia.com/blog/introduction-to-loss-functions

[9] **Kingma, D.P. and Ba, J.**, *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980 2014.
https://arxiv.org/abs/1412.6980

[10] **Sanket Doshi**, *Various Optimization Algorithms For Training Neural Network*. Last access: 10/07/2021
https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[11] **Zong, M., Wang, R., Chen, Z., Wang, M., Wang, X., & Potgieter, J**, *Multi-cue based 3D residual network for action recognition*. Neural Computing and Applications, 33(10), 5167-5181, 2020.
https://doi.org/10.1007/s00521-020-05313-8

[12] **Ioffe, S., & Szegedy, C.**, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. International conference on machine learning (pp. 448-456), 2015.
http://proceedings.mlr.press/v37/ioffe15.html

DOCUMENTO I

MEMORIA

Índice

Índice de figuras

Acrónimos

3DCNN	3 Dimensional Convolutional Neural Network
AMP	Automatic Mixed Precision
ANN	Artificial Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FCNN	Fully Connected Neural Network
FPS	Fotogramas Por Segundo
GPU	Graphics Processing Unit
IA	Inteligencia Artificial
NLP	Natural Language Processing
ODS	Objetivos de Desarrollo Sostenible
RAM	Random Access Memory
SGD	Stochastic Gradient Descent
SSD	Solid State Drive

PARTE I

MEMORIA

Capítulo 1

Introducción y planteamiento del proyecto

1.1. Introducción

El proyecto surge a raíz del gran avance que está teniendo la inteligencia artificial estas últimas décadas y del gran impacto que este puede tener en el mundo empresarial y en la sociedad. Estos avances en los inicios del machine learning se produjeron en el campo de la información numérica estructurada, pero con el desarrollo y abaratamiento del poder de computación el campo de la visión artificial ha ido en aumento.

En 1958 Frank Rosenblatt inventó el perceptrón [1], un modelo probabilístico que imitaba la forma en el que el cerebro almacena y organiza la información mediante neuronas, el cual se ha convertido en un elemento esencial de las actuales redes neuronales. A finales de los 60 se empezó a trabajar con la visión artificial de forma teórica debido a las limitaciones tecnológicas de la época, y se planteó como una forma de imitar la capacidad visual humana que permitiese a robots y máquinas entender su entorno. Cabe destacar la importancia también que tiene este campo para entender el funcionamiento de la capacidad de visión de animales y humanos. En los 70 se trabajó en desarrollar los conceptos de reconocimientos de bordes y líneas en imágenes, entendiendo que los objetos más complejos se componen de combinaciones de estos dando lugar a la creación de Neocognitron en 1980 [2]. Se trataba de una red neuronal artificial multicapa o ANN, por sus siglas en inglés, capaz de reconocer caracteres japoneses u otras tareas de reconocimiento de patrones sirviendo de inspiración para la posterior creación de redes neuronales convolucionales (CNN). En los 90 se trabajó con el reconocimiento de profundidad con el objetivo de reconstruir a partir de imágenes formas en 3D. Todos estos avances estaban limitados por el poder de computación de la época y es ahora en las últimas décadas cuando se ha aplicado la teoría desarrollada anteriormente.

En un principio, ésta permitió el análisis de imágenes, reconociendo en ellas caracteres escritos a mano e impresos. Posteriormente se empezó a analizar imágenes de mayor resolución consiguiendo clasificar el objeto que contenían y posteriormente identificar los distintos objetos en ella, su posición y segmentar o marcar los píxeles que corresponden a un objeto concreto. Los avances en "Deep Learning.º aprendizaje profundo han potenciado aún más la capacidad de procesar problemas cada vez más complejos.

Es aquí donde se encuentra el primer intento de clasificar imágenes estáticas según la acción que está realizando la persona que contiene. Las críticas con este método es que no necesariamente está entendiendo que la postura de esa persona corresponde con una acción en concreto, sino que puede estar identificando la acción por el contexto de la imagen. Por ejemplo, una persona jugando al fútbol se la puede identificar por la postura de estar dando una patada a un balón o por estar en un campo de fútbol y por el uniforme que lleve.

Esto puede no ser un problema ya que, si el objetivo final es clasificar imágenes estáticas sobre acciones no complejas, esta información es valiosa y debe ser usada. El problema surge cuando acciones más complejas ajenas al entorno no proporcionan esta información y por lo tanto nunca el algoritmo va a ser capaz de comprender lo que identifica realmente una acción. Una persona es capaz de identificar en un juego de mímica cuando una persona está jugando al fútbol, independientemente de si está en un campo de fútbol o tiene un balón, tan solo con observar la imitación de la acción. Es por eso por lo que para identificar acciones más complejas es necesario proveer de más información mediante el análisis de imágenes con relación temporal o, en otras palabras, analizar vídeos para así evitar recurrir al contexto e identificar realmente los gestos que dan lugar a una acción.

Las posibilidades de conseguir identificar acciones en vídeo son enormes ya que revolucionaría sectores como el de la seguridad donde se podría mejorar la detección de robos o de acciones violentas y donde se requiere que una sola persona vigile un gran número de cámaras al mismo tiempo.

1.2. Motivación

El proyecto por tanto surge como respuesta a esta necesidad de innovar en el campo de la visión artificial y de donde diferentes empresas pueden encontrar ventajas competitivas si logran este objetivo.

Este avance no solo tiene que verse reflejado en los resultados de precisión, sino que también tiene que hacerse teniendo en cuenta las limitaciones de recursos, ya sea tanto en la posibilidad de conseguir vídeos como de capacidad de procesamiento para entrenar y para ejecutar la red. Si el modelo necesita de muchos recursos las imágenes se tienen que enviar a un servidor externo, conocido como Cloud Computing, frente a procesarlo localmente, conocido como Edge Computing.

Es por todo esto que se ha elegido la estructura 3 Dimensional Convolutional Neural Network (3D CNN) como base para probar diferentes arquitecturas que se utilicen de alguna forma en ésta. Una ventaja de utilizar redes convolucionales es que permiten aumentar la velocidad de procesamiento con él uso de la GPU, al estas estar optimizadas para trabajar con matrices que es justamente la base de una convolucional, y de esta forma poder tanto aligerar la carga al entrenar y al aplicarlas. Además, frente a las redes recurrentes son más eficientes por el menor número de pesos necesarios.

1.3. Objetivos del proyecto

El objetivo es desarrollar un algoritmo capaz de detectar acciones en vídeos de cámaras de seguridad. Para ello se construirá la base de datos y se desarrollará el algoritmo.

1.3.1. Obtención de la base de datos

Se creará la base de datos con la que se probaran los distintos algoritmos y métodos de preprocesamiento. Ésta debe cumplir los siguientes criterios:

- **Debe estar balanceada:** debe tener el mismo número de datos correspondientes a cada categoría.
- **Debe ser preferiblemente de acceso público:** si se quiere permitir que sean reproducibles los resultados, todo el mundo debe tener acceso a los vídeos sobre los que se basa el trabajo.
- **Deben estar bien clasificados:** el algoritmo es tan bueno como la calidad de la clasificación de la base de datos.
- **Deben estar recortados:** se debe dejar exclusivamente el contenido que contenga la acción.
- **Deben de tener la máxima calidad posible:** posteriormente puede reducirse la calidad antes de pasarlos por la red, pero esta decisión se puede tomar solo si está acotado superiormente por una calidad decentemente alta.

- **Debe haber una cantidad considerable de datos:** si se quiere poder dividir la base de datos en Train, Validation y Test y que estos sean estadísticamente representativos se debe tener un número alto de vídeos siempre que sea posible. De no ser así los resultados obtenidos pueden no ser extrapolables a situaciones reales.

1.3.2. Desarrollo del algoritmo

Se desarrollará diferentes algoritmos que utilicen la estructura de 3D CNN parcialmente o en su totalidad atendiendo a los siguientes criterios:

- **Que sea ligero:** tiene que ser posible entrenarlo en un espacio de tiempo razonable con los recursos disponibles y ser fácilmente ejecutable.

- **Lenguaje de programación:** que utilice el lenguaje de programación Python [3] y alguno de los dos Deep Learning frameworks más importantes como son Tensorflow [4] o PyTorch [5] para facilitar su integración en una aplicación final.

- **Precisión:** que tenga una precisión de al menos 70 % en validación.

1.4. Alineación con los objetivos de desarrollo sostenible

El 25 de septiembre de 2015 se definieron un conjunto de objetivos globales para erradicar la pobreza, proteger el planeta y asegurar la prosperidad de todos [6]. Estos objetivos deben cumplirse en los próximos 15 años y abarcan a los gobiernos, las empresas y la sociedad civil. Es por eso por lo que es de vital importancia que se tengan en cuenta a la hora de desarrollar nuevos proyectos y que estos se integren de la mejor forma posible.

La tecnología de la visión artificial puede aplicarse a muchos campos, pero los ODS [6] que mejor se alinean son:

- **Objetivo 8: Promover el crecimiento económico inclusivo y sostenible, el empleo y el trabajo decente para todos.** Tareas que sean repetitivas y con poco valor añadido como puede ser la vigilancia pueden ser automatizadas mediante esta tecnología.

Figura 1. *ODS 8 - Promover el crecimiento económico inclusivo y sostenible, el empleo y el trabajo decente para todos.*

- **Objetivo 16: Promover sociedades justas, pacíficas e inclusivas.** Gracias a la visión artificial se puede crear sociedades más seguras gracias a poder detectar acciones como los robos, atracos o atentados terroristas.

Figura 2. *ODS 16 - Promover sociedades justas, pacíficas e inclusivas.*

1.5. Metodología de trabajo

La metodología de trabajo se basará en los objetivos del proyecto expuesto. Se empezará por crear la base de datos, recopilando la información de bases de datos públicas como UCF 101 [7] o Kinetics [8] y de plataformas de intercambio de vídeos como YouTube. Una vez creada la base de datos se tendrán que diseñar los algoritmos de preprocesado de vídeos, que serán dependientes de la estructura del modelo por lo que a pesar de que está planeado acabar ese proceso en octubre puede que se tenga que modificar más adelante. Posteriormente se creará el modelo lo cual exigirá una extensa lectura de las publicaciones disponibles sobre el tema, probando los resultados en la base de datos. Finalmente se irá probando a la par de la creación del modelo distintos hiperparámetros para optimizar el modelo sobre validación y comprobar su precisión real en test.

El cronograma inicial por tanto sería el siguiente:

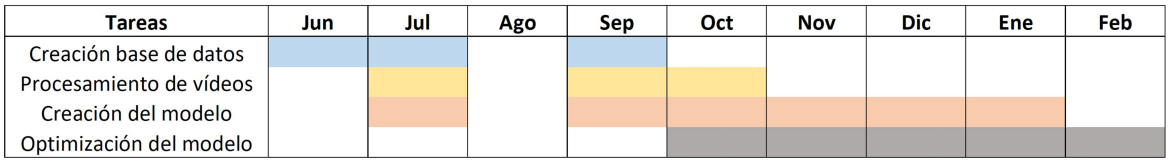

Figura 3. *Cronograma*

1.6. Recursos a emplear

Los recursos a emplear son los siguientes:

- **Google Colab [9] y tarjeta gráfica local para ejecutar los algoritmos desarrollados:** las ventajas de Google Colab es el acceso gratuito a más poder de computación, aunque tiene restricciones como el tener que estar activo o el límite de 12 horas de ejecución.
- **Bases de datos u otras fuentes de vídeos como YouTube:** para construir la base de datos sobre la que entrenar.
- **Python** [3] y un gestor de entornos como **Anaconda** [10].
- **Github** [11] para llevar un seguimiento de los cambios realizados.
- Entorno de desarrollo integrado como por ejemplo **PyCharm** [12] o **Visual Studio Code** [13] para editar el código y ejecutarlo.
- **Tensorflow [4] o PyTorch [5]:** como frameworks desde el que entrenar los algoritmos y los cuales nos permiten usar la GPU gracias a su integración con CUDA.

Capítulo 2

Estado de la técnica

ESTE capítulo pretende ofrecer un breve contexto de las matemáticas y los algoritmos que sirvieron como precursores de las técnicas de análisis de vídeos así como el estado del arte en ese campo.

2.1. ¿Qué es la Inteligencia Artificial?

Este término es usado indiscriminadamente en la actualidad por los departamentos de marketing y en películas distópicas por su marcada connotación futurista, diluyendo su significado. Todo el mundo tiene claro que algo artificial es el producto del trabajo de la humanidad y ajeno a la naturaleza pero, ¿qué es la inteligencia? Esta pregunta transciende el campo de la ciencia y entra en el mundo de la filosofía. Incluso al analizar el mundo animal, la humanidad ha pasado de pensar que los animales no poseían inteligencia alguna a reconocer la profundidad a la que llega en algunas especies. ¿Es inteligencia el saber responder a estímulos externos o debe haber comprensión del entorno? Si consideramos solo el primero, se puede argumentar que una planta posee inteligencia cuando reacciona para crecer hacia las zonas con más luz. De ser así, un simple robot con un programa cuyo objetivo es seguir una línea pintada en el suelo puede enmarcarse en del concepto de la inteligencia artificial.

Estas cuestiones filosóficas escapan el alcance de este trabajo, pero nos ayudan a entender la dificultad de describir que es la inteligencia artificial. Ante la necesidad de encontrar una definición que abarque todas estas cuestiones, la proporcionada por Elaine Rich es la más completa según Wolfgang Ertel en su libro Ïntroduction to Artificial Intelligence"[14]:

> **Artificial Intelligence** is the study of how to make computers do things at which, at the moment, people are better. [La inteligencia artificial es el estudio de cómo hacer que los ordenadores hagan cosas que, por el momento, a los humanos se nos da mejor hacer.] (Elaine Rich [15])

Esta definición pone la inteligencia artificial siempre en los límites de las capacidades actuales de la tecnología, asegurando por tanto su relevancia. El propósito de la IA es habilitar a los ordenadores para que sean capaces de realizar tareas que por el momento solo los humanos somos capaces de realizar por su complejidad. Es importante hacer hincapié que el problema no es una falta de poder de computación o capacidad de memoria, ya que en ambos aspectos los ordenadores nos superan, sino esa complejidad que acompañan a problemas y tareas del mundo real que para los humanos nos resulta casi innata. Ya sea reconocer caracteres, distinguir que "pixelesçorresponden a que objeto o simplemente reconocer la cara de un amigo.

La diversidad de problemas que puede resolver la IA y las formas de resolverlos hace que nos podamos encontrar subcampos dentro de su estudio, como notablemente son el Machine Learning y el Deep Learning (Figura 4).

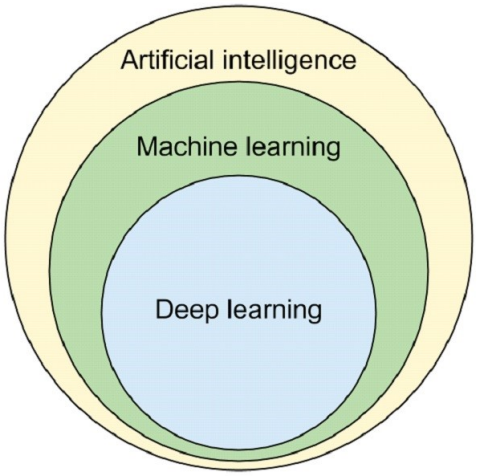

Figura 4. *Subcampos de la IA [16]*

Hay un debate abierto sobre si la IA contiene enteramente estos dos campos. Por una parte, unos argumentan que la IA debe implicar la existencia de un agente, definido como una entidad artificial que no solo observa el entorno sino que participa e interactúa con él (como puede ser el caso de un coche autónomo), mientras que los campos de Machine Learning y Deep Learning no requieren necesariamente de esta interacción. Además, se podría argumentar que no todas las técnicas de Machine Learning poseen ïnteligencia.ªunque los detractores apuntan a la dificultad de definir que es que un algoritmo tenga inteligencia. En el ámbito de este TFM, se

considerará la definición clásica mostrada en la Figura 4 la cual se adapta mejor a la aportada por Elaine Rich.

2.2. Machine Learning y Deep Learning

Una de las propiedades fundamentales de la inteligencia es el aprendizaje y, por lo tanto, si queremos crear una inteligencia artificial debemos abordar como hacer que una maquina aprenda. Es aquí donde entra el campo del Machine Learning. Una posible definición puede ser la siguiente:

> **Machine Learning** es el estudio de los mecanismos de aprendizaje y los algoritmos que los aplican [14].

El objetivo de una entidad que aprende es generalizar de su experiencia. En el caso de los algoritmos de Machine Learning, este aprendizaje viene de la interacción con un entorno o el procesamiento de una base de datos suficientemente grande y diversa para que sea representativa de la realidad. Tras el aprendizaje, este proceso nos permitiría en teoría generalizar el problema aprendido a la realidad, obteniendo idílicamente el mismo comportamiento que con la base de datos o el entorno controlado.

Normalmente los algoritmos se pueden clasificar en tres principales categorías dependiendo de cómo este interactúa con los datos y de cómo estos son generados [17]:

- **Aprendizaje supervisado:** el algoritmo recibe el input y el output esperado por parte de un "profesor"para que este aprenda las normas generales de cómo funciona el proceso. El inconveniente de este método es la necesidad de obtener datos ya clasificados que casi necesariamente tienen que ser fabricados de procesos automatizados o, si son más complejos, gracias al trabajo de personas que realicen esta operación de clasificación. Generalmente son usado para resolver los problemas de clasificación y regresión.
- **Aprendizaje no supervisado:** el algoritmo recibe datos no clasificados y es la tarea del algoritmo descubrir los patrones para generar un output. Debido a la falta de clasificación inicial, es más fácil y barato generar bases de datos con esta característica. Entre sus objetivos se encuentra encontrar patrones y explicar los datos.
- **Aprendizaje por refuerzo:** cuando la tarea que se quiere aprender supone la interacción con un entorno, la creación de una base de datos no es posible. Como no siempre es

práctico generar un entorno en el mundo real con el que el algoritmo aprenda, se crea un entorno virtual simulado con el que aprende en tiempo acelerado con el objetivo de maximizar la puntuación generada para reforzar el comportamiento deseado.

Dentro de todos los problemas que los algoritmos de Machine Learning pueden resolver, el que más nos interesa en el contexto de este TFM es el de clasificación. Algunos de los algoritmos más usados para resolver este problema en el ámbito de datos estructurados son [17]:

- **Logistic Regression o Regresión Logística (Figura 5):** usado para modelar la probabilidad de las categorías a clasificar.

- **Support Vector Machine (SVM) (Figura 6):** genera un hiperplano que separa y maximiza el margen entre dos clases.

- **Neural Networks o Redes Neuronales (Figura 7):** red de ecuaciones matemáticas que fueron diseñadas imitando la estructura de las neuronas.

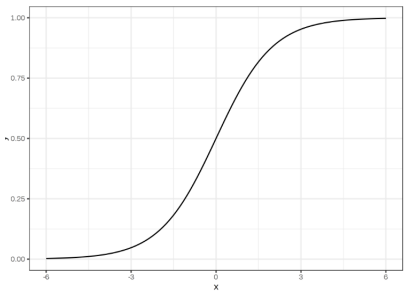

Figura 5. *Logistic Regression o Regresión Logística*

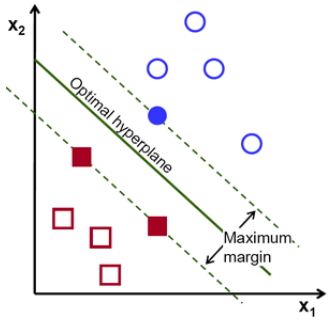

Figura 6. *Support Vector Machine (SVM)*

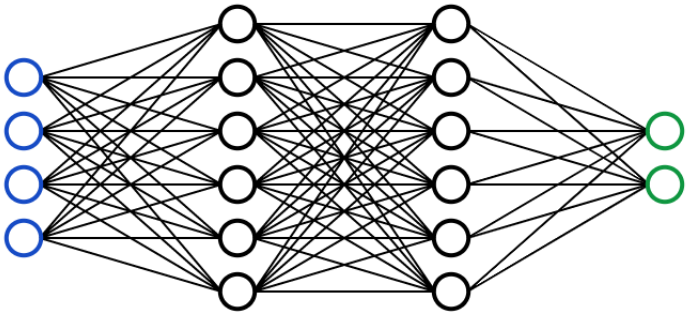

Figura 7. *Neural Networks o Redes Neuronales*

La mayoría de los algoritmos hacen suposiciones de como son los datos y es el trabajo del científico de datos que los aplica de asegurarse su correcta aplicación gracias a una comprensión de estos. No existe por tanto un método perfecto que pueda resolver cualquier problema, aunque las redes neuronales son las que mejor se aproximan a este algoritmo ideal. Al ser una de las bases de los algoritmos aplicados en la visión artificial, es necesario profundizar en ellas y detallar el teorema matemático en el que se sustentan: el teorema del aproximador universal.

Las redes neuronales, como su nombre indica, consiste en una red de neuronas o nodos basados en el perceptron (Figura 8) desarrollado por Frank Rosenblatt [1]. El calculo de la salida de una neurona se realiza tomando los valores de salida de las nodos/neuronas predecesoras, multiplicarlas por el peso sináptico correspondiente y se le aplica un sumatorio al resultado de estas operaciones. Finalmente, se suma el bias y se introduce el resultado en la función de activación. El resultado de esta función es la salida de la neurona.

El teorema del aproximador universal demuestra que siempre existe una red neuronal, de una sola capa, que use la función de activación sigmoide y con un número arbitrario de neuronas que sea capaz de aproximar con la precisión que se le requiera cualquier función continua. Más tarde se extendió el teorema a otras funciones de activación y profundidad de la red. Esto en efecto implica que con una red neuronal arbitrariamente grande se es capaz de resolver cualquier problema.

En la práctica, la existencia de una red neuronal con estas características no implica que sea posible encontrarla, que el tamaño de esta sea razonable para su aplicación o que sea capaz de generalizar el problema. Esto se explica ante la dificultad del proceso de encontrar la red neuronal con estas características deseadas. Este proceso se denomina entrenamiento. Al no saber cúal será la configuración de la red más óptima, se debe definir con anterioridad la forma, función de activación e inicializar los pesos y el bias con valores aleatorios. El objetivo por

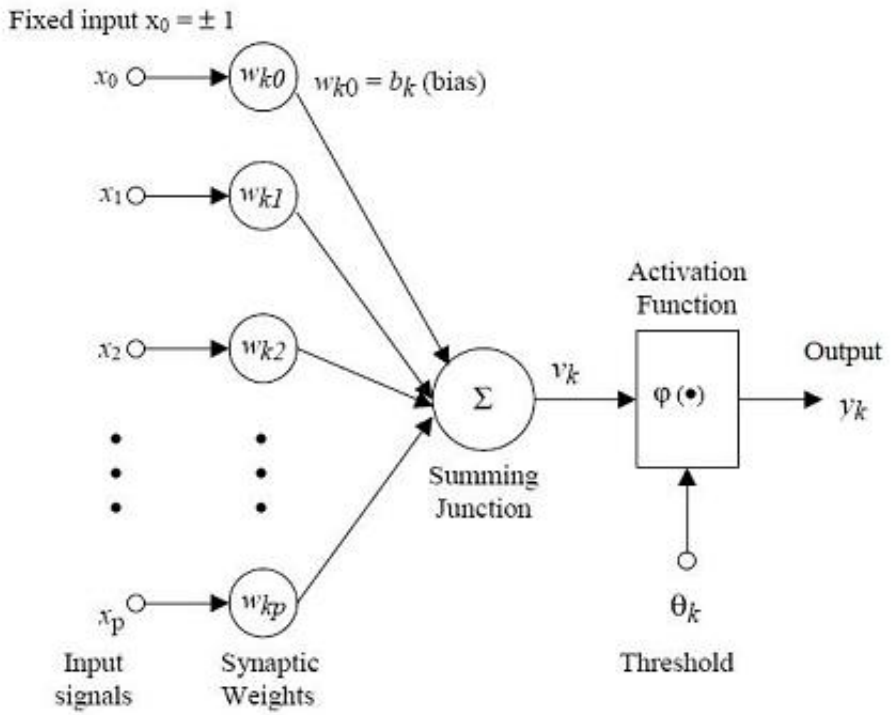

Figura 8. *Estructura del perceptron [18]*

tanto del entrenamiento es converger los parámetros hacia el óptimo, el cual se encuentra en el mínimo de una función denominada función de perdidas.

La definición de la función de perdidas es una parte crítica del entrenamiento ya que debe representar el error del algoritmo al realizar predicciones durante el proceso del entrenamiento. A menor error, mejor es el algoritmo, aunque se debe tener en cuenta que no todos los errores son iguales. Es posible que dos algoritmos tengan el mismo valor en la función de perdida cuando el primero se esté equivocando mucho en ciertos datos y que el resto tenga un comportamiento perfecto y el segundo tenga incertidumbre en la mayoría de los datos. Es aquí donde el entendimiento de los datos que se tratan entra en juego para valorar si la función de perdida debe ser cambiada o el problema viene de la existencia de ciertos outliers que deben ser eliminados de la base de datos.

Nos falta definir finalmente qué es el Deep Learning. El Deep Learning o aprendizaje profundo, como su nombre indica, es un caso particular de las redes neuronales artificiales con un gran número de capas. Esto nos permite afrontar problemas de más complejidad, siendo capaces de extraer características más complejas de los datos proporcionados. Ejemplos

de aplicación del Deep Learning puede ser el Natural Language Processing (NLP) o el procesamiento de lenguaje natural, procesos de detección de fraude o la visión artificial.

2.3. Visión Artificial

La visión artificial es la disciplina que estudia cómo hacer que los ordenadores puedan extraer información pertinente de imágenes, vídeos u otras entradas visuales de información para posteriormente tomar acciones y decisiones basadas en esta información [19].

El potencial que tiene esta tecnología es inmenso en un gran rango de industrias debido a la alta velocidad de computación y la escalabilidad de los servicios que puede proporcionar un ordenador frente a un humano en tareas cómo detección de defectos en líneas de producción, detección de enfermedades en radiografías, conducción autónoma o la vigilancia en cámaras de seguridad.

Todos estos problemas pueden solo requerir el análisis de imágenes estáticas debido a que la información a detectar está contenida en un instante, como puede ser la detección de objetos, o puede necesitar de la información codificada en distintos instantes de tiempo como puede requerir la detección de acciones. El tratamiento de imágenes estáticas y vídeos requieren de técnicas diferentes por lo que se analizará cada caso por separado.

2.3.1. Visión Artificial en imágenes

La visión artificial en imágenes lleva años desarrollándose, siendo de los primeros problemas resueltos el de reconocimiento de caracteres, donde destaca Neocognitron en 1980 [2]. A diferencia de las técnicas de machine learning en datos más çlásicos", el tratamiento de imágenes presenta ciertos retos propios al formato que se deben tener en cuenta. Entre ellos podemos encontrar:

- **Tamaño:** el peso de cada "dato.[es] sustancialmente mayor al tratarse de una matriz de números con un ancho, alto y de profundidad variable. Esto presenta varios inconvenientes al requerir más tamaño de memoria y de procesamiento.
- **Color/blanco y negro:** las imágenes pueden ser tratadas en color o en blanco y negro, dato que se reflejará en la profundidad de la matriz siendo de 3 en el caso de imágenes en RGB y de 1 en blanco y negro. Al tener que diseñar el algoritmo con una de las dos opciones en mente, se tiene que tomar la decisión de antemano, limitando el uso posterior de algoritmo a solo ese tipo.

- **Resolución:** las imágenes de entrada pueden tener resoluciones muy diversas y en muchos algoritmos es un dato predefinido. Es por ello por lo que las imágenes de entrada se tendrán en muchos casos que normalizar a un tamaño constante con transformaciones de reescalado. La implicación de usar distintas resoluciones influye en el tamaño de la red, en el detalle que se podrá apreciar en la foto y en la memoria que será necesaria para entrenar el algoritmo.

Pasadas estos retos, la visión artificial puede resolver tres problemas principales: clasificación, detección de objetos y segmentación. Cada problema supone el siguiente paso lógico frente al previo, aumentando significativamente su utilidad y dificultad.

2.3.1.1. Clasificación

El problema de clasificación consiste en la capacidad de categorizar una imagen en su totalidad. El objetivo es darle a cada imagen una etiqueta específica. Es el caso más simple al contener exclusivamente una de las categorías dentro de la imagen. Uno de los problemas más importantes a tener en cuenta para crear un algoritmo es la dificultad de obtener una base de datos de calidad. Esto se resolvió con la creación en 2009 de ImageNet [24], una base de datos que actualmente contiene más de 20 000 categorías y 14 millones de imágenes.

Inicialmente se utilizaron redes neuronales que conectaban todos los píxeles de la imagen a la primera capa de la red también llamadas Fully Connected Neural Networks (FCNN) (Figura 9). Este método no era muy eficiente ya que, al aumentar el tamaño de la imagen, el número de pesos aumentaba sustancialmente convirtiendo la red en estructuras muy pesadas. Además, la capacidad para clasificar imágenes de una misma categoría con distorsiones visuales provocadas por la perspectiva o la rotación de la imagen se veía muy afectada por esta estructura tan rígida.

Por todo esto se desarrollaron las redes convolucionales o Convolutional Neural Networks (CNN) (Figura 10). Estas redes consisten en capas compuestas de filtros o kernels con parámetros entrenables que realizan operaciones de convolución a lo largo de la imagen. De esta forma los filtros aprenden a activarse cuando detectan ciertos rasgos en las imágenes, permitiéndonos extraer información de las imágenes con un coste mucho menor en número de parámetros. El algoritmo más famoso que aplicó esta estrategia fue Alexnet [27], creado en 2012 por Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton para la competición de ImageNet.

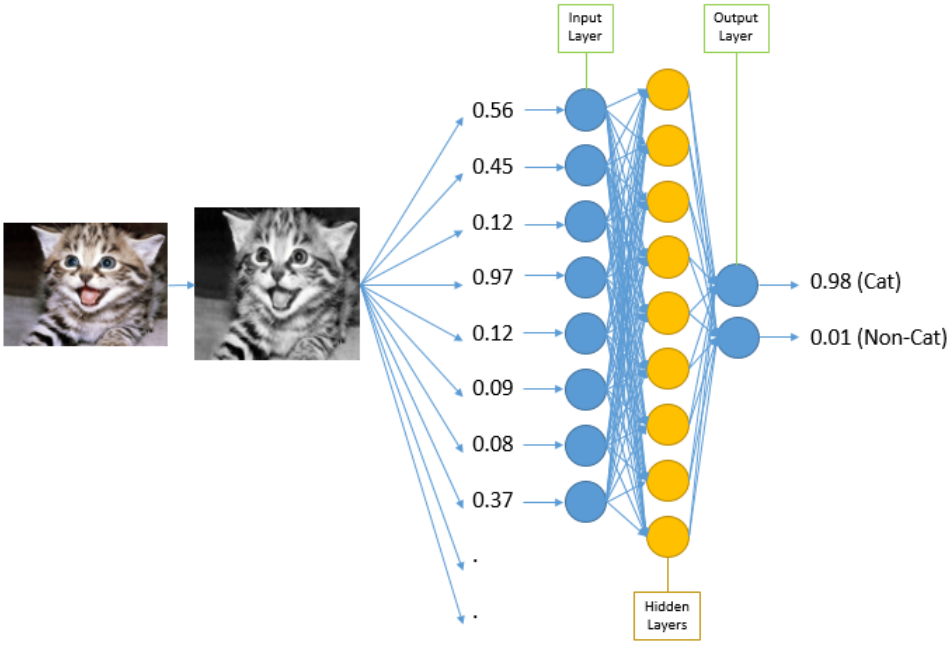

Figura 9. *Clasificación de imagenes con redes neuronales [25].*

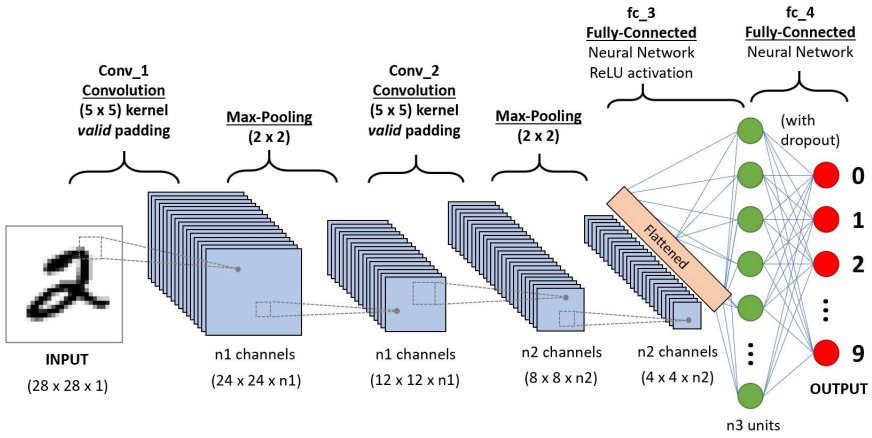

Figura 10. *Estructura clásica de la implementación de redes neuronales convolucionales [26].*

2.3.1.2. Detección de objetos

Una vez resuelto el problema de clasificación de imágenes, nos damos cuenta de que en el mundo real rara vez las imágenes que queremos analizar tienen un solo objeto. Un objetivo más realista sería el poder detectar los distintos objetos presentes en una imagen y su localización mediante coordenadas del recuadro que lo contiene (Figura 11). Este es por tanto el problema de detección de objetos que se puede dividir en dos categorías:

- **Métodos .ºne-stage":** se prioriza la velocidad de inferencia. Dentro de esta categoría podemos encontrar algortimos como YOLO, SSD o RetinaNet.

- **Métodos "two-stage":** se prioriza la precisión de detección. Encontramos modelos como Faster R-CNN, Mask R-CNN y Cascade R-CNN.

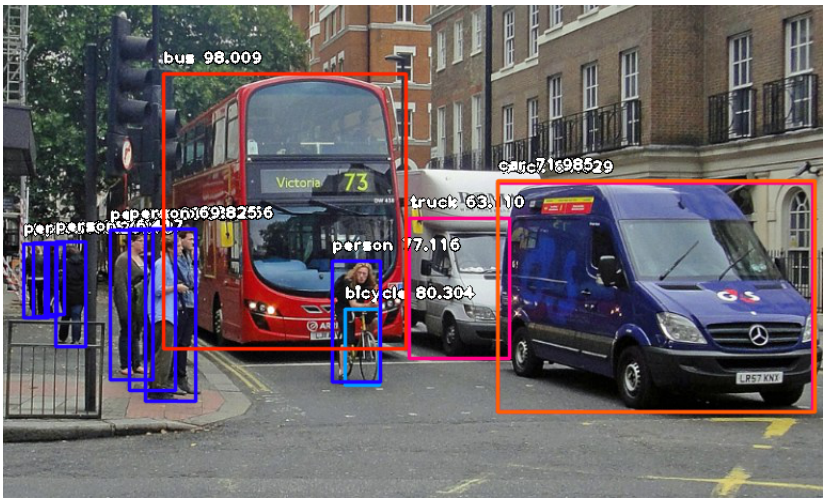

Figura 11. *Detección de objetos en imágenes [28].*

2.3.1.3. Segmentación

El ultimo problema que nos encontramos sería el de segmentación. El objetivo consiste en detectar que píxeles en la imagen corresponden a que objeto de la misma categoría (Figura 12). De esta forma no solo se pude saber dónde está en la imagen sino también su forma y volumen. Este problema se encuentra en pleno apogeo en el ámbito de la investigación para aplicaciones desde la conducción autónoma hasta la implementación en robots para detectar la forma de los objetos para planificar como agarrarlos de forma óptima.

Figura 12. *Segmentación en imágenes [29].*

2.3.2. Visión Artificial en vídeos

El problema de reconocer acciones en vídeos se encuentra en una fase de investigación puntera, apareciendo nuevos métodos y estructuras cada año. Como se ha mencionado

anteriormente, el peso de los archivos de vídeos es muy superior a otros formatos y es por eso que se puede observar una clara relación entre el acceso a un mayor poder de computación y la complejidad de los modelos. Al igual que con el análisis de imágenes, nos encontramos con las mismas restricciones de tamaño, color y resolución de los vídeos añadiendo el nuevo factor de cuantos "frames.º fotogramas por segundo se van a procesar. Y es que los vídeos, además de las dos dimensiones proporcionadas por la resolución horizontal y vertical y la dimensión otorgada por el color, posee una cuarta dimensión temporal reflejada en el número de fotogramas por segundo.

Los problemas que se pueden resolver analizando vídeos son muy diversos, aunque muchos de ellos se pueden resolver analizando cada fotograma por separado con las técnicas mencionadas en el apartado anterior e implementando otros algoritmos que lo complementen. Este sería el caso del seguimiento de objetos o de la segmentación de objetos en vídeos. Por otra parte, hay problemas que no pueden ser resueltos con esta técnica debido a la necesidad de utilizar la información codificada en la dimensión temporal, como puede ser la detección de acciones. Es de este problema donde vamos a profundizar en tres algoritmos comúnmente usados:

2.3.2.1. ConvNet + LSTM [20]

Esta estructura se basa en reutilizar los mismos algoritmos que identifican patrones en imágenes y pasar la información que extraiga a una red que analiza la relación temporal entre los diferentes fotogramas. Más específicamente, se aplica una red convolucional (ConvNet) de 2D que está formada por filtros en forma de matriz de dimensión variable que nos permite identificar patrones en las imágenes estáticas (Figura 13). Esos patrones se definirán por los pesos en la fase de entrenamiento y ocasionalmente pueden ser importados de otras redes entrenadas en aplicaciones parecidas definiendo así una técnica denominada Transfer Learning. Esto nos da la ventaja de que no se empieza de cero lo que permite necesitar menos datos para entrenar el algoritmo. Finalmente, los datos pasan por la LSTM o Long Short-Term Memory la cual es una estructura de redes neuronales recurrentes (RNN) que tienen retroalimentación temporal arbitraria que permite analizar información en función del tiempo.

2.3.2.2. 3D ConvNets [21]

Para analizar imágenes estáticas se usaban redes convolucionales 2D que permitían extraer patrones mediante la aplicación recurrente de filtros (Figura 14). En esta técnica por tanto para añadir el factor temporal se añade a los filtros una nueva dimensión que corresponda a los

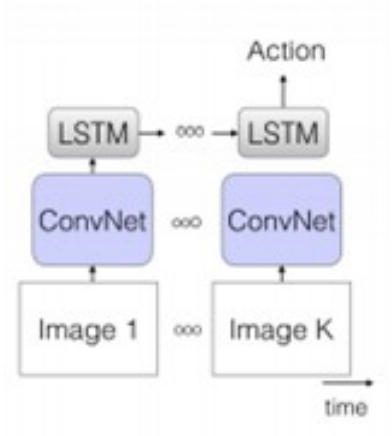

Figura 13. *Estructura de ConvNet + LSTM [20].*

fotogramas consecutivos a los que se le va a aplicar el filtro. Esto nos permite detectar no solo patrones en el espacio sino también en el tiempo. Después de aplicar una secuencia de estos filtros, se procede a pasar la matriz resultante por una red neuronal totalmente conectada que nos permite dar una probabilidad resultante de la clasificación del video. Con vídeos de alta resolución o de larga longitud nos encontramos el problema de que el tamaño de la red neuronal totalmente conectada es muy grande y supone gran parte de los pesos entrenados, resultando en un algoritmo de gran tamaño y difícil de entrenar por las limitaciones computacionales.

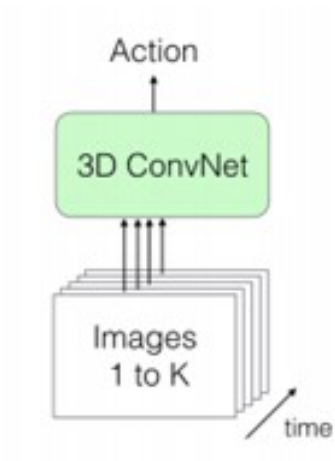

Figura 14. *Estructura de 3D ConvNets [21].*

2.3.2.3. Two-Stream Networks [22]

Esta red gira en torno a combinar dos algoritmos distintos para obtener una predicción. Primero analizamos las cualidades espaciales de un fotograma mediante una red convolucional 2D que es combinada con otra red convolucional 2D que recibe el flujo óptico (optical Flow) preprocesado por el sistema anteriormente (Figura 15). El flujo óptico es el patrón de

movimiento aparente de los elementos de una secuencia de imágenes respecto al observador. De esta forma conseguimos analizar tanto la información espacial y la temporal para después combinarla y tomar la decisión de clasificación.

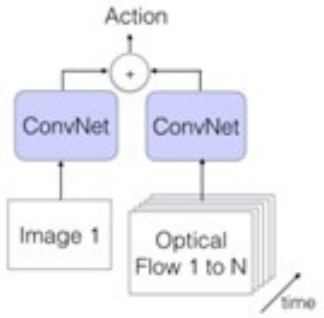

Figura 15. *Estructura de Two-Stream Networks [22].*

Estos tres métodos son populares, pero traen consigo varios problemas que deben ser resueltos:

- La necesidad de que los videos de entrada sean de tamaño regular lo que implica mantener constante el número de fotogramas y por tanto variar los fotogramas por segundo. A consecuencia de esto, acciones que tengan una alta variabilidad en el tiempo de acción hacen que estos algoritmos sean incapaces de aprendérselas.
- El gran número de pesos que tiene en función de la calidad del video de entrada y por tanto la necesidad de más capacidad de procesamiento.
- En el caso del Two-Stream Network el tener que calcular el flujo óptico significa tener que hacer un preprocesado de que requiere un uso intensivo de la CPU y por la tanto reduce su eficiencia.
- La precisión obtenida en las distintas bases de datos como UCF101 o Kinetics puede mejorarse [23].

Capítulo 3

Descripción del modelo desarrollado

ESTE capítulo pretende explicar todo lo relacionado con la generación de la base de datos, el pipeline y el algoritmo empleado, desde su elección hasta su desarrollo y posterior optimización.

3.1. Objetivos y especificación

El objetivo de este trabajo es desarrollar un algoritmo que reconozca acciones en vídeos de cámaras de seguridad. Estas acciones no pueden ser reconocidas en una imagen estática, como puede ser el caso de una persona remando o jugando al fútbol, y por tanto se tendrá que diseñar todo el sistema para que perciba e interprete la información temporal. Como ya se ha mencionada en el apartado 1.3, se creará la base de datos, el pipeline que alimente los datos y el algoritmo cumpliendo con las siguientes características.

En cuanto a la base de datos:

- **Debe estar balanceada:** debe tener el mismo número de datos correspondientes a cada categoría.
- **Debe ser preferiblemente de acceso público:** si se quiere permitir que sean reproducibles los resultados, todo el mundo debe tener acceso a los vídeos sobre los que se basa el trabajo.
- **Deben estar bien clasificados:** el algoritmo es tan bueno como la calidad de la clasificación de la base de datos.
- **Deben estar recortados:** se debe dejar exclusivamente el contenido que contenga la acción.

- **Deben de tener la máxima calidad posible:** posteriormente puede reducirse la calidad antes de pasarlos por la red, pero esta decisión se puede tomar solo si está acotado superiormente por una calidad decentemente alta.
- **Debe haber una cantidad considerable de datos:** si se quiere poder dividir la base de datos en Train, Validation y Test y que estos sean estadísticamente representativos se debe tener un número alto de vídeos siempre que sea posible. De no ser así los resultados obtenidos pueden no ser extrapolables a situaciones reales.

En cuanto al algoritmo:

- **3D CNN:** que utilice la estructura de 3D CNN parcialmente o en su totalidad.
- **Que sea ligero:** tiene que ser posible entrenarlo en un espacio de tiempo razonable con los recursos disponibles y ser fácilmente ejecutable.
- **Lenguaje de programación:** que utilice el lenguaje de programación Python [3] y alguno de los dos Deep Learning frameworks más importantes como son Tensorflow [4] o PyTorch [5] para facilitar su integración en una aplicación final.
- **Precisión:** que tenga una precisión de al menos 70 % en validación.

El pipeline, además, deberá tener en cuenta las limitaciones del poder de computación y de las plataformas que se utilizarán, así como manejar la variabilidad en el formato y forma de los datos de entrada.

3.2. Ética en la IA

A la hora de diseñar un sistema que implemente algoritmos de inteligencia artificial, es muy importante mencionar la ética. Estos tipos de sistemas eliminan la intervención de seres humanos en la toma de decisiones y, por lo tanto, se corre el riesgo de crear sistemas que intencionada o accidentalmente tomen decisiones basadas en raza, sexo u otras categorías. Esto es conocido como bias en el contexto de la ética en la IA.

3.2.1. Casos reales

Para entender la importancia de tener en cuenta este concepto a la hora de diseñar algoritmos y bases de datos, se estudiará brevemente dos casos reales con sus respectivas consecuencias.

El primero se dio en la red social Twitter, donde se implementó un algoritmo para que, en caso de que una foto subida fuese más grande que la ventana donde se iba a representar la miniatura de la imagen, este fuese capaz de detectar que parte de la imagen era más importante y mostrar esa zona. El criterio para decidir qué parte de la imagen es importante en un principio se realizó detectando caras, pero este método se descartó por los errores que cometía. Finalmente se aplicó un método denominado en la neurociencia como prominencia (Figura 16), que consiste en el estudio de las partes de la imagen en las que las personas se fijan más, y diseñar una red neuronal que imitase este proceso. De esta forma el algoritmo no solo serviría para las imágenes donde haya presencia de personas, sino que también determinaría las regiones de interés del resto.

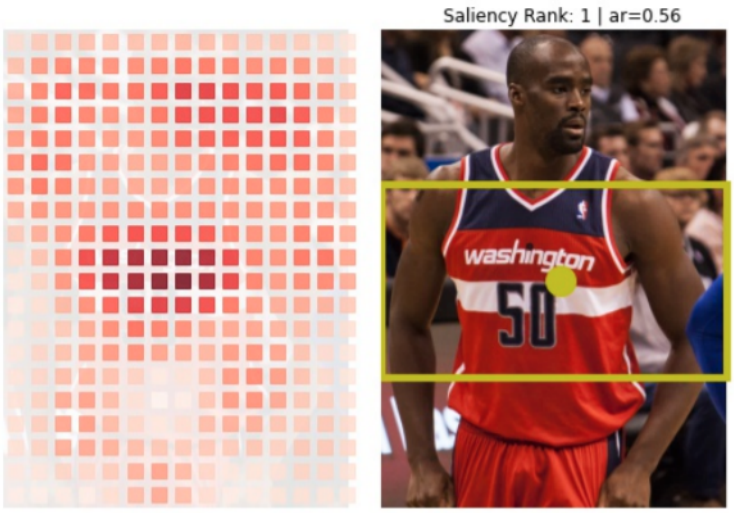

Figura 16. *Mapa de calor de las zonas con más prominencia [30]*

El problema se detectó cuando el grupo de investigadores [30] comprobó cual sería el comportamiento ante la presencia de personas de distinta raza en imágenes lo suficiente grandes para que el algoritmo tuviese que tomar la decisión de a cuál enfocar la atención. Se comprobó que sistemáticamente el algoritmo daba prioridad a personas blancas frente a afroamericanas o a recortar el cuerpo de las mujeres frente a su cara, independientemente de variables como posición en la imagen, color de fondo, etc. La explicación para este resultado no fue una intención maliciosa por parte de los diseñadores del algoritmo, sino la composición de la base de datos utilizada.

La importancia de balancear correctamente la base de datos reside en el proceso de optimización de los algoritmos. Este proceso tiene como objetivo minimizar el error del algoritmo, dando el mismo peso al error cometido al estimar cualquiera de los datos. Si una de las clases está infrarrepresentada, nos encontramos con un bias que de menos importancia al error total de una de las clases, dando como resultado un algoritmo que discrimina por el color

de piel. Existe una gran dificultad para crear bases de datos bien balanceadas, ya que los datos se extraen en general de internet, donde las desigualdades sociales pueden verse reflejadas. Así pues, este caso refleja la necesidad de verificar el bias en las bases de datos, tanto públicas como privadas, en las que los algoritmos se basan. Finalmente, Twitter decidió eliminar esta funcionalidad.

El segundo caso se dio en un algoritmo comercial ampliamente utilizado por las compañías de seguros médicos en Estados Unidos [31]. El objetivo del algoritmo era minimizar los gastos médicos de sus afiliados mediante la priorización de los tratamientos y recursos extras más costosos a las personas cuyos gastos médicos crónicos futuros fuesen a ser más caros, mejorando además la satisfacción y los resultados. La métrica para predecir estos gastos médicos futuros y la necesidad para el pago de cuidados suplementarios eran los gastos médicos pasados, datos fácilmente obtenibles por la compañía. Es aquí donde nos encontramos el problema. Si se hace un estudio profundo de las predicciones del algoritmo, se puede observar un bias para priorizar el pago de cuidados y tratamientos suplementarios de personas blancas frente a personas afroamericanas o con menos poder adquisitivo.

Esta vez, el bias no surge de la base de datos, sino de la métrica utilizada. Al usar los gastos pasados para predecir gastos futuros y así dar prioridad al pago de un tratamiento, se presupone un mismo comportamiento de la población a la hora de acudir al médico en las primeras etapas de una enfermedad. Las diferencias socio económicas de una población dan lugar a que ciertos grupos decidan no acudir al médico ante dolencias leves debido al potencial coste económico de las visitas y el tratamiento. Al no ser tratadas a tiempo, estas pueden derivar en enfermedades más graves con un mayor coste económico. El algoritmo tomando estos datos como referencia, dio prioridad al tratamiento de personas que hiciesen más uso del sistema sanitario por su favorable situación económica y discriminó a las poblaciones más desfavorecidas teniendo las mismas necesidades médicas. Este caso refleja la gran importancia de estudiar el bias a la hora de elegir la métrica a utilizar y del impacto real en las vidas de las personas cuando no se tiene esta supervisión. Tras corregir este bias, el porcentaje de personas afroamericanas que recibirían cuidados extra pasaría de 17.7 % a 46.5 % [31].

3.2.2. Causas y soluciones

Como se ha podido observar, el impacto de una mala gestión de la ética en la IA no pertenece a un mundo distópico, sino que se encuentra ya en el día a día de los desarrolladores de estos algoritmos y tiene consecuencias reales en la vida de las personas. El origen del bias es muy diverso pero tres de los principales vectores son [32]:

- **Bias en el Diseño:** a la hora de formular un problema para resolver con un modelo, este se plantea para cumplir el objetivo de la empresa sin valorar si ha sido formulado para asegurar que sea justo. Podría darse el caso de que defina parámetros intrínsecamente injustos a la hora de tomar decisiones como puede ser que decidir el acceso a crédito según el barrio en el que vivas.
- **Bias en los Datos:** las bases de datos públicas más populares se obtuvieron de datos públicos. Estos datos no están necesariamente balanceados intrínsecamente, sino más bien todo lo contrario al reflejar las desigualdades sociales de acceso a los recursos digitales. El reducido tamaño de ciertas bases de datos puede impedir que sean representativas de la realidad.
- **Bias en la Selección:** a la hora de seleccionar que parámetros sirven como entrada del modelo, si no se hace un estudio del bias que puede generar tanto individualmente como un su conjunto, el modelo puede acabar discriminando ciertos sectores de población. Aparentemente un grupo de parámetros pueden parecer justos, pero si se analiza el conjunto se puede llegar a un modelo discriminatorio dependiendo del peso que se le de a cada uno.

Para solucionar estos problemas, se debe seguir una serie de pautas que garanticen la transparencia, la capacidad de verificar que parámetros afectan a la toma de decisiones del algoritmo y el impacto y resultados en los diferentes grupos demográficos. Una solución propuesta por el departamento de IA de Google es la creación de "model cards.º tarjetas de cada modelo [33]. Estas tarjetas contendrían la siguiente información:

- Como funciona el modelo
- El uso previsto
- El origen de los datos
- El rendimiento en subgrupos

Todos estos datos permitirían evaluar internamente la existencia de bias y el impacto sobre los diferentes grupos de población, proporcionando la transparencia necesaria a la hora de implementar modelos que tomen decisiones y que tengan un gran impacto en la sociedad.

3.3. Datos

En esta sección se pretende explicar el proceso de obtención, análisis y procesamiento de los vídeos que componen la base de datos utilizada para entrenar los algoritmos. Como se ha mencionado anteriormente, la base de datos debe cumplir las siguientes características:

- **Debe estar balanceada:** debe tener el mismo número de datos correspondientes a cada categoría.
- **Debe ser preferiblemente de acceso público:** si se quiere permitir que sean reproducibles los resultados, todo el mundo debe tener acceso a los vídeos sobre los que se basa el trabajo.
- **Deben estar bien clasificados:** el algoritmo es tan bueno como la calidad de la clasificación de la base de datos.
- **Deben estar recortados:** se debe dejar exclusivamente el contenido que contenga la acción.
- **Deben de tener la máxima calidad posible:** posteriormente puede reducirse la calidad antes de pasarlos por la red, pero esta decisión se puede tomar solo si está acotado superiormente por una calidad decentemente alta.
- **Debe haber una cantidad considerable de datos:** si se quiere poder dividir la base de datos en Train, Validation y Test y que estos sean estadísticamente representativos se debe tener un número alto de vídeos siempre que sea posible. De no ser así los resultados obtenidos pueden no ser extrapolables a situaciones reales.

En los siguientes subapartados se abordará la obtención y el procesamiento de los vídeos para cumplir con estos objetivos.

3.3.1. Obtención de la base de datos

En la comunidad de desarrolladores de algoritmos de inteligencia artificial, está muy extendida la práctica de compartir bases de datos públicas para poder comparar los resultados de diferentes arquitecturas sobre unas bases comunes. De esta forma, es posible encontrar fácilmente todo tipo de bases de datos de forma gratuita, con hasta implementaciones específicas en las librerías de machine learning más populares.

Primero se exploraron bases de datos públicas como UCF 101 [7] o Kinetics [8], buscando categorías de vídeos de cámaras de seguridad. Al no haber una cantidad razonable para conseguir una base de datos robusta, se realizó un script en Python para descargar vídeos de YouTube con las características ya mencionadas. El número total de vídeos extraídos fueron 167 con muy diversas características de resolución, fotogramas por segundo (FPS) y escenarios o entornos. Esto hará que se tenga que procesar los datos para conseguir uniformidad o que el modelo sea capaz de aceptar datos con variabilidad.

Analizando el bias de la base de datos, teniendo en cuenta el contexto del uso de vídeos de cámaras de seguridad, las leyes de protección de datos a las que este contenido está sujeto resulta en una mayor representación de vídeos en países con leyes más laxas. Esto implica un gran desequilibrio en la representación de distintos tonos de color de piel y estilos culturales. Por lo tanto, a pesar de que la base de datos se puede seguir usando para probar distintos tipos de algoritmos, los algoritmos entrenados a partir de la base de datos no pueden ser usados en entornos de producción ya que encontraríamos muchas situaciones en las que clasifique personas de las etnias más representadas en las categorías que se han definido.

3.3.2. Post-procesamiento de la base de datos

Una vez hemos obtenido los vídeos, es necesario recortarlos y clasificarlos correctamente como vídeos normales y vídeos en los que se comete la acción. Para ello, se generó en un archivo .txt una lista con los respectivos nombres de cada archivo y los segundos que queremos etiquetar junto al nombre de esta. Tras esta clasificación, un script de Python recortó automáticamente los vídeos y los clasificó según la etiqueta. El resultado final fue la creación de 204 vídeos con situaciones normales y 208 vídeos con la acción. Para mejorar la calidad de la base de datos se han tenido en cuenta las siguientes consideraciones.

3.3.2.1. Recorte de vídeos

Los vídeos recortados deben tener exclusivamente la acción a detectar, dando como resultado clips de duración muy variable en un rango de entre 2 y 10 segundos. Para evitar que haya un bias por la duración de estos vídeos recortados, por cada uno que contenga una acción dentro de lo posible se ha intentado obtener otro sin ninguna acción con la misma duración.

3.3.2.2. Bias del fondo y las personas

Al no tener una cantidad muy grande de datos, se tiene que intentar reducir al máximo posible el bias. Una de las formas en el que el algoritmo podría falsamente clasificar los vídeos podría ser detectando características propias de las personas que cometen la acción frente a las que no las cometen. Es por eso por lo que se intentó siempre que fuese posible obtener recortes de cada vídeo donde se muestre a la misma persona cometiendo la acción y en una situación normal. De esta forma, es imposible relacionar la persona con la acción en sí, evitando así el que se pueda resolver por memorización de los rasgos de las personas como puede ser raza, color de ropa o sexo.

Esto ocurre también con el entorno de la grabación, que en muchos de los algoritmos de detección de acciones puede resultar en uno de los elementos más importantes a la hora de conseguir buena precisión, en especial en la detección de acciones en imágenes. Esto se

Figura 17. *Ejemplo de cómo se puede determinar la acción sin ver a la persona que la comete. De izquierda a derecha: salto, golf, canoa y patinaje sobre hielo. [34]*

refleja en el paper "Why Cant I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition"[34]. En este artículo se demuestra que tanto los algoritmos como los humanos hacemos uso del contexto para determinar, sin ver a la persona, la acción que se está cometiendo. En la figura 17 se puede observar un ejemplo de cómo tras recortar a la persona se es capaz de reconocer y usar el entorno para clasificar la acción. Eliminando este bias, es posible mejorar la generalización del algoritmo pero también dificulta la tarea al reducir la información disponible. En nuestro caso, al tener pocos vídeos y estos ser en general en un entorno parecido, se ha eliminado el bias generando vídeos de situaciones normales y con la acción en el mismo entorno evitando la memorización de características de este para la clasificación.

3.3.2.3. Filtro de movimiento

Otra forma de resolver este problema definitivamente sería capturando solo el movimiento que se detecta en cada vídeo. Al ser vídeos sacados de cámaras de seguridad, el ángulo de visión y posición son constantes y por tanto son las personas las únicas que modifican los píxeles en el tiempo cuando se mueven. Además, elimina elementos estáticos como letras y números introducidos en los vídeos por las cámaras de seguridad. Esto se consigue calculando la diferencia absoluta entre fotogramas consecutivos en blanco y negro, resultando en píxeles negros donde no ha habido movimiento y píxeles blancos donde sí. Para ser más resiliente al ruido, es preferible calcular la diferencia con una media móvil ponderada de los fotogramas anteriores aunque esto produce la aparición del efecto secundario de imagen fantasma. Esta operación se puede realizar con la librería cv2 de OpenCV [35].

En la práctica, el ruido generado por la baja calidad de la cámara y la compresión a la que ha sido sometido al subirse a plataformas como YouTube da lugar a un resultado con calidad muy baja como se puede ver en las figuras 18 y 19, siendo este uno de los ejemplos de aplicación más favorables. Se pierde por tanto los detalles importantes de manos y cuerpo que pueden ser clave para la detección de movimiento y, por tanto, acciones.

Figura 18. *Fotograma de uno de los vídeos sin aplicar el filtro de movimiento.*

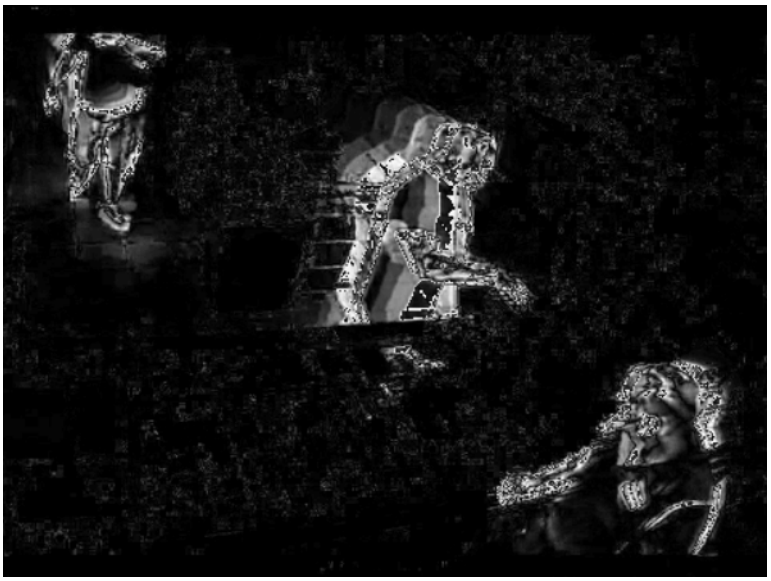

Figura 19. *Fotograma de uno de los vídeos al aplicar el filtro de movimiento.*

3.3.2.4. División de la base de datos

El último paso para preparar la base de datos para el proceso de aprendizaje es su división en tres grupos:

- **Train o entrenamiento:** este subgrupo será utilizado para entrenar el modelo automáticamente. Es por tanto el más grande, con un 70 % de los vídeos, 144 normales y 141 con la acción.
- **Validation o validación:** este subgrupo será utilizado para manual o automáticamente afinar los hiperparámetros del modelo. Estos parámetros definen el proceso de entrenamiento y es por eso por lo que no pueden ser obtenidos con los datos de entrenamiento.

De esta forma nos aseguramos de optimizarlos ante datos junto a los que no ha sido entrenado el modelo. Contará con un 15 % de los vídeos, 31 normales y 35 con la acción.

- **Test o prueba:** este último subgrupo sirve para validar los resultados del modelo frente a datos que no han sido vistos por el proceso de entrenamiento ni de optimización de los hiperparámetros. Debe ser representativo de la realidad ya que el rendimiento que el modelo obtenga al hacer predicciones sobre los datos representan como de bueno el algoritmo sería al aplicarlo en la vida real. Los hiperparámetros no deben ser modificados para mejorar el resultado en este subgrupo. Contará con un 15 % de los vídeos, 31 normales y 30 con la acción.

Es importante asegurarse que no exista un bias entre los subgrupos, como puede ser la aparición de la misma persona, para evitar que si el algoritmo usase para clasificar otros rasgos ajenos a la acción que queremos detectar esto se viese reflejado en los resultados de estimar el resto de los subgrupos. De ser así, los resultados obtenidos serían inválidos. Cumpliendo todas las restricciones anteriormente mencionadas, dentro de los subgrupos se ha intentado balancear las categorías lo máximo posible para no crear un bias de clases, el cual hace que adivinando aleatoriamente se pueda tener una precisión distinta al 50/50 por ciento.

3.3.2.5. Data Augmentation

Al no disponer de muchos datos, es necesario recurrir a un conjunto de técnicas denominado Data Augmentation o aumentación de datos. El objetivo de las técnicas es incrementar el número de datos disponible aplicando modificaciones a los datos ya existentes para así permitir que el algoritmo generalice mejor a datos un poco diferentes a los ya existentes. Las modificaciones pueden ser:

- **Espaciales:** los vídeos se reflejan horizontalmente para así conservar en la misma orientación el eje vertical.

- **Color:** modificar el espectro de colores, contraste y brillo entre otros parámetros permite al algoritmo ser más resiliente a estas variaciones entre vídeos.

- **Ruido:** introducir ruido en las imágenes permite al algoritmo trabajar con distintas calidades de vídeos y ser más robusto ante estas variaciones.

Con todos estos cambios se limita la capacidad del algoritmo para memorizarse los datos del entrenamiento en vez de reconocer las acciones en sí.

3.4. Pipeline

El trabajo del pipeline es gestionar de la forma más óptima posible el proceso de entrenamiento. Esto abarca desde realizar las transformaciones pertinentes a los datos, cargarlos en la RAM, CPU o la GPU, realizar las operaciones de entrenamiento, validación y prueba y finalmente guardar los resultados y el modelo.

3.4.1. Limitaciones del hardware

El primer aspecto a tener en cuenta al diseñar el pipeline son los recursos disponibles durante todo el proceso y los requerimientos de los algoritmos durante el entrenamiento. El objetivo es maximizar la velocidad de entrenamiento optimizando el uso del hardware del sistema. En este proyecto, hay dos recursos limitantes a la hora de minimizar el tiempo de entrenamiento: la RAM y la GPU.

En cuanto a la RAM, durante el proceso de entrenamiento se requiere poder acceder a los datos de la forma más rápida posible. Los datos pueden estar almacenados en la unidad de memoria (como puede ser un disco duro o una SSD o Solid State Drive) o transferirlos a la RAM que suele tener velocidades de lectura 30 veces superiores a las de una unidad SSD de almacenamiento. La limitación de transferir todos los datos a la RAM viene dada por su limitada capacidad de memoria en comparación con las unidades de almacenamiento. Esto se acentúa cuando se trabaja con vídeos, los cuales tienen un peso mucho mayor frente al resto de tipos de datos. Si no es posible cargar todos los datos en la RAM, se tendrán que cargar parcialmente y transferir los datos cada vez que sea necesario. Esto implica un aumento del tiempo de entrenamiento al tener que transferir datos constantemente, con el aumento de tiempo que supone esta actividad.

En cuanto a la GPU, su uso es requerido gracias a su capacidad de realizar operaciones con matrices y tensores de forma más óptima que una CPU. Como el entrenamiento de redes neuronales se realiza mediante operaciones con tensores, su uso está muy extendido en el campo de la IA. Los limitantes en este hardware son su memoria interna y su capacidad de procesamiento. Este último influye directamente en la velocidad de las operaciones, pero es el primero el que puede resultar más problemático. Durante el proceso de entrenamiento, en la memoria interna de la GPU se tiene que almacenar tanto los parámetros de la red que estamos entrenando, así como los datos de la operación actual. Esto quiere decir que con una memoria limitada o se reduce la complejidad del modelo, se limita la cantidad de datos usados en cada momento del entrenamiento o ambas a la vez.

El hardware utilizado durante el transcurso de este proyecto posee 16GB de RAM y una GPU RTX 2060 super con 8 GB GDDR6 de memoria interna. Estas dos características limitan mucho tanto los vídeos que se pueden cargar en la RAM, así como la complejidad del algoritmo, problemas que se han mitigado con técnicas descritas más adelante. Además, se ha tenido acceso a la plataforma Google Colab [9], la cual permite gratuitamente entrenar modelos con la capacidad de procesamiento no utilizada en los servidores de la empresa. Gracias a ella tenemos acceso [36] a GPUs como la K80, T4, P4 y P100 de NVIDIA y 12 GB de RAM. Las limitaciones son un máximo de 12 horas de entrenamiento y la posibilidad de reducir los recursos disponibles según la demanda del servicio lo que implica mucha incertidumbre y falta de estabilidad en los tiempos de procesamiento. Finalmente se decantó por usar el hardware físico debido a la estabilidad y repetibilidad que proporciona.

3.4.2. Machine Learning Frameworks

Para facilitar y optimizar la definición de los algoritmos, el proceso de entrenamiento y la reproducibilidad de todo el proceso, se han desarrollado diversos frameworks. Entre ellos, los más populares son Tensorflow [4], desarrollado por Google, y PyTorch [5], producido por Facebook.

Ambos nos permiten de forma sencilla definir las distintas capas del modelo, así como el proceso de entrenamiento. Además, estos nos permiten aprovecharnos del entrenamiento en GPU mediante la implementación de la plataforma de computación en paralelo de NVIDIA: Compute Unified Device Architecture o CUDA. Esta plataforma nos permite aprovecharnos de las ventajas anteriormente mencionadas al usar GPUs en el proceso de entrenamiento de una forma fácil y sencilla al no requerir ningún cambio sustancial en el código.

A pesar de todas las similitudes, PyTorch nos permite acceder más fácilmente a los bucles de entrenamiento para poder modificarlos según nuestras necesidades, tanto para probar nuevas arquitecturas como para visualizar los datos. Además, implementa funcionalidades de CUDA como el paquete Automatic Mixed Precision o AMP [37] que permite modificar automáticamente la precisión de lo tensores para reducir las necesidades de computación sin penalizar el resultado final de los modelos. Por todo esto y por la gran comunidad entorno a este framework, se ha utilizado PyTorch finalmente en el proyecto.

3.4.3. Carga de datos

Una vez ya tenemos toda la base de datos generada, se debe configurar la interfaz que permita leer los datos, procesarlos y cargarlos en el dispositivo que realice el entrenamiento.

Como se ha mencionado anteriormente, no se dispone de suficiente RAM ni espacio en la memoria de la GPU para alojar toda la base de datos. Por lo tanto, la estrategia a seguir será cargar los datos poco a poco según se vaya necesitando durante el proceso de entrenamiento. Para ello se utilizaran dos clases: Dataset y DataLoader.

3.4.3.1. Dataset

La clase abstracta Dataset [38] de PyTorch del paquete torch.utils.data nos permite crear subclases personalizadas para cada base de datos según como esté construida. Para ello, podemos sobrescribir las funciones siguientes:

- **__init__():** inicializa la base de datos y las transformaciones que queramos aplicar a cada elemento. En caso de que los datos sean muy grandes para almacenarlos en la RAM, cargamos un CSV o determinamos las diferentes carpetas que nos indican donde encontrar los datos.
- **__len__():** permite cuantificar la cantidad de datos disponibles en la base de datos. Esto es necesario para la clase DataLoader que analizaremos más adelante.
- **__getitem__():** tras pasarle un índice nos devuelve el elemento correspondiente tras aplicar la transformación si se ha definido una.

En nuestro caso, tenemos una serie de vídeos almacenados en carpetas que dividen los archivos en las tres subdivisiones de Train, Validation y Test. Estas a su vez contienen subcarpetas que clasifican cada vídeo según su clase. Estos vídeos se pueden cargar directamente o se puede guardar previamente cada fotograma como una imagen para que la operación sea más sencilla y ligera. A pesar de estas ventajas, se decidió conservar los archivos como vídeos para simplificar operaciones como cambiar los fotogramas por segundo o limitar el número de fotogramas totales mediante transformaciones.

Al tratarse de una estructura con carpetas que determinan la etiqueta de la clase a la que pertenece cada vídeo, podemos usar la clase DatasetFolder [39] de la librería TorchVision [40] como se puede observar en el extracto de código 3.1.. Esta librería de PyTorch contiene funcionalidades específicas al entrenamiento de modelos aplicados en el campo de la visión artificial. DatasetFolder nos permite cargar una base de datos proporcionándole los siguientes parámetros/objetos:

- **root** (string): la dirección donde se encuentran las carpetas con los vídeos.
- **loader** (callable): una función que devuelve una muestra tras proporcionarle la dirección de esta. En este caso, devolverá el vídeo como un tensor.

- **extensions** (tuple[string]): lista de extensiones aceptadas. En nuestro caso es .MP4.
- **transform** (callable, optional): una función con las transformaciones a aplicar a cada elemento cargado.

En cuanto al loader, se ha utilizado la librería pytorch-VideoDataset [41] creada por Yuxin Zhao como se puede observar en el extracto de código 3.1. En concreto, usamos la función VideoFilePathToTensor, la cual nos permite de una dirección a un vídeo devolver un tensor con las siguientes transformaciones:

- **max_len:** el máximo número de fotogramas totales.
- **fps:** el número de fotogramas por segundo que se van a extraer del vídeo.
- **padding_mode:** en caso de que no hubiese suficientes fotogramas para cumplir la restricción de máxima longitud, se puede definir qué tipo de padding o relleno a aplicar. Las opciones son None (no haría nada y por lo tanto los vídeos pueden tener longitud variable), 'zero' (rellenaría los fotogramas restantes con ceros) y 'last' (rellenaría los fotogramas restantes con copias del último fotograma).

En cuanto a la transformación a aplicar a cada vídeo, se ha decidido finalmente aplicar redimensionamiento para reducir la carga de computación y unificar la resolución y proporciones de todos los vídeos, mantener en escala de colores RGB o en escala de grises para reducir la carga de computación y normalizar las imágenes y finalmente aplicar aleatoriamente una transformación de volteo horizontal como técnica de aumentación de datos. De esta forma seguimos conservando la orientación vertical, la cual nunca puede darse al ser imposible la situación en la que una persona se encuentre invertida, y aumentamos el número de muestras con las que el algoritmo puede encontrarse, evitando aún más que se aprenda los datos. No se ha incluido las técnicas de aumentación de modificación de saturación o similares al trabajar con vídeos en escala de grises, así como tampoco la adición de ruido al no tener mucha calidad de origen lo que puede hacer imposible su clasificación.

Estas transformaciones se han realizado mediante las funciones VideoResize, VideoRandomHorizontalFlip y VideoGrayscale de la librería pytorch-VideoDataset [41] creada por Yuxin Zhao. Para pasarlas a DatasetFolder, se ha concatenado todas las transformaciones mediante la función Compose de torchvision.transforms [42] como se puede observar en el extracto de código 3.1. Utilizar los vídeos en color o en escala de grises así como los valores de resolución no son definitivos y deben tratarse como hiperparámetros en el proceso de entrenamiento. Esto se explicará más adelante.

```
transform = torchvision.transforms.Compose([
    transforms.VideoResize([128, 128]),
    transforms.VideoRandomHorizontalFlip(),
    transforms.VideoGrayscale(num_output_channels=channel),
])

loader = transforms.VideoFilePathToTensor(max_len=60, fps=10,
                                          padding_mode='last')

dataset_train = datasets.DatasetFolder(
    os.path.abspath('Dataset/Train'),
    loader=loader, extensions=('.mp4'), transform=transform)
```

Extracto de código 3.1. *Configuración del DatasetFolder, la función loader y las transformaciones.*

3.4.3.2. DataLoader

Tras generar el objeto de la base de datos, es el trabajo de la clase DataLoader de la librería torch.utils.data [43] de suministrar los datos durante todo el proceso de entrenamiento. Para ello se le debe pasar a la función el objeto de la base de datos junto a los siguientes parámetros:

- **batch_size:** representa el número de muestras con las que se entrena en cada iteración en la que se calcula la salida del algoritmo y se entrenan los pesos mediante propagación hacia atrás de errores calculando el gradiente para llegar al mínimo de la función de pérdidas. Este método se explicará más adelante.
- **num_workers:** en caso de usar una GPU, se puede definir un número de trabajadores o subprocesos para que procesen en paralelo las operaciones de carga de datos desde la memoria SSD a eventualmente la GPU. De esta forma no se crea un cuello de botella en este proceso que ralentizaría el proceso de entrenamiento.
- **pin_memory:** en caso de usar CUDA y por la tanto una GPU, esta opción nos permite guardar directamente en la memoria RAM los datos para luego pasarlos a la memoria de la GPU antes de devolverlos a la función que la esté llamando, ahorrando por tanto tiempo en el proceso al ser la memoria RAM más rápida que la SSD.
- **shuffle:** al no tener datos dependientes entre sí, esta opción nos permite mezclar aleatoriamente los datos al procesarlos en cada época de entrenamiento.

La selección óptima del batch_size óptimo se estudiará más adelante en el apartado 3.4.6 Optimización de hiperparámetros. Respecto a los otros tres parámetros, tanto num_workers

como pin_memory nos va a permitir mejorar los tiempos de entrenamiento aprovechando mejor la capacidad disponible al evitar secuencialmente realizar las tareas de cargado y transformaciones junto con el proceso de entrenamiento. Es la CPU y la RAM la que procesarían en paralelo los datos mientras la GPU continua con el proceso de entrenamiento, teniendo disponible siempre un nuevo lote en cuanto acaba con el anterior.

En cuanto a la selección del número de trabajadores, la técnica realizada fue empezar con un número elevado e ir reduciendo hasta que se encontró una bajada de la velocidad de entrenamiento por cada época, conservando el número anterior que fue de 4 trabajadores. El efecto en los recursos del método es un aumento en el consumo de memoria RAM, aunque si el lote es pequeño el impacto es menor.

El efecto que tiene hacer shuffle o mezclar los datos aleatoriamente en cada época de entrenamiento es el de hacer más robusto el algoritmo permitiéndole generalizar mejor, así como reducir las posibilidades de hacer over/underfitting al reducir la varianza. Se quiere activar este parámetro cuando tenemos una lista de datos de clases diferentes y queremos evitar que analice todos los datos de cada una de forma consecutiva. Así los batch son más representativos de la distribución general de los datos y el gradiente de cada uno es más similar al gradiente del conjunto de la base de datos. Se realiza en cada época para minimizar el riesgo de generar lotes que no sean representativos y que estos se mantengan durante todo el entrenamiento. Además, si evitamos que el orden sea constante eliminamos el bias generado por el cálculo del gradiente de una secuencia específica de datos, manteniendo la independencia de los cambios que cada muestra realiza en el modelo.

En el extracto de código 3.2 se muestra el procedimiento de la configuración del dataloader. Se puede observar que solo cuando se hace uso de una GPU se puede activar la funcionalidad pin_memory y el uso de trabajadores.

```
train_kwargs = {'batch_size': batch_size, 'shuffle': True}
test_kwargs = {'batch_size': test_batch_size, 'shuffle': True}

if use_cuda:
    cuda_kwargs = {'num_workers': 4,
                   'pin_memory': True}
    train_kwargs.update(cuda_kwargs)
    test_kwargs.update(cuda_kwargs)

train_loader = torch.utils.data.DataLoader(dataset_train, **
   ↪ train_kwargs)
```

```
test_loader = torch.utils.data.DataLoader(dataset_test, **test_kwargs)
```

Extracto de código 3.2. *Configuración del DataLoader.*

3.4.4. Optimizador

El proceso de entrenar un algoritmo de inteligencia artificial consiste en definir una función de pérdidas que represente como de bien realiza su cometido y mejorar este resultado a lo largo de las épocas de entrenamiento. El problema de minimizar esta función es que en general no son convexas y por lo tanto no tienen un único mínimo sino que tienen mínimos locales como puede verse en la figura 20. La estrategia con la que se intenta encontrar el mínimo global es muy importante para evitar quedarse atascado en mínimos locales.

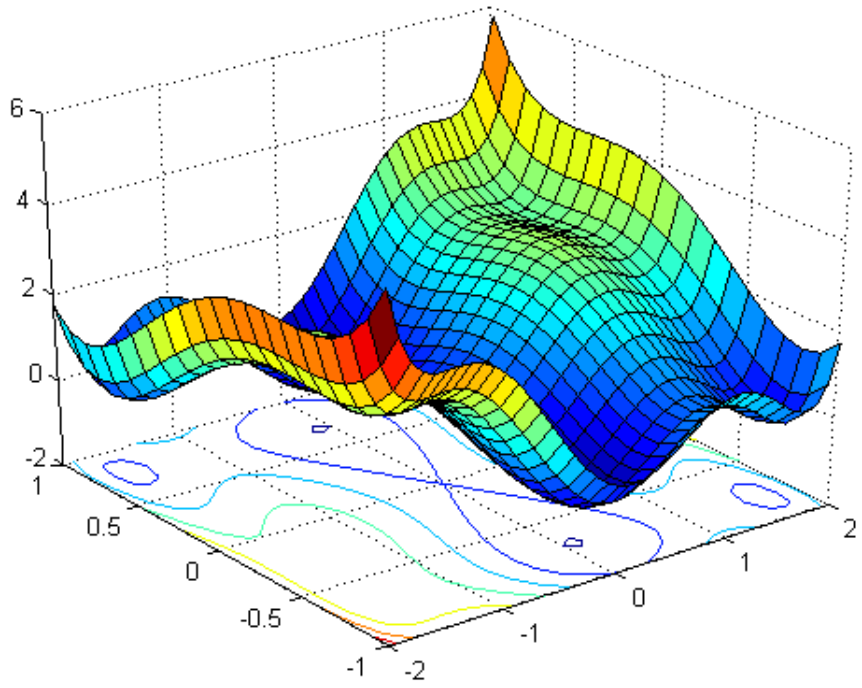

Figura 20. *Representación 2D de una función de pérdida con mínimos locales [45]*

Para minimizar la función de pérdidas de un modelo se utiliza el método llamado backpropagation o propagación hacia atrás de errores. Para ello, se calcula la derivada de la función de pérdidas con respecto a los parámetros de la red neuronal para una muestra o lote proporcionado. Esta derivada nos indica la dirección en la que nos debemos desplazar para ir hacia un mínimo de la función de pérdidas. Esta dirección se denomina el gradiente de la red neuronal.

Antes de actualizar el modelo con el gradiente, se debe multiplicar por un parámetro denominado Learning Rate o tasa de aprendizaje. Este hiperparámetro define la velocidad a la que nos desplazamos en la dirección del gradiente durante el proceso de entrenamiento. De esta forma, con un learning rate bajo podemos tener un proceso de entrenamiento muy lento y/o acabar en un mínimo local no óptimo. Por el contrario, si tenemos un learning rate alto podemos encontrarnos con un sistema inestable, oscilando constantemente en torno al mínimo sin llegar nunca a alcanzarlo. Estos dos fenómenos se pueden observar en la figura 21.

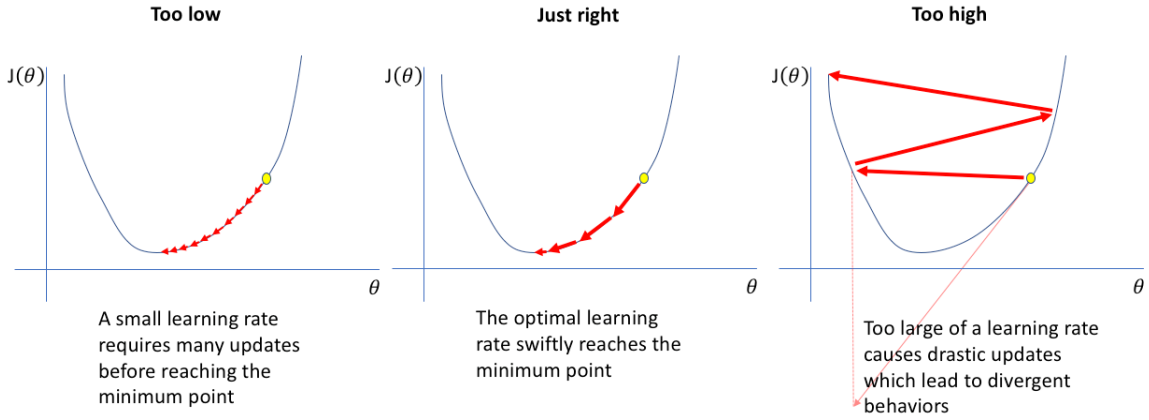

Figura 21. *Comparativa de learning rates. De izquierda a derecha: muy bajo, bueno y demasiado alto [46].*

Para evitar este problema, se puede modificar durante el proceso de entrenamiento el learning rate, el más simple siendo reducir gradualmente el valor para en un principio llegar a la zona general del mínimo y más tarde aproximarse a él lentamente. Este proceso se denomina adaptive learning rate o tasa de aprendizaje adaptativa y se modifica definiendo un learning rate scheduler o un planificador de la tasa de aprendizaje que será el que defina el valor que toma el parámetro en cada etapa del entrenamiento. Existen muchos métodos para modificar el learning rate aunque los principales serían los siguientes [46]:

- **Learning rate annealing:** este es el método mencionado anteriormente que consiste en empezar con un valor elevado y reducirlo gradualmente durante el proceso de entrenamiento. Con esta técnica se permite en un principio viajar rápidamente desde los parámetros iniciales a un rango bueno donde se encuentre el mínimo para posteriormente explorar la región más profunda y compleja de la función de pérdidas. El método más popular dentro de esta categoría es el step decay (figura 22) con el que se reduce en un porcentaje cada n épocas de entrenamiento.

- **Cyclical learning rates:** en el paper Cyclical Learning Rates for Training Neural Networks [47] de Leslie Smith se propone el uso de un planificador que actualice el parámetro con una regla triangular entre un máximo y un mínimo. Combina además

este método con una degradación fija del parámetro o una degradación exponencial. Los distintos métodos se pueden ver en la figura 23. La lógica según Smith de tener una tasa de aprendizaje cíclica es que al aumentar este parámetro se consigue, sacrificando resultados a corto plazo, un beneficio a largo plazo al converger en una solución más óptima.

- **Stochastic Gradient Descent con Warm Restarts (SGDR):** similar al método cíclico, el método SGDR combina un agresivo planificador annealing mediante una función coseno con restarts periódicos a la tasa de aprendizaje original (ver figura 24). Este método permite gracias a los restarts periódicos escapar de mínimos locales para continuar explorando la función de pérdidas.

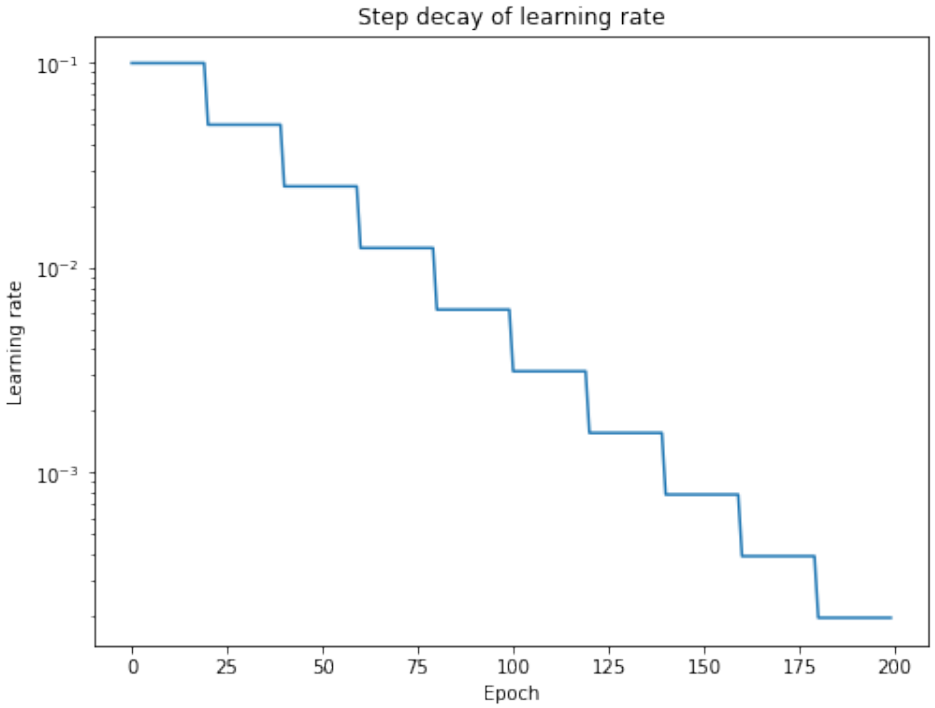

Figura 22. *Learning Rate Annealing: Step Decay [46].*

Los optimizadores son unas herramientas para resolver todos estos problemas. La mayoría de los optimizadores calculan el learning rate automáticamente y aplican el gradiente al modelo haciendo que este aprenda. Un buen optimizador es aquel que entrena rápido y a la vez evita que el modelo se quede atascado en un mínimo local.

Existen muchos optimizadores con distintas características por lo que se va a describir los más importantes para posteriormente explicar la elección del aplicado en el modelo.

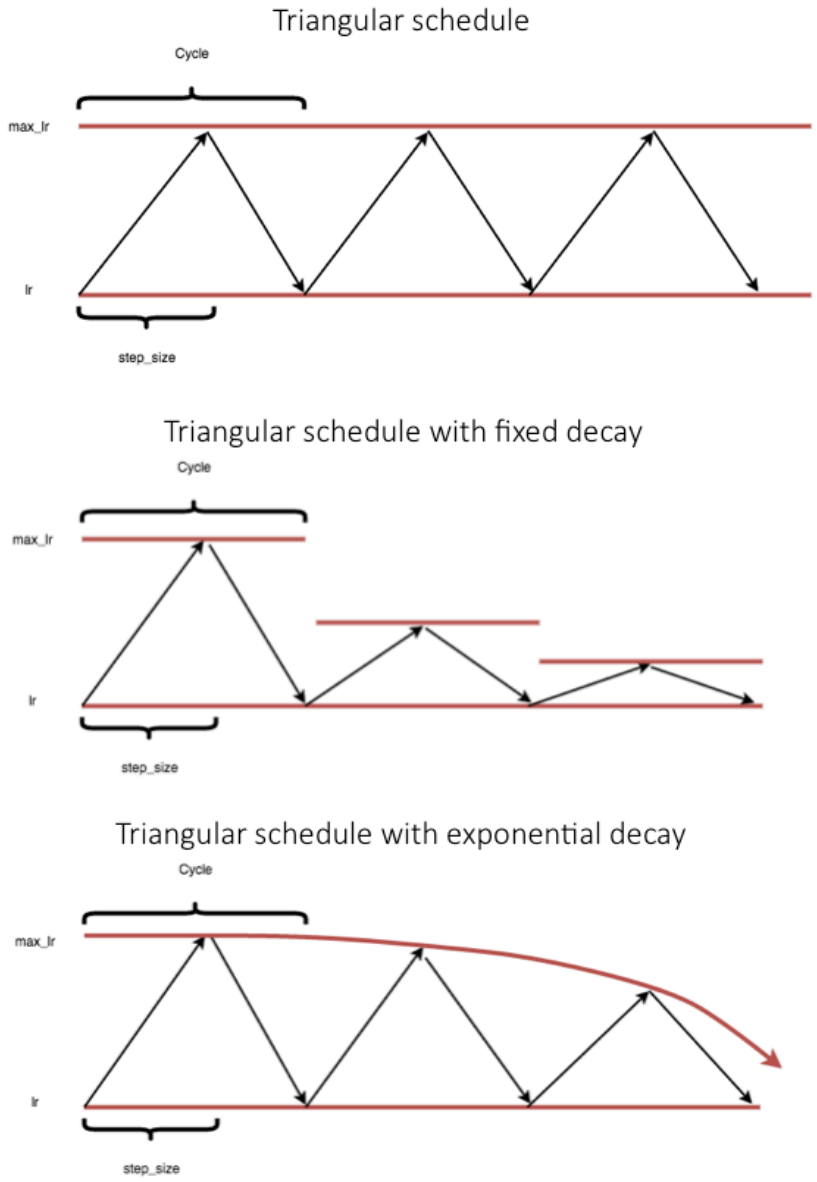

Figura 23. *Cyclical learning rates. De arriba a abajo: planificador triangular, planificador triangular con degradación fija y planificador triangular con degradación exponencial [46].*

3.4.4.1. Stochastic Gradient Descent o SGD

Stochastic Gradient Descent es uno de los optimizadores más simples al utilizar solo una tasa de aprendizaje estática durante todo el proceso de entrenamiento. El tener la tasa constante no implica que se actualicen los pesos de forma igual durante todas las etapas ya que, como se ha mencionado anteriormente, el gradiente se multiplica por la tasa de aprendizaje antes de actualizar el modelo y este gradiente dependerá de la pendiente en ese punto de la función de pérdidas. A medida que se acerque a un punto óptimo o subóptimo, los gradientes se harán cada vez más pequeños.

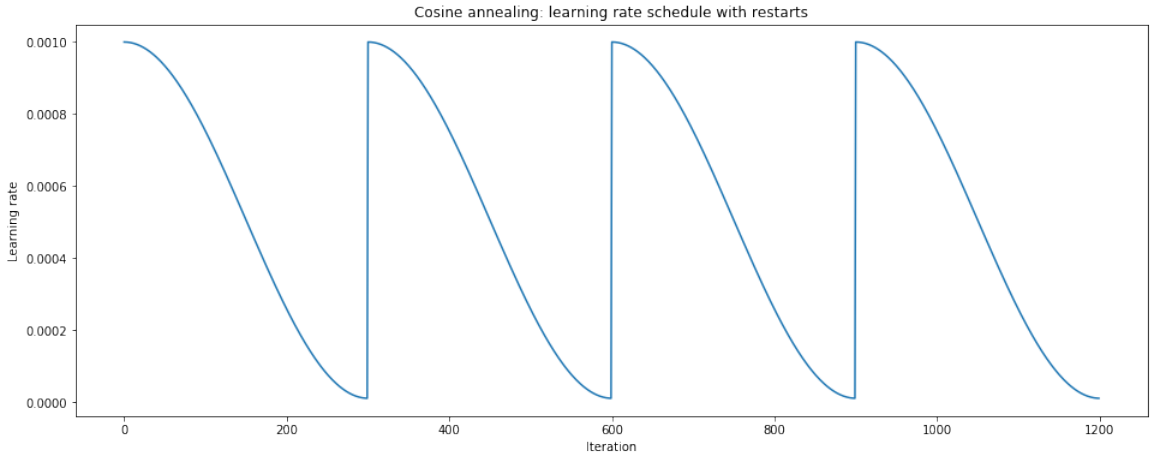

Figura 24. *Stochastic Gradient Descent con Warm Restarts con la función coseno [46].*

3.4.4.2. AdaGrad

AdaGrad o Adaptative Gradients [48] es muy similar a SGD. La diferencia clave es el uso de gradientes adaptativos, teniendo una tasa de aprendizaje distinta para cada parámetro de la red neuronal. Como no todos los parámetros tienen la misma importancia a la hora de resolver el problema, esta técnica permite incluir esta lógica. El mecanismo para determinar la tasa de aprendizaje para cada parámetro depende de la frecuencia con la que han sido actualizados. Usar learning rates pasados para contribuir a determinar el actual se denomina momento. De esta forma, parámetros que son actualizados frecuentemente debe ser entrenados más lentamente ya que puede ser un indicador de oscilación en torno a un mínimo y por lo tanto se le asignará un learning rate más bajo. Por el contrario, si un parámetro apenas se actualiza se debe aumentar su learning rate en un intento de hacerlo más efectivo, a pesar del riesgo de salir de la región de interés ya que el parámetro carece de impacto en ese momento.

3.4.4.3. RMSProp

RMSProp o Root Mean Square Propagation [49] surgió a raíz de resolver el problema que tiene AdaGrad con la excesiva reducción de sus tasas de aprendizaje tras unos pocos batches de entrenamiento lo cual acaba con un proceso de aprendizaje muy lento. RMSProp lo resuelve degradando la tasa de aprendizaje exponencialmente haciendo que sea más volátil. Este método también usa momento como AdaGrad para actualizar los learning rates lo cual permite salir de mínimos locales cuando la tasa de aprendizaje ha sido grande en el pasado cercano. De esta forma el optimizador es más robusto para evitar atascarse en mínimos locales cercanos al punto de partida inicial.

3.4.4.4. Adadelta

Adadelta surgió al mismo tiempo que RMSProp con el mismo objetivo de resolver los problemas de AdaGRad. Lo resuelve definiendo una ventana de los gradientes que se utilizará para calcular el momento del learning rate. De esta forma se limita la gran reducción que sufre AdaGrad al acumular el momento desde el inicio del entrenamiento.

3.4.4.5. Adam

Adam o Adaptive Moment Estimation [51] utiliza momento de pasados learning rates como AdaGrad, Adadelta y RMSProp para actualizar los learning rates actuales. Además de eso, también utiliza gradientes pasados para acelerar el aprendizaje. El momento juega por tanto un papel más importante y permite mantener aún mejor un learning rate alto a lo largo de las primeras etapas del aprendizaje.

3.4.4.6. Elección de optimizador

La tarea de elegir el optimizador más adecuado para un problema es concreto de forma teórica es muy complicado ya que su desempeño depende en gran medida de la forma de la función de pérdidas.

Se han realizado diversos experimentos a lo largo de la literatura para determinar cuáles desempeñan mejor en un gran rango de problemas y de esta forma poder elegir el que mejores resultados aporte para en un primer momento resolver el problema. Más adelante, una vez el modelo esté construido y se haya obtenido resultados aceptables se puede experimentar cambiando el optimizador para acelerar la convergencia en una solución durante el proceso de optimización de hiperparámetros.

De todas las pruebas se ha encontrado que el optimizador Adam es el que mejor se desenvuelve en la mayoría de escenarios [52] y es por eso por lo que actualmente es el más usado. Se caracteriza por ser [53]:

- Sencillo de implementar
- Eficiente computacionalmente
- Requiere poca memoria
- Desempeña bien es problemas con gradiente ruidoso o disperso
- Sus hiperparámetros son intuitivos y requieren poca optimización

Es verdad que en algunas situaciones el SGD puede desempeñar mejor que Adam pero requiere de una mejor optimización de hiperparámetros para llegar a ese resultado. Para definir optimizador en pytorch se puede usar la librería torch.optim [54] donde podemos encontrar la clase Adam. Le pasamos los parámetros del modelo y solo modificaremos el hiperparámetro learning rate, dejando el resto en su configuración de serie. La implementación se puede observar en el extracto de código 3.3.

```
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

Extracto de código 3.3. *Configuración del optimizador.*

3.4.5. Métricas y función de pérdidas

A la hora de evaluar como de bien un algoritmo ha realizado una tarea se debe definir una métrica para cuantificarlo matemáticamente. En el caso del problema que este trabajo quiere resolver, debemos resolver un problema de clasificación y por lo tanto debemos evaluar como de bien ha realizado esta tarea en concreto. Una vez se haya estudiado todas las métricas que nos transmiten información relevante sobre las estimaciones que realiza el modelo, se debe elegir una que represente la función de pérdidas. Esta función será la que se minimice durante el proceso de entrenamiento y por lo tanto no debe solo representar como de bien ha realizado la clasificación, sino también su confianza tanto al acertar como al fallar.

3.4.5.1. Accuracy, confusion matrix, precision y recall

Accuracy, confusion matrix, precision y recall (en español exactitud, matriz de confusión, precisión y exhaustividad) son las métricas más sencillas y fáciles de interpretar y por lo tanto nos aportarán información relevante del desempeño del modelo. Vamos a analizarlas individualmente para entender sus puntos fuertes y sus deficiencias.

Accuracy o exactitud

Accuracy o exactitud es el número total de etiquetas clasificadas correctamente entre el número total de predicciones (ver ecuación 1). Esta métrica es muy sencilla, pero le falta trasmitir información importante sobre su desempeño en cada categoría. Además, si las clases están desbalanceadas es posible que el valor de la precisión venga de predecir que todas las muestras son de la misma clase y por lo tanto pensar que el modelo lo está teniendo buenos resultados.

$$Exactitud = \frac{\text{Número de etiquetas correctas}}{\text{Número total de predicciones}} \tag{1}$$

Confusion matrix o matriz de confusión

Para compensar estas deficiencias, se creó la confusion matrix o matriz de confusión. En ella se representa las etiquetas reales en uno de los ejes y en el otro las etiquetas con las que se ha clasificado cada muestra. De esta forma podemos observar cuantas muestras han sido bien clasificadas por cada categoría y de las que han sido mal clasificadas con que categorías se confunde. Cuando se habla de clasificaciones binarías, como es el caso del problema tratado en este trabajo, se usan los términos falsos negativos y falsos positivos (figura 25). En este documento se hablará de falsos positivos cuando la muestra ha sido clasificada como la acción a identificar cuando realmente era la categoría normal y falsos negativos cuando la etiqueta real era la acción pero se ha clasificado como normal. Es muy importante tener en cuenta estas distinciones ya que no necesariamente es igual de grave tener falsos positivos frente a falsos negativos. En el contexto de este trabajo, se prefiere tener falsos positivos frente a falsos negativos ya que es más importante no fallar al detectar la acción.

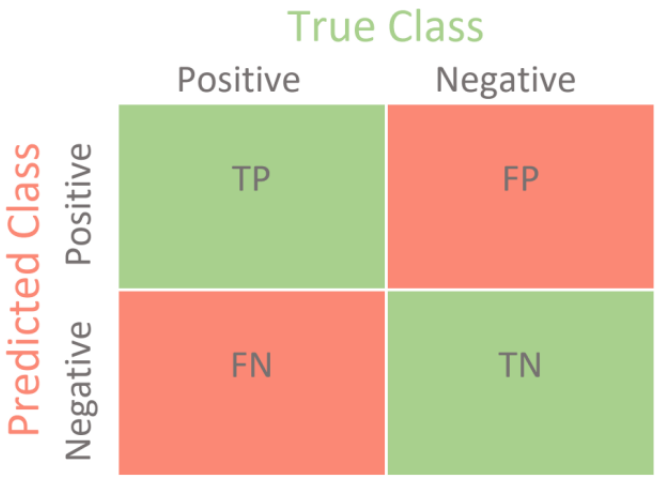

Figura 25. *Matriz de confusión de un problema de clasificación binario [55].*

Precision o precisión y recall o exhaustividad

Partiendo de esta matriz, se puede calcular la precision o precisión y la recall o exhaustividad. Estas dos métricas nos permiten evaluar dos proporciones clave de la matriz. La precisión se define como el cociente entre los verdaderos positivos o true positives (TP) y las muestras clasificadas en esa categoría que es la suma de los verdaderos positivos y los falsos negativos (ecuación 2). Este parámetro nos muestra el desempeño de la clasificación cuando estima que es la acción. Cuanto más próximo a 1 mejor.

$$\text{Precisión} = \frac{TP}{TP + FP} = \frac{TP}{\text{Todas las muestras clasificadas como positivas}} \tag{2}$$

La exhaustividad se define como el cociente de los verdaderos positivos entre todas las muestras de esa categoría o la suma de los falsos negativos y los verdaderos positivos (ecuación 3). Esta métrica nos indica cuantas de las muestras positivas ha clasificado correctamente frente al número total de muestras positivas reales. Cuanto más próximo a 1 mejor.

$$\text{Exhaustividad} \ = \ \frac{TP}{TP + FN} \ = \ \frac{TP}{\text{Todas las muestras de esa categoría}} \tag{3}$$

Todas estas métricas tienen el inconveniente de no tener en cuenta la confianza de las predicciones del modelo. No es lo mismo que las probabilidades de los fallos estén cercanas al 50 % frente a que sea el 90 %. Las próximas métricas se han diseñado para introducir esta variable en el cálculo.

3.4.5.2. Negative Log Likelihood Loss

Negative Log Likelihood Loss o NLLL se aplica solamente si el modelo tiene la función Softmax como función de activación de salida. Softmax es una función que calcula el logaritmo de la función exponencial normalizada de cada unidad de la capa (ecuación 4), por lo que permite su uso en problemas de clasificación con múltiples categorías.

$$S(f_{y_i}) \ = \ \frac{e^{f_{y_i}}}{\sum_{i=j} e^{f_j}} \tag{4}$$

Esta función toma un vector de tamaño N y lo modifica para que los valores estén acotados entre 0 y 1. Posteriormente normaliza los valores para que la suma de todos los elementos del vector sea igual a 1. NLLL aplica por tanto el logaritmo a los elementos y al estar acotados entre 0 y 1, el resultado es negativo y por tanto el NLLL lo multiplica por -1 para obtener valores positivos en la función de pérdidas. En la ecuación 5 se puede ver la combinación de la capa Softmax y NLLL. En pytorch la función NLLL no aplica el logaritmo por lo que la última capa del modelo debe ser LogSoftmax en vez de Softmax [56].

$$loss(x, y) \ = \ -log(\frac{e^{f_{y_i}}}{\sum_{i=j} e^{f_j}}) \tag{5}$$

Minimizar esta función de activación permite no solo mejorar el número de aciertos, sino también la confianza con la que hace las predicciones al premiar el incremento de las probabilidades que acierta y premia los aciertos con altas probabilidades gracias al logaritmo.

3.4.5.3. Cross-Entropy Loss Function

Esta función de pérdidas calcula la diferencia entre dos distribuciones de probabilidad de una serie de variables aleatorias. Permite por tanto resumir la media de las diferencias entre los valores predichos y los reales. Para mejorar el modelo se intenta minimizar su valor que está acotado entre 1 y 0, siendo 0 un resultado perfecto.

En cuanto a su comportamiento, penaliza enormemente las predicciones erróneas con alta confianza, así como predicciones correctas pero con baja confianza. Entre muchas de sus variantes se encuentra la Binary Cross-Entropy o BCE (ecuación 6). Está diseñado para tareas de clasificación binaria como es el problema que se quiere resolver en este trabajo.

$$loss(x, y) \; = \; -\sum x \cdot log(y) \tag{6}$$

En la práctica, en pytorch aplicar la función de pérdidas Cross-Entropy Loss es equivalente matemáticamente a la combinación de la capa LogSoftmax y la función de pérdidas NLLL aunque conceptualmente sean problemas de minimización distintos [57].

3.4.5.4. Hinge Embedding Loss Function

Esta función de pérdidas calcula su valor comparando las predicciones y los valores reales que deben estar acotados entre 1 y -1. Este rango convierte esta función en una buena opción para los problemas de clasificación binarios.

Se caracteriza por generar un error mayor cuando el signo de la predicción con el de la etiqueta real es distinto lo que promueve acertar el signo y por lo tanto la clase de las predicciones. La fórmula de esta función se puede ver en la ecuación 7.

$$loss(x, y) = \begin{cases} x & \text{si } y = 1 \\ max\{0, \Delta - x\} & \text{si } y = -1 \end{cases} \tag{7}$$

No penaliza errar con porcentajes de confianza altos por lo que su aplicación se reduce generalmente a la comparación de similitud de dos entradas en semi-supervided learning [58].

3.4.5.5. Selección de la función de pérdidas

Entre las funciones de perdidas estudiadas, la que mejor cumple con los requisitos del problema y del modelo es la función Cross-Entropy Loss. Los motivos son los siguientes:

- Usar la última capa Softmax nos permite comprobar fácilmente las probabilidades que el modelo otorga a lada categoría. De esta forma se puede visualizar intuitivamente lo que hace el modelo.

- Puede ser aplicada tanto a problemas de clasificación binarios como a multiclase. Esto nos permite no tener que modificar la estructura del modelo en caso de añadir más categorías.

- Es una función probada extensamente en la literatura con resultados comprobados.

Como se ha mencionado anteriormente, la función Negative Log Likelihood Loss es equivalente matemáticamente pero no nos permite visualizar las probabilidades de las predicciones al requerir una estructura distinta en la salida del modelo. Es por eso por lo que se ha descartado esta opción.

3.4.6. Optimización de hiperparámetros

La optimización de hiperparámetros es una de las fases más importantes en el proceso de entrenamiento de un modelo. Si normalmente los parámetros son entrenados automáticamente, los hiperparámetros son aquello que describen o configuran el proceso de entrenamiento y por lo tanto están fijos durante el proceso. El impacto de cada uno es diferente y existen diversos procesos de optimización.

3.4.6.1. Hiperparámetros

Batch size

El batch size o tamaño del lote es uno de los hiperparámetros más importantes a la hora de optimizar un modelo por su impacto en los resultados y en el consumo de recursos. Cuanto más grande es el tamaño, más datos se tienen que cargar en memoria, tanto de la RAM como de la GPU, pero a cambio nos permite acelerar el proceso. Se ha demostrado que un número más reducido de lote permite generalizar mejor y converger más rápido hacia una solución buena, aunque no garantiza llegar a la mejor [44].

Esto se debe a que durante el entrenamiento, las muestras compiten para reducir la función de pérdidas, dando lugar a la cancelación de los gradientes individuales al calcular el gradiente total. Cada muestra quiere mover los pesos en un sentido posiblemente opuesto y esta lucha simultanea reduce la capacidad de entrenamiento. En cambio, si el entrenamiento fuese realizado en secuencia, el gradiente permitiría que se fuese convergiendo a una solución óptima al mover los pesos individualmente sin competencia.

Learning rate

El learning rate o tasa de aprendizaje determina como de rápido se recorre la superficie de la función de pérdidas durante el proceso de entrenamiento. Su efecto en el proceso de aprendizaje depende en gran medida en el optimizador elegido como se ha explicado en el apartado 3.4.4. En el caso del optimizador elegido, Adam tiene un mecanismo que adapta el learning rate de cada parámetro por separado. Esto no quiere decir que no haya un impacto en el proceso de entrenamiento ya que el optimizador parte de ese valor en el comienzo del proceso.

Resolución

La resolución de los vídeos a la que se convierten todos los vídeos antes de pasarlos al modelo tiene un impacto tanto en el resultado final como también en la carga de procesamiento. En un primer lugar, aumentar la resolución de los fotogramas aumentan la información disponible al modelo.

Existe el peligro que al reducir mucho la resolución de los vídeos se pierda la calidad suficiente para que hasta un humano reconozca la acción. Por otra parte, su disminución reduce la complejidad del modelo al tener un número menor de parámetros que entrenar y datos de menor tamaño que cargar en la GPU. Todo esto reduce por tanto la carga de computación permitiendo aumentar el tamaño de lote.

Fotogramas por segundo

Los fotogramas por segundo representan la frecuencia de información temporal que vamos a aportar al modelo. Al igual que con la resolución, usar el estándar de 30 fotogramas por segundo de la industria de las cámaras de vídeo supone un aumento de la complejidad del modelo y la memoria necesaria para entrenarlo. A diferencia del caso anterior, su reducción no implica necesariamente una perdida de información relevante ya que la velocidad de movimiento del cuerpo humano es limitada y por lo tanto se puede captar con menos fotogramas por segundo.

En el artículo Investigating the impact of frame rate towards robust human action recognition [59] se estudia el impacto de los fotogramas por segundo en la tarea del reconocimiento de imágenes. Destacan que un humano puede reconocer acciones en vídeos de 8 fotogramas por segundo y, como las cámaras de seguridad guardan los vídeos con 15 fotogramas por segundo, se puede concluir que el rango de 8 a 15 es aceptable para fijar este hiperparámetro.

Fotogramas totales

Si el anterior parámetro determinaba la frecuencia temporal, lo fotogramas totales definen la longitud total de la muestra que se va a tomar. Los vídeos que conforman la base de datos se recortaron para que contengan solo el momento en el que se comete la acción y por lo tanto la longitud total debería, teniendo en cuenta los fotogramas por segundo, abarcar hasta las muestras de más corta duración. Como los vídeos más cortos son de en torno a 2 segundos a 30 fps, si se tomasen 60 frames totales no encontraríamos ningún problema, En caso de modificar los fps, se debería modificar también este parámetro para no incurrir en ningún problema. En caso de que no hubiese suficientes fotogramas, se copiará el último hasta llegar al valor definido.

Escala de colores

Cuando queremos analizar un vídeo, debemos plantear si la información contenida en la escala de colores es relevante para el problema o podría ser suficiente trabajar en escala de grises. Trabajar a color permite transmitir información más clara sobre el borde de los objetos y distinguirlos frente a fondos que en escala de grises puede ser muy similar. El impacto sobre la carga de computación es sustancial ya que los vídeos a color están compuestos por fotogramas con 3 capas distintas. Cada una corresponde a uno de los tres colores de la gama RGB siendo estos rojo, verde y azul. Al reducir estas 3 capas a 1 al pasar a escala de grises, los parámetros necesarios durante el entrenamiento se reducen sustancialmente liberando memoria de la GPU. Como no es necesariamente evidente responder a esta pregunta de forma teórica, se tratará como un hiperparámetro la elección de usar 1 canal o 3 según si se usa la escala de grises o RGB respectivamente.

3.4.6.2. Métodos de optimización

La correcta elección de hiperparámetros afecta en gran medida los resultados obtenidos con el modelo. Como los hiperparámetros definen como es la estructura del modelo, no pueden elegirse de forma teórica ya que estos modificarán la forma de la función de pérdidas y es la tarea del proceso de entrenamiento de encontrar el nuevo mínimo. Es por eso por lo que se han desarrollado distintas técnicas para facilitar este proceso y llegar de forma constante a la mejor versión posible. Su desempeño se medirá en el conjunto de datos de validación. A continuación se explicaran los métodos más importantes [60].

Manual Search

El método manual search o búsqueda manual consiste en iterar en la elección de hiperparámetros según nuestra experiencia o juicio. Una vez se ha elegido unos valores, se entrena el modelo y se evalúa para una vez más hacer cambios. De todas las pruebas se elige la que mejor desempeño haya tenido, habiendo llegado al final del proceso de optimización.

Este método tiene la ventaja de ser eficiente en cuanto al uso de recursos si la persona que está realizando la optimización tiene experiencia con el proceso ya que estaremos reduciendo el número de pruebas a realizar.

Grid Search

Grid Search o busqueda por cuadrícula es un método de optimización que consiste en la definición de una serie de valores para cada hiperparámetro que forma una cuadrícula con la que se hará una búsqueda exhaustiva con todas sus combinaciones posibles (figura 26). La mejor configuración en el conjunto de datos de validación serán los valores optimizados. Este método requiere de una gran carga de computación ya que se van a probar todas las combinaciones posibles y por lo tanto entrenar el modelo ese mismo número de veces hasta llegar a su óptimo. Además, es posible no encontrar los puntos de interés por haber seleccionado manualmente sin ningún criterio previo valores con demasiado espacio entre ellos. Tiene la ventaja de ser un proceso muy paralelizable, acelerando la optimización si se dispone de recursos necesarios.

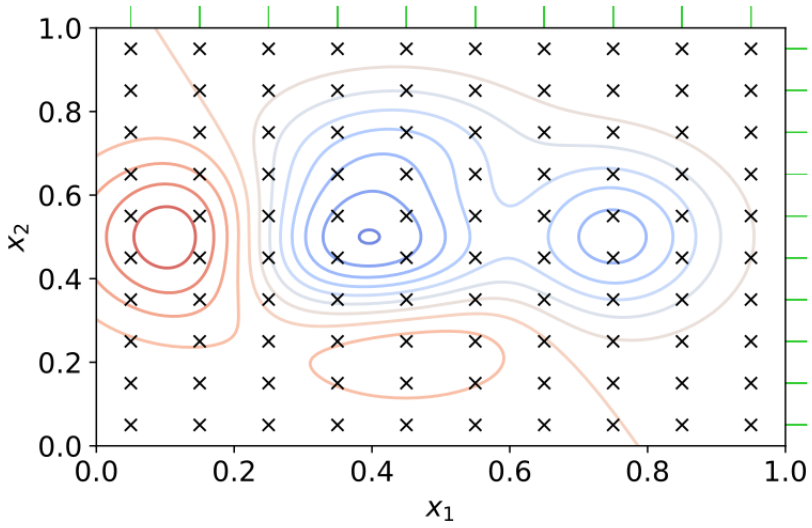

Figura 26. *Optimización de hiperparámetros: Grid Search [61].*

Random Search

Random Search o búsqueda aleatoria es similar al proceso anterior con la diferencia de que las combinaciones de valores no son definidas por el usuario. En este caso, se define un rango para cada hiperparámetro y se generan combinaciones aleatorias de valores contenidos en estos, tanto para espacios continuos como discretos (figura 27). De esta forma, la búsqueda no está sujeta al bias del usuario y permite obtener mejores resultados cuando el número de hiperparámetros es reducido. Este método es posible de paralelizar y es común su uso

combinado con grid search para refinar la búsqueda a las zonas de interés encontradas a posteriori.

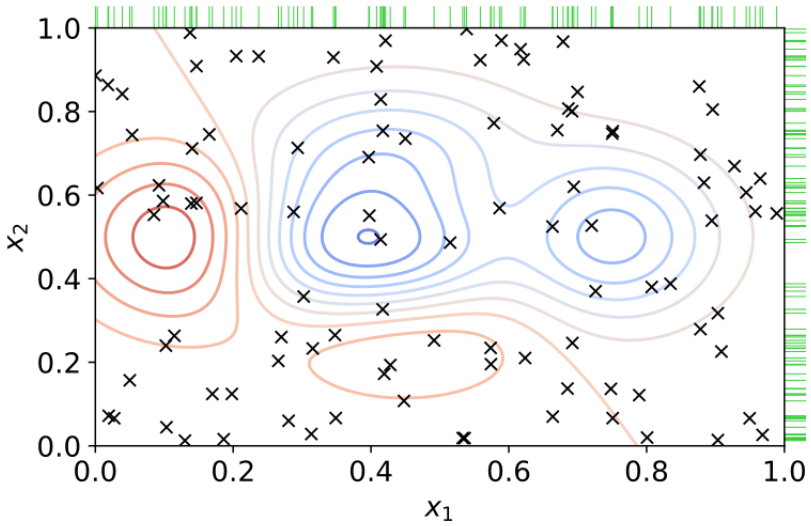

Figura 27. *Optimización de hiperparámetros: Random Search [62].*

Automated Hyperparameter Tuning

Automated Hyperparameter Tuning o optimización automática de hiperparámetros es un conjunto de métodos que realizan la operación de forma automática basándose en los resultados obtenidos tras cada iteración. Es el equivalente a aplicar métodos de machine learning para optimizar esta búsqueda. Destacan dos métodos: Bayesian Optimization y Genetic Algorithms.

Bayesian Optimization o optimización bayesiana es un método global de optimización que permite construir un modelo probabilístico de la función que relaciona los valores de los hiperparámetros con el mínimo de la función de pérdidas en el conjunto de validación. Este método ha sido demostrado que obtiene mejores resultados con menores pruebas [63] en comparación con grid search o random search por su habilidad para evaluar la calidad de un conjunto de valores antes del proceso de entrenamiento. Esto se puede observar en la figura 28, donde los putos explorados se concentran en torno al punto óptimo.

Genetic Algorithms o algoritmos genéticos son un conjunto de métodos globales de optimización que permiten, inspirados las leyes de la evolución biológica, explorar un espacio de hiperparámetros para encontrar el óptimo. Se crea una población inicial de soluciones aleatorias para posteriormente evaluarlas. Las combinaciones que mejores resultados hayan tenido se mantendrán en la muestra descartándose el resto. Finalmente, se generarán nuevas combinaciones mediante mutaciones o combinaciones de las mejores para volver a repetir

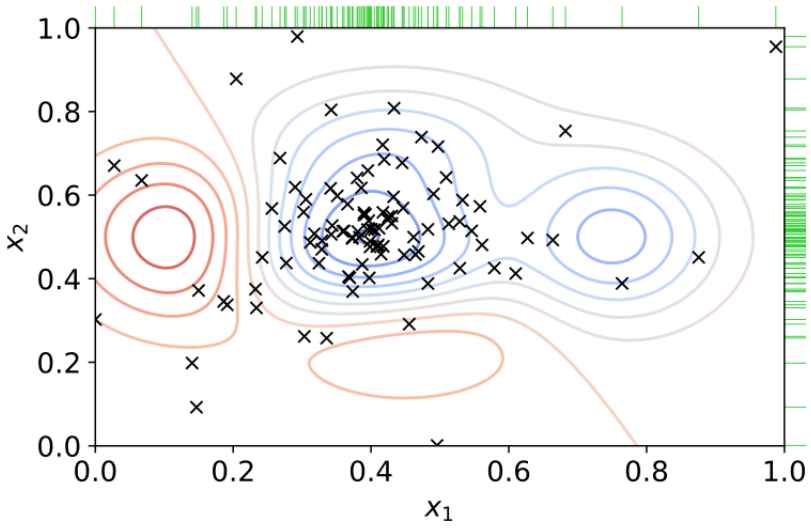

Figura 28. *Optimización de hiperparámetros: Bayesian Optimization [62].*

el proceso, generando nuevas generaciones con cada iteración. Con este método se pueden conseguir muy buenos resultados, pero puede requerir de mucha carga de computación ya que el número de combinaciones en cada población es crítico para su correcto desarrollo.

Método elegido

Como su nombre indica, estos métodos son de optimización y por lo tanto se aplicarán una vez el modelo haya conseguido comenzar a aprender a identificar las acciones. Es por esto que en un primer momento se utilizará el método manual ya que nos permite hacer una exploración reducida e iterar más rápidamente en el diseño del modelo. Además, la capacidad de entrenamiento es reducida, obligando a tenerlo en cuenta a la hora optimizar los hiperparámetros. En el caso de lograr llegar al punto donde tiene sentido realizar la optimización, se considera la mejor opción para esta aplicación el uso de la combinación de los métodos Grid Search y Random Search ya que nos permiten controlar el número de entrenamientos a realizar, así como elegir manualmente la región a explorar, reduciendo la carga de computación necesaria. En un futuro se recomienda aplicar los métodos más avanzados por sus beneficios a la hora de llegar a la combinación óptima global siempre y cuando se disponga de la suficiente capacidad de computación.

3.4.7. Visualización

Tras el entrenamiento de cada modelo, es necesario analizar el porqué de las decisiones que toma. En el mundo de la visión artificial es especialmente útil ya que nos permite identificar que partes de las imágenes o vídeos son las determinantes a la hora de realizar la clasificación. La generación de estas imágenes se denomina Visual Explanations o explicaciones visuales. De

esta forma se puede descubrir si un algoritmo realiza las predicciones a raíz de la acción en si o es un bias el que está siendo usado.

Un claro ejemplo de este fenómeno se estudió en algoritmos de clasificación de objetos donde se le pedía al algoritmo detectar pesas. Dentro de las imágenes proporcionadas en el entrenamiento se podía encontrar imágenes donde un brazo está sujetando la pesa. Al realizar este análisis se descubrió que la existencia del brazo era usado para clasificar este objeto, siendo claramente un comportamiento no deseado.

En un caso más relacionado con el problema que se pretende resolver, en el paper "Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition"[34] se trata el problema del uso, por parte de los modelos, del entorno para reconocer acciones (figura 29). Al tener una base de datos pequeña, este fenómeno se tiene que tener muy en cuenta a la hora de analizar los resultados.

Figura 29. *Visualización de la detección de acciones por el entorno donde se desarrollan y el resultado una vez se ha eliminado el bias [34].*

Estas visualizaciones se pueden realizar a través de varios métodos. El primero y más simple seria la obtención de las activaciones de las neuronas en las capas intermedias de las redes convolucionales. Esto nos aportaría la información de que zonas de las imágenes activan los distintos filtros. Las primeras capas son activadas por patrones simples mientras que las últimas combinan los patrones simples para llegar a detectar objetos o acciones complejos. Esta técnica está más desarrollada en el mundo de la visión artificial con imágenes, siendo fácil representar la detección en la imagen original. En este proyecto, se trabaja con convolucionales de tres dimensiones para analizar no solo la información espacial sino también la temporal, complicando su representación.

La segunda técnica sería el uso de los gradientes para calcular de entre todos los filtros aplicados cual es el que toma mayor peso para realizar la clasificación. Tiene la ventaja que no solo nos informa de las zonas que activan las diferentes capas, sino también de si esa

activación se usa para realizar las predicciones. Este método se denomina Grad-CAM [65] y Grad-CAM++ [66] y, a pesar de estar desarrollado para imágenes, puede ser aplicado tras adaptarlo a convolucionales de tres dimensiones.

Para simplificar el problema de visualización, se ha decidido aplicar el método de las activaciones de las capas por su simplicidad y fácil aplicación. En un futuro se puede plantear modificar el Grad-CAM++ para su aplicación en vídeos, al aportar información más representativa de las zonas que influyen la decisión final del modelo.

3.4.8. Guardado de resultados

Finalmente, la última fase del entrenamiento de un modelo es el guardado de los resultados. Durante las distintas fases del entrenamiento debemos guardar el avance de las métricas para evaluar cómo se desarrolla y posteriormente compararlas.

Para ello se va a utilizar la herramienta TensorBoard [67] desarrollada por TensorFlow aunque es compatible con PyTorch. Con ella podemos registrar fácilmente las métricas durante cada entrenamiento, la estructura del modelo, los hiperparámetros utilizados e imágenes de interés (figura 30). Posteriormente, se puede realizar la comparación de los resultados obtenidos para detectar cual ha sido el modelo con el mejor desempeño.

Figura 30. *Ejemplo de un dashboard de TensorFlow [67].*

Además, se deben guardar los parámetros entrenados para posteriormente usar el modelo para realizar predicciones. El guardado debe hacerse en el momento adecuado del entrenamiento para evitar el fenómeno de Overfitting. Este fenómeno se da cuando el algoritmo es capaz de seguir reduciendo el valor de la función de pérdidas en el conjunto de datos de entrenamiento,

pero su desempeño en los conjuntos de validación y prueba empeoran. Esto es debido a la capacidad que tiene el algoritmo de aprenderse los datos de entrenamiento perdiendo capacidad de generalización con datos con los que no ha sido entrenado. Para evitar este fenómeno, se puede guardar los pesos cada vez que se llegue a un mínimo en la función de pérdidas de validación y si no esta no se recupera parar el entrenamiento en su conjunto. Esta técnica se denomina Early Stopping.

De cara a su implementación en PyTorch, se guardarán solo los pesos con la función torch.save [68] con model.state_dict() y no la clase del modelo para evitar posibles incompatibilidades con futuras versiones como describe la documentación. Para conservar la estructura del modelo se guardará también una copia del archivo y facilitar así su reproducción. Todos estos archivos se almacenarán en la carpeta creada por TensorBoard para el entrenamiento que se está desarrollando (extracto de código 3.4), con la fecha y el nombre del dispositivo que lo realiza, para así fácilmente obtener los parámetros y el modelo de una configuración concreta.

```
torch.save(model.state_dict(), os.path.join(writer.log_dir, "cnn3d.pt"))

out_filename = os.path.join(writer.log_dir, "cnn_3d_model_structure.py")

with open(__file__, 'r') as f:
with open(out_filename, 'w') as out:
    for line in f.readlines():
        print(line, end='', file=out)
    out.close()
f.close()
```

Extracto de código 3.4. *Guardado de los pesos entrenados y una copia del código.*

3.5. Algoritmos

Una vez hemos diseñado un pipeline que es capaz de proporcionar los datos y gestionar el proceso de entrenamiento, debemos enfrentarnos al diseño de un algoritmo capaz de resolver el problema de detección de acciones en vídeos de cámaras de seguridad. En un primer lugar se analizará los problemas intrínsecos que presenta este objetivo y finalmente las distintas propuestas de modelos.

3.5.1. Problemas intrínsecos del objetivo

Al resolver problemas con modelos de inteligencia artificial, se tiene que tener mucho cuidado al analizar las dificultades que los datos pueden presentar. Si se consideran todas las fuentes de bias, es posible dirigir el diseño del modelo para evitar estos problemas de raíz. Se han identificado los siguientes desafíos.

3.5.1.1. Múltiples categorías en el mismo vídeo

La primera dificultad proviene de la aparición de múltiples personas en el mismo fotograma. Debido a este hecho, nos encontramos con el dilema de que en vídeos que representan la acción a detectar también contienen a personas que no la están cometiendo. Esto aumenta sustancialmente la dificultad del problema, al probablemente activar simultáneamente las activaciones de ambas categorías.

3.5.1.2. Superposición de sujetos

La segunda se deriva también de la aparición de múltiples personas y de escenarios complejos. Se trata de la superposición de unas personas sobre otras y de objetos que tapan parcialmente el cuerpo. Esto puede confundir al algoritmo al no saber distinguir entre personas y dificultar la identificación de la acción. Además, presenta la aparición de bias debido al número reducido de vídeos que limitan la capacidad de representar todo tipo de situaciones complejas en la base de datos. El algoritmo podría relacionar situaciones circunstanciales como el número de personas o la superposición de sujetos como indicadores para realizar la clasificación por la falta de vídeos representativos en la base de datos de entrenamiento.

3.5.1.3. Entorno no identificativo

Como se ha mencionado anteriormente, este problema no contiene la .ªyuda"de un fondo identificativo que asista en la decisión de clasificación, como puede ser en otras tareas de clasificación de acciones. Esto se refleja en el artículo "Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition"[34]. Esto aumenta significativamente la dificultad frente a modelos que se pueden encontrar en el estado del arte de bases de datos que si lo presentan.

3.5.2. Modelo

Una vez hemos identificado las dificultades, se han propuesto varios modelos o combinaciones de ellos para superarlas. Además de estas consideraciones, se ha tenido en cuenta las limitaciones de la capacidad de computación y de la base de datos generada. Se comentarán

primero las posibles soluciones para paliar los problemas mencionados y posteriormente los modelos de clasificación.

3.5.2.1. Estrategias de eliminación de problemas intrínsecos

El principal problema intrínseco a resolver que podría presentar las mayores dificultades es el de la aparición de múltiples personas con diferentes categorías. Para resolverlo, se proponen dos métodos distintos: uso del filtro de movimiento descrito en el apartado 3.3.2.3 y YOLO v4 con Deep Sort.

Filtro de movimiento

El filtro de movimiento permite aislar las entidades que se mueven permitiendo recortar a las personas individualmente. Esta técnica se consigue aplicando al resultado mostrado en la figura 19 una función delta que destaca el movimiento para posteriormente producir la segmentación de cada persona como se puede observar en la figura 31 gracias a la librería OpenCV. Permite por tanto detectar a individuos, aislarlos del entorno y recortar solo a la persona para analizarlo con el modelo. Al ser muy simple, tiene dificultades en los casos con múltiples personas y su superposición, así como asignar a las personas un código de identificación único durante todo el vídeo. Además, no distingue entre objetos que se mueven y personas lo que aumenta las posibilidades de error.

Figura 31. *Segmentación de personas gracias al detector de movimiento [69].*

YOLO v4 con Deep Sort

Esta combinación de modelos de inteligencia artificial ya entrenados [70] tiene como objetivo detectar personas en cada fotograma gracias al algoritmo YOLO o You Only Look Once y asignarle el mismo número de identificación a cada persona a lo largo del vídeo mediante el modelo Deep Sort. El análisis se produce por tanto dividiendo en fotogramas el vídeo, detectando a las personas y posteriormente relacionar la información temporal mediante la asignación de identificadores únicos a lo largo del vídeo. Permite identificar únicamente a las

personas y es capaz de discernir entre ellas cuando están superpuestas. Esto nos permitiría en teoría obtener un recorte individual de las personas enfocando la atención únicamente a cada uno. Tras recortarlas, se pasa los recortes individuales al algoritmo con la etiqueta real de cada persona, eliminando el problema de la aparición simultánea de personas que realizan la acción y otras que no.

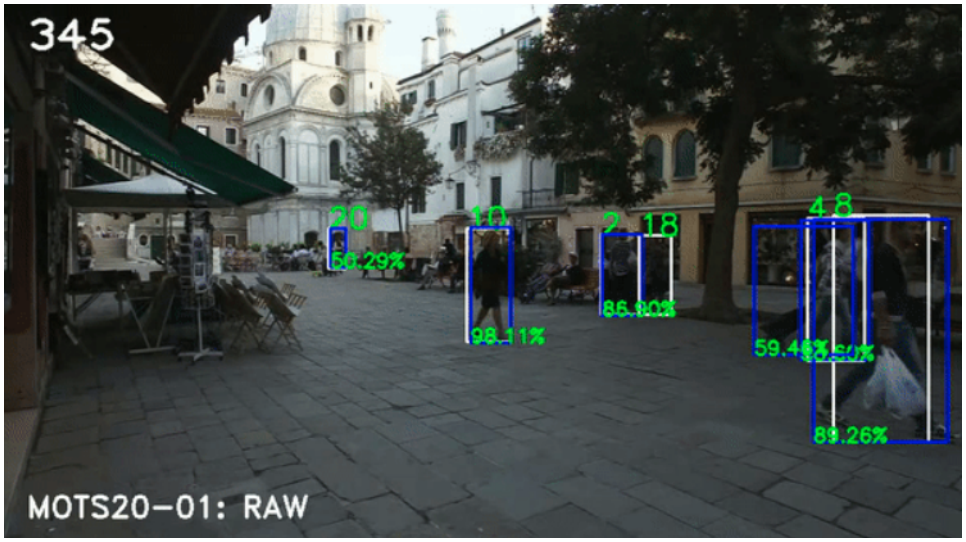

Figura 32. *Ejemplo de uso del algoritmo YOLO v4 y Deep Sort [70].*

3.5.2.2. Algoritmo principal de clasificación

El objetivo de este trabajo es desarrollar un algoritmo basándose en la capa 3DCNN, la cual consiste en operaciones convolucionales de dos dimensiones espaciales y una dimensión temporal. Existen muchos modelos que utilizan esta estructura con complejidad variable pero debido al gran número de parámetros a los que se llega y la limitada capacidad de computación, se ha decidido explorar la arquitectura más simple que la aplica como prueba de concepto.

Esta arquitectura está compuesta por 5 capas convolucionales en serie seguidas de 2 capas fully connected o conectadas completamente con la operación softmax al final para calcular las probabilidades asignadas para cada clase (figura 33). Esta arquitectura se describe en el artículo "Multi-cue based 3D residual network for action recognition"[71] como modelo base de esta capa.

Esta arquitectura se ha modificado para resolver varios de los problemas clásicos que los modelos de Deep Learning tienen:

- **Overfitting:** como ya se ha descrito anteriormente, este fenómeno se da cuando el algoritmo se aprende los datos de entrenamiento, reduciendo el valor de la función de

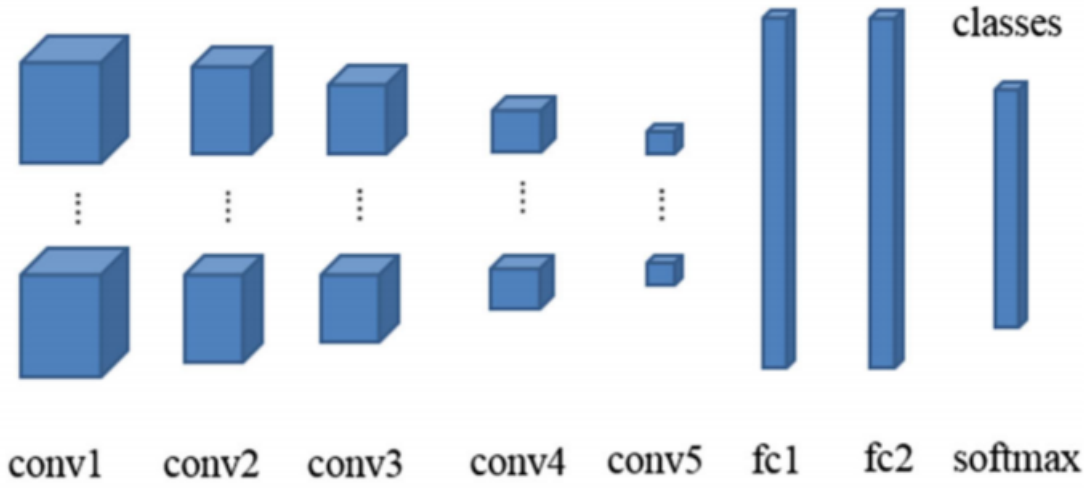

Figura 33. *Capas del modelo 3DCNN [71].*

pérdidas en esos datos, pero aumentando en el resto al no ser capaz de generalizar. Esto también se describe como un aumento de la varianza del modelo en la descomposición bias-varianza.

- **Underfitting:** al contrario del caso anterior, este fenómeno se da cuando el algoritmo no es capaz de aprender las características importantes. Esto también se describe como un aumento del bias del modelo en la descomposición bias-varianza.
- **Carga computacional:** los modelos de Deep Learning tienden a utilizar muchos recursos al tener que entrenar un número elevado de parámetros entrenables.
- **Internal covariate shift:** se describe como los cambios en la distribución de las activaciones en la red causado por los cambios en los parámetros durante el proceso de entrenamiento. En otras palabras, en la red neuronal los datos de entrada de una capa son los de la salida de la anterior y cuando estos parámetros cambian, cambia también la distribución de las entradas de las siguientes capas. Estos cambios son problemáticos durante el entrenamiento debido a la necesidad de reducir la tasa de entrenamiento e inicializar los parámetros con cuidado.

Las capas utilizadas para combatir estos problemas se describirán a continuación.

Max pooling

Max pooling es un proceso de discretización cuyo objetivo es reducir el tamaño de su entrada al transformar la ventana de la operación en el valor máximo contenido en ella. Así reducimos el número de parámetros necesarios conservando las características de la región y

reducimos el underfitting al proveer una forma abstracta de las subregiones a las que se les ha aplicado el filtro. Un ejemplo en dos dimensiones se puede observar en la figura 34.

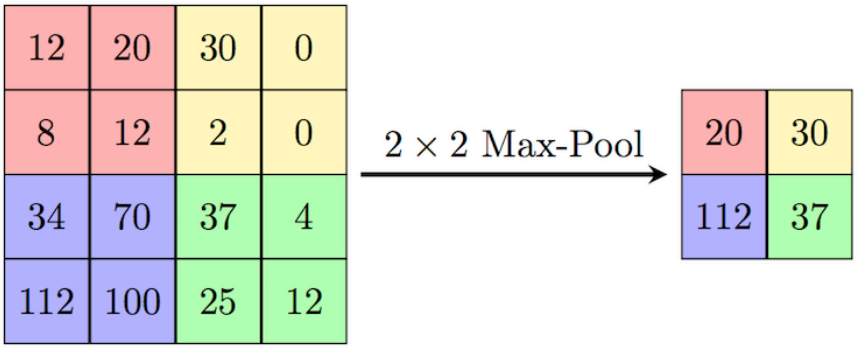

Figura 34. *Ejemplo de la operación Maxpooling en 2D [72].*

Dropout

Esta técnica de regularización tiene como objetivo prevenir el overfitting. Se consigue ignorando un porcentaje de unidades durante el proceso de entrenamiento elegidas de forma aleatoria en cada época (figura 35). Ignorándolas se permite no modificar su valor durante el proceso y eliminar co-dependencias entre los parámetros.

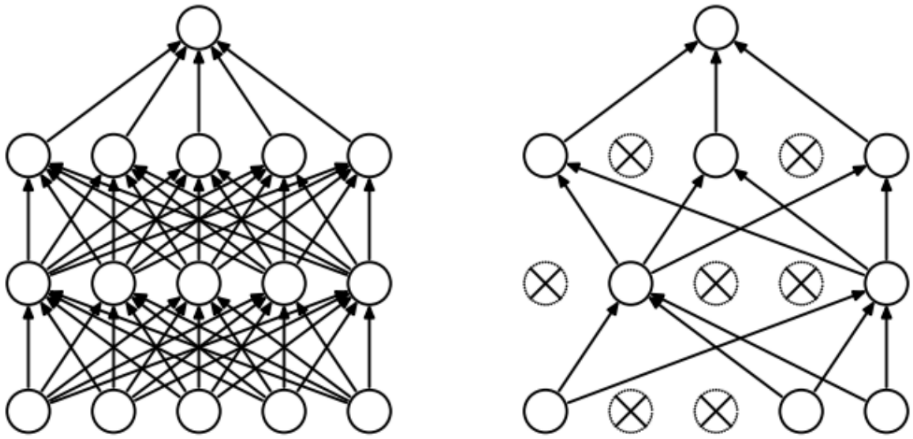

Figura 35. *Ejemplo de la operación Dropout en una red neuronal [73].*

Batch normalization

Esta capa se diseñó en el artículo "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"[74] para atajar el problema del internal covariate shift. Permite acelerar el entrenamiento reduciendo el número de épocas y suavizar la función de pérdidas.

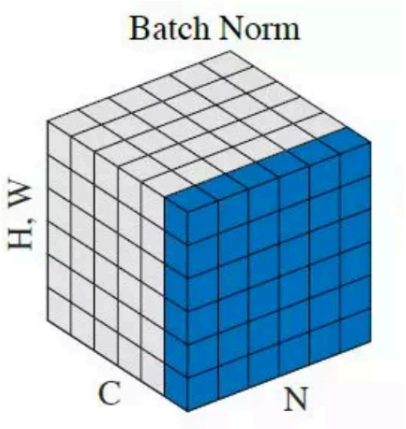

Figura 36. *Ejemplo de la operación Batch Normalization [74].*

Tras aplicar estas técnicas, las capas finales del modelo y sus parámetros se pueden observar en el extracto de código 3.5.

```
self.conv1 = nn.Conv3d(in_channels=channel, out_channels=64,
                       kernel_size=(3, 3, 3), padding=2)
self.bn1 = nn.BatchNorm3d(64)
self.relu1 = torch.nn.ReLU()
self.mp1 = nn.MaxPool3d(kernel_size=(2, 2, 2))
self.drop1 = nn.Dropout3d(p=0.15)

self.conv2 = nn.Conv3d(64, 32, kernel_size=(3, 3, 3), padding=2)
self.bn2 = nn.BatchNorm3d(32)
self.relu2 = torch.nn.ReLU()
self.mp2 = nn.MaxPool3d(kernel_size=(2, 2, 2))
self.drop2 = nn.Dropout3d(p=0.15)

self.conv3 = nn.Conv3d(32, 32, kernel_size=(3, 3, 3), padding=2)
self.bn3 = nn.BatchNorm3d(32)
self.relu3 = torch.nn.ReLU()
self.mp3 = nn.MaxPool3d(kernel_size=(2, 2, 2))
self.drop3 = nn.Dropout3d(p=0.15)

self.conv4 = nn.Conv3d(32, 16, kernel_size=(3, 3, 3), padding=2)
self.bn4 = nn.BatchNorm3d(16)
self.relu4 = torch.nn.ReLU()
self.mp4 = nn.MaxPool3d(kernel_size=(2, 2, 2))
self.drop4 = nn.Dropout3d(p=0.15)

self.conv5 = nn.Conv3d(16, 8, kernel_size=(3, 3, 3), padding=2)
self.bn5 = nn.BatchNorm3d(8)
self.relu5 = torch.nn.ReLU()
self.mp5 = nn.MaxPool3d(kernel_size=(2, 2, 2))
self.drop5 = nn.Dropout3d(p=0.15)

self.flat1 = nn.Flatten()
self.l_1 = nn.Linear(216, 16)
self.bn6 = nn.BatchNorm1d(16)
self.relu6 = torch.nn.ReLU()
self.drop6 = nn.Dropout(p=0.15)

self.l_2 = nn.Linear(16, nclass)
self.softmax  = torch.nn.Softmax()
```

Extracto de código 3.5. *Inicialización de las capas y sus parámetros.*

Capítulo 4

Análisis de resultados y conclusiones

ESTE capítulo pretende exponer los resultados obtenidos tras la optimización de los procesos y el modelo así como su interpretación. Se transmitirán una serie de sugerencias para avanzar esta línea de investigación en un futuro, así como la de otros proyectos de esta índole.

4.1. Resultados

El desarrollo del modelo constaba de tres partes: construcción de la base de datos, desarrollo del pipeline y diseño y entrenamiento del modelo. Se analizará por tanto los resultados en esos objetivos.

4.1.1. Base de datos

La construcción de la base de datos fue compleja debido a la escasez de vídeos que cumpliesen las características. Aun así, se consiguieron 412 vídeos totales, 204 de ellos con la categoría normal y 208 de la acción a detectar. Estos se dividieron en tres particiones correspondientes a train, validation y test con el 70 %, 15 % y 15 % correspondientemente. Se aseguró su balanceo en categorías en cada subdivisión.

Muchos de ellos eran de calidad deficiente por lo que se decidió crear una segunda base de datos con solo aquellos que cumpliesen con unos requisitos mínimos de resolución y que se acogiesen a una definición más clara de la acción. Esta segunda base de datos contiene 210 vídeos, 105 de cada categoría. La repartición entre las subdivisiones de train, validation y test es de 64, 24 y 17 vídeos respectivamente de cada categoría, estando por tanto perfectamente balanceadas las categorías.

Como se puede observar en ambos casos la cantidad de muestras disponibles es insuficiente para un problema de estas características y esto se reflejará en los resultados del modelo. Esta deficiencia de datos se debe a las restricciones legales a las que las grabaciones de las cámaras de seguridad están sometidas. En cuanto a la calidad ética de la base de datos, se puede observar que no existe una gran diversidad étnica y sociocultural debido a las mayores restricciones que este contenido tiene en los países occidentales y a su mayor capacidad para aplicarlas. Es por eso que no es recomendable su uso para desarrollar modelos de producción.

4.1.2. Pipeline

Tras aplicar todas las técnicas mencionadas, se ha conseguido un pipeline eficiente con los recursos disponibles. Se ha comprobado la correcta carga y transmisión de los datos al algoritmo, así como la aplicación de las transformaciones pertinentes antes del entrenamiento. No se ha podido aplicar el algoritmo YOLO junto con Deep SORT debido a la falta de fiabilidad durante el recorte de las personas. Esto se debe a la baja calidad de los vídeos y a la excesiva superposición de múltiples personas.

El optimizador Adam fue el usado finalmente debido a su fácil implementación y los buenos resultados que se obtienen en la mayoría de las aplicaciones de deep learning. En cuanto a la función de perdidas, la función aplicada fue Cross-Entropy Loss por su versatilidad en cuanto al número de clases con las que se puede aplicar, los buenos resultados en la literatura y permitirnos observar directamente las probabilidades que el modelo asigna. La optimización de hiperparámetros se realizó manualmente, tomando decisiones basadas en la información registrada en TensorBoard de entrenamientos pasados.

4.1.3. Modelo

Tras muchas iteraciones en la estructura del modelo y de los hiperparámetros, se ha conseguido obtener overfitting en los datos de entrenamiento, lo que implica que el modelo tiene la complejidad suficiente para aprenderse el problema. Esto se puede observar en la figura 37, donde la función de perdidas decrece a lo largo de las épocas en los datos de entrenamiento, pero aumenta en los datos de validación.

A pesar de ello, no se ha conseguido que el modelo obtenga mejores resultados en el proceso de clasificación que hacer predicciones aleatorias, dando lugar a una precisión de en torno el 50 % en validación como es el caso cuando tenemos una base de datos de dos categorías balanceada. Estas observaciones se reflejan en la figura 38, donde obtenemos una precisión en

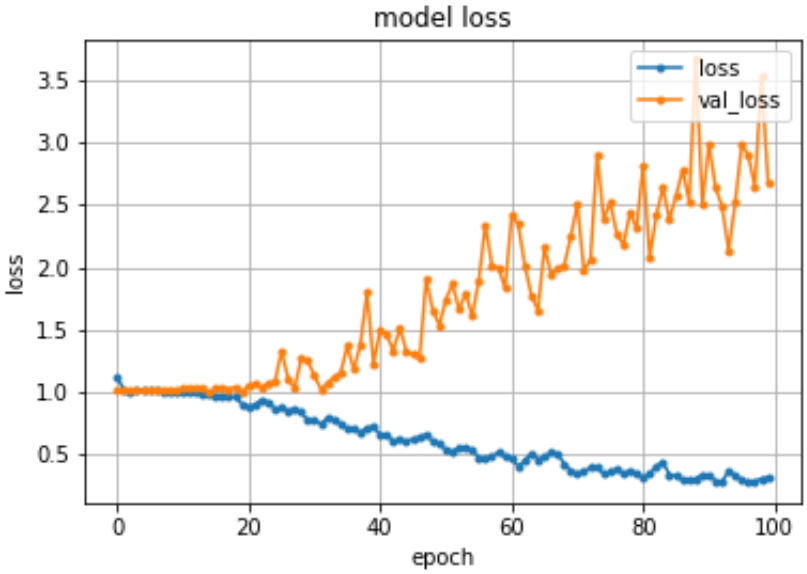

Figura 37. *Evolución de la función de perdidas.*

los datos de entrenamiento cercana al 100 % debido al overfitting pero en los datos de validación oscila en torno al 50 %.

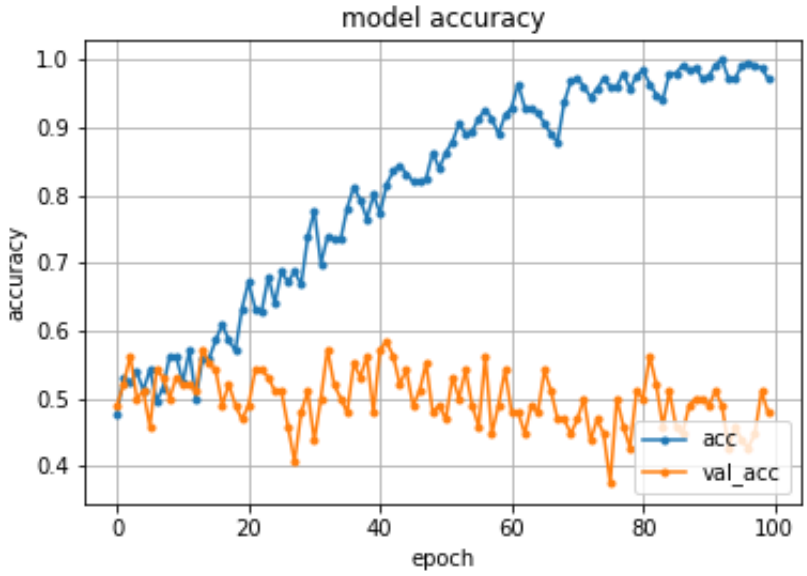

Figura 38. *Evolución de la precisión.*

Representando la matriz de confusión (figura 39), se observa que el algoritmo prefiere clasificar como la acción los casos normales dando lugar a falsos positivos. En este caso, es preferible a los falsos negativos los cuales además se encuentran en menor proporción.

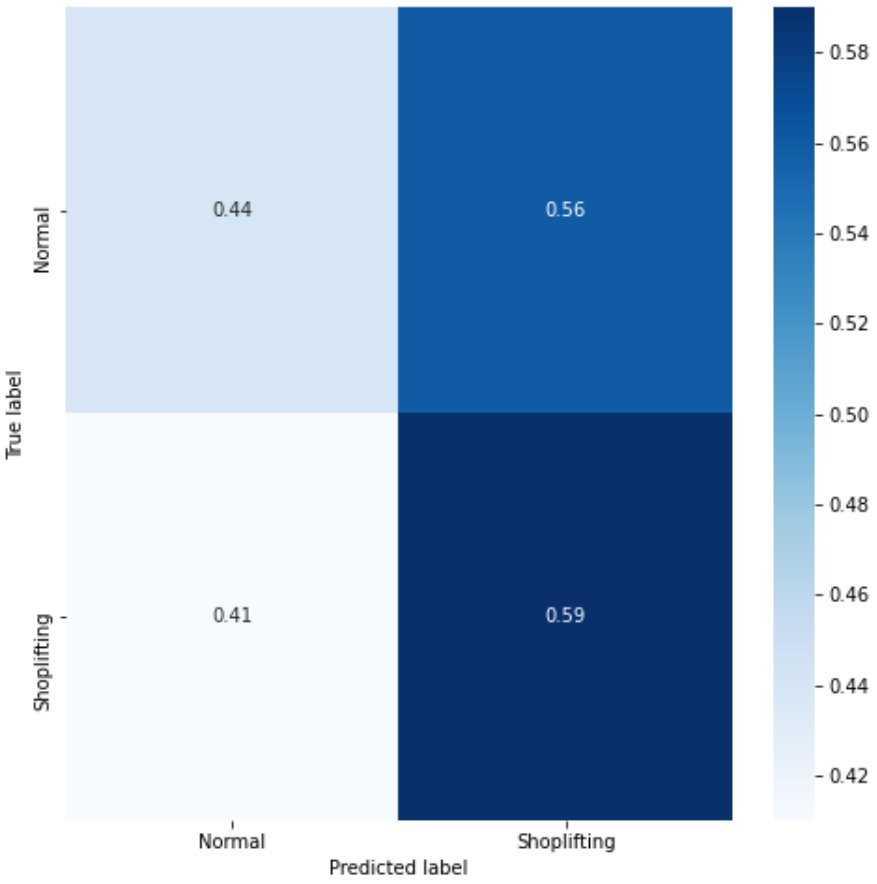

Figura 39. *Matriz de confusión.*

4.2. Conclusiones

De los resultados obtenidos, se pueden extraer diferentes causas de porque no se ha obtenido un modelo capaz de realizar predicciones. En un primer lugar vamos a analizar la base de datos y posteriormente el modelo.

4.2.1. Base de datos

En cuanto a la base de datos, se ha cumplido los siguientes objetivos:

- Construir una base de datos balanceada.
- Usar datos de acceso público.
- La correcta clasificación de los vídeos.

- Recortados para solo representar el tramo donde aparece la acción.

No se han recortado finalmente a las personas individuales con YOLO junto con Deep SORT ya que no es muy fiable a la hora de realizar bien la operación en situaciones con muchas personas cercanas. Al no poder ser aplicado en modelos de producción, se ha preferido realizar la clasificación directamente para así intentar desarrollar una herramienta más fiable. Respecto a la calidad de los vídeos y al número encontrado, las grandes restricciones impiden encontrar vídeos de calidad y en cantidad, limitando la variedad disponible y con grandes deficiencias de cara al aspecto ético. Es muy importante, especialmente cuando se trata con vídeos de cámaras de seguridad, tener en cuenta la eliminación de bias por raza, sexo y aspectos socioculturales. Debido a esto, los algoritmos que se desarrollen con esta base de datos no pueden ser aplicados en producción.

4.2.2. Modelo

En cuanto al modelo, se a cumplido los siguientes objetivos:

- Utilizar la capa 3DCNN.
- Ser lo suficientemente ligero para ser entrenado con los recursos disponibles.
- El uso de python como lenguaje de programación y el framework PyTorch.

No se ha conseguido la precisión del 70 % debido a varios motivos. El primero es la falta de datos suficientes para entrenar un modelo que resuelva una tarea tan compleja como la detección de acciones. Segundo, no se disponían de suficientes recursos para entrenar modelos más complejos que son capaces de compensar el problema de la escasez de datos. Finalmente, la baja calidad de los vídeos limitaba la capacidad del algoritmo de apreciar el detalle de la acción, especialmente al aplicar transformaciones.

4.3. Recomendaciones para futuros estudios

Para resolver los problemas encontrados, se recomienda explorar los siguientes puntos.

Ampliar la base de datos

Es necesario aumentar la calidad y cantidad de vídeos en la base de datos. Esto permitirá evitar bias y aumentar la calidad de imagen de los recortes de cada persona si se aplica esa estrategia. Además, se deberá tener en cuenta el aspecto ético para realizar esta operación. Para cumplir todo esto, se propone generar los vídeos mediante el uso de actores para asegurarse de cumplir con las obligaciones éticas y respetar la legislación vigente.

Transfer learning

En caso de no poder aumentar la base de datos, se propone entrenar un modelo en otras bases de datos de acciones humanas con categorías más pobladas y con mejor calidad para posteriormente transferir lo aprendido a nuestra base de datos. El concepto detrás de esta técnica consiste en mantener las primeras capas convolucionales del modelo intactas ya que en ellas se ha aprendido a detectar movimientos básicos comunes a todas las acciones. El resto de las capas se entrenarían por tanto con la base de datos y las dos categorías, pudiendo así aprender a detectar la acción con menos datos que si entrenásemos de cero.

Detección de personas

Continuar explorando algoritmos de detección de personas como YOLO + Deep SORT para aislarlas. Si se aumentase su fiabilidad, ya sea por la mejora de la técnica o la mejora de la resolución de los vídeos, se podría entrenar el modelo solo con la representación de la persona. Esto minimizaría el bias de entorno o la aparición de personas de distintas categorías en la misma imagen. Existen también otra categoría de modelos que permiten detectar los puntos clave de las personas como brazos, manos, cabeza, etc. Su uso podría complementar la funcionalidad de las convolucionales para detectar acciones.

Poder de computación

El acceso a más poder de computación es necesario de cara al entrenamiento de modelos con vídeos. Estos datos son muy pesados y requieren tanto de mucha memoria en la RAM y la GPU como de modelos de mayor complejidad. El aumento de las capacidades de computación permitiría reducir significativamente los tiempos de entrenamiento, los cuales se encuentran actualmente en el orden de horas y días, para poder probar nuevas arquitecturas y realizar una optimización de hiperparámetros más rigurosa y automatizada.

Bibliografía

[1] **Rosenblatt, F.**, *The perceptron: A probabilistic model for information storage and organization in the brain.*. Psychological Review, 65(6), 386408 (1958).
https://doi.org/10.1037/h0042519

[2] **Fukushima, K.**, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.*. Biol. Cybernetics 36, 193202 (1980).
https://doi.org/10.1007/BF00344251

[3] **Python**, *Python Software Foundation*. Last access: 06/06/2021
https://www.python.org/

[4] **Tensorflow**, *Open Source Machine Learning Framework for Everyone*. Last access: 06/06/2021
https://www.tensorflow.org/

[5] **PyTorch**, *An open source machine learning framework that accelerates the path from research prototyping to production deployment.*. Last access: 06/06/2021
https://pytorch.org/

[6] **Naciones Unidas**, *Objetivos de Desarrollo Sostenible*. Last access: 05/06/2021
https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[7] **Center for Research in Computer Vision (CRCV)**, *UCF101 - Action Recognition Data Set*. Last access: 05/06/2021
https://www.crcv.ucf.edu/data/UCF101.php

[8] **DeepMind**, *Kinetics*. Last access: 05/06/2021
https://deepmind.com/research/open-source/kinetics

[9] **Google Colaboratory**, *Ejecutar y programar en Python en tu navegador*. Last access: 05/06/2021
https://colab.research.google.com/

[10] **Anaconda**, *Sistema de gestión de paquetes*. Last access: 05/06/2021
https://www.anaconda.com/

[11] **GitHub**, *Where the world builds software*. Last access: 05/06/2021
https://github.com/

[12] **PyCharm**, *IDE de Python para desarrolladores profesionales*. Last access: 05/06/2021
https://www.jetbrains.com/es-es/pycharm/

[13] **Visual Studio Code**, *Code Editing. Redefined*. Last access: 05/06/2021
https://code.visualstudio.com/

[14] **Wolfgang Ertel**, *Introduction to Artificial Intelligence*. Springer International Publishing (2017).
https://doi.org/10.1007/978-3-319-58487-4

[15] **E. Rich**, *Artificial intelligence*. Graw-Hill, New York (1983).
https://doi.org/10.1016/0004-3702(86)90034-2

[16] **Sindhu V, Nivedha S, Prakash M**, *An empirical science research on bioinformatics in machine learning*. Journal of Mechanics of Continua and Mathematical Sciences , vol. spl7, no. 1, Feb. 2020.
https://doi.org/10.26782/jmcms.spl.7/2020.02.00006

[17] **Terence Shin**, *All Machine Learning Models Explained*. Last access: 08/06/2021
https://towardsdatascience.com/all-machine-learning-models-explained-in-6-minutes-9fe30ff6776a

[18] **Joe Klein**, *A Simple Proof of the Universal Approximation Theorem*. Last access: 10/06/2021
https://blog.goodaudience.com/neural-networks-part-1-a-simple-proof-of-the-universal-approximation-theorem-b7864964dbd3

[19] **IBM**, *What is computer vision?*. Last access: 11/06/2021
https://www.ibm.com/topics/computer-vision

[20] **J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko**, *Long-term recurrent convolutional networks for visual recognition and description*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2015.
https://doi.org/10.1109/cvpr.2015.7298878

[21] **S. Ji, W. Xu, M. Yang, and K. Yu**, *3D Convolutional Neural Networks for Human Action Recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 1, pp. 221231, Jan. 2013.
https://doi.org/10.1109/tpami.2012.59

[22] **Karen Simonyan, Andrew Zisserman**, *Two-Stream Convolutional Networks for Action Recognition in Videos*. Computer Vision and Pattern Recognition (cs.CV) (2014).
https://arxiv.org/abs/1406.2199

[23] **J. Carreira and A. Zisserman**, *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jul. 2017.
https://doi.org/10.1109/cvpr.2017.502

[24] **Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei**, *ImageNet: A large-scale hierarchical image database*. 2009 IEEE Conference on Computer Vision and Pattern Recognition, Jun. 2009.
https://doi.org/10.1109/cvprw.2009.5206848

[25] **Code Project**, *Classification using Fully Connected Networks*. Last access: 13/06/2021
https://www.codeproject.com/Articles/5160467/Image-Classification-Using-Neural-Networks-in-NET

[26] **Sumit Saha**, *A Comprehensive Guide to Convolutional Neural Networks*. Last access: 13/06/2021
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[27] **A. Krizhevsky, I. Sutskever, and G. E. Hinton**, *ImageNet classification with deep convolutional neural networks*. Communications of the ACM, vol. 60, no. 6, pp. 8490, May 2017.
https://doi.org/10.1145/3065386

[28] **Moses Olafenwa**, *Object Detection*. Last access: 13/06/2021
https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606

[29] **Paperswithcode**, *Semantic Segmentation*. Last access: 13/06/2021
https://paperswithcode.com/task/semantic-segmentation

[30] **Yee, K., Tantipongpipat, U., & Mishra, S.**, *Image Cropping on Twitter: Fairness Metrics, their Limitations, and the Importance of Representation, Design, and Agency*.

arXiv preprint 2021.
https://arxiv.org/abs/2105.08667

[31] **Z. Obermeyer, B. Powers, C. Vogeli, and S. Mullainathan,**, *Dissecting racial bias in an algorithm used to manage the health of populations*. Science, vol. 366, no. 6464, pp. 447453, Oct. 2019.
https://doi.org/10.1126/science.aax2342

[32] **EXL Service**, *Ethics and bias in artificial intelligence What every executive needs to know*. Last access: 15/06/2021
https://www.exlservice.com/ethics-and-bias-in-artificial-intelligence-what-every-executive-needs-to-know

[33] **M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru**, *Model Cards for Model Reporting*. Proceedings of the Conference on Fairness, Accountability, and Transparency, Jan. 2019.
https://doi.org/10.1145/3287560.3287596

[34] **Choi, J., Gao, C., Messou, J. C., & Huang, J. B.**, *Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition*. arXiv preprint 2019.
https://arxiv.org/abs/1912.05534

[35] **OpenCV**, *Open Source Computer Vision*. Last access: 15/06/2021
https://docs.opencv.org/4.5.2/

[36] **Google**, *Colaboratory: Preguntas frecuentes*. Last access: 20/06/2021
https://research.google.com/colaboratory/faq.html#resource-limits

[37] **PyTorch**, *AUTOMATIC MIXED PRECISION PACKAGE - TORCH.CUDA.AMP*. Last access: 24/06/2021
https://pytorch.org/docs/stable/amp.html

[38] **PyTorch**, *torch.utils.data.Dataset*. Last access: 26/06/2021
https://pytorch.org/docs/stable/data.html?highlight=dataset#torch.utils.data.Dataset

[39] **PyTorch**, *torchvision.datasets.DatasetFolder*. Last access: 26/06/2021
https://pytorch.org/vision/stable/datasets.html?highlight=datasetfolder#torchvision.datasets.DatasetFolder

[40] **PyTorch**, *Torchvision*. Last access: 26/06/2021
https://pytorch.org/vision/stable/index.html

[41] **YuxinZhaozyx**, *pytorch-VideoDataset*. Last access: 26/06/2021
https://github.com/YuxinZhaozyx/pytorch-VideoDataset

[42] **PyTorch**, *torchvision.transforms.Compose*. Last access: 30/06/2021
https://pytorch.org/vision/stable/transforms.html?highlight=compose#torchvision.transforms.Compose

[43] **PyTorch**, *torch.utils.data.DataLoader*. Last access: 30/06/2021
https://pytorch.org/docs/stable/data.html?highlight=dataloader#torch.utils.data.DataLoader

[44] **Smith, S. L., Kindermans, P. J., Ying, C., & Le, Q. V.**, *Don't decay the learning rate, increase the batch size*. arXiv preprint arXiv:1711.00489 2017.
https://arxiv.org/abs/1711.00489

[45] **Algorithmia**, *Introduction to loss functions*. Last access: 05/07/2021
https://algorithmia.com/blog/introduction-to-loss-functions

[46] **Jeremy Jordan**, *Setting the learning rate of your neural network.*. Last access: 05/07/2021
https://www.jeremyjordan.me/nn-learning-rate/

[47] **Smith, L. N.**, *Cyclical learning rates for training neural networks*. 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), 2017, pp. 464-472.
https://doi.org/10.1109/wacv.2017.58

[48] **Duchi, J., Hazan, E., & Singer, Y**, *Adaptive subgradient methods for online learning and stochastic optimization*. Journal of machine learning research 12.7 (2011).
https://stanford.edu/j̃duchi/projects/DuchiHaSi10_colt.pdf

[49] **Kurbiel, T. and Khaleghian, S.**, *Training of deep neural networks based on distance measures using RMSProp*. arXiv preprint arXiv:1708.01911 2017.
https://arxiv.org/abs/1708.01911

[50] **Zeiler, M.D.**, *Adadelta: an adaptive learning rate method*. arXiv preprint arXiv:1212.5701 2012.
https://arxiv.org/abs/1212.5701

[51] **Kingma, D.P. and Ba, J.**, *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980 2014.
https://arxiv.org/abs/1412.6980

[52] **Sanket Doshi**, *Various Optimization Algorithms For Training Neural Network*. Last access: 10/07/2021
https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[53] **Gunand Mayanglambam**, *Deep Learning Optimizers: SGD with momentum, Adagrad, Adadelta, Adam optimizer*. Last access: 10/07/2021
https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f

[54] **PyTorch**, *torch.optim*. Last access: 10/07/2021
https://pytorch.org/docs/stable/optim.html

[55] **Bryan Shalloway**, *Weighting Confusion Matrices by Outcomes and Observations*. Last access: 12/07/2021
https://www.r-bloggers.com/2020/12/weighting-confusion-matrices-by-outcomes-and-observations/

[56] **PyTorch**, *torch.nn.NLLLoss*. Last access: 11/07/2021
https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html#torch.nn.NLLLoss

[57] **PyTorch**, *torch.nn.CrossEntropyLoss*. Last access: 11/07/2021
https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss

[58] **PyTorch**, *torch.nn.HingeEmbeddingLoss*. Last access: 11/07/2021
https://pytorch.org/docs/stable/generated/torch.nn.HingeEmbeddingLoss.html#torch.nn.HingeEmbeddingLoss

[59] **Harjanto, F., Wang, Z., Lu, S., Tsoi, A. C., & Feng, D. D.**, *Investigating the impact of frame rate towards robust human action recognition*. Signal Processing, 124, 220-232 (2016).
https://doi.org/10.1016/j.sigpro.2015.08.006

[60] **Pier Paolo Ippolito**, *Hyperparameters Optimization*. Last access: 12/07/2021
https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d

[61] **Alexander Elvers**, *Grid Search*. Last access: 12/07/2021
https://upload.wikimedia.org/wikipedia/commons/thumb/b/b6/Hyperparameter_Optimization_using_Grid_Search.svg/810px-Hyperparameter_Optimization_using_Grid_Search.svg.png

[62] **Alexander Elvers**, *Random Search*. Last access: 12/07/2021
https://upload.wikimedia.org/wikipedia/commons/thumb/7/74/Hyperparameter_

Optimization_using_Random_Search.svg/810px-Hyperparameter_
Optimization_using_Random_Search.svg.png

[63] **Snoek, J., Larochelle, H., & Adams, R. P.**, *Practical bayesian optimization of machine learning algorithms.*. arXiv preprint arXiv:1206.2944 2012.
https://arxiv.org/abs/1206.2944

[64] **Alexander Elvers**, *Bayesian Optimization*. Last access: 12/07/2021
https://upload.wikimedia.org/wikipedia/commons/thumb/3/3c/Hyperparameter_
Optimization_using_Tree-Structured_Parzen_Estimators.svg/810px-
Hyperparameter_Optimization_using_Tree-Structured_Parzen_Estimators.svg.png

[65] **Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D.**, *Grad-cam: Visual explanations from deep networks via gradient-based localization*. arXiv:1610.02391 2016.
https://arxiv.org/abs/1610.02391

[66] **Chattopadhay, A., Sarkar, A., Howlader, P., & Balasubramanian, V. N.**, *Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks*. arXiv:1710.11063 2018.
https://arxiv.org/abs/1710.11063

[67] **TensorFlow**, *TensorBoard*. Last access: 14/07/2021
https://www.tensorflow.org/tensorboard?hl=es-419

[68] **PyTorch**, *torch.save*. Last access: 14/07/2021
https://pytorch.org/docs/stable/generated/torch.save.html

[69] **Adrian Rosebrock**, *Basic motion detection and tracking with Python and OpenCV*. Last access: 16/07/2021
https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-
with-python-and-opencv/

[70] **Leon Lok**, *Deep-SORT-YOLOv4*. Last access: 16/07/2021
https://github.com/LeonLok/Deep-SORT-YOLOv4

[71] **Zong, M., Wang, R., Chen, Z., Wang, M., Wang, X., & Potgieter, J**, *Multi-cue based 3D residual network for action recognition*. Neural Computing and Applications, 33(10), 5167-5181, 2020.
https://doi.org/10.1007/s00521-020-05313-8

[72] **Computersciencewiki**, *Max-pooling / Pooling*. Last access: 16/07/2021
https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

[73] **Amar Budhiraja**, *Dropout in (Deep) Machine learning*. Last access: 16/07/2021
https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[74] **Ioffe, S., & Szegedy, C.**, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. International conference on machine learning (pp. 448-456), 2015.
http://proceedings.mlr.press/v37/ioffe15.html