

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

"IDENTIFICACIÓN DE ACCIONES EN FORMATO VÍDEO CON UN MODELO DE RED CONVOLUCIONAL 2D + LSTM"

Autor: Luis Navarro Velasco

Directores: Álvaro Jesús López López

Lucía Güitta López

Madrid

Junio de 2021

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título "Identificación de acciones en formato vídeo con un modelo de red Convolucional 2D + LSTM"

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

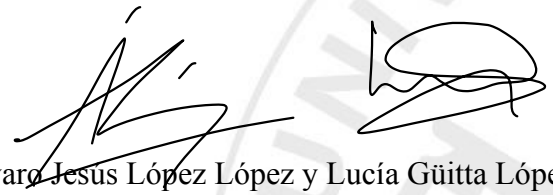
de otros documentos está debidamente referenciada.



Fdo.: Luis Navarro Velasco Fecha: 15/06/2021

Autorizada la entrega del proyecto

LOS DIRECTORES DEL PROYECTO



Fdo.: Álvaro Jesus López López y Lucía Güitta López Fecha: 16 / 06 / 21



MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

"IDENTIFICACIÓN DE ACCIONES EN FORMATO VÍDEO CON UN MODELO DE RED CONVOLUCIONAL 2D + LSTM"

Autor: Luis Navarro Velasco

Directores: Álvaro Jesús López López

Lucía Güitta López

Madrid

Junio de 2021

IDENTIFICACIÓN DE ACCIONES EN FORMATO VÍDEO CON UN MODELO DE RED CONVOLUCIONAL 2D + LSTM

Autor: Navarro Velasco, Luis

Directores: López López, Álvaro Jesús

Güitta López, Lucía

Entidad colaboradora: ICAI – Universidad Pontificia de Comillas.

RESUMEN DEL PROYECTO

1. *Introducción*

En la actualidad, es creciente el interés por desarrollar modelos de Inteligencia Artificial que faciliten las tareas que debe desempeñar el hombre. Cada vez son mayores y más variados los problemas cuya solución se busca mediante la aplicación de este tipo de algoritmos, los cuales van desde la detección personas y objetos en imágenes hasta la predicción de la bolsa o conducción de coches autónomos. Pese a que pueda parecer que la Inteligencia Artificial acaba de aterrizar en el mundo, esto no es así. El primer modelo de red neuronal fue diseñado por Frank Rosenblatt hace ya más de sesenta años; más concretamente en 1957 [1]. Este modelo presentaba por primera vez el Perceptrón, que era capaz no solo de aprender, sino también de generalizar, y aún se utiliza en la actualidad para el reconocimiento de patrones.

Estos modelos de redes neuronales han evolucionado mucho desde el primer diseño de Frank Rosenblatt, dando lugar a arquitecturas muy diferentes que dan solución a problemas muy variados. Esta rápida evolución en tiempos recientes se ha visto favorecida por el rápido aumento de la capacidad de procesamiento, que aumenta de forma exponencial y que está permitiendo elaborar modelos de gran complejidad.

Una de las vertientes que ha adoptado la Inteligencia Artificial es el Deep Learning, el cual ha supuesto un cambio de paradigma, ya que antes de la aparición de este tipo de modelos, era

necesario extraer las características de los datos previamente a entrenar el modelo de Inteligencia Artificial, lo cual hacía el trabajo más complejo y sesgaba la toma de decisiones hacia las características que los desarrolladores consideraban importantes. Con la llegada del Deep Learning, ya no es necesario extraer estas características, sino que gracias a la profundidad de las redes (es decir, al número de capas que estas contienen conectadas unas con otras), son capaces de extraer dichas características por sí mismas, llegando a comprender cuáles son las relevantes a la hora de tomar decisiones.

En la actualidad, los modelos de Deep Learning son utilizados para múltiples soluciones: predecir valores futuros, clasificar los datos de entrada, comprender el contenido de un texto, detección de objetos en imágenes.... Son muchas sus aplicaciones, pero la que se va a tratar en profundidad en este documento es la denominada Visión Artificial. Esta consiste en la creación de modelos capaces de analizar imágenes y extraer información de éstas. Hay mucha información diferente que se puede extraer de una imagen, y es por ello por lo que existen diferentes familias de modelos de Visión Artificial. Las principales aplicaciones o familias de modelos son las siguientes:

- **Clasificación de Imágenes**: el objetivo de este tipo de modelos es clasificar una imagen según su contenido.
- **Detección de objetos**: consiste en detectar ciertos objetos en una imagen, y localizarlos en la misma. Para ello se sitúan unas cajas sobre los objetos, denominadas "bounding boxes".
- **Segmentación de imágenes**: es un caso de uso parecido al anterior, pero en este caso se pretende especificar a qué objeto corresponde cada pixel de la imagen.
- **Detección de puntos clave**: consiste en detectar puntos característicos en una imagen. Esta técnica se emplea para hacer reconocimiento facial y para detectar la postura de las personas, ya que se consigue localizar en la imagen la ubicación de las articulaciones.

Aunque la aplicación de la Visión Artificial más desarrollada hasta ahora sea el análisis de imágenes, esta no se limita únicamente a dicha tarea, sino que también tiene por objetivo analizar vídeos.

El análisis de vídeos mediante técnicas de Visión Artificial presenta una solución compleja pero interesante a muchos problemas que actualmente debe resolver el hombre sin ayuda de las máquinas. Este tipo de técnicas están en auge en la actualidad, habiendo un gran interés por parte de la comunidad de desarrolladores debido a sus potenciales aplicaciones. La aplicación con mayor popularidad en la actualidad es la detección de acciones en cámaras de videovigilancia, que es en lo que se centra este proyecto.

En el mundo hay millones de cámaras de videovigilancia destinadas a asegurar que las calles o locales en los que están instalados son seguros. Actualmente, todas estas cámaras requieren de la supervisión constante de una persona, que debe asegurarse de que no hay actividad anómala en ninguna una de las cámaras, y por lo tanto que no está ocurriendo ningún crimen ni hay nadie en peligro. El hecho de desarrollar un modelo capaz de analizar lo que está ocurriendo en cada cámara de seguridad y extraer conclusiones de dichos vídeos supondría un ahorro de millones de euros en todo el mundo, además de conseguir que una máquina realice un trabajo ingrato como es la supervisión constante de estas cámaras. Es por ello por lo que es creciente el interés por lograr un modelo estable capaz de resolver este problema.

Este proyecto tiene por objetivo precisamente esa aplicación de la Visión Artificial: detectar acciones en vídeos de cámaras de videovigilancia, lo cual convierte el problema en uno de clasificación, en el que el modelo debe ser capaz de indicar si la acción en cuestión ha ocurrido o no en un vídeo.

Para ello, se ha optado por afrontar el problema mediante el diseño de un modelo de Deep Learning, lo cual permite que la extracción de características no se deba hacer de manera manual, sino que esta tarea la hace el propio modelo junto con la clasificación a partir de estas características extraídas de los datos.

El hecho de que la acción que se pretende detectar ocurra en grabaciones de cámaras de videovigilancia aumenta la complejidad del modelo. Esto se debe al hecho de no poder emplear la información que aporta el entorno, tal y como se comentará más adelante. Además, no hay disponibles en internet una cantidad suficiente de vídeos para la acción que se quiere detectar en este caso, y la calidad de estos es muy reducida.

La detección de acciones en vídeo está experimentando un gran desarrollo en los últimos años, en los que están surgiendo numerosos modelos que pretenden afrontar este problema de maneras diferentes. Algunos de estos modelos, se limitan a estudiar cada fotograma del vídeo por separado, mientras que otros más ambiciosos tienen por objetivo no solo extraer información de los fotogramas, sino también estudiar su evolución en el tiempo.

La primera aproximación, la de solo extraer información de los fotogramas, es la más sencilla y la más desarrollada en la actualidad. Esto consiste en emplear las técnicas de Visión Artificial destinadas al análisis de imágenes y aplicarlas a cada uno de los fotogramas del vídeo. Esto podría llevar a no comprender bien qué acción se está desarrollando, ya que el modelo no podrá extraer información del movimiento de la persona sino solo de su situación en cada fotograma por separado. Esto hace que el modelo se base en gran parte en el entorno en el que sucede la acción para determinar la misma. Sin embargo, esta técnica tiene gran aplicación en otros problemas de análisis de vídeo, como son la detección de objetos o la identificación de personas.

Por otro lado, el segundo enfoque mencionado para el análisis de vídeo mediante Visión Artificial consiste no solo en extraer información de cada fotograma separado, sino además de cómo evoluciona la escena. Esta forma de afrontar el problema de detección de acciones es más completa que la anterior, y permite centrarse realmente en la acción humana y no tanto en la situación en la que se encuentra la persona en la imagen.

Actualmente se están empleando dos métodos distintos a la hora de analizar vídeos de esta manera: un primer enfoque que consiste preprocesar las imágenes para extraer características de estas para posteriormente introducir los datos procesados en un modelo, y un segundo enfoque que consiste en usar solamente Deep Learning con los datos en bruto, y hacer que sea el propio modelo el que extraiga las características.

El primer método mencionado consiste en realizar un procesado previo al entrenamiento que permita mostrar únicamente la información relevante de las imágenes, y emplear dichos nuevos datos para entrenar una red neuronal. Un ejemplo de esto es la publicación “Real-Time Action Detection in Vídeo Surveillance using Sub-Action Descriptor with Multi-CNN” [2]. En este caso se propone procesar cada fotograma para extraer de cada uno de ellos la posición y movimiento de la persona, para posteriormente alimentar una red Convolucional con estos nuevos fotogramas procesados.

El segundo método mencionado permite emplear los vídeos en bruto, es decir, sin la necesidad de realizar ninguna transformación previa en los datos de entrada mediante el uso de Deep Learning. Para solucionar el problema de clasificación de esta manera, se están empleando estructuras de red neuronal como la Conv3D, que permite realizar un análisis espaciotemporal de los vídeos. Estos modelos están surgiendo desde hace tan solo 3 años y se encuentran aún en fase de investigación y desarrollo, y es el tipo de modelo que se pretende diseñar en este proyecto. Algunos de los modelos que cabe mencionar son el Conv + LSTM (el cual se ha desarrollado en este proyecto), Conv3D, Two-Stream Networks y Two-Stream Inflated 3D ConvNets. [3]

El uso de Deep Learning ha permitido combinar varios tipos de redes para extraer las características deseadas tanto de los fotogramas como de la componente temporal. Ejemplo de esto es la posibilidad de detectar puntos clave de las personas (sus articulaciones) y estudiar su evolución en el tiempo o incluso llevar a cabo una segmentación de la imagen y extraer los píxeles que corresponden a las personas para posteriormente analizarlos temporalmente.

Además de estos algoritmos de Deep Learning que se han mencionado, cabe mencionar el denominado Optical Flow. Este modelo pretende conocer el movimiento de los objetos en la imagen mediante el estudio del "movimiento" de los píxeles que la conforman, es decir, se trata de encontrar en el fotograma siguiente los valores de píxel que se encuentran en el fotograma actual, y así poder comparar la posición inicial y la siguiente simplemente observando su posición en ambos fotogramas. Esto puede permitir no solo conocer el movimiento de los objetos en un vídeo, sino además predecir cómo se moverán estos objetos en los fotogramas posteriores.

En este proyecto se ha decidido analizar tanto los fotogramas como la secuencia temporal porque, aunque se presenta como una solución más compleja, presenta una mayor precisión en vídeos con entornos estáticos, es decir, en aquellos vídeos que se graban desde una misma cámara posicionada en un mismo lugar, ofreciendo así una mejor solución para aplicaciones de videovigilancia, la cual apunta a ser la que más beneficios aporte a la sociedad en lo que a detección de acciones respecta. Además, tras explorar los dos métodos mencionados se ha apostado por hacer uso de Deep Learning, ya que la extracción previa de características puede sesgar las decisiones del modelo final.

2. *Objetivos*

Los objetivos de este proyecto son los siguientes:

- Desarrollar un modelo de detección de acciones en vídeo lo más robusto posible y agnóstico a la calidad del vídeo, es decir, que su funcionalidad no se vea comprometida por la calidad del vídeo de entrada.
- Desarrollar un modelo agnóstico al entorno, es decir, que el modelo no se base en el entorno en el que se desarrolla el vídeo para predecir la acción.
- Desarrollar un modelo que permita analizar vídeos de diferentes longitudes.
- Implementar una metodología de pre procesado de vídeos genérica, es decir, que acepte cualquier formato y tamaño de vídeo.
- Conseguir una precisión del 75% en el set de entrenamiento y del 65% en los sets de validación y test.

3. *Metodología*

El desarrollo de este proyecto se puede dividir en cuatro partes sucesivas, las cuales conforman un pipeline de Visión Artificial:

- Creación del dataset.
- Procesamiento de los vídeos que conforman el dataset.
- Creación de un modelo funcional.
- Análisis del entrenamiento.
- Optimización del modelo.

La primera tarea del proyecto es la de crear un dataset equilibrado (con una cantidad parecida de vídeos de cada acción a detectar) y significativo, de manera que permita al modelo tener una cantidad suficiente y no sesgada de los datos para que pueda generalizar, pudiendo así determinar la acción que se muestra en vídeos que no forman parte de la base de datos con la que este ha sido entrenado.

Antes de comenzar la creación de un dataset, se deben comprender aquellas características del mismo que podrían llevar a unos malos resultados en el entrenamiento del modelo. Además de lo comentado en el párrafo anterior de que el dataset debe ser equilibrado y significativo, de manera que todas las clases queden adecuadamente representadas para obtener buenos resultados de predicción en todas ellas, se debe tener en cuenta la influencia del entorno si se quiere desarrollar un modelo que sea agnóstico al mismo, como es el caso de este proyecto. Esta necesidad de un modelo agnóstico al entorno se debe a que lo que se pretende detectar son acciones en cámaras de videovigilancia, en las cuales no hay cambio de entorno, ya que están apuntando a un punto fijo.

Esto es opuesto a la gran mayoría de soluciones de detección de acciones en vídeo actuales, las cuales se apoyan mucho en el entorno para detectar acciones, tal y como hace el cerebro humano en muchos casos. Esta problemática se presenta en el paper "Why Can't I Dance in the Mall?" [4], en el que se explica que, al igual que las personas, los modelos que han sido entrenados sobre acciones en diferentes entornos son capaces de detectar la acción sin realmente fijarse en la persona, sino analizando el entorno. Este efecto se puede comprender en las siguientes imágenes, donde la persona que está desarrollando la acción queda oculta con un bloque verde, pero a pesar de ello sabemos qué acción está realizando en cada caso gracias al entorno:



Figura 1. Imágenes de acciones ocultando a la persona

El hecho de no poder apoyar la decisión en el entorno hace que la complejidad de este proyecto aumente considerablemente, ya que el modelo debe ser capaz de comprender totalmente los movimientos humanos, como si de un juego de mímica se tratara. Esto hace que el modelo deba ser más complejo y que la cantidad de datos necesaria sea mayor.

La primera tarea que realizar para generar el dataset consiste en realizar una búsqueda en internet de aquellas bases de datos ya creadas que puedan resultar útiles para el problema que se pretende solucionar (detección de una acción en concreto en cámaras de videovigilancia). Estas se deben filtrar para obtener únicamente aquellos vídeos de interés, es decir, aquellos que contengan escenas de las acciones que se pretenden detectar. En este caso, no hay disponibles en internet bases de datos que contengan la acción buscada, por lo que se continuó la búsqueda en repositorios públicos que pudieran contener dichos vídeos.

Finalmente, tras realizar una búsqueda por varios repositorios públicos, se hallaron vídeos en YouTube que contenían la clase deseada, aunque cabe destacar la escasez y la mala calidad de los mismos. Para poder generar el dataset, se creó un código que permite descargar tanto vídeos como listas de vídeos de YouTube, lo cual permitió obtener el contenido de forma rápida y automatizada. Tras este procedimiento, se obtuvo un total de 210 vídeos.

Una vez se dispone de los vídeos que compondrán el dataset, es necesaria una labor de procesamiento de dichos datos. Esto consiste en escoger el tramo de vídeo en el que se desarrolla la acción en concreto, desechando el resto del vídeo para no confundir al modelo. Además, se deben formatear los vídeos para que todos tengan las mismas dimensiones, ya que el modelo requiere que todos los datos de entrada tengan el mismo formato. Para ello se deben escalar los vídeos, modificando sus dimensiones de alto y ancho de fotograma y el número de fotogramas del vídeo. Algo a tener en cuenta, además, es que una buena práctica es mantener constante el número de fotogramas por segundo, de manera que el salto temporal entre cada fotograma sea constante en todos los vídeos.

Tras el formateo de los vídeos, se han llevado a cabo técnicas diferentes para aumentar la cantidad de datos disponibles en el dataset. Esto es una técnica denominada Data Augmentation. Para ello se implementaron códigos que permiten generar un modo espejo del vídeo (simetría sobre eje vertical) y vídeos con ruido gaussiano. Esta última forma de Data Augmentation se desarrolló basado en la idea del Adversarial Training. Con la implementación de estos métodos, se aumentó el tamaño del dataset hasta obtener un total de 630 vídeos. Estos vídeos se repartieron entre los conjuntos de entrenamiento, validación y test, con una proporción de 60% de los vídeos para entrenamiento, 20% para validación y el 20% restante para test.

Una vez se completó la creación del dataset, se procedió al diseño e implementación del modelo. El modelo diseñado para tratar de predecir acciones en vídeo sigue una estructura de Conv2D + LSTM. Siguiendo esta estructura, se crearon tres modelos diferentes. El primero de ellos se diseñó empleando la librería Keras. Tras el estudio de la flexibilidad que proporcionaba la librería PyTorch, se decidió migrar el modelo a dicho framework. El modelo inicial desarrollado en PyTorch sigue la siguiente estructura:

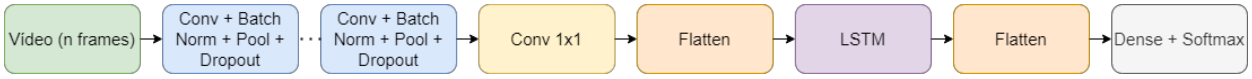


Figura 2. Arquitectura del modelo Conv2D + LSTM

Lo que se pretende con este modelo, es extraer características tanto de los fotogramas del vídeo, de lo cual se encarga el bloque convolucional (en azul en la imagen) como de la evolución de dichos fotogramas en el tiempo, de lo que se encarga la LSTM. Cabe destacar el uso de una capa convolucional 1x1, lo cual permite reducir el número de canales de entrada a la LSTM, reduciendo así drásticamente el número de parámetros de dicha capa. Este flujo de datos y las diferentes dimensiones de los mismos en su paso por el modelo se representa a continuación:

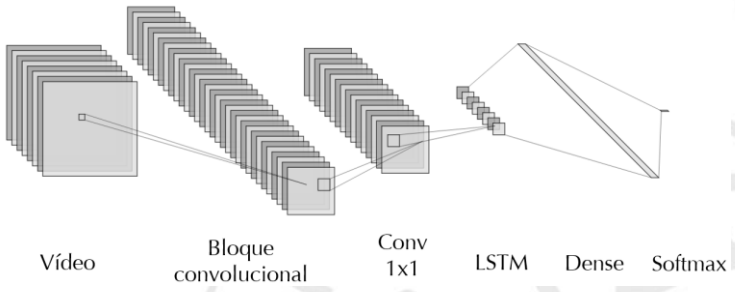


Figura 3. Dimensiones de las características en el modelo Conv2D + LSTM

Son numerosas las modificaciones que se han introducido en esta estructura inicial con el objetivo de lograr mejores resulatdos. Algunas de las modificaciones más destacadas son la eliminación de gran parte de las capas Pool, la eliminación de las capas Dropout debido a su escaso

efecto en la red y la implementación de un bloque convolucional que sigue una estructura basada en las redes residuales (ResNet).

A pesar de estas modificaciones en la arquitectura y de realizar una búsqueda de los hiperparámetros óptimos, no se consiguió evitar el overfitting. Esto llevó a la implementación de una tercera red, para el cual se hizo uso de Transfer Learning. En este caso, el Transfer Learning consistió en introducir una red ResNet18 preentrenada como estimador, de forma que extraiga las características necesarias de las imágenes y las pasara como entrada a una LSTM, dando lugar a un modelo de estructura ResNet18 + LSTM:

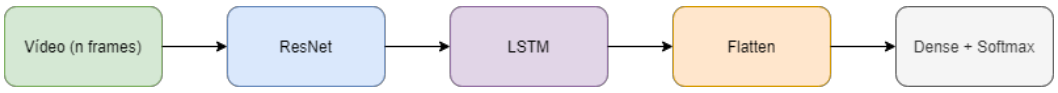


Figura 4. Arquitectura del modelo ResNet18 + LSTM

Tras la creación del modelo, y durante la iteración sobre los hiperparámetros del mismo, se desarrollaron diferentes métodos de optimización del entrenamiento, que permitieron reducir los tiempos del mismo. Estas técnicas desarrolladas son la aceleración por hardware mediante el uso de GPU y la implementación de Mixed Precission.

Este último método permite entrenar la red tanto con pesos de tipo float32 como float16 mientras se asegura la estabilidad del modelo. Esto se consigue evaluando el tamaño del peso, y si sus cifras significativas pueden ser correctamente representadas en float16, se emplea dicho formato de datos, y de lo contrario se emplea float32. Esto permite no solo aumentar la velocidad de entrenamiento al reducir el tiempo requerido para realizar operaciones con los pesos de la red, sino que además permite reducir los requerimientos de computación, lo cual lo convierte en una técnica muy interesante de aplicar en modelos de Machine Learning.

En cuanto a la evaluación del modelo, se emplearon varias técnicas que permiten conocer el estado del entrenamiento durante el mismo y su posterior análisis. Durante el entrenamiento, se muestran los valores de precisión y pérdidas tras cada época, así como el tiempo que ha llevado completar cada una de ellas para poder comprender el impacto de los diferentes métodos de optimización en el modelo. En cuanto al análisis del entrenamiento a posteriori, se genera las gráficas

de precisión y pérdidas del modelo, lo cual permite analizar el comportamiento del mismo y poder así decidir cuáles serán las siguientes modificaciones en la arquitectura y en los hiperparámetros. Además, se analiza la matriz de confusión y se generan los mapas de activación de cada una de las capas, lo cual permite comprender en qué se está fijando el bloque funcional a la hora de extraer características de los fotogramas.

El análisis de estas gráficas permite comprender el estado actual del modelo y cómo se está comportando, lo cual es esencial para continuar con el proceso iterativo de optimización de sus hiperparámetros.

4. Resultados

Los resultados obtenidos en este proyecto no se limitan a los resultados del modelo, ya que para obtenerlos, se ha desarrollado un pipeline completo de Visión Artificial. Es por ello que en este apartado cabe hacer mención también a los desarrollos que se han completado en cada una de las partes del pipeline, los cuales han permitido llevar a la implementación de un modelo con capacidad de extraer características de vídeos.

En primer lugar, se ha generado una base de datos con vídeos encontrados en internet. Para ello, no solo se ha realizado una búsqueda exhaustiva que ha permitido recolectar 210 vídeos de un tipo de acción con escasos datos en internet, sino que, para agilizar la descarga, se ha implementado un código que permite descargar tanto vídeos individuales como listas de YouTube de forma automática.

Una vez que se recolectaron todos los datos, se implementó un código que permite dividir los vídeos para extraer los fotogramas en los que se desarrolla la acción de interés, y que etiqueta el vídeo según si aparece dicha acción o no.

A continuación, teniendo los vídeos clasificados, se implementaron con éxito dos técnicas de Data Augmentation (modo espejo y ruido gaussiano), que permitieron aumentar el tamaño del dataset en un 300%, hasta tener un total de 630 vídeos. Además, se implementaron funciones que permitieron formatear cada uno de los vídeos para conseguir unas dimensiones iguales en cada uno de ellos, de forma que quedaran totalmente preparados para ser procesados por la red.

En lo que al modelo se refiere, se desarrollaron tres modelos diferentes. El primer modelo se desarrolló haciendo uso de la librería Keras. Este modelo se terminó por descartar, y se realizó una migración a PyTorch, donde se desarrolló un modelo similar pero más complejo. Este modelo de Conv2D + LSTM presenta los siguientes resultados:

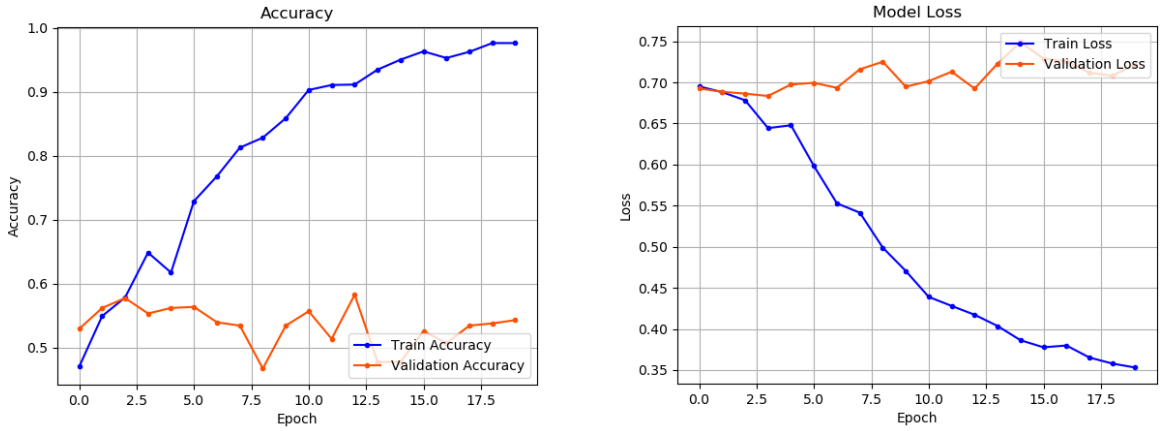


Figura 5. Gráficas de precisión y pérdidas del modelo Conv2D + LSTM

Tal y como se puede observar, hay un gran overfitting en el modelo a pesar de las diversas técnicas que se han empleado para evitarlo. En la búsqueda de reducción de este overfitting, se ha introducido Dropout, implementado Regularización L2, aumentar el volumen de datos (lo que llevó a implementar las técnicas de Data Augmentation mencionadas anteriormente) y reducción de la complejidad del modelo. Dado que la implementación de estas técnicas no favoreció la capacidad de generalización, se decidió implementar Transfer Learning para intentar reducir este overfitting.

Esta decisión de hacer uso de Transfer Learning es lo que llevó al desarrollo del tercer y último modelo que se ha implementado en este proyecto. Este modelo es un ResNet18 + LSTM. De esta manera, se sustituyó el bloque convolucional por una ResNet18 preentrenada, lo cual permitió reducir sustancialmente el número de parámetros de la red que se debían entrenar, ya que la ResNet18 se emplea como un estimador, de forma que extrae las características de los fotogramas, pero no se entrena con dichos datos. Los resultados se pueden observar a continuación:

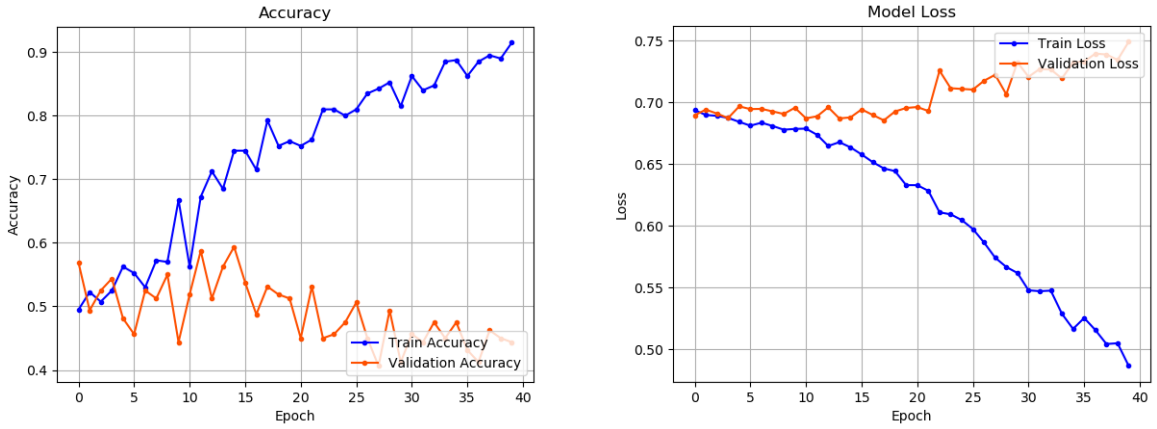


Figura 6. Gráficas de precisión y pérdidas del modelo ResNet18 + LSTM

Tal y como se puede observar, a pesar de que el comienzo del entrenamiento presenta mayores valores de precisión en validación, no se ha logrado evitar el overfitting que también presentaba el modelo anterior. En este modelo se han aplicado las mismas técnicas que en el anterior para intentar evitar este sobreaprendizaje, pero a pesar de su implementación no se ha conseguido resolver el problema.

A pesar de este problema en los resultados, cabe destacar que el hecho de que el modelo haga overfitting, indica que es un modelo funcional, lo cual supone un punto de partida para continuar con el desarrollo en la búsqueda de mejores resultados de predicción. Esto además muestra que los pasos previos desarrollados en el pipeline funcionan correctamente, y permiten que el modelo aprenda a extraer características de los datos.

Cabe mencionar, por último, que además de todo esto mencionado, en este proyecto se ha desarrollado con éxito técnicas de optimización del entrenamiento como la aceleración por hardware, que ha permitido reducir en gran medida los tiempos de entrenamiento mediante el uso de GPU y Mixed Precission, que ha permitido reducir los tiempos de entrenamiento y los requerimientos de computación.

5. *Conclusiones*

Son millones las cámaras de videovigilancia presentes en el mundo para garantizar la seguridad tanto en las calles como en interiores. El análisis de todos estos vídeos es una tarea que requiere de la presencia de una persona controlando varias de estas cámaras simultáneamente, lo cual es una tarea ardua y sujeta a posibles errores. Es por ello por lo que cada vez hay un mayor interés en automatizar esta labor, empleando tecnología que apoye a la persona encargada de tal actividad.

Con el objetivo de resolver parte del problema, en este proyecto se desarrolla un modelo de Visión Artificial que pretende detectar una acción en concreto en cámaras de videovigilancia. Para ello, se han estudiado diferentes arquitecturas y se ha optado por implementar un modelo de Conv2D+LSTM.

En este proyecto no solo se ha implementado el modelo, sino que se ha desarrollado de forma completa un pipeline de Visión Artificial, llevando a cabo desde la recolección de datos para la formación del dataset hasta la optimización del modelo, pasando por el procesado de los vídeos, el diseño de la arquitectura del modelo, análisis del entrenamiento…

A la hora de analizar los resultados del modelo, se puede observar que es capaz de extraer características, lo cual indica que el pipeline desarrollado funciona adecuadamente. A pesar de ello, no se ha conseguido evitar el overfitting en el modelo pese a las numerosas técnicas aplicadas para intentar solventar este problema tan común en los modelos de Machine Learning. Por otro lado, estos resultados llevan a pensar que pueda haber un error en los datos (bien en la calidad, la cantidad o en la manera de clasificarlos) o que el modelo diseñado no es el adecuado para este tipo de problema, no siendo capaz de extraer las características necesarias a pesar de las múltiples modificaciones que se han realizado en sus hiperparámetros. Esta segunda idea es la que lleva a que en los futuros desarrollos de este proyecto se planteen diferentes estructuras de red para poder afrontar el problema con un enfoque distinto al actual.

Una de las grandes problemáticas y retos de este proyecto ha sido la creación del dataset. Esto se debe a la escasez de datos presentes en internet y a la imposibilidad de generar dichos datos. Por otro lado, la calidad de los vídeos disponibles en internet es muy baja, no superando en muchos casos una calidad de 240 píxeles. Esto ha podido limitar en gran medida la capacidad de aprendizaje del

modelo, ya que, si los datos con los que se entrena no son de calidad, tampoco pueden serlo sus predicciones.

El desarrollo de algoritmos de Machine Learning es complejo y largo, y no siempre resulta exitoso, pero siempre permite aprender cuáles pueden ser los siguientes pasos a seguir para lograr una implementación exitosa.

6. *Referencias*

[1] «Historia de las Redes Neuronales,» Itnuevolaredo, [En línea]. Available: http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/RNA/Redes%20Neuronales2.pdf.

[2] C.-B. Jin, «Real-Time Action Detection in Video Surveillance using Sub-Action,» Arxiv, [En línea]. Available: https://arxiv.org/ftp/arxiv/papers/1710/1710.03383.pdf.

[3] J. C. y. A. Zisserman, «Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset,» Arxiv, 12 Febrero 2018. [En línea]. Available: https://arxiv.org/pdf/1705.07750.pdf.

[4] e. a. Jinwoo Choi, «Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition,» Arxiv, 11 Diciembre 2019. [En línea]. Available: https://arxiv.org/pdf/1912.05534.pdf.

ACTION RECOGNITION IN VÍDEO WITH A CONVOLUTIONAL 2D + LSTM MODEL

Autor: Navarro Velasco, Luis

Directores: López López, Álvaro Jesús

Güitta López, Lucía

Collaborating entity: ICAI – Pontifical University of Comillas

PROJECT ABSTRACT

1. *Introduction*

Currently, there is a growing interest in developing Artificial Intelligence models that facilitate the tasks to be performed by humans. The problems whose solution is sought through the application of this type of algorithms are becoming increasingly larger and more varied, ranging from the detection of people and objects in images to the prediction of the stock market or the driving of autonomous cars. Although it may seem that Artificial Intelligence has just landed in the world, this is not so. The first neural network model was designed by Frank Rosenblatt more than sixty years ago; more specifically in 1957 [1]. This model introduced for the first time the Perceptron, which was capable not only of learning, but also of generalizing, and is still used today for pattern recognition.

These neural network models have evolved greatly since Frank Rosenblatt's first design, giving rise to very different architectures that provide solutions to a wide variety of problems. This rapid evolution in recent times has been favored by the rapid increase in processing capacity, which is growing exponentially and is making it possible to develop models of great complexity.

One of the aspects that Artificial Intelligence has adopted is Deep Learning, which has meant a paradigm shift, since before the appearance of this type of models, it was necessary to extract the characteristics of the data before training the Artificial Intelligence model, which made the work more complex and biased the decision making towards the characteristics that the developers considered important. With the advent of Deep Learning, it is no longer necessary to extract these features, but

thanks to the depth of the networks (i.e. the number of layers they contain connected to each other), they are able to extract these features by themselves, understanding which ones are relevant when making decisions.

Deep Learning models are currently used for multiple solutions: predicting future values, classifying input data, understanding the content of a text, detecting objects in images.... There are many applications, but the one that will be discussed in depth in this paper is called Machine Vision. This consists of the creation of models capable of analyzing images and extracting information from them. There is a lot of different information that can be extracted from an image, and that is why there are different families of Computer Vision models. The main applications or families of models are the following:

- **Image Classification**: the objective of this type of models is to classify an image according to its content.
- **Object detection**: it consists of detecting certain objects in an image and locating them in the image. For this purpose, boxes called "bounding boxes" are placed over the objects.
- **Image segmentation**: this use case is similar to the previous one, but in this case it is intended to specify to which object each pixel of the image corresponds.
- **Key point detection:** consists of detecting characteristic points in an image. This technique is used for facial recognition and to detect the posture of people, since it is possible to locate the location of the joints in the image.

Although the most developed application of Computer Vision so far is image analysis, it is not limited only to this task, but also aims to analyze videos.

The analysis of videos by means of Artificial Vision techniques presents a complex but interesting solution to many problems that man must currently solve without the help of machines. This type of techniques are currently booming, with a great interest from the developer community due to their potential applications. The most popular application at present is the detection of actions in video surveillance cameras, which is the focus of this project.

Around the world there are millions of video surveillance cameras designed to ensure that the streets or premises in which they are installed are safe. Currently, all of these cameras require constant monitoring by a person, who must ensure that there is no anomalous activity on any of the cameras, and therefore that no crime is occurring and no one is in danger. Developing a model capable of analyzing what is happening in each security camera and drawing conclusions from these videos would save millions of euros worldwide, as well as getting a machine to do the thankless job of constantly monitoring these cameras. This is why there is growing interest in achieving a stable model capable of solving this problem.

This project aims precisely at this application of Computer Vision: detecting actions in videos from video surveillance cameras, which turns the problem into a classification problem, in which the model must be able to indicate whether the action in question has occurred or not in a video.

To this end, we have chosen to tackle the problem by designing a Deep Learning model, which allows the extraction of features not to be done manually, but rather this task is done by the model itself along with the classification from these features extracted from the data.

The fact that the action to be detected occurs in video surveillance camera recordings increases the complexity of the model. This is due to the fact that it is not possible to use the information provided by the environment, as will be discussed later. In addition, there are not enough videos available on the Internet for the action to be detected in this case, and the quality of these videos is very low.

Video action detection is undergoing a great development in recent years, in which numerous models are emerging that seek to address this problem in different ways. Some of these models are limited to study each frame of the video separately, while other more ambitious models aim not only to extract information from the frames, but also to study their evolution over time.

The first approach, that of only extracting information from the frames, is the simplest and currently the most developed. This consists of using Computer Vision techniques for image analysis and applying them to each of the video frames. This could lead to misunderstanding what action is taking place, since the model will not be able to extract information from the movement of the person, but only from his situation in each frame separately. This makes the model rely heavily on the

environment in which the action is taking place to determine the action. However, this technique has great application in other video analysis problems, such as object detection or person identification.

On the other hand, the second approach mentioned for video analysis using Computer Vision consists not only in extracting information from each separate frame, but also how the scene evolves. This approach to the problem of action detection is more complete than the previous one, and allows to really focus on the human action and not so much on the situation in which the person is in the image.

Two different methods are currently being used when analyzing videos in this way: a first approach that consists of pre-processing the images to extract features from them to later introduce the processed data into a model, and a second approach that consists of using only Deep Learning with the raw data, and having the model itself extract the features.

The first approach is to perform pre-training processing to display only the relevant information from the images, and use the new data to train a neural network. An example of this is the publication "Real-Time Action Detection in Video Surveillance using Sub-Action Descriptor with Multi-CNN" [2]. In this case it is proposed to process each frame to extract from each of them the position and movement of the person, and then feed a Convolutional network with these new processed frames.

The second method mentioned above makes it possible to use the raw videos, i.e. without the need to perform any prior transformation on the input data by using Deep Learning. To solve the classification problem in this way, neural network structures such as Conv3D are being used, which allows a spatio-temporal analysis of the videos. These models have been emerging for only 3 years and are still in the research and development phase, and it is the type of model that is intended to be designed in this project. Some of the models worth mentioning are the Conv + LSTM (which has been developed in this project), Conv3D, Two-Stream Networks and Two-Stream Inflated 3D ConvNets. [3]

The use of Deep Learning has made it possible to combine several types of networks to extract desired features from both the frames and the temporal component. An example of this is the possibility of detecting key points of people (their joints) and studying their evolution over time or

even carrying out a segmentation of the image and extracting the pixels that correspond to people to later analyze them temporally.

In addition to these Deep Learning algorithms mentioned above, it is worth mentioning the so-called Optical Flow. This model aims to know the movement of objects in the image by studying the "movement" of the pixels that make up the image, i.e., it is to find in the next frame the pixel values found in the current frame, and thus compare the initial position and the next by simply observing their position in both frames. This can allow not only to know the movement of objects in a video, but also to predict how these objects will move in subsequent frames.

In this project, it has been decided to analyze both the frames and the time sequence because, although it is presented as a more complex solution, it presents a higher accuracy in videos with static environments, that is, in those videos that are recorded from the same camera positioned in the same place, thus offering a better solution for video surveillance applications, which aims to be the one that brings more benefits to society in terms of action detection. In addition, after exploring the two mentioned methods, it has been decided to make use of Deep Learning, since the previous extraction of features can bias the decisions of the final model.

2. *Objetives*

The objectives of this project are the following:

- To develop a video action detection model that is as robust as possible and agnostic to video quality, i.e., that its functionality is not compromised by the quality of the input video.
- Develop a model that is environment agnostic, i.e., the model does not rely on the environment in which the video takes place to predict the action.
- Develop a model that can analyze videos of different lengths.
- Implement a generic video pre-processing methodology, i.e., one that accepts any video format and size.
- Achieve an accuracy of 75% in the training set and 65% in the validation and test sets.

3. *Methodology*

The development of this project can be divided into four successive parts, which make up a Machine Vision pipeline:

- Creation of the dataset.
- Processing of the videos that make up the dataset.
- Creation of a functional model.
- Training analysis.
- Model optimization.

The first task of the project is to create a balanced (with a similar amount of videos of each action to be detected) and meaningful dataset, in order to allow the model to have a sufficient and unbiased amount of data so that it can generalize, thus being able to determine the action shown in videos that are not part of the database with which it has been trained.

Before starting the creation of a dataset, it is necessary to understand those characteristics of the dataset that could lead to poor results in the training of the model. In addition to the previous paragraph that the dataset must be balanced and meaningful, so that all classes are adequately represented to obtain good prediction results in all of them, the influence of the environment must be taken into account if we want to develop a model that is agnostic to it, as in the case of this project. This need for an environment-agnostic model is due to the fact that what is intended to be detected are actions in video surveillance cameras, in which there is no change of environment, since they are pointing to a fixed point.

This is opposite to the vast majority of current video action detection solutions, which rely heavily on the environment to detect actions, just as the human brain does in many cases. This problem is presented in the paper "Why Can't I Dance in the Mall?" [4], which explains that, like people, models that have been trained on actions in different environments are able to detect the action without actually looking at the person, but by analyzing the environment. This effect can be understood in the following images, where the person performing the action is hidden by a green block, but nevertheless we know which action is being performed in each case thanks to the environment:



Figura 1. Images hiding the person

Not being able to support the decision in the environment increases the complexity of this project considerably, since the model must be able to fully understand human movements, as if it were a mime game. This means that the model must be more complex and the amount of data required is greater.

The first task to be performed to generate the dataset is to search the Internet for those databases already created that may be useful for the problem to be solved (detection of a specific action in video surveillance cameras). These must be filtered to obtain only those videos of interest, i.e., those containing scenes of the actions to be detected. In this case, there are no databases available on the Internet that contain the desired action, so the search was continued in public repositories that could contain such videos.

Finally, after a search of several public repositories, videos were found on YouTube that contained the desired class, although it is worth noting the scarcity and poor quality of these videos. In order to generate the dataset, a code was created to download both videos and lists of videos from YouTube, which made it possible to obtain the content quickly and automatically. After this procedure, a total of 210 videos were obtained.

Once the videos that will compose the dataset are available, it is necessary to process the data. This consists of choosing the video section in which the specific action takes place, discarding the rest of the video so as not to confuse the model. In addition, the videos must be formatted so that they all have the same dimensions, since the model requires that all input data have the same format. To do this, the videos must be scaled, modifying their frame height and width dimensions and the number of frames in the video. It is also a good practice to keep the number of frames per second constant, so that the time jump between each frame is constant in all videos.

After formatting the videos, different techniques have been carried out to increase the amount of data available in the dataset. This is a technique called Data Augmentation. For this purpose, codes were implemented to generate a mirror mode of the video (symmetry on vertical axis) and videos with Gaussian noise. This last form of Data Augmentation was developed based on the idea of Adversarial Training. With the implementation of these methods, the dataset size was increased to a total of 630 videos. These videos were divided between the training, validation and test sets, with a proportion of 60% of the videos for training, 20% for validation and the remaining 20% for testing.

Once the creation of the dataset was completed, the model design and implementation proceeded. The model designed to try to predict video actions follows a Conv2D + LSTM structure. Following this structure, three different models were created. The first one was designed using the Keras library. After studying the flexibility provided by the PyTorch library, it was decided to migrate the model to this framework. The initial model developed in PyTorch follows the following structure:

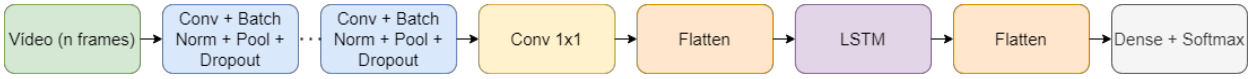


Figura 2. Conv2D + LSTM model architecture

The purpose of this model is to extract features both from the video frames, which is handled by the convolutional block (in blue in the image), and from the evolution of these frames over time, which is handled by the LSTM. It is worth noting the use of a 1x1 convolutional layer, which makes it possible to reduce the number of input channels to the LSTM, thus drastically reducing the number of parameters of this layer. This data flow and the different dimensions of the data as it passes through the model are shown below:

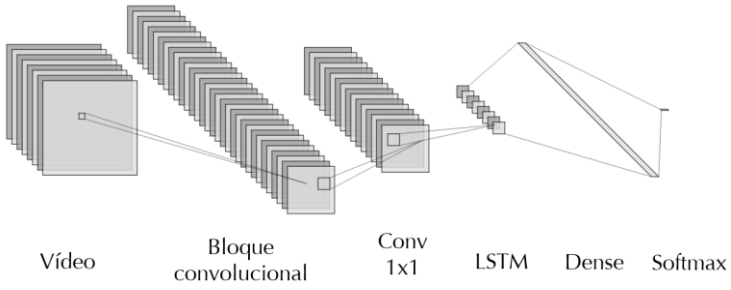


Figura 3. Conv2D + LSTM characteristic dimensions

Numerous modifications have been made to this initial structure in order to achieve better results. Some of the most outstanding modifications are the elimination of most of the Pool layers, the elimination of the Dropout layers due to their low effect on the network and the implementation of a convolutional block that follows a structure based on the residual networks (ResNet).

Despite these modifications to the architecture and a search for optimal hyperparameters, it was not possible to avoid overfitting. This led to the implementation of a third network, for which Transfer Learning was used. In this case, Transfer Learning consisted of introducing a pre-trained ResNet18 network as an estimator, so that it extracts the necessary features from the images and passes them as input to an LSTM, resulting in a ResNet18 + LSTM model:

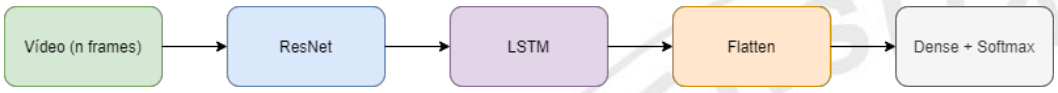


Figura 4. ResNet18 + LSTM model architecture

After the creation of the model, and during the iteration on the hyperparameters of the model, different training optimization methods were developed, which allowed to reduce the training times. These developed techniques are hardware acceleration using GPU and the implementation of Mixed Precission.

The latter method allows training the network with both float32 and float16 weights while ensuring model stability. This is achieved by evaluating the size of the weight, and if its significant figures can be correctly represented in float16, that data format is used, and otherwise float32 is used.

This allows not only to increase the training speed by reducing the time required to perform operations with the weights of the network, but also to reduce the computational requirements, which makes it a very interesting technique to apply in Machine Learning models.

Regarding the evaluation of the model, several techniques were used to know the training status during training and its subsequent analysis. During training, the accuracy and loss values after each epoch are shown, as well as the time taken to complete each epoch in order to understand the impact of the different optimization methods on the model. As for the a posteriori training analysis, the precision and loss plots of the model are generated, which allows to analyze the behavior of the model and thus to be able to decide which will be the next modifications in the architecture and in the hyperparameters. In addition, the confusion matrix is analyzed and activation maps are generated for each of the layers, which allows understanding what the functional block is focusing on when extracting features from the frames.

The analysis of these graphs allows understanding the current state of the model and how it is behaving, which is essential to continue with the iterative process of optimizing its hyperparameters.

4. *Results*

The results obtained in this project are not limited to the results of the model, since in order to obtain them, a complete Artificial Vision pipeline has been developed. That is why in this section it is also worth mentioning the developments that have been completed in each of the parts of the pipeline, which have led to the implementation of a model capable of extracting features from videos.

First of all, a database has been generated with videos found on the Internet. For this purpose, not only an exhaustive search was performed to collect 210 videos of a type of action with scarce data on the Internet, but also, to speed up the download, a code was implemented to download both individual videos and lists from YouTube automatically.

Once all the data was collected, a code was implemented that allows splitting the videos to extract the frames in which the action of interest takes place, and that tags the video according to whether that action appears or not.

Then, having the videos classified, two Data Augmentation techniques were successfully implemented (mirror mode and Gaussian noise), which allowed to increase the size of the dataset by 300%, up to a total of 630 videos. In addition, functions were implemented that made it possible to format each of the videos to achieve equal dimensions in each of them, so that they would be fully prepared for processing by the network.

As far as the model is concerned, three different models were developed. The first model was developed using the Keras library. This model was eventually discarded, and a migration was made to PyTorch, where a similar but more complex model was developed. This Conv2D + LSTM model presents the following results:

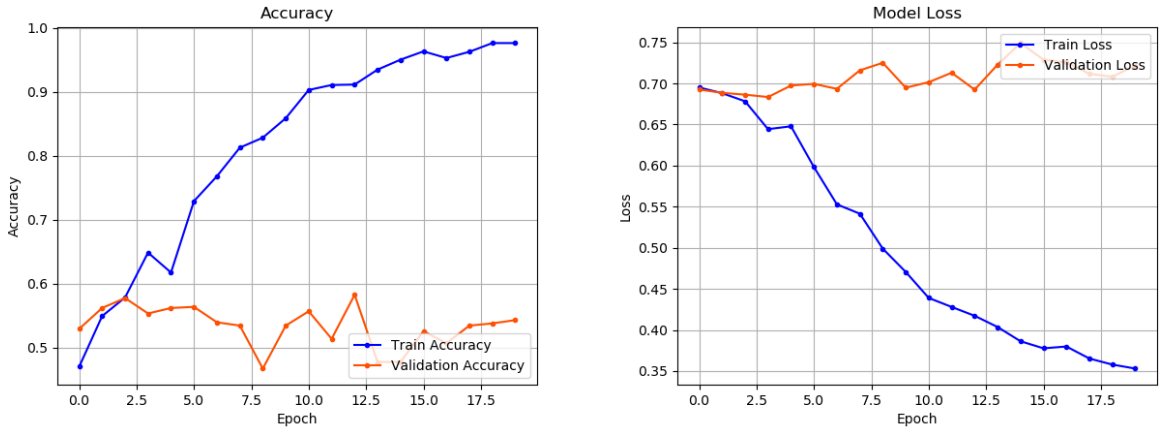


Figura 5. Conv2D + LSTM accuracy and loss plots

As can be seen, there is a great deal of overfitting in the model despite the various techniques that have been employed to avoid it. In the quest to reduce this overfitting, Dropout has been introduced, L2 Regularization has been implemented, the data volume has been increased (which led to implement the Data Augmentation techniques mentioned above) and the complexity of the model has been reduced. Since the implementation of these techniques did not favor the generalization capability, it was decided to implement Transfer Learning to try to reduce this overfitting.

This decision to make use of Transfer Learning is what led to the development of the third and last model implemented in this project. This model is a ResNet18 + LSTM. In this way, the convolutional block was replaced by a pre-trained ResNet18, which allowed to substantially reduce

the number of network parameters to be trained, since the ResNet18 is used as an estimator, so that it extracts the features from the frames, but is not trained with such data. The results can be seen below:

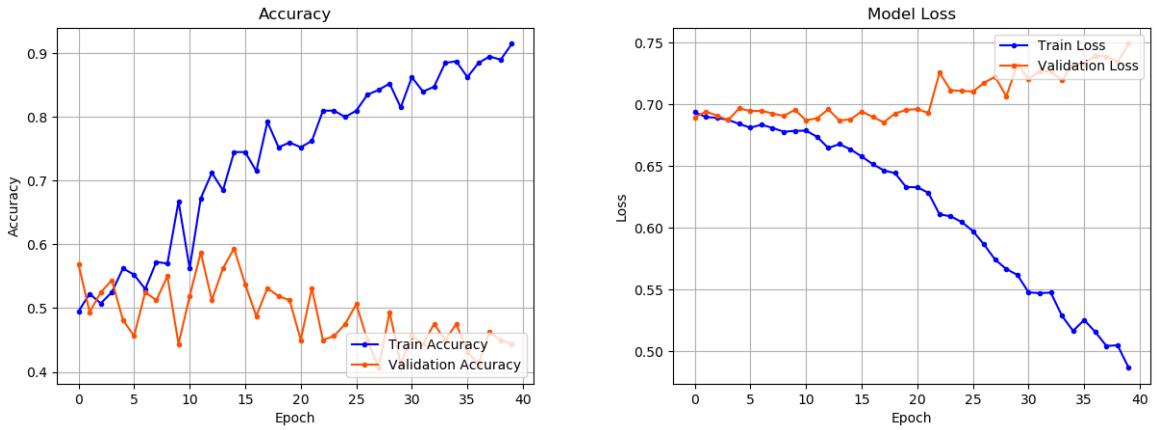


Figura 6. ResNet18 + LSTM accuracy and loss plots

As can be seen, despite the fact that the beginning of training presents higher validation accuracy values, it has not been possible to avoid the overfitting that was also present in the previous model. In this model, the same techniques have been applied as in the previous one to try to avoid this overlearning, but despite their implementation, the problem has not been solved.

Despite this problem in the results, it should be noted that the fact that the model does overfitting indicates that it is a functional model, which is a starting point for further development in the search for better prediction results. This also shows that the previous steps developed in the pipeline work correctly, and allow the model to learn to extract features from the data.

Finally, it is worth mentioning that in addition to all of the above, this project has successfully developed training optimization techniques such as hardware acceleration, which has greatly reduced training times through the use of GPU and Mixed Precission, which has reduced training times and computational requirements.

5. *Conclusions*

There are millions of video surveillance cameras in the world to ensure security both on the streets and indoors. The analysis of all these videos is a task that requires the presence of a person controlling several of these cameras simultaneously, which is an arduous task and subject to possible errors. This is why there is an increasing interest in automating this task, using technology that supports the person in charge of this activity.

In order to solve part of the problem, this project develops a Computer Vision model that aims to detect a specific action in video surveillance cameras. For this purpose, different architectures have been studied and it has been chosen to implement a Conv2D+LSTM model.

In this project, not only the model has been implemented, but also a complete Artificial Vision pipeline has been developed, from data collection for dataset training to model optimization, including video processing, model architecture design, training analysis...

When analyzing the results of the model, it can be observed that it is able to extract features, which indicates that the developed pipeline works properly. In spite of this, it has not been possible to avoid overfitting in the model despite the numerous techniques applied to try to solve this common problem in Machine Learning models. On the other hand, these results lead us to think that there may be an error in the data (either in quality, quantity or in the way of classifying them) or that the model designed is not suitable for this type of problem, not being able to extract the necessary features despite the multiple modifications that have been made to its hyperparameters. This second idea is the one that leads to the fact that in future developments of this project different network structures will be proposed in order to face the problem with a different approach than the current one.

One of the major problems and challenges of this project has been the creation of the dataset. This is due to the scarcity of data on the Internet and the impossibility of generating such data. On the other hand, the quality of the videos available on the Internet is very low, in many cases not exceeding a quality of 240 pixels. This may have greatly limited the learning capacity of the model, since, if the data with which it is trained are not of high quality, neither can its predictions be of high quality.

The development of Machine Learning algorithms is complex and time-consuming, and is not always successful, but it is always possible to learn what the next steps can be to achieve a successful implementation.

6. *References*

[1] «Historia de las Redes Neuronales,» Itnuevolaredo, [En línea]. Available: http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/RNA/Redes%20Neuronales2.pdf.

[2] C.-B. Jin, «Real-Time Action Detection in Video Surveillance using Sub-Action,» Arxiv, [En línea]. Available: https://arxiv.org/ftp/arxiv/papers/1710/1710.03383.pdf.

[3] J. C. y. A. Zisserman, «Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset,» Arxiv, 12 Febrero 2018. [En línea]. Available: https://arxiv.org/pdf/1705.07750.pdf.

[4] e. a. Jinwoo Choi, «Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition,» Arxiv, 11 Diciembre 2019. [En línea]. Available: https://arxiv.org/pdf/1912.05534.pdf.



MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

"IDENTIFICACIÓN DE ACCIONES EN FORMATO VÍDEO CON UN MODELO DE RED CONVOLUCIONAL 2D + LSTM"

Autor: Luis Navarro Velasco

Directores: Alvaro Jesús López López

Lucía Güitta López

Madrid

Junio de 2021

TABLA DE CONTENIDOS

CAPÍTULO I: INTRODUCCIÓN Y OBJETIVOS 7
CAPÍTULO II: ESTADO DEL ARTE 15
CAPÍTULO III: METODOLOGÍA 29
CAPÍTULO IV: DISEÑO DE LA SOLUCIÓN 31
1. INFLUENCIA DEL ENTORNO EN LA DETECCIÓN DE ACCIONES 31
2. CREACIÓN DEL DATASET 32
3. DATA AUGMENTATION 39
4. NORMALIZACIÓN DE LOS DATOS 47
5. PROCESAMIENTO DE VÍDEOS 49
6. DESARROLLO DEL ALGORITMO 55
6.1. LIBRERÍAS PARA LA CREACIÓN DE ALGORITMOS 55
TENSORFLOW 56
KERAS 59
PYTORCH 62
COMPARATIVA 65
6.2. CARGA DE DATOS CON PYTORCH 66
TORCH DATASET 67
TORCH DATALOADER 69
6.3. TRANSFER LEARNING 71
TRANSFER LEARNING CON PYTORCH 77
6.4. CONV2D + LSTM 78
6.5. BATCH NORMALIZATION 86
6.6. OVERFITTING Y CÓMO EVITARLO 87
6.7. ACELERACIÓN DEL ENTRENAMIENTO CON GPU 93
7. ANÁLISIS DEL ENTRENAMIENTO 96
7.1. MÉTRICAS DE EVALUACIÓN 96
7.2. GRÁFICAS DE ENTRENAMIENTO 100
7.3. MAPAS DE ACTIVACIÓN 108
8. OPTIMIZACIÓN DEL ENTRENAMIENTO 110
8.1. OPTIMIZACIÓN DE HIPERPARÁMETROS 110
8.2. MIXED PRECISSION 118
CAPÍTULO V: RESULTADOS 121
CAPÍTULO VI: CONCLUSIONES 125
CAPÍTULO VII: FUTUROS DESARROLLOS 127
CAPÍTULO VIII: ALINEACIÓN CON LOS ODS 129
CAPÍTULO IX: REFERENCIAS 131

TABLA DE FIGURAS

Figura 1. Velocidad de reloj en los procesadores [2] 7
Figura 2. Machine Learning vs Deep Learning [3] 8
Figura 3. Ejemplo de clasificación de imágenes [4] 9
Figura 4. Ejemplo de detección de objetos [4] 10
Figura 5. Segmentación semántica vs Segmentación por instancia [5] 11
Figura 6. Ejemplo de detección de puntos clave [6] 11
Figura 7. Operación de convolución [8] 16
Figura 8. Conv3D [9] 17
Figura 9. Estructura de las RNN [10] 17
Figura 10. Capa densa [11] 18
Figura 11. Comparación de los modelos de clasificación de imágenes [12] 19
Figura 12. Módulo Inception [13] 20
Figura 13. Función identidad en modelo ResNet [15] 21
Figura 14. Salida del YOLO [17] 22
Figura 15. Intersection Over Union [18] 22
Figura 16. Detección de puntos clave [19] 24
Figura 17. Detección de acciones [20] 25
Figura 18. Ejemplo de Optical Flow [24] 28
Figura 19. Imágenes de acciones ocultando a la persona 31
Figura 20. Curvas de entrenamiento y validación [33] 38
Figura 21. Ejemplo de recortado aleatorio [34] 41
Figura 22. Ejemplo de volteado de imagen [35] 41
Figura 23. Ejemplo de enmarcado [36] 42
Figura 24. Ejemplo de traslación [36] 42
Figura 25. Ejemplo de borrado aleatorio [37] 43
Figura 26. Ejemplos de cambios en los canales [38] 44
Figura 27. Ejemplo de difuminado [39] 45
Figura 28. Ejemplo de inyección de ruido [39] 45

Figura 29. Ejemplo de imagen (izquierda) y su adversaria (derecha) [40] 46
Figura 30. Efecto de normalizar los datos [41] 48
Figura 31. Dimensiones de una imagen [42] 50
Figura 32. Escalado con Nearest Neighbor [44] 51
Figura 33. Escalado con el método Bilinear [43] 51
Figura 34. Escalado mediante Bicubic Interpolation [43] 52
Figura 35. Número de canales en RGB y en escala de grises 53
Figura 36. Comparativa de los métodos de conversión a escala de grises [45] 55
Figura 37. Estructura de TensorFlow [46] 57
Figura 38. Ejemplo de gráfico computacional [46] 59
Figura 39. Métodos de Keras para la creación de modelos de Machine Learning [48] 60
Figura 40. Estructura de carpetas para DatasetFolder [53] 68
Figura 41. Transfer Learning [54] 72
Figura 42. Congelación de capas [54] 76
Figura 43. Código de Transfer Learning 77
Figura 44. Estructura del modelo desarrollado 78
Figura 45. Max Pooling y Average Pooling [55] 80
Figura 46. Dropout [56] 81
Figura 47. Conv 1x1 [34] 82
Figura 48. Flattening 83
Figura 49. Celda LSTM [57] 83
Figura 50. Tamaño de las características 84
Figura 51. Estructura del modelo con ResNet 86
Figura 52. Overfitting [58] 88
Figura 53. Ajuste a los datos según la complejidad del modelo [59] 89
Figura 54. Regularización L1 [60] 90
Figura 55. Regularización L2 [60] 91
Figura 56. Estructura de una CPU y una GPU [61] 93
Figura 57. Comparativa de computación de CPU y GPU [62] 94
Figura 58. Tiempo de entrenamiento con CPU y GPU [63] 95

Figura 59. Fórmula de Accuracy 97
Figura 60. Fórmula de "Precision" 98
Figura 61. Matriz de confusión [64] 99
Figura 62. Sensibilidad 99
Figura 63. Especificidad 100
Figura 64. F1-Score 100
Figura 65. Curvas de precisión y pérdidas [65] 101
Figura 66. Overfitting [33] 103
Figura 67. Underfitting [33] 104
Figura 68. Underfitting [33] 105
Figura 69. Conjunto de entrenamiento no representativo [33] 106
Figura 70. Conjunto de validación no representativo [33] 107
Figura 71. Efecto de normalizar los datos [41] 107
Figura 72. Mapas de activación [66] 109
Figura 73. Learning Rate 111
Figura 74. Función Sigmoide [67] 113
Figura 75. Función Tangente Hiperbólica [68] 114
Figura 76. Función ReLU [67] 115
Figura 77. ReLU vs Leaky ReLU [67] 116
Figura 78. Grid Search [69] 117
Figura 79. Random Search [69] 118
Figura 80. Código de Mixed Precission [72] 120
Figura 81. Resultados del modelo Conv2D + LSTM 122
Figura 82. Resultados del modelo ResNet18 + LSTM 123



CAPÍTULO I: INTRODUCCIÓN Y OBJETIVOS

En la actualidad, es creciente el interés por desarrollar modelos de Inteligencia Artificial que faciliten las tareas que debe desempeñar el hombre. Cada vez son mayores y más variados los problemas cuya solución se busca mediante la aplicación de este tipo de algoritmos, los cuales van desde la detección personas y objetos en imágenes hasta la predicción de la bolsa o conducción de coches autónomos. Pese a que pueda parecer que la Inteligencia Artificial acaba de aterrizar en el mundo, esto no es así. El primer modelo de red neuronal fue diseñado por Frank Rosenblatt hace ya más de sesenta años; más concretamente en 1957 [1]. Este modelo presentaba por primera vez el Perceptrón, que era capaz no solo de aprender, sino también de generalizar, y aún se utiliza en la actualidad para el reconocimiento de patrones.

A pesar de la "antigüedad" de este tipo de algoritmos, estos no han producido un impacto real en la sociedad hasta tiempos recientes. Esta relativa tardanza en la aplicación de modelos basados en el diseñado por Rosenblatt se debe a los requerimientos computacionales que este tipo de modelos presenta, y los cuales no se han conseguido a un bajo coste hasta hace tan solo unos pocos años. Esta capacidad computacional crece exponencialmente en la actualidad, y está permitiendo crear modelos de Inteligencia Artificial cada vez más complejos, y que permiten llevar a cabo acciones más complicadas. Este incremento se puede observar en la siguiente gráfica, en la cual se muestra el aumento de la velocidad de reloj de los procesadores desde 1970:

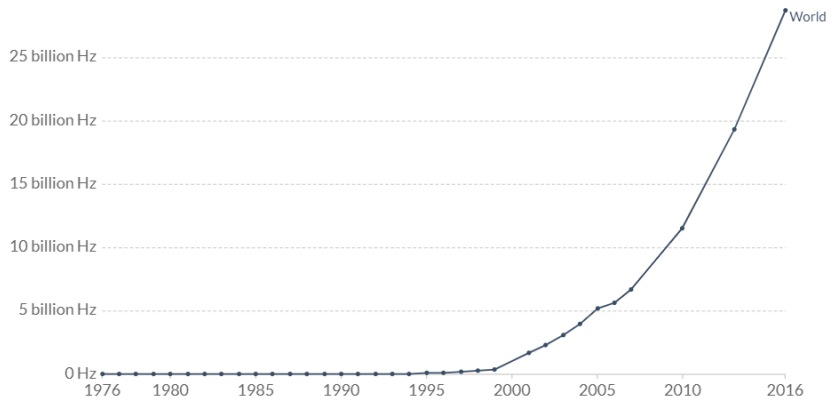


Figura 1. Velocidad de reloj en los procesadores [2]

Tal y como se acaba de mencionar, el hecho de tener acceso a equipos con una capacidad computacional mucho más elevada que hace unos pocos años a un precio asequible ha hecho que los modelos de Inteligencia Artificial hayan podido crecer en tamaño y complejidad. Esto llevó a la creación del denominado Deep Learning. Este tipo de redes neuronales se caracterizan por tener una capacidad de abstracción mayor que las redes neuronales convencionales. El hecho de tener muchas capas interconectadas hace que los cálculos que se tienen que llevar a cabo para entrenar dichas redes sean mucho más complejos y, por tanto, requieran del uso de unidades de procesamiento con una capacidad mucho mayor. El hecho de que en la actualidad dispongamos de los medios para poder entrenar este tipo de red neuronal con una gran cantidad de datos, ha permitido que se popularice la investigación y el uso de modelos de Deep Learning, y que se utilicen para más y más aplicaciones de diferente índole.

El Deep Learning ha supuesto un cambio de paradigma, ya que antes de la aparición de este tipo de modelos, era necesario extraer las características de los datos previamente a entrenar el modelo de Inteligencia Artificial, lo cual hacía el trabajo más complejo y sesgaba la toma de decisiones hacia las características que los desarrolladores consideraban importantes. Con la llegada del Deep Learning, ya no es necesario extraer estas características, sino que gracias a la profundidad de las redes (es decir, al número de capas que estas contienen conectadas unas con otras), son capaces de extraer dichas características por sí mismas, llegando a comprender cuáles son las relevantes a la hora de tomar decisiones.

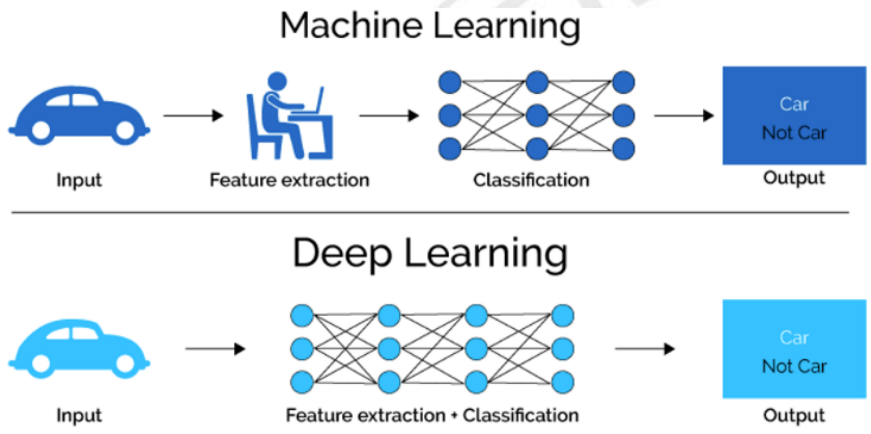


Figura 2. Machine Learning vs Deep Learning [3]

Esto ha permitido no solo facilitar mucho el proceso de creación de los modelos, sino que además permite evitar el posible sesgo introducido por los diseñadores del mismo, ya que ahora es la propia red la que descubre las características en los datos que considera que son determinantes a la hora de predecir o clasificar la salida. Es por todo esto por lo que el Deep Learning está cobrando tanta popularidad en la actualidad.

Estos modelos son utilizados para múltiples soluciones: predecir valores futuros, clasificar los datos de entrada, comprender el contenido de un texto, detección de objetos en imágenes.... Son muchas las aplicaciones del Deep Learning, pero la que se va a tratar en profundidad en este documento es la denominada Visión Artificial. Esta consiste en la creación de modelos capaces de analizar imágenes y extraer información de éstas. Hay mucha información diferente que se puede extraer de una imagen, y es por ello por lo que existen diferentes familias de modelos de Visión Artificial. Las principales aplicaciones o familias de modelos son las siguientes:

- **Clasificación de imágenes**: el objetivo de este tipo de algoritmos es clasificar una imagen según su contenido, es decir, llegar a comprender qué es lo que aparece en dicha imagen y ser capaz de clasificar la escena. Por ejemplo, si poseemos un dataset compuesto por imágenes de perros y gatos, podemos crear un clasificador capaz de establecer cuál de estos dos animales aparece en la imagen.



Figura 3. Ejemplo de clasificación de imágenes [4]

- **Detección de objetos**: a diferencia de la familia anterior, este tipo de modelos no clasifica una imagen en su conjunto, sino que es capaz de detectar los diferentes objetos que aparecen en la misma. Para ello, crea unas cajas ("bounding boxes") que sitúa sobre cada uno de ellos, junto con la clase detectada (tipo de objeto) y la

confianza con la que detecta al mismo. Por ejemplo, si disponemos de imágenes de una calle, podríamos detectar los coches, personas, animales, farolas, etc. que aparezcan en la misma.

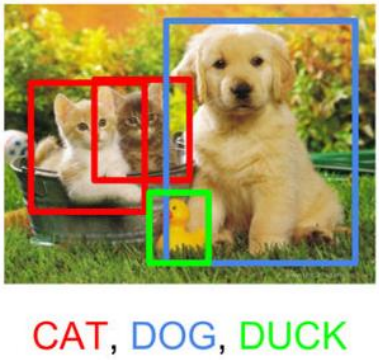


Figura 4. Ejemplo de detección de objetos [4]

- **Segmentación de imágenes**: este tipo de modelos es parecido al de detección de objetos, pero en este caso, en vez de situar una caja alrededor del objeto detectado se especifican todos los píxeles de la imagen que corresponden a dicho objeto. Esto permite no solo detectar el objeto, sino además conocer la forma exacta que tiene el mismo y el área que ocupa en la imagen. Existen dos tipos de modelos de segmentación de imágenes: segmentación semántica y segmentación por instancia. El primero clasifica un objeto por su tipo, mientras que el segundo no solo hace esto, sino que permite diferenciar los diferentes objetos de un mismo tipo que aparecen en una imagen. Por ejemplo, si en una imagen aparecen varias personas, la segmentación semántica clasificará todos los píxeles que corresponden a las personas con la etiqueta de persona, mientras que la segmentación por instancia será capaz de diferenciar entre cada una de las personas, etiquetando a cada una con una etiqueta distinta dentro de la clase persona.

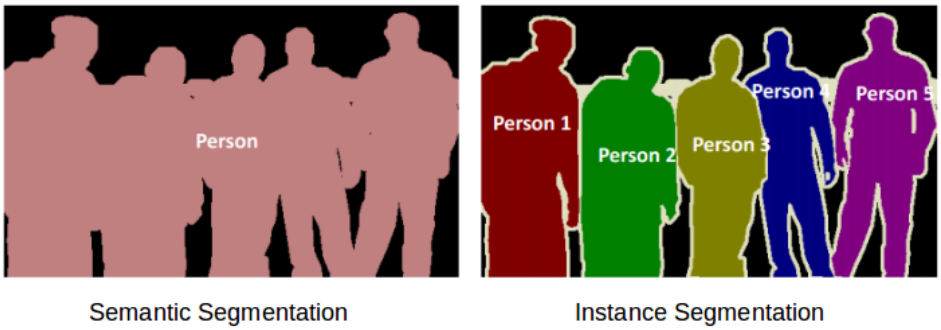


Figura 5. Segmentación semántica vs Segmentación por instancia [5]

- **Detección de puntos clave**: este tipo de algoritmos tiene por objetivo conseguir detectar puntos clave en una imagen, es decir, puntos característicos que se presentan en la misma. Estos modelos son de gran utilidad para detectar la pose de una persona (en el cual se detectan las articulaciones de la misma para ser capaz de determinar cómo están situadas sus extremidades) o realizar una tarea de reconocimiento facial (en la cual se detectan puntos clave del rostro como los extremos de los ojos, nariz boca, etc).

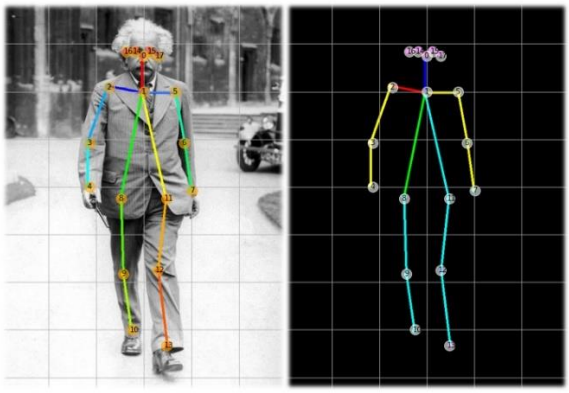


Figura 6. Ejemplo de detección de puntos clave [6]

Tal y como se puede intuir, lo que caracteriza las soluciones de Visión Artificial del resto de soluciones de Deep Learning es el tipo de dato con el que se debe tratar, ya que los datos a analizar son imágenes. Esto añade una complejidad extra, ya que cada imagen contiene mucha información, lo que hace aumentar las necesidades de memoria para desarrollar este tipo de soluciones. Para poder

ilustrar esto, tomemos como ejemplo una imagen que tomamos con un teléfono móvil. Un móvil medio actualmente es capaz de realizar imágenes en Full-HD, es decir, con unas dimensiones de 1920x1080 píxeles. Sabiendo que cada píxel es un valor numérico y que en cada imagen hay 3 canales (RGB), podemos calcular el número de valores numéricos que tiene la imagen: 1920*1080*3 = 61.624.800 valores numéricos. Es decir, cada imagen que sacamos con nuestro teléfono móvil con cámara Full-HD está compuesto por 61.624.800 valores.

Este ejemplo ha permitido ver las dimensiones de los datos que se manejan, pero cabe mencionar que en las soluciones de Visión Artificial no se suelen manejar imágenes con tanta resolución, siendo la dimensión más común 224x224x3, las cuales suponen un total de 150.528 valores por cada imagen. Si tratáramos de crear un modelo con una red totalmente conectada o densa ("Fully Connected Layer" o "Dense Layer"), la cual era la más común antes de la aparición de la Visión Artificial, necesitaríamos 150.528 neuronas en la capa de entrada, es decir, un tamaño de red desorbitado. Debido a esta problemática a la hora de analizar imágenes con las redes más clásicas, surgieron las denominadas Redes Neuronales Convolucionales.

Las Redes Neuronales Convolucionales (CNN) son un tipo de Red Neuronal diseñadas para el análisis de imágenes. Este tipo de redes se basan en una operación denominada convolución, de la cual reciben el nombre. Esta operación, que se explica en la sección de estado del arte en este documento, permite reducir drásticamente el número de parámetros de la red. Haciendo uso del ejemplo anterior que implicaba usar 150.528 neuronas en la capa de entrada (y por tanto 150.528*2=301.056 parámetros únicamente en esta primera capa), si usamos una capa convolucional de dimensiones 7x7x3, pasaríamos a tener 147 parámetros en esta capa. Esto reduce en gran medida la capacidad computacional necesitada para entrenar esta red. Es por ello que podemos decir que las Redes Neuronales Convolucionales han hecho posible aplicar la Inteligencia Artificial al análisis de imágenes, y son las que se utilizan en este proyecto con dicha finalidad.

Tal y como se ha comentado, este tipo de Redes Neuronales permiten resolver uno de los problemas que la Visión Artificial trata de resolver; analizar imágenes. Sin embargo, esta técnica también tiene por objetivo analizar vídeos, es decir, secuencias de imágenes. En esta aplicación más compleja de la Visión Artificial, no basta con usar únicamente este tipo de redes, sino que se deben desarrollar modelos más complejos que permitan no solo extraer patrones de los fotogramas del vídeo

(es decir, de las diferentes imágenes que lo componen), sino que además se debe llevar a cabo un estudio de la relación de dichos patrones en el tiempo. Esta necesidad es la que ha llevado al desarrollo de capas como la Convolucional 3D (la cual se explica en la sección de estado del arte en este documento) o el modelo Convolucional 2D + LSTM, que es el desarrollado en este proyecto.

El análisis de vídeos mediante técnicas de Visión Artificial presenta una solución compleja pero interesante a muchos problemas que actualmente debe resolver el hombre sin ayuda de las máquinas. Este tipo de técnicas están en auge en la actualidad, habiendo un gran interés por parte de la comunidad de desarrolladores debido a sus potenciales aplicaciones. La aplicación con mayor popularidad en la actualidad es la detección de acciones en cámaras de videovigilancia.

En el mundo hay millones de cámaras de videovigilancia destinadas a asegurar que las calles o locales en los que están instalados son seguros. Actualmente, todas estas cámaras requieren de la supervisión constante de una persona, que debe asegurarse de que no hay actividad anómala en ninguna una de las cámaras, y por lo tanto que no está ocurriendo ningún crimen ni hay nadie en peligro. El hecho de desarrollar un modelo capaz de analizar lo que está ocurriendo en cada cámara de seguridad y extraer conclusiones de dichos vídeos supondría un ahorro de millones de euros en todo el mundo, además de conseguir que una máquina realice un trabajo ingrato como es la supervisión constante de estas cámaras. Es por ello por lo que es creciente el interés por lograr un modelo estable capaz de resolver este problema.

Un gran problema del que no se debe perder foco al desarrollar modelos de Inteligencia Artificial es el de la privacidad de los datos que se manejan. En el caso de este tipo de soluciones, es especialmente crítico debido a que se trata de grabaciones de personas. Esta necesidad de proteger la privacidad hace que no sea fácil conseguir los datos necesarios para desarrollar una solución de este tipo, ya que estos datos, por razones de protección de imagen, no son públicos. Esto es algo que está ralentizando la creación de este tipo de modelos.

Debido al gran potencial que la Visión Artificial presenta, el interés por este tipo de modelos va en aumento, lo cual está llevando a la creación de cada vez más modelos y más robustos. Es tal su popularidad que alrededor del mundo se realizan hackatones en las que participan cientos de equipos, y cuyo objetivo es diseñar un modelo que presente los mejores resultados para una base de datos dada.

En definitiva, esta tecnología ya forma parte de nuestras vidas, y ha llegado para quedarse.

Ahora que se han introducido los temas que se tratarán en este documento, cabe hablar sobre los objetivos establecidos para este proyecto. Dichos objetivos son los siguientes:

- Desarrollar un modelo de detección de acciones en vídeo lo más robusto posible y agnóstico a la calidad del vídeo, es decir, que su funcionalidad no se vea comprometida por la calidad del vídeo de entrada.
- Desarrollar un modelo agnóstico al entorno, es decir, que el modelo no se base en el entorno en el que se desarrolla el vídeo para predecir la acción.
- Desarrollar un modelo que permita analizar vídeos de diferentes longitudes.
- Implementar una metodología de pre procesado de vídeos genérica, es decir, que acepte cualquier formato y tamaño de vídeo.
- Conseguir una precisión del 75% en el set de entrenamiento y del 65% en los sets de validación y test.

CAPÍTULO II: ESTADO DEL ARTE

Tal y como se ha comentado en el apartado de introducción de este documento, la Inteligencia Artificial está a la orden del día en la sociedad actual. Estando este campo de actualidad, cabe destacar la gran popularidad que ha adquirido el Deep Learning en los últimos años. Este tipo de algoritmos está siendo de gran utilidad por ejemplo en el campo de la medicina, donde múltiples modelos están siendo utilizados para la detección de diferentes enfermedades mediante el análisis de los resultados obtenidos por el paciente en pruebas médicas convencionales.

El campo del Deep Learning, tal y como se ha comentado, es relativamente nuevo. Se inició principalmente a raíz de tres publicaciones: Hinton et al, 2016; Bengio et al, 2007 y Ranzato et al, 2007 [7]. Lo que estas publicaciones presentaban y que los distinguían del estado del arte del momento, era el uso de capas intermedias en las redes neuronales. Esto dotó a estas nuevas redes de la capacidad de llevar a cabo clasificaciones no lineales complejas y de extraer atributos relevantes de los datos en bruto, evitando así que le usuario tenga que llevar a cabo de forma manual la extracción de características de los datos antes de introducirlos a la red.

El hecho de disponer de capas intermedias ha permitido la creación de modelos que incluyen diferentes tipos de capas de red neuronal, pudiendo así extraer diferentes características de los datos de entrada según el problema que se pretenda resolver. Los principales tipos de capa utilizados en Deep Learning en la actualidad son los siguientes:

- **CNN (Convolutional Neural Network):** este tipo de capa permite analizar imágenes de entrada mediante el uso de una operación de convolución. El funcionamiento de estas capas consiste en crear una matriz (denominada matriz de convolución y que inicialmente tiene pesos creados de forma aleatoria) que realice una operación de convolución sobre los diferentes píxeles de una imagen, deslizándose a través de esta para ir obteniendo resultados sobre las diferentes partes de la imagen.

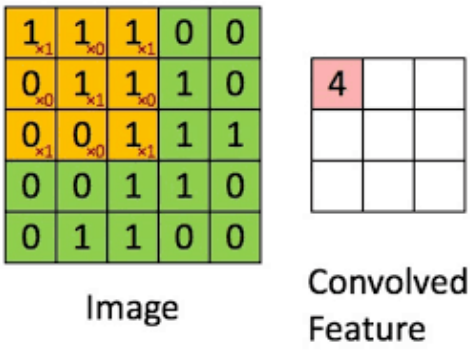


Figura 7. Operación de convolución [8]

Estas capas se emplean para extraer características de las imágenes. En la actualidad, la aplicación que se le está dando es la detección de diferentes objetos presentes en una imagen y la identificación de personas. Una limitación que presenta este tipo de capa es que solo permite analizar un fotograma. Esto implica que no pueden ser utilizadas por sí solas en el análisis de una secuencia de fotogramas, es decir, en el análisis de vídeos. Esto choca con el actual interés por generar modelos que permitan analizar vídeos, y es por ello por lo que se ha creado una variante de la red convolucional que permite analizar una secuencia de fotogramas, no solo extrayendo las características de cada uno de ellos por separado sino también información de su conjunto. Este nuevo modelo de CNN, denominado Convolucional 3D, consiste en la generación de filtros con profundidad. Esto permite pasar por cada filtro (de profundidad n * número de canales) un número n de imágenes que son analizadas al mismo tiempo por dicho filtro. Como se puede apreciar, a diferencia de la CNN convencional, en lugar de tener únicamente una matriz, el filtro de la capa es un conjunto de matrices que forman un tetraedro. Esto se ve representado en la siguiente imagen:

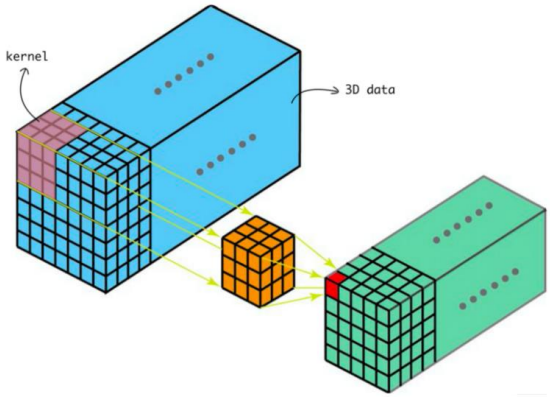


Figura 8. Conv3D [9]

- **RNN (Recurrent Neural Network):** este tipo de capa permite extraer características de una secuencia de datos, es decir, tiene como entrada un conjunto de datos secuenciales y consigue extraer características temporales de los mismos. Esto se consigue utilizando parte de la información extraída del dato anterior en el dato posterior. De esta manera, no solo influirá el dato en sí en la salida de la red, sino también la información obtenida del dato anterior. Esta estructura mencionada se presenta a continuación:

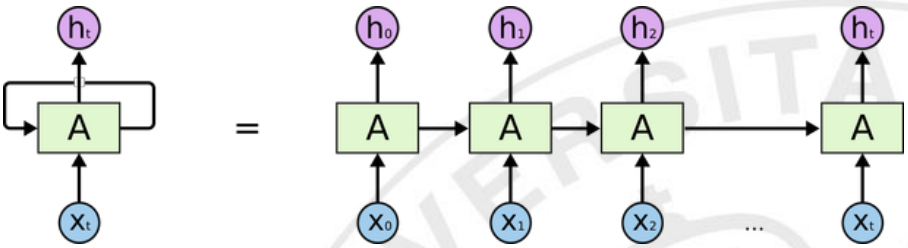


Figura 9. Estructura de las RNN [10]

Esta capacidad de extraer información temporal ha hecho que este tipo de capas se utilicen comúnmente en el análisis de vídeos, ya que, combinadas con capas Convolucionales, permiten extraer información de la relación secuencial que siguen las imágenes de un vídeo, pudiendo así clasificar las acciones que están sucediendo en el mismo.

- **Dense Layer or Fully Connected Layer:** es una capa totalmente conectada, es decir, una capa de neuronas en la que el dato de entrada se conecta a todas y cada una de las neuronas que conforman la capa. Este tipo de capas permiten llevar a cabo tareas de clasificación. Tal y como se puede observar en la siguiente imagen, lo que caracteriza a este tipo de capas es que se conectan totalmente a la capa anterior, es decir, cada neurona de la capa i obtiene información de todas las neuronas de la capa i-1. Además, en el caso de la capa de entrada, hay una neurona por cada uno de los datos.

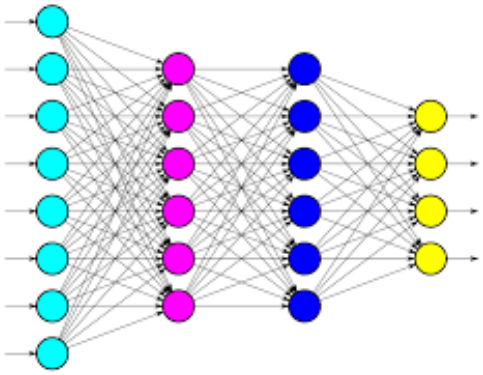


Figura 10. Capa densa [11]

Ahora que se han presentado las principales capas empleadas en el diseño de los actuales modelos de Deep Learning, cabe hablar de los algoritmos que se están desarrollando con ellas y las aplicaciones a las que están destinados los mismos. Este documento se centrará en el estudio de la denominada Visión Artificial, dentro de la cual cabe diferenciar entre los algoritmos destinados a analizar imágenes y los destinados a analizar vídeos.

En lo que a extracción de características en imágenes se refiere, y tal y como se ha comentado en la introducción de este documento, los algoritmos más recientes que se han creado en esta vertiente del Deep Learning son los de clasificación de imágenes, detección de objetos, segmentación de imágenes y detección de puntos clave.

CLASIFICACIÓN DE IMÁGENES

La clasificación de imágenes es la rama más desarrollada de la Visión Artificial, tanto por ser la más sencilla de resolver como por ser aquella en la que es más fácil etiquetar los datos. Son múltiples los modelos existentes para resolver este problema, el cual goza de gran popularidad. Entre los modelos más usados actualmente encontramos AlexNet (2012), GoogLeNet (Inception) (2014), VGGNet (2014) y ResNet (2015).

Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

Figura 11. Comparación de los modelos de clasificación de imágenes [12]

De entre estas redes, cabe destacar la novedad introducida en las ResNet y el módulo de Inception introducido por GoogLeNet.

GoogLeNet introdujo el módulo Inception, el cual era totalmente novedoso dado el estado del arte del momento. Lo que se pretendía conseguir con este novedoso módulo es conseguir aumentar el tamaño de la red sin incurrir en tantos costes computacionales, de forma que el crecimiento de la red se llevara a cabo de una forma más eficiente computacionalmente. Para ello, se crearon los módulos Inception, que pretendían crecer la red con un incremento menor en los requerimientos computacionales que si simplemente se añadieran capas una detrás de otra. Este módulo se representa a continuación:

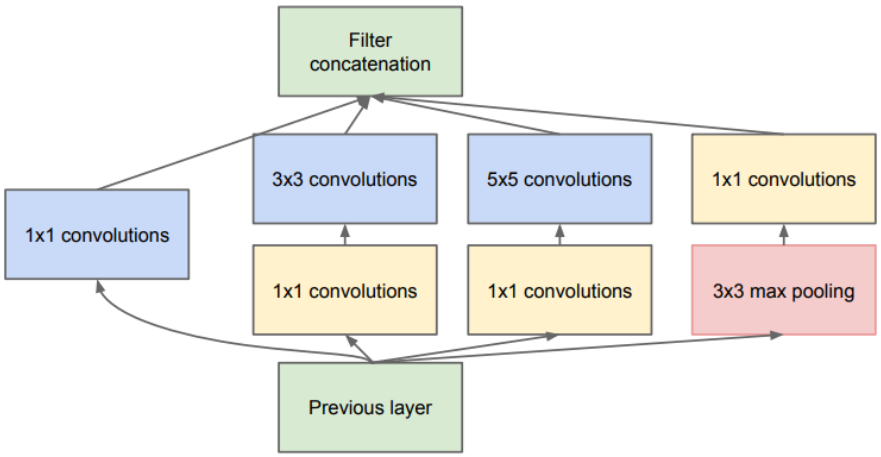


Figura 12. Módulo Inception [13]

Con esta técnica de separar, transformar y volver a unir, se no solo se consiguen los beneficios computacionales mencionados, sino que además permite extraer información de la imagen en diferentes escalas

Algo que se puede observar en el diseño de estas redes es la clara tendencia a crear redes cada vez más profundas (AlexNet está formada por 5 capas convolucionales, VGG por 19 y GoogLeNet por 22). Esto se debe a que, a pesar de que el Teorema de Aproximación General establezca que una red neuronal con una sola capa es capaz de representar cualquier función (conteniendo un número elevado de neuronas para ciertas funciones), el diseño de capas muy grandes lleva al sobre aprendizaje de la red ("overfitting") mucho antes que si se crean más capas más pequeñas. [14]

Por otro lado, el hecho de hacer una red demasiado profunda introduce un nuevo problema que degrada la precisión del modelo: el problema de desvanecimiento de gradiente. Esto es un fenómeno que ocurre en las redes profundas, y que consiste en que tras el cálculo de la regla de la cadena durante el proceso de "backpropagation", si los gradientes de las capas finales son pequeños, harán que el gradiente de las capas iniciales se reduzca exponencialmente según nos propagamos por la red (ya que se multiplican muchos valores pequeños muchas veces). Esto hace que el aprendizaje pase a ser muy pequeño, debido a que al hacer el descenso del gradiente no nos distanciamos mucho de la posición actual.

El modelo ResNet se creó con el objetivo de eliminar este problema, de forma que se permitiera crear modelos muy profundos evitando esta limitación en el entrenamiento. Para ello, sus creadores introdujeron la operación identidad:

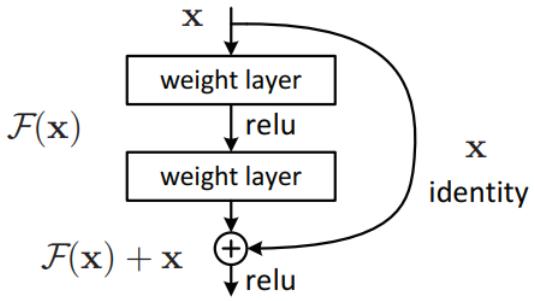


Figura 13. Función identidad en modelo ResNet [15]

Este bloque residual basado en la función identidad consiste en crear un atajo entre la imagen inicial, de forma que se sume a su transformada tras parar por varios bloques convolucionales.

DETECCIÓN DE OBJETOS

El objetivo de los modelos de este tipo consiste en localizar diferentes objetos en una imagen. Los objetos que detecte el modelo dependen del dataset con el que ha sido entrenado el mismo, existiendo modelos preentrenados para los objetos más comunes sobre los que se quiere realizar una detección. De entre los modelos más utilizados actualmente, podemos encontrar Fast R-CNN, Faster R-CNN, YOLO (You Only Look Once) y SSD (Single Shot Detection).

De entre los mencionados, cabe destacar el reciente desarrollo de una familia de modelos denominada YOLO (You Only Look Once). La primera aparición de este tipo de algoritmos es reciente, ya que se presentaron por primera vez en una publicación en 2016 [16]. Es tal el potencial de este tipo de modelos, que la comunidad de investigación se está volcando en su desarrollo, habiendo creado desde la primera publicación 4 versiones (cada cual más robusta y profunda) de esta familia de modelos. Este tipo de modelos tiene como entrada una imagen, y presenta como salida dicha imagen con cajas que bordean los objetos, indicando de qué objeto se trata cada uno de ellos. Estas

cajas se denominan “Bounding Boxes”. A continuación, se presenta un ejemplo de la salida de un modelo YOLO:

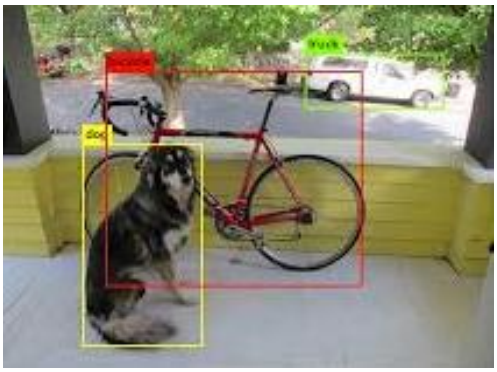


Figura 14. Salida del YOLO [17]

La evaluación de la precisión de este tipo de modelos es diferente a la del resto, ya que en este caso lo que interesa evaluar es el solape entre la caja real que rodea al objeto y la caja generada por le modelo, para lo cual se usa una operación denominada IoU (Intersection Over Union). Esta operación consiste en calcular el ratio entre el solape de la caja real y la caja predicha por el modelo y la suma del tamaño de ambas cajas. Esta operación se puede observar a continuación:

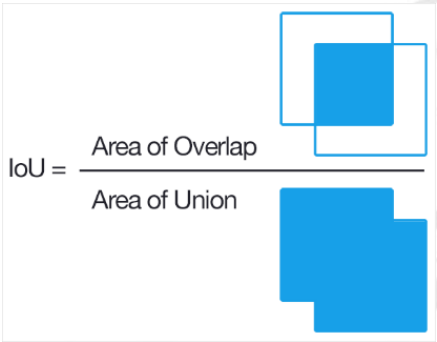


Figura 15. Intersection Over Union [18]

SEGMENTACIÓN DE IMÁGENES

Este tipo de algoritmos de Visión Artificial, están cogiendo más y más popularidad debido a su uso en los coches de conducción autónoma. Eso está impulsando el desarrollo de modelos capaces de dar una respuesta en tiempo real acerca de los diferentes objetos presentes en una imagen.

Entre los modelos actuales, podemos encontrar U-Net, Fast FCN (Fast Fully Connected Network) y Mask R-CNN.

Este último modelo mencionado es una extensión del modelo de detección de objetos Faster R-CNN. Es por ello que la salida que este modelo genera consiste en unas cajas (al igual que el Faster R-CNN) y una clasificación a nivel de píxel de qué objeto hay situado en cada posición de la imagen. Este modelo consigue un procesamiento de imágenes de hasta 5 fotogramas por segundo, lo cual permite analizar vídeos en tiempo real.

En la actualidad, hay varias formas de llevar a cabo una segmentación: [5]

- En base a la región: una primera solución a este problema es basándonos en la diferencia entre dos píxeles consecutivos. Dos objetos diferentes tendrán colores diferentes en la imagen, por lo que, si hay una diferencia clara entre el color de dos píxeles consecutivos, podremos concluir que se trata de objetos distintos.
- En base a los bordes: todos los objetos tienen bordes, por lo que la forma de encontrar la frontera entre un objeto y su aledaño puede ser detectando bordes en la imagen.
- Usando clustering: cada píxel en un objeto tiene cercanía con el resto de píxeles de ese mismo objeto, además de cierto parecido en lo que se refiere al valor del píxel. Es por ello que se puede implementar una solución de clustering para resolver este problema

Algo a tener en cuenta a la hora de desarrollar este tipo de modelos es la dificultad que conlleva etiquetar un dataset destinado a esta solución, ya que se debe marcar en cada una de las imágenes cada uno de los objetos que se desean diferenciar.

DETECCIÓN DE PUNTOS CLAVE

Al igual que en los modelos destinados a segmentación de imágenes, los modelos que pretenden resolver el problema de detección de puntos clave implican la necesidad de dedicar más tiempo a la creación del dataset que en el caso de detección de objetos o clasificación de imágenes. Esto se debe a que hay que etiquetar cada uno de los puntos clave que se desean detectar en la imagen. En el caso de detección de pose, se suelen situar unos 18 puntos por persona, los cuales permiten identificar la postura de la persona. Estos puntos se deben etiquetar en cada una de las personas que aparecen en la imagen.

Cabe mencionar que este tipo de algoritmos se emplean también para el reconocimiento facial y el estudio de expresiones humanas, para lo cual se detectan diferentes puntos clave en la cara de la persona y se estudia la posición de cada uno de ellos.

Algunos algoritmos a destacar para esta solución son OpenPose, AlphaPose y Keypoint R-CNN, los cuales conforman el estado del arte actual.



Figura 16. Detección de puntos clave [19]

Ahora que se han descrito los últimos algoritmos diseñados para cada uno de los métodos de análisis de imágenes (los cuales se emplean también para el desarrollo de modelos de análisis de vídeo), cabe hablar de los algoritmos de Visión Artificial que analizan vídeos, ya que el objetivo de este proyecto es desarrollar un modelo capaz de detectar acciones en vídeo. Esta es la aplicación más reciente dentro del campo de la Visión Artificial, y la que aún está en una fase más inicial de desarrollo. Un ejemplo de detección de acciones se presenta a continuación:

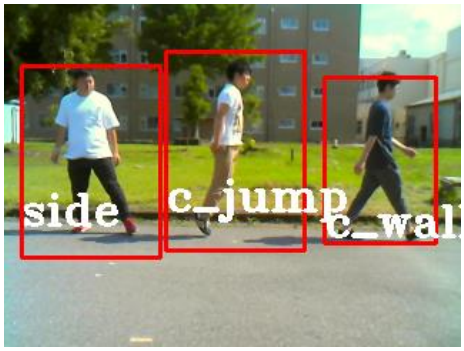


Figura 17. Detección de acciones [20]

A diferencia de los otros dos tipos de modelo que se acaban de presentar, en este caso cada dato no está compuesto por una única imagen, sino por un conjunto de ellas que se suceden en el tiempo. Esto implica que los datos a manejar son mucho más pesados que en el caso de análisis de imágenes, lo que hace eleva las necesidades computacionales y de almacenamiento.

La siguiente sección está centrada en la aplicación concreta de detección de acciones en vídeo, ya que es el problema que se pretende resolver en este proyecto. Hay dos tipos de enfoque distintos a la hora de llevar a cabo esta tarea: analizar cada fotograma de forma aislada o analizar tanto los fotogramas como la secuencia de estos; esto es, comprendiendo la secuencia temporal.

DETECCIÓN DE ACCIONES ESTUDIANDO CADA FOTOGRAMA POR SEPARADO

En la actualidad, la mayoría de los modelos de detección de acciones en vídeo se limitan a estudiar cada fotograma por separado, pero esto podría llevar a no comprender bien qué acción se está llevando a cabo realmente sino a clasificar la acción estudiando la pose de la persona en ese instante o el entorno en el que se ubica y los objetos que está usando la persona. Por ejemplo, si se introduce en el modelo una imagen de una persona haciendo kayak, el modelo verá que la persona se encuentra en el río y que tiene en sus manos unos remos de kayak, lo cual le ayudará a determinar la acción. [21]

Este tipo de modelos se basan en el uso de Redes Convolucionales, que permiten extraer información de la imagen, pero se pierde totalmente la información acerca de cómo se está moviendo la persona o los objetos en el vídeo. De este modo, los modelos se basan en la extracción de

características estáticas como la pose, el entorno y los objetos que rodean a la persona. Para ello se utilizan las técnicas de análisis de imágenes explicadas anteriormente.

Este tipo de modelos son más sencillos de desarrollar que los que pretenden comprender cómo evolucionan la escena en el tiempo, pero tienen peor precisión potencialmente y no son capaces de resolver el problema de clasificar una acción en un entorno inesperado, por ejemplo, hacer gimnasia en un campo de futbol.

Tal y como se ha comentado en el apartado de introducción de este documento, uno de los grandes motivos por los que la detección de acciones está ganando popularidad es por su potencial uso en sistemas de seguridad, donde estos modelos podrían ser de gran apoyo en la vigilancia en tiempo real de las diferentes ubicaciones controladas por cámaras de vídeo vigilancia. En esta gran aplicación de la Visión Artificial, estos modelos no tienen la suficiente precisión, ya que no tienen el apoyo del entorno a la hora de determinar la acción de la persona y, al no extraer información temporal, no disponen de información suficiente para determinar la acción que está llevando a cabo cada una de las personas que aparecen en la escena.

Este método de estudio de cada fotograma por separado permite sin embargo hacer seguimiento de personas en vídeo. Esto consiste en generar un vector de 128 características de la persona (al igual que en reconocimiento facial) y asignarle un ID a cada persona, de forma que en cada fotograma se detectan a todas las personas presentes y se comprueba la identidad de cada una, pudiendo así hacer un seguimiento a lo largo de los diferentes fotogramas de un vídeo.

DETECCIÓN DE ACCIONES ANALIZANDO LOS FOTOGRAMAS Y SU RELACIÓN TEMPORAL

Este enfoque más complejo permite extraer información no solo de cada fotograma separado, sino además de cómo evoluciona la escena. Esta forma de afrontar el problema de detección de acciones es más completa que la anterior, y permite centrarse realmente en la acción humana y no tanto en la situación en la que se encuentra en la imagen.

Actualmente se están empleando dos métodos distintos a la hora de analizar vídeos de esta manera: un primer enfoque que consiste preprocesar las imágenes para extraer características de estas

para posteriormente introducir los datos procesados en un modelo, y un segundo enfoque que consiste en usar solamente Deep Learning con los datos en bruto, y hacer que sea el propio modelo el que extraiga las características.

El primer método mencionado consiste en realizar un procesado previo al entrenamiento que permita mostrar únicamente la información relevante de las imágenes, y emplear dichos nuevos datos para entrenar una red neuronal. Un ejemplo de esto es la publicación "Real-Time Action Detection in Vídeo Surveillance using Sub-Action Descriptor with Multi-CNN" [22]. En este caso se propone procesar cada fotograma para extraer de cada uno de ellos la posición y movimiento de la persona, para posteriormente alimentar una red Convolucional con estos nuevos fotogramas procesados.

El segundo método mencionado permite emplear los vídeos en bruto, es decir, sin la necesidad de realizar ninguna transformación previa en los datos de entrada. Para solucionar el problema de clasificación de esta manera, se están empleando estructuras de red neuronal como la Conv3D, que permite realizar un análisis espaciotemporal de los vídeos. Estos modelos están surgiendo desde hace tan solo 3 años y se encuentran aún en fase de investigación y desarrollo, y es el tipo de modelo que se pretende diseñar en este proyecto. Algunos de los modelos que cabe mencionar son el Conv + LSTM (el cual se ha desarrollado en este proyecto), Conv3D, Two-Stream Networks y Two-Stream Inflated 3D ConvNets. [23]

El uso de Deep Learning ha permitido combinar varios tipos de redes para extraer las características deseadas tanto de los fotogramas como de la componente temporal. Ejemplo de esto es la posibilidad de detectar puntos clave de las personas (sus articulaciones) y estudiar su evolución en el tiempo o incluso llevar a cabo una segmentación de la imagen y extraer los píxeles que corresponden a las personas para posteriormente analizarlos temporalmente.

Además de estos algoritmos de Deep Learning que se han mencionado, cabe mencionar el denominado Optical Flow. Este modelo pretende conocer el movimiento de los objetos en la imagen mediante el estudio del "movimiento" de los píxeles que la conforman, es decir, se trata de encontrar en el fotograma siguiente los valores de píxel que se encuentran en el fotograma actual, y así poder comparar la posición inicial y la siguiente simplemente observando su posición en ambos fotogramas. Esto puede permitir no solo conocer el movimiento de los objetos en un vídeo, sino además predecir cómo se moverán estos objetos en los fotogramas posteriores.

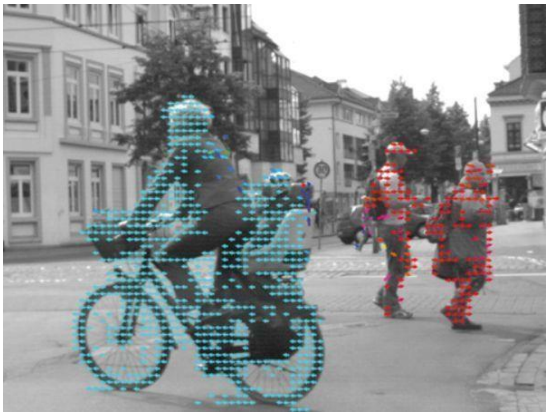


Figura 18. Ejemplo de Optical Flow [24]

En este proyecto se ha decidido analizar tanto los fotogramas como la secuencia temporal porque, aunque se presenta como una solución más compleja, presenta una mayor precisión en vídeos con entornos estáticos, es decir, en aquellos vídeos que se graban desde una misma cámara posicionada en un mismo lugar, ofreciendo así una mejor solución para aplicaciones de videovigilancia, la cual apunta a ser la que más uso tenga en la sociedad en lo que a detección de acciones respecta. Además, tras explorar los dos métodos mencionados se ha apostado por hacer uso de Deep Learning, ya que la extracción previa de características puede limitar la precisión final del modelo.

CAPÍTULO III: METODOLOGÍA

El desarrollo de este proyecto se puede dividir en cuatro partes:

- Creación del dataset.
- Procesamiento de los vídeos que conforman el dataset.
- Creación de un modelo funcional.
- Análisis del entrenamiento.
- Optimización del modelo.

La primera tarea del proyecto es la de crear un dataset equilibrado (con una cantidad parecida de vídeos de cada acción a detectar) y significativo, de manera que permita al modelo tener una cantidad suficiente y no sesgada de los datos para que pueda generalizar, pudiendo así determinar la acción que se muestra en vídeos que no forman parte de la base de datos con la que este ha sido entrenado. Para completar esta tarea, se hará una búsqueda en internet de aquellas bases de datos ya creadas que puedan resultar interesantes para el problema que se pretende solucionar (detección de acciones). Estas se deben filtrar para obtener únicamente aquellos vídeos que interesen, es decir, aquellos vídeos que contengan escenas de las acciones que se pretenden detectar. Adicionalmente a estas bases de datos, se buscarán vídeos en repositorios públicos en los que se puedan encontrar más muestras de dichas acciones.

Tras haber generado el dataset, se debe llevar a cabo un procesamiento de los vídeos seleccionados para poder usar un formato unificado a la hora de introducir los datos en la red neuronal. Para ello, se deberán recortar los vídeos tanto por píxeles (reduciendo las proporciones de los fotogramas que lo componen) como por número de fotogramas (manteniendo fijo el número de imágenes por cada vídeo introducido en el modelo).

El proceso de crear un modelo funcional se basará en la programación de toda la estructura necesaria para que el modelo funcione correctamente: carga de datos, inicialización del modelo, creación de la función de entrenamiento, evaluación de los datos de test y las funciones adicionales que se le añadan al modelo, tales como la representación de los resultados en gráficas o presentación

de la activación de cada una de las capas. El modelo debe ejecutarse correctamente y aprenderse el set de entrenamiento para dar esta fase por completada.

A continuación, se analizarán diferentes técnicas para evaluar el entrenamiento y sacar conclusiones del comportamiento de la red.

Por último, se debe optimizar el modelo. Esto es la iteración sobre los hiperparámetros para conseguir que el modelo generalice y obtener los mejores resultados de precisión posibles. Para ello, se llevará a cabo un seguimiento del entrenamiento con diferentes combinaciones de hiperparámetros, evaluando cuál puede ser la combinación que optimice los resultados. Además, se llevará a cabo una optimización del tiempo de entrenamiento empleando varias técnicas.

CAPÍTULO IV: DISEÑO DE LA SOLUCIÓN

La solución presentada en este proyecto, tal y como se puede observar en el apartado de metodología de este documento, sigue un pipeline típico de un proyecto de Visión Artificial. En los siguientes apartados se desarrollarán cada una de estas fases seguidas, explicando las diferentes opciones que se han explorado para su desarrollo, así como el diseño implementado en la solución final.

1. INFLUENCIA DEL ENTORNO EN LA DETECCIÓN DE ACCIONES

Antes de comenzar el desarrollo de la solución que lleva a la detección de acciones mediante Visión Artificial, se debe tener en cuenta aquellas cosas que pueden dar lugar a un entrenamiento erróneo del modelo o que puedan limitar su aprendizaje. En este apartado se presentará el problema de la influencia del fondo en los modelos de Visión Artificial, el cual hay que tener en mente durante el desarrollo de cada una de las partes del pipeline para reducir su efecto en el aprendizaje final que adquiera el modelo.

Para presentar la problemática, y basando la argumentación en el paper "Why Can't I Dance in the Mall?" [21], se muestran a continuación unas imágenes en las que la persona ha sido tapada con un rectángulo verde mientras realizaba una acción:



Figura 19. Imágenes de acciones ocultando a la persona

En las imágenes que se han presentado, se puede adivinar fácilmente la acción que se está llevando a cabo cada una de ellas, aunque no se vea a la persona en cuestión. Esto se debe a que las personas nos apoyamos en el contexto de la imagen para sacar ciertas conclusiones, y es por ello por

lo que el juego de la mímica supone un reto para aquellas personas que tratan de adivinar la acción que se está imitando sin poseer un contexto de dónde se ubica la misma y qué objetos está empleando para llevar a cabo la acción.

Al igual que las personas necesitamos de este contexto para poder llegar a una conclusión de una forma más fácil, los modelos de Visión Artificial hacen uso de esta información también. Esto se hace mediante la extracción de características en el bloque convolucional que forman los modelos y que, si no se vigila su entrenamiento, podrían hacer que la clasificación de la acción se base meramente en el fondo de la imagen o en los objetos de los que hace uso la persona que aparece en ella.

En la actualidad se conoce esta problemática, y se intenta lidiar con el diseño de modelos que centren su atención y sus decisiones en el movimiento humano y no en el entorno. Esto es lo que lleva al uso de máscaras que aíslen a la persona del entorno en el que se encuentra o de modelos como Optical Flow, el cual está enfocado al estudio del movimiento de los objetos en los vídeos.

Este efecto se ve aumentado en aquellos casos en los que el entorno no brinda ninguna información (por ejemplo, en el caso en el que los vídeos en los que clasificar acciones procedan de una cámara de seguridad, la cual graba siempre un mismo entorno), ya que en el caso en el que el modelo entrenado se base en el fondo para tomar decisiones, tendrá muy baja precisión si las acciones se llevan a cabo en entornos diferentes.

Es por ello por lo que esta problemática se debe tener en cuenta en el diseño de cada una de las partes del pipeline de Visión Artificial para diseñar adecuadamente un modelo capaz de inferir la acción que está llevando a cabo la persona sin importar el entorno en el que se desarrolle la misma, y por tanto el entorno en el que se quiera llevar a cabo la clasificación.

2. CREACIÓN DEL DATASET

El paso de creación de un dataset representativo y con suficientes datos es esencial para el buen desarrollo de cualquier proyecto de Inteligencia Artificial. Los datos son lo que alimentan el modelo, y lo que le permiten aprender a llevar a cabo la tarea para la que ha sido diseñado. Si no se

dispone de datos suficientes, el modelo podría no llegar extraer los patrones necesarios para la toma de decisiones. Además, si estos dados no son de calidad o representativos, es decir, no se asemejan a los datos que recibirá como entrada el modelo cuando se utilice para inferencia, el modelo no será capaz de resolver el problema cuando salga a producción. Es por todo esto por lo que esta primera parte del pipeline es crucial para el buen desarrollo del proyecto.

Algo a tener en cuenta además, es que cuanto más complejo sea el problema a resolver, más complejo deberá ser el modelo diseñado, ya que deberá encontrar patrones más complejos entre los datos, para lo que necesitará de más capas o capas más complejas para resolver el problema que se le plantea. La necesidad de diseñar un modelo más complejo lleva a la necesidad de disponer de más datos para entrenar dicho modelo, ya que si no se dará lugar a overfitting. Esto ha sido una gran problemática en este proyecto, en el cual se disponía de muy pocos datos para el entrenamiento del modelo debido a la falta de datos en internet de la acción a clasificar y de no poder crear dicho contenido.

A la hora de crear un dataset para cualquier proyecto de Inteligencia Artificial, es clave tener en cuenta también el equilibrio de los datos. Esto implica tener un número de ocurrencias similar de cada clase, de forma que todas ellas sean representativas en el dataset. El hecho de no equilibrar las clases puede dar lugar a errores en la interpretación de los resultados de precisión del modelo, así como una menor precisión del modelo a la hora de llevar a cabo la inferencia. Por ejemplo, si se lleva a cabo una clasificación entre dos clases y una de ellas constituye el 90% del dataset mientras que la otra supone tan solo el 10%, el modelo podría obtener un 90% de precisión si en vez de extraer patrones aprende que obtienen una gran precisión prediciendo que todos los datos son de la clase mayoritaria. Esto llevaría a pensar que el modelo tiene una gran precisión, pero la realidad es que está dando como salida siempre la misma clase, y que por tanto a la hora de llevar a cabo la inferencia seguirá este procedimiento.

En el caso de detección de acciones, tal y como se ha comentado anteriormente, hay que ser cuidadoso a la hora de tratar con el entorno de las imágenes. En el caso de la creación del dataset es crucial equilibrar no solo las clases en el set de entrenamiento, sino también disponer de vídeos en un mismo entorno en el que se desarrollen diferentes acciones. De este modo, el modelo tendrá más difícil basarse en el fondo para tomar la decisión sobre qué clase se desarrolla en cada vídeo. El hecho

de equilibrar los fondos de esta forma, aunque no lo garantiza, permite aumentar la seguridad de que el modelo no está basando sus decisiones únicamente en el entorno para tomar una decisión.

A la hora de crear un dataset de Visión Artificial, no solo se deben tomar decisiones que favorezcan la precisión del modelo, sino que se debe ser cuidadoso con los posibles sesgos que pueda haber en los datos. El modelo aprenderá de cualquier sesgo introducido en los datos, y trasladará ese sesgo a su toma de decisiones cuando se emplee el modelo para realizar inferencia. Esto es un gran problema que puede derivar en problemas éticos en la aplicación de la Inteligencia Artificial. Son muchos los casos de modelos en los que se han introducido sesgos en los datos, derivando en modelos sesgados también, lo que lleva a tener que retirar los modelos de producción. Se debe tener este riesgo en cuenta para desarrollar modelos de Inteligencia Artificial éticos.

Ahora que se han comentado los principales puntos a tener en cuenta a la hora de generar un dataset de calidad que permita desarrollar un proyecto de forma exitosa, cabe comentar las diferentes formas de generar un dataset. Hay diferentes formas de crear un dataset, y escoger una u otra depende de la aplicación del modelo, y por lo tanto de las imágenes o vídeos que se deben obtener.

- **Usar un dataset público**: hay ciertas aplicaciones comunes de Visión Artificial para las que existen datasets ya generados en internet, y que contienen los datos etiquetados. Si hay disponible uno de estos datasets para la aplicación del proyecto a desarrollar, puede suponer una gran reducción en el tiempo de desarrollo del proyecto, ya que la tarea de recolección y limpieza de datos es una de las que más tiempo consume a la hora de desarrollar un proyecto de este tipo. En algunos casos el dataset tendrá las clases que se necesitan, pero en otros casos, habrá que extraer de estos datasets las clases que son de interés e incluso combinar datasets. En el caso concreto de detección de acciones en vídeo, los datasets más populares son:
    - UCF-101: es una extensión de UCF-50 que consta de 13.320 vídeos repartidos entre 101 acciones diferentes. Estas acciones están divididas en 5 grupos principales: interacciones persona-objeto, movimiento corporal, interacción persona-persona, personas tocando instrumentos musicales y personas haciendo deporte. [25]

- ActivityNet: es un dataset que contiene 200 clases de acciones diferentes con un total de 100 vídeos por cada clase. Esto hace un total de 648 horas de vídeo. [26]
- HMDB-51: dataset formado por 6766 vídeos en 51 clases. Estas clases se pueden repartir en 5 grandes grupos de acciones: acciones faciales (sonreír, hablar, masticar, reír), acciones faciales manipulando un objeto (fumar, comer, beber), movimientos de cuerpo generales (aplaudir, subir escaleras, etc.), movimientos de cuerpo interactuando con objetos (dibujar, jugar al golf, etc.) y movimientos humanos para interacción con otra persona (abrazarse, dar la mano, etc.). [27]
- Something-Something: es un dataset que contiene vídeos de acciones humanas interactuando con objetos del día a día. Estos vídeos están grabados en primera persona y se observa solo la mano interactuando con el objeto. Algunos ejemplos de acción son poner algo en una superficie, mover algo hacia arriba o cubrir una cosa con otra. Este dataset está formado por un total de 220.847 vídeos y un total de 147 clases. [28]
- Sports-1M: contiene 1.133.158 enlaces a vídeos de YouTube de deportes clasificados en 487 clases distintas. [29]
- AVA: dataset creado por Google que contiene anotaciones audiovisuales en vídeo para poder comprender la actividad que se desarrolla en cada uno de ellos. Hay 3 datasets distintos de esta familia: AVA-Kinetics Dataset, AVA Actions Dataset y AVA Spoken Activity Dataset. [30]
- Youtube-8M: dataset creado por Google que contiene 6,1 millones de IDs de vídeos de YouTube que contienen acciones. Está compuesto por un total de 3862 vídeos, lo que genera un total de 350.000 horas de vídeo. [31]
- Kinetics: esta familia de datasets consta de 3 datasets diferentes: Kinetics-400, Kinetics-600 y Kinetics-700. Al igual que en el caso anterior, estos datasets están compuestos de links a vídeos en internet. Cada vídeo tiene una duración

media de 10 segundos, y la diferencia entre estos datasets son el número de clases que tiene cada uno, de forma que tienen 400, 600 y 700 clases con un total de 650.000 vídeos en el caso del dataset más grande (Kinetics-700). [32]

- **Generar el dataset**: esta segunda opción es la empleada en aquellos casos en los que las clases a clasificar no son muy comunes, y por lo tanto no se dispone de un dataset público que pueda emplearse. Para ello, se deben buscar los datos en internet o incluso generarlos en el caso en el que los disponibles en internet no sean suficientes.

En el caso de este proyecto, la acción a detectar no es común, y por lo tanto no existe un dataset para la misma. Es por ello por lo que se generó el dataset de forma manual. Este tipo de vídeos no se puede generar para aumentar el tamaño de dataset, por lo que la única fuente de datos es la búsqueda en internet. Además, este tipo de vídeos es poco común, por lo que la cantidad de datos en el dataset era reducida, lo cual dificultó mucho el desarrollo del proyecto. La principal fuente de vídeos para la generación del dataset fue la descarga de contenido de Youtube, para lo que se creó un código en Python que permitiera la descarga tanto de vídeos como de listas de reproducción completas.

Paralelamente con la colecta de datos, se debe etiquetar cada uno de ellos con la clase a la que corresponde. Es crucial no cometer errores en este proceso, ya que etiquetar datos de forma errónea llevará a un aprendizaje erróneo por parte del modelo. Para llevar este procedimiento a cabo hay varias técnicas. Una posibilidad sería vincular en un csv el nombre del archivo que contiene el vídeo con la etiqueta de dicho vídeo. De este modo, al cargar los datos se realiza una búsqueda del vídeo por su nombre en el csv y se obtienen las etiquetas del mismo. Esta técnica se emplea sobre todo cuando un vídeo tiene varias etiquetas (por ejemplo en el caso de detección de objetos en vídeo), aunque también se puede emplear en los casos de clasificación de vídeo en los que cada vídeo solo tendrá una etiqueta asociada (por ejemplo en detección de acciones cada vídeo solo tendrá una etiqueta indicando la acción que ocurre en el mismo). Otro método que se puede emplear, y que es el que se ha aplicado en este proyecto, es incluir la etiqueta en el propio nombre del archivo. De este modo, a la hora de nombrar un vídeo se debe utilizar un nombre que permita extraer del mismo la etiqueta que corresponde a dicho vídeo. La nomenclatura empleada en este proyecto es _Etiqueta_NombreVídeo.

Este método empleado permite agilizar el proceso de carga de vídeos, ya que al cargar el vídeo se conoce la etiqueta que le corresponde sin tener que acceder a otro archivo para obtenerla. Tal y como se ha comentado, este método de etiquetado es posible porque el vídeo solo requiere de una etiqueta por tratarse de un problema de clasificación, pero si el problema a resolver fuera de otro tipo (por ejemplo detección de objetos) habría que recurrir al etiquetado mediante el uso de un archivo csv.

El objetivo que se debe alcanzar a la hora de diseñar un modelo de Inteligencia Artificial es que el modelo generalice, es decir, que sea capaz de resolver el problema que se le planteó durante el entrenamiento, pero con datos que no ha visto anteriormente. En el caso de emplear todos los datos para entrenar, no se dispondría de la capacidad de conocer la precisión del modelo a la hora de clasificar acciones en vídeos que no ha visto aún, es decir, no conocemos la capacidad del modelo para generalizar. Esto es lo que lleva al uso de tres conjuntos de datos diferentes: el conjunto de entrenamiento, el conjunto de validación y el conjunto de test. La finalidad de cada uno se detalla a continuación:

- **Conjunto de entrenamiento**: este conjunto de datos es el que se emplea para entrenar el modelo, es decir, estos datos se pasarán hacia delante por la red (forward propagation) para realizar una estimación de la clase del vídeo y posteriormente se hará una pasada hacia atrás (backpropagation) para ajustar los parámetros de la misma y poder así mejorar en la estimación de la clase final. Este conjunto es el que contiene una porción mayor de los datos, ya que suele suponer entre el 50% y el 80% de los datos. Los datos que conforman este conjunto son los únicos que permiten que el modelo aprenda, ya que los otros dos conjuntos no están destinados a modificar los parámetros de la red.

- **Conjunto de validación**: este conjunto de datos permite ver la evolución de la capacidad de generalizar del modelo mientras se lleva a cabo el entrenamiento. Esto se consigue debido a que estos datos no se emplean para ajustar los parámetros del entrenamiento, es decir, se pasan hacia delante por la red pero no se realiza "backpropagation". La finalidad de crear este conjunto de datos es encontrar unos valores óptimos para los hiperparámetros del modelo, ya que permite estudiar el

impacto de la variación de cada hiperparámetro en los resultados que presenta el modelo a la hora de generalizar. En algunos casos no se destina un conjunto de datos para esta finalidad, sino que se lleva a cabo una técnica denominada "cross-validation" que permite emplear datos del conjunto de entrenamiento para realizar esta tarea. En los casos en los que, si se utiliza este conjunto de datos, suele constituir entre el 10% y el 20% del total de los datos del dataset. A continuación, se muestra un ejemplo de las curvas que se analizan con el conjunto de validación para determinar el rendimiento del modelo.

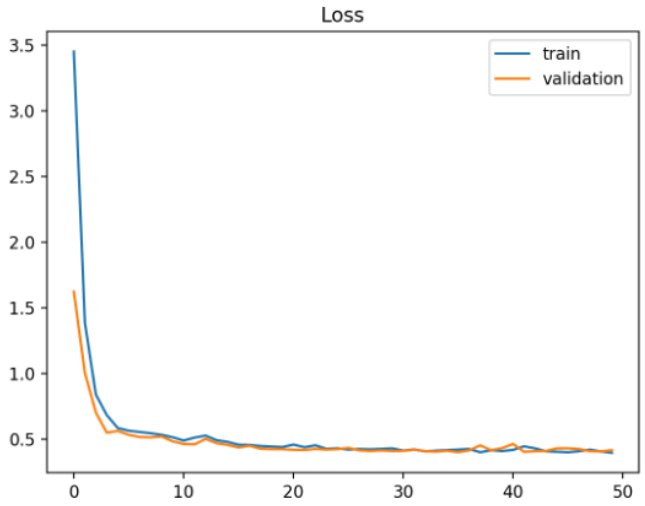


Figura 20. Curvas de entrenamiento y validación [33]

- **Conjunto de test:** este conjunto de datos permite conocer la capacidad de generalizar del modelo con datos que el modelo nunca ha visto antes. Mientras que los datos destinados en el conjunto de entrenamiento han sido empleados para ajustar los parámetros de la red y los datos del conjunto de validación han sido empleados para ajustar los hiperparámetros del modelo, los datos que constituyen el conjunto de test no interfieren en la modificación de ningún parámetro, sino que se emplean para conocer la precisión real del modelo con datos que no ha visto antes, es decir, la capacidad del modelo de generalizar. Los resultados de este conjunto son los que se presentan como precisión final del modelo a la hora de evaluar la calidad de un modelo

y de comparar unos con otros. El conjunto de test suele constituir entre el 10% y el 30% de los datos totales del dataset.

En el caso de este proyecto, se generó un dataset con un total de 210 vídeos (105 vídeos de la clase a y 105 vídeos de la clase b), y una separación en conjuntos de datos del 60% en entrenamiento, 20% en validación y 20% en test.

A la hora de generar este dataset, se prestó especial atención no solo a balancear las clases en cada uno de estos conjuntos para que cada uno de ellos estuviera constituido en proporción 50%-50% entre las dos clases, sino que además se balanceó el fondo debido a la importancia que esto supone tal y como se ha mencionado anteriormente en este documento. Para realizar esta tarea, se ha incluido en cada uno de los conjuntos uno o más vídeos de la acción a en un fondo y la misma cantidad de vídeos de la acción b en ese mismo fondo.

Además, algo a tener en cuenta a la hora de trabajar con vídeo es que se debe tratar de mantener constantes los fotogramas por segundo. Esto se debe a que el modelo obtendrá mejores resultados si el salto temporal de cada fotograma es consistente entre los diferentes vídeos, de manera que los cambios que se puedan producir entre un fotograma y el siguiente de vídeos distintos sea constante. Para escoger el número de fotogramas adecuado, se debe tener en cuenta el tipo de vídeos con los que se trata, y por tanto cuánta variación de información hay entre fotogramas, o lo que es lo mismo, la velocidad a la que se mueven los objetos en el vídeo. Teniendo en cuenta que los vídeos analizados en este proyecto son vídeos en los que se analiza el movimiento humano, se ha decidido mantener un valor de fotogramas por segundo constante de 10 fps, el cual es un valor razonable para analizar el movimiento humano.

3. DATA AUGMENTATION

Una vez que se han recolectado los vídeos que conformarán el dataset, se pueden llevar a cabo diferentes transformaciones en los mismos que permitan aumentar el tamaño del dataset mediante la creación de vídeos similares a los originales, pero con ciertas variaciones. Este proceso de aumentar el tamaño del dataset mediante la creación de datos a partir de la transformación de los datos originales se denomina “data augmentation”.

Esta técnica puede ser especialmente útil en aquellos casos en los que el dataset es limitado, ya que pueden resolver un problema de falta de datos. Pero no solo eso, sino que, mediante la introducción de transformaciones en los datos originales, el modelo puede llegar a extraer información más relevante de las imágenes que analiza. Por ejemplo, a la hora de detectar objetos es muy interesante que el modelo sea capaz de detectarlos incluso en aquellos casos en los que haya algo obstruyendo parte del objeto o incluso cuando el objeto aparece parcialmente en la imagen o a diferentes escalas.

De este modo, se pueden emplear diferentes técnicas de data augmentation para generar datos que representen ciertas problemáticas que el modelo encontrará a la hora de trabajar en inferencia, de modo que pueda entrenar con dichos datos y sea capaz de extraer características más relevantes, aumentando así su precisión a la hora de predecir cuando trabaje en un entorno de producción. Esto se puede conseguir mediante la introducción de datos que representen esta problemática en el dataset. Además, en algunos casos puede actuar como método de regularización para prevenir el overfitting, como por ejemplo al introducir ruido en las imágenes tal y como se explicará más adelante.

Los métodos de data augmentation son variados, y según cuál sea el proyecto tiene sentido emplear unos u otros. Es necesario realizar un análisis previo para comprender cuáles de estas técnicas permitirán generar datos que realmente sirvan para que el modelo aprenda, y no que supongan un dato no relevante o ambiguo a la hora de entrenar. Por ejemplo, si se entrena un modelo para distinguir entre razas de perro, no tiene sentido generar datos girando la imagen 180 grados, ya que en ningún caso se introducirán en el modelo imágenes de perros invertidas cuando el modelo esté trabajando en inferencia.

Algunas de las técnicas de data augmentation para imágenes son las siguientes:

- **Recortado aleatorio (“random cropping”):** este es uno de los métodos más conocidos y utilizados a la hora de realizar data augmentation para proyectos de detección de objetos, ya que permite que el modelo aprenda a detectar el objeto en aquellos casos en los que solo aparezca parcialmente en una imagen. Esta técnica consiste en realizar un recorte aleatorio de la imagen para generar una imagen nueva. Se pueden realizar varios recortes por cada imagen, dando lugar así a varias imágenes por cada imagen inicial del dataset original. El resultado es el siguiente:

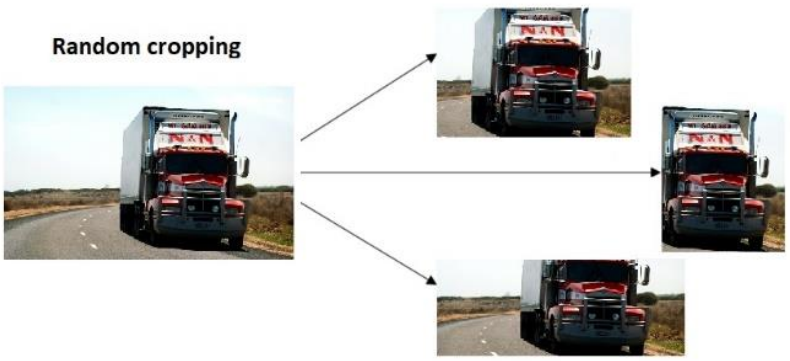


Figura 21. Ejemplo de recortado aleatorio [34]

- **Volteado ("flipping"):** esta técnica consiste en voltear la imagen sobre el eje x o sobre el eje y de la misma, es decir, generar una imagen que sea la original boca abajo (volteado sobre el eje x) o una imagen que sea el modo espejo de la original (volteado sobre el eje y). También se puede crear una combinación de ambos volteos, tal y como se muestra en la siguiente imagen:



Figura 22. Ejemplo de volteado de imagen [35]

De entre estas opciones de volteo, es más común realizar la transformación sobre el eje y (modo espejo) que sobre el eje x, ya que, a la hora de emplear el modelo en inferencia, es poco probable que se le introduzcan datos que sean escenas "boca

abajo", pero si que debe conocer las diferentes orientaciones que pueden tener las escenas.

- **Enmarcado ("padding"):** el método de enmarcado consiste en generar un marco en la imagen, de modo que los bordes de la misma pasen a estar formados por píxeles de un color fijo (generalmente negro).

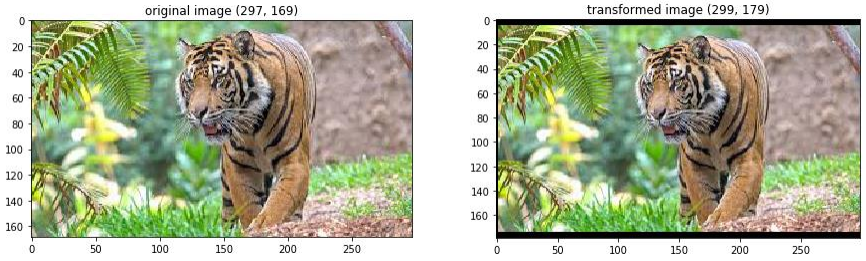


Figura 23. Ejemplo de enmarcado [36]

- **Rotación ("rotation"):** esta técnica consiste en rotar la imagen un número de grados determinado o aleatorio. A partir de una misma imagen se pueden generar varias imágenes con rotaciones distintas, aumentando así en gran cantidad el dataset.

- **Traslación ("translation"):** esta técnica de data augmentation consiste en trasladar la imagen en una dirección, ya sea en una dirección aleatoria o predefinida. Tras desplazar la imagen, parte de esta desaparece, mientras que los píxeles sobrantes son rellenados con píxeles de un color fijo (generalmente negro), tal y como se muestra en la siguiente imagen:

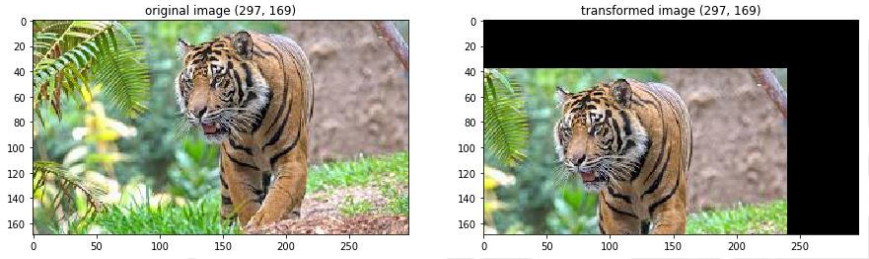


Figura 24. Ejemplo de traslación [36]

El efecto que se consigue con este método es similar al que se consigue con el de recortado, ya que la imagen final está constituida únicamente por parte de la imagen inicial. Sin embargo, lo que diferencia a este método del de recortado es que la imagen final mantiene las mismas dimensiones que la imagen inicial.

- **Borrado aleatorio ("random erasing"):** este método consiste en borrar de forma aleatoria parte de la imagen, sustituyéndola por un rectángulo de un color fijo o con ruido (valores aleatorios de píxeles). Al igual que la técnica de recorte aleatorio, este es uno de los métodos más conocidos y utilizados a la hora de realizar data augmentation para proyectos de detección de objetos. Es realmente útil para generar datos para entrenar un modelo que sea capaz de detectar objetos a pesar de que estos estén parcialmente ocultos por otro objeto, es decir, cuando en la imagen hay obstrucción del objeto.

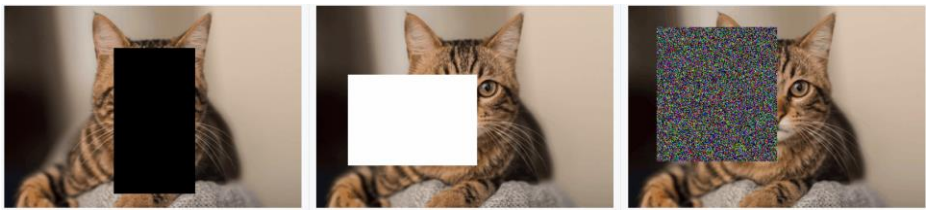


Figura 25. Ejemplo de borrado aleatorio [37]

- **Color ("color space"):** tal y como se ha comentado previamente en este documento, las imágenes están formadas por tres matrices (R, G y B) gracias a las cuales se representa el color de cada píxel. Cada una de estas matrices (o canal) representa la intensidad en uno de los tres colores: rojo, verde y azul respectivamente. Este tipo de data augmentation consiste en realizar cambios en los valores de estas matrices de manera que cambie la apariencia de la foto, por ejemplo, aumentando la luminosidad, cambiando los colores de la imagen o aumentando la saturación. Se pueden ver algunos ejemplos a continuación:



Figura 26. Ejemplos de cambios en los canales [38]

Uno de los sesgos más comunes a la hora de analizar imágenes es el debido a la iluminación de la imagen. Este tipo de data augmentation es muy eficaz a la hora de atacar directamente este problema, ya que permite representar una misma imagen con diferentes iluminaciones, reduciendo el overfitting debido a esta característica de las imágenes. La manera de aumentar o reducir el brillo de una imagen es simplemente aumentar o reducir en un valor constante el valor de todos los píxeles en los tres canales. De este modo, si se aumentan todos los valores, el brillo global de la imagen aumentará, mientras que si se reducen todos los valores de píxel, el brillo disminuirá. El número de este tipo de transformaciones que se puede realizar es muy elevado, ya que una imagen se puede editar de muchas maneras distintas. Estas imágenes se pueden editar tanto con programas específicos de edición como con programas que realicen esto de manera automatizada (por ejemplo, empleando Python con alguna de sus librerías de tratamiento de imágenes).

- **Difuminado ("blur"):** la técnica de difuminado consiste en generar una versión borrosa de la imagen original.



Figura 27. Ejemplo de difuminado [39]

Al igual que la introducción de ruido en la imagen, el difuminado tiene un efecto regularizador, y se emplea como método de Adversarial Training (el cual se explicará más adelante).

- **Inyección de ruido ("noise injection"):** este método de data augmentation consiste en superponer sobre la imagen una matriz de valores aleatorios, que comúnmente siguen una distribución Gaussiana, es decir, se introduce ruido blanco o Gaussiano. En la siguiente imagen se puede ver un ejemplo de esta técnica:



Figura 28. Ejemplo de inyección de ruido [39]

Esta técnica puede ayudar a las capas convolucionales a aprender características de las imágenes de una forma más robusta, produciendo un efecto regularizador que ayuda a evitar el overfitting.

En este proyecto, las técnicas de data augmentation que se han empleado son volteado respecto del eje y (modo espejo de los vídeos) y la introducción de ruido en los fotogramas. La primera técnica se ha empleado simplemente por el hecho de tener más datos representativos para entrenar le modelo,

mientras que la decisión de introducir ruido en los datos de entrenamiento se debe a la necesidad de reducir el overfitting, para lo cual se aprovechó el efecto regulador de esta técnica. Otro motivo que llevó al uso de la técnica de ruido como data augmentation es la inspiración en el Adversarial Machine Learning.

Para comprender el concepto de Adversarial Machine Learning, se presenta las siguientes imágenes:



Figura 29. Ejemplo de imagen (izquierda) y su adversaria (derecha) [40]

Aunque para un humano estas dos imágenes son idénticas, para un modelo de Machine Learning no lo son. Es más, el modelo de Visión Artificial de Google, GoogLeNet, clasificó la imagen izquierda como un panda y la imagen derecha como un gibón; un primate de color marrón que a ojos de un humano en nada se parece a un panda. Esto se debe a que los algoritmos de Machine Learning no funcionan exactamente como lo hace un humano. En este caso en concreto, lo que hizo que el modelo de Google clasificara la imagen de la derecha como un gibón con un 99,3% de confianza fue la introducción de ruido sobre la imagen de la izquierda, lo cual cambió drásticamente la predicción del modelo con respecto a la imagen original, la cual clasificó como un panda con un 57,7% de confianza. [40]

Tal y como se ha observado con el ejemplo anterior, ciertas modificaciones en los datos pueden llevar a errores en los resultados de los algoritmos si no se tiene en cuenta esta problemática a la hora de entrenar los modelos. Estos errores pueden ser desde algo inocente como confundir un perro y un gato como algo grave que pone en peligro la vida de las personas, como puede ser que un coche autónomo confunda un stop con una señal de límite de velocidad. Es por ello que el Adversarial

Machine Learning trata de explotar la manera en la que funcionan estos algoritmos para intentar inducir errores en los mismos en tareas en las que un humano no fallaría (como es el ejemplo de la imagen superior). Esta concienciación y búsqueda del error del modelo mediante un entrenamiento exhaustivo del mismo (e introduciendo "trampas" en los datos) se denomina Adversarial Training.

La concienciación sobre esta problemática es, tal y como se ha mencionado, uno de los motivos que llevó a introducir ruido en los datos como forma de Data Augmentation a la hora de desarrollar este proyecto.

4. NORMALIZACIÓN DE LOS DATOS

La normalización de los datos en Machine Learning es algo fundamental para cualquier modelo que se pretenda desarrollar. Esta idea consiste en escalar los datos para que todos los datos de entrada al modelo estén centrados y sigan una distribución normal, de manera que se evite que la escala de los mismos afecte a la predicción del modelo.

Por ejemplo, en el caso de un problema de regresión, si se dispone de una variable que indica el número de horas que ha estado viajando un avión y otro con el número de kilómetros que ha viajado, la diferencia de magnitudes es realmente elevada. En este caso, intrínsecamente el modelo le dará una mayor importancia a los kilómetros por el simple hecho de ser un valor más grande, ya que provocará una mayor activación en las neuronas. El hecho de normalizar permite darle la misma importancia a ambas variables, de manera que el modelo no se vea influenciado por su magnitud total sino por la distancia a la media de cada una de ellas.

Este procedimiento tan sencillo de normalizar los datos aumenta en gran medida la precisión del modelo, y reduce drásticamente los tiempos de entrenamiento, tal y como se puede observar en la siguiente imagen, donde a la izquierda se presentan los resultados de entrenamiento de un modelo en el que no se ha llevado a cabo una normalización de datos y a la derecha la gráfica de entrenamiento de un modelo en el que si se ha aplicado esta técnica:

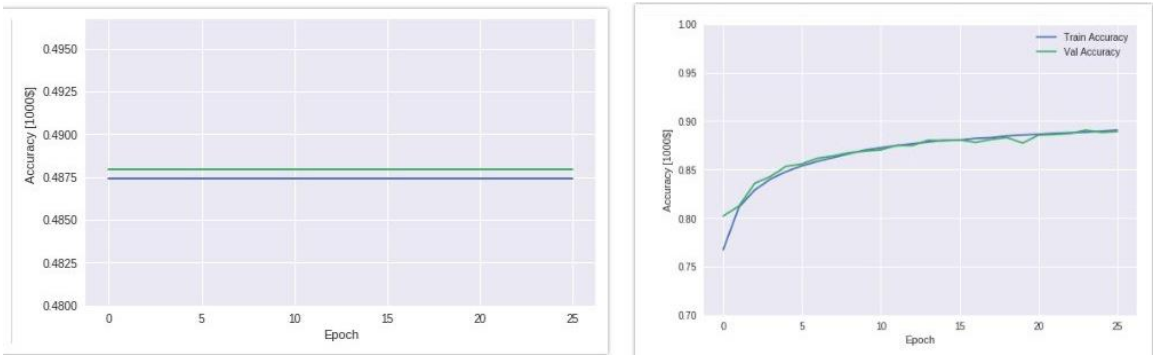


Figura 30. Efecto de normalizar los datos [41]

Como se puede observar en la imagen, en el caso en el que no se ha aplicado una normalización de datos, el modelo no llega a aprender en las 26 épocas de entrenamiento. Por otro lado, en el caso en el que sí se ha aplicado esta técnica, el modelo llega a casi un 90% de precisión tanto en entrenamiento como en validación tras las mismas épocas de entrenamiento que en el caso anterior.

Al igual que en el resto de las aplicaciones de Machine Learning, en el caso de Visión Artificial esta técnica es muy importante también, y mejora en gran medida los resultados del modelo. En el caso de las imágenes, aquello que puede influir en el modelo a la hora de entrenar si no se normalizan los datos es el brillo en las mismas. Esto se debe a que una imagen con más brillo tenderá a tener valores de pixel más altos (más cercanos a 255, que es el color blanco), mientras que aquellas imágenes más oscuras tenderán a valores cercanos a 0 (que es el color negro).

En cuanto a la normalización en imágenes, lo que se busca es normalizar los valores de pixel, de manera que sigan una distribución normal para evitar valores muy dispares debidos al brillo. Si disponemos de dos imágenes exactamente iguales, pero una con más brillo y otra con menos, la diferencia entre los valores de píxeles contiguos será la misma en ambas imágenes, mientras que los valores absolutos de los píxeles de la imagen con mayor brillo serán más elevados que los de menor brillo (tal y como se acaba de explicar en el párrafo anterior). Es decir, a pesar de que estas dos imágenes deberían ser tratadas de igual manera por el modelo, esto podría no ser así en el caso en el que no se llevara a cabo un proceso de normalización debido a esa diferencia en los datos de entrada.

Una técnica de normalización aplicada en proyectos de Visión Artificial consiste en normalizar cada una de las imágenes por separado. Esto consiste en calcular la media de los valores de cada pixel de una imagen y normalizar todos los píxeles de la misma en base a eso. De este modo, estos valores seguirán una distribución normal en cada una de las imágenes. Algo a mencionar es que por el hecho de normalizar cada imagen por separado, la media de cada una de las imágenes será distinta del resto, ya que si por ejemplo tenemos una imagen más clara y otra más oscura, aunque los píxeles de cada una estén normalizados, la media de ambas imágenes será distinta. Este inconveniente ha llevado al desarrollo de otras técnicas como es el Batch Normalization, el cual se desarrolla más adelante en este documento

5. PROCESAMIENTO DE VÍDEOS

Una vez que se dispone de los vídeos que conformarán el dataset, estos deben ser procesados de forma que se obtenga un formato unificado para poder introducirlos en el modelo. Esto implica aplicar transformaciones a cada uno de los vídeos para que todos ellos tengan las mismas dimensiones tanto en número de píxeles como en profundidad (o lo que es lo mismo, en número de fotogramas).

Los modelos de Inteligencia Artificial necesitan de unos datos que tengan un formato consistente, es decir, que todos los datos que se introducen en el modelo deben tener las mismas características para que este pueda tratarlos. En el caso de los vídeos, los parámetros que deben compartir son el ancho y alto de fotograma y la profundidad (el número de fotogramas), además del número de canales que tiene cada fotograma. Este último parámetro determina si un vídeo se muestra a color (con canales RGB) o en escala de grises.

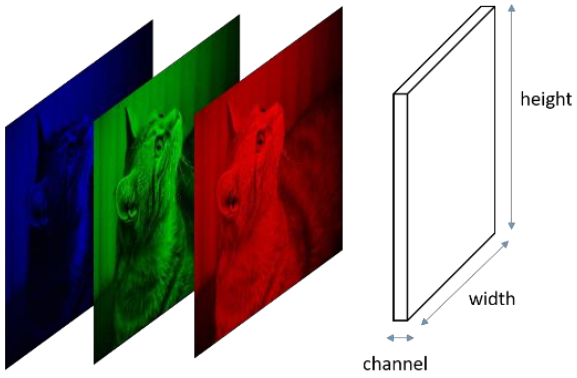


Figura 31. Dimensiones de una imagen [42]

La importancia de mantener fijas estas dimensiones se debe a que las capas convolucionales encargadas de extraer parámetros de las imágenes requieren de un tamaño fijo de imagen, ya que se debe especificar al diseñar el modelo cuál va a ser el tamaño del dato de entrada en cada una de las capas. Es por ello por lo que todos los fotogramas de todos los vídeos del dataset deben tener un número de píxeles a lo ancho y a lo alto iguales y el mismo número de canales, lo cual lleva a la necesidad de realizar un preprocesado de todos los datos de entrada al modelo.

El proceso de escalado de las imágenes se puede realizar empleando diferentes técnicas: [43]

- **Nearest Neighbor:** es la forma más simple de llevar a cabo un escalado de una imagen, pero a la vez la que peores resultados presenta en cuanto a la calidad de la imagen final. Esta técnica se basa en determinar cuál es el píxel más cercano a un píxel vacío y rellenarlo con los mismos valores que este vecino más cercano. En el caso de escalar la imagen a una de tamaño superior, habrá varios píxeles cercanos que tengan el mismo color, tal y como se puede observar en la imagen siguiente:

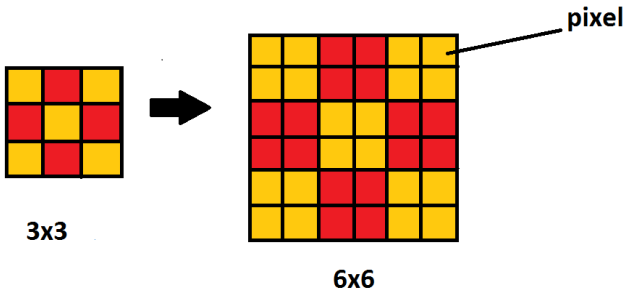


Figura 32. Escalado con Nearest Neighbor [44]

Algo a tener en cuenta a la hora de emplear este método es la necesidad de retocar la imagen final en el caso en el que se deseen obtener resultados de calidad cuando hay una gran diferencia entre el tamaño inicial de la imagen y el final, ya que el resultado se verá pixelado.

- **Bilinear:** esta técnica emplea una aproximación al problema similar a Nearest Neighbors, pero en este caso en vez de asignar el valor de un píxel al del píxel más cercano, genera pesos en función de la distancia a cada píxel de la imagen, y con esos pesos y los valores del píxel al que están asociados genera el píxel final. Esto hace que la computación sea mucho más compleja que en Nearest Neighbors, pero los resultados son sustancialmente mejores.

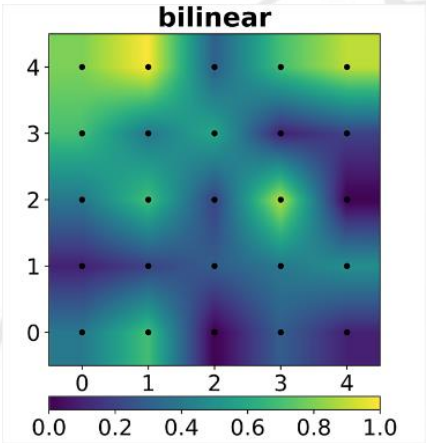


Figura 33. Escalado con el método Bilinear [43]

- **Bicubic Interpolation:** este último método de escalado de imágenes es el que ofrece mejores resultados de los tres presentados en esta sección, aunque también cabe mencionar que es el más lento en ejecución. Este método consiste en calcular la media de los valores de los 16 píxeles que rodean al píxel que se debe calcular. Esto hace que la imagen resultante tenga un aspecto más suave que en los casos anteriores.

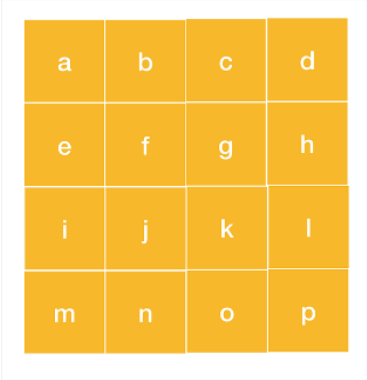


Figura 34. Escalado mediante Bicubic Interpolation [43]

Ahora que se han explicado los diferentes métodos de escalado de imágenes, cabe hablar sobre la transformación de imágenes de color en escala de grises. Esta conversión puede suponer muy interesante en aquellos casos en los que el color no aporte información relevante al modelo, y por tanto no base sus decisiones en él.

El hecho de convertir de RGB a escala de grises supone una reducción del espacio ocupado en memoria por el dataset y una disminución del número de parámetros requeridos en la red.

Tanto la reducción de memoria como en número de parámetros de la red se deben a que las imágenes en formato de escala de grises están formadas únicamente por 1 canal, mientras que las imágenes en RGB contienen 3 canales (rojo, verde y azul). Esto supone que cada imagen contendrá dos tercios menos de información en escala de grises que en RGB, tal y como se puede observar a continuación:

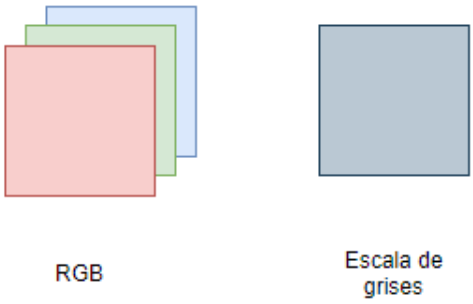


Figura 35. Número de canales en RGB y en escala de grises

Tal y como se puede ver en la imagen anterior, el tamaño de una imagen en escala de grises supone tan solo un tercio del tamaño de la imagen a color. Esto hace que los requerimientos de procesamiento y de almacenamiento se reduzcan al emplear escala de grises en vez de imágenes a color.

Además, el modelo requerirá de menos parámetros, ya que cada capa de la red deberá analizar menos información cada vez, ya que el tamaño del dato de entrada será menor. Esto se puede comprender con un ejemplo sencillo. El caso de tener una capa convolucional con un filtro de dimensiones 7x7, en el caso de tener que analizar una imagen en RGB, el número de parámetros de esta capa será 7x7x3+1 = 148 parámetros (el +1 se debe al parámetro bias de la red), mientras que, en el caso de tener un único canal de entrada a la capa, es decir, en el caso de utilizar imágenes en escala de grises, el número de parámetros de esta capa serán 7x7+1 = 50 parámetros. Tal y como se puede observar, el número de parámetros en el caso de utilizar imágenes en escala de grises supone una reducción de dos tercios de los parámetros que se requieren al emplear imágenes en RGB.

Esto a la hora de diseñar redes muy profundas puede suponer una gran diferencia en los requerimientos y los tiempos de computación, agilizando mucho el proceso de iteración a la hora de ajustar los hiperparámetros del modelo. Sin embargo, esta técnica solo se debe emplear en aquellos casos en los que el color no aporte información al modelo, ya que de lo contrario se podría estar limitando el rendimiento del mismo al estar quitándole información.

Hay tres métodos distintos a la hora de convertir una imagen de color a escala de grises: [45]

- **Basado en la luz:** este método consiste en realizar una media entre el valor de píxel más grande de entre los tres canales junto con el valor de píxel más pequeño de entre estos canales. Por tanto, la fórmula que se aplica a uno de los píxeles en color para calcular su valor en escala de grises es (max(R,G,B) + min(R,G,B)) / 2.

- **Haciendo la media:** este es el método más rápido y sencillo, y consiste en realizar la media entre el valor del píxel en cada uno de los tres canales, es decir, (R+G+B)/2.

- **Basado en la luminosidad:** este último método es una versión más sofisticada del método de la media, ya que en este caso no se trata de una media ponderada entre los tres canales, sino que emplea unos pesos a la hora de realizar esta media para que la imagen final sea mejor visualmente dada la percepción humana. El ojo humano es más sensible al verde, posteriormente al rojo y por último al azul, y siguiendo esto, se ajustan unos pesos de forma que la imagen final le dé una importancia mayor al valor del canal verde, una menor al rojo y una aún menor al azul. De este modo, la fórmula que se emplea es 0,21*R + 0,72*G + 0,07*B.

A continuación, se puede observar un ejemplo de los resultados que se obtiene con cada uno de estos métodos.

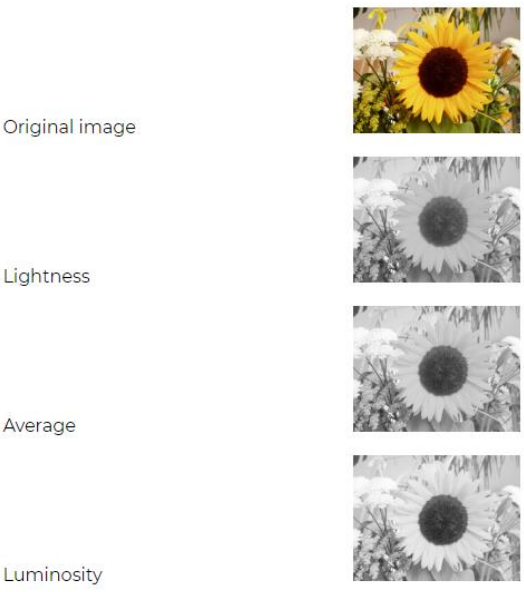


Figura 36. Comparativa de los métodos de conversión a escala de grises [45]

6. DESARROLLO DEL ALGORITMO

Ahora que se dispone de un dataset, se procede a diseñar el algoritmo. Aunque en esta sección se presente como un proceso secuencial, el proceso de creación de un algoritmo es iterativo. No solo se debe encontrar la arquitectura óptima para el modelo, sino que además se deben buscar aquellos hiperparámetros que optimicen los resultados de este a la hora de resolver el problema que debe resolver.

Las diferentes fases que se desarrollarán en este apartado son un análisis de las librerías o "frameworks" para desarrollar modelos, la carga de datos, Transfer Learning, la arquitectura empleada en este proyecto, métodos de regularización y entrenamiento con GPU.

6.1. LIBRERÍAS PARA LA CREACIÓN DE ALGORITMOS

La primera decisión a tomar a la hora de desarrollar un modelo de Visión Artificial es el entorno en el que se va a programar la solución. Aunque el modelo se puede desarrollar definiendo

matemáticamente cada una de las capas de la red y sus derivadas para realizar backpropagation, esto no tiene sentido actualmente dada la existencia de librerías que ya tienen esto predefinido y que permiten al usuario programar modelos a alto nivel. Estas librerías poseen las capas necesarias para desarrollar modelos, de manera que no es necesario programar cada una de las ecuaciones matemáticas que las definen, sino solo llamar a ciertas funciones que lleven a cabo las acciones necesarias para entrenar el modelo.

Actualmente existen tres grandes librerías para el desarrollo de modelos de Inteligencia Artificial: TensorFlow, Keras y PyTorch. Aunque también hay disponible otras librerías como Caffe, Theano, MXNet y Chainer, las tres mencionadas anteriormente son las que gozan de mayor popularidad actualmente y las que tienen una mayor comunidad de desarrolladores. Es por esto por lo que son las tres opciones que se han planteado para el proyecto y las que se desarrollarán en esta sección.

Cabe destacar que las tres librerías que se presentan son de código abierto, lo cual ha agilizado mucho el proceso de desarrollo de las mismas y ha permitido que su uso se extienda rápidamente. Además, la comunidad de desarrolladores es muy colaborativa, ofreciendo sus códigos de forma abierta a todo el mundo para fomentar el desarrollo de la Inteligencia Artificial.

Algo que se puede observar en todas las librerías de Machine Learning es que el lenguaje de programación predominante es Python. Este lenguaje está gozando de gran popularidad en la actualidad, y gran parte de esta popularidad se debe al extendido uso que se le da en este sector. Aunque algunas librerías únicamente admiten Python como lenguaje de programación, otras son más flexibles y permiten otros lenguajes adicionales como C++ o Java.

TENSORFLOW

TensorFlow es una librería genérica de Machine Learning escrita tanto en C++ como en Python y desarrollada por Google. Es una plataforma que ofrece una gran variedad de recursos, ya que permite crear todo tipo de modelos de Machine Learning: desde Deep Learning hasta Aprendizaje por refuerzo. Aunque actualmente sea conocida por su oferta en soluciones de Machine Learning, inicialmente no fue creada con ese fin. TensorFlow nació como una librería matemática simbólica

que permitía programar flujos de datos para una gran variedad de aplicaciones y ejecutarlos con gran rendimiento y precisión.

Esta librería se basa en el uso y creación de dos elementos: los tensores y los gráficos computacionales. Un tensor es una matriz con n dimensiones. Los gráficos computacionales representan el flujo ("flow") de información entre un elemento del proceso y otro. Estos gráficos se crean a la hora de diseñar un algoritmo, y en ellos se define la manera en la que la información fluye de una operación matemática a otra hasta que se consigue el resultado final y explican la manera en la que se ejecutará un modelo. De estos dos elementos recibe su nombre esta plataforma: Tensor por los tensores y Flow por ese flujo de información entre cada uno de los nodos del modelo que se ha diseñado.

El gran equilibrio entre flexibilidad y escalabilidad que presenta TensorFlow, ha permitido que los desarrolladores e investigadores del ámbito de la Inteligencia Artificial puedan probar modelos nuevos y variados en poco tiempo, y que dichos modelos puedan ser usados por millones de usuarios y en todo tipo de dispositivo, ya que además es una solución portable.

Algo interesante que presenta TensorFlow es que proporciona APIs de todos los niveles: desde nivel bajo para los desarrolladores más expertos y que buscan crear modelos totalmente nuevos hasta los niveles más altos para usuarios principiantes o que quieren crear un modelo de forma rápida y sencilla. En la siguiente imagen se pueden ver los diferentes niveles de los que dispone TensorFlow.

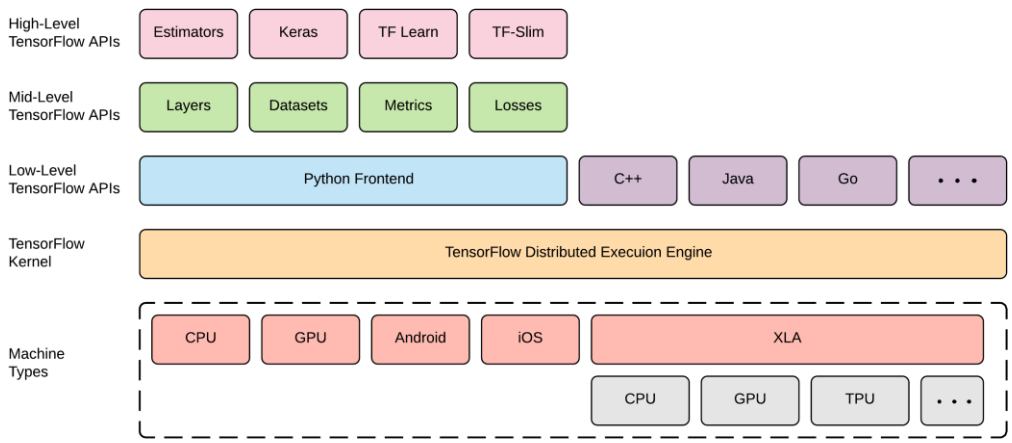


Figura 37. Estructura de TensorFlow [46]

Algo que se puede observar también en el diagrama es que TensorFlow se puede ejecutar sobre cualquier dispositivo. Esta portabilidad que ofrece la plataforma la convierte en idónea para llevar modelos a producción, ya que el usuario final podrá ejecutar los modelos se cual sea el dispositivo del que disponga.

Entre las capas de nivel alto se puede observar la presencia de la librería Keras, la cual se tratará en el siguiente apartado de este documento. Tanto Keras como el resto de APIs presentes en su nivel más alto permiten desarrollar un pipeline de Machine Learning de forma rápida y sencilla, haciéndolo idóneo para aquellas soluciones comunes y conocidas y también para aquellos usuarios que se estén iniciando en el desarrollo de modelos de Machine Learning. Cabe destacar que estos niveles más altos de TensorFlow están desarrollados únicamente en Python.

En el nivel medio de APIs podemos encontrar diferentes paquetes que ofrece la plataforma para facilitar el desarrollo de modelos. Estos paquetes permiten que el desarrollador no tenga que escribir cada una de las ecuaciones matemáticas que conforman el modelo, sino que por el contrario ofrecen una librería con métricas, funciones de coste, capas de Redes Neuronales y datasets ya creados para que el desarrollador del modelo solo tenga que hacer una llamada a la función a la hora de emplear elementos comunes en la creación de algoritmos de Machine Learning.

Por otro lado, el nivel bajo permite programar el modelo en varios lenguajes distintos, lo cual aporta mayor flexibilidad a los desarrolladores, permitiendo el desarrollo de modelos en C++, Java, Go, JavaScript y Swift. Es especialmente interesante la posibilidad de programar modelos en C++, ya que es un lenguaje mucho más rápido en términos de tiempos de ejecución que Python lo cual lo hace ideal para aquellos modelos que en producción deben dar una respuesta en tiempo real.

Es en este nivel bajo en el que se construyen los gráficos computacionales mencionados al comienzo de esta sección. Estos gráficos permiten especificar el flujo de información del proceso que se lleva a cabo para entrenar un modelo, y además permite paralelizar cada uno de los subprocesos especificando entre qué procesadores se reparte la carga de trabajo. Estos procesadores pueden ser de tipo CPU (Central Processing Unit), GPU (Graphics Processing Unit) o TPU (Tensor Processing Unit). A continuación, se muestra un ejemplo de cómo sería un gráfico computacional para una operación simple:

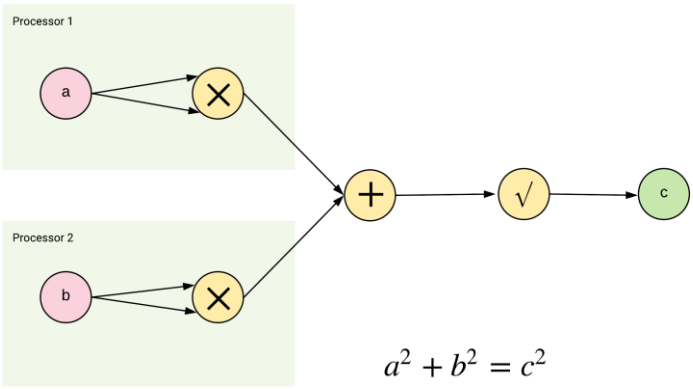


Figura 38. Ejemplo de gráfico computacional [46]

Para crear un programa en TensorFlow, primero debemos diseñar uno de estos grafos que especifiquen qué operaciones se realizarán y cómo fluye la información entre operaciones, y posteriormente ejecutar el grafo. La gran capacidad de paralelización de procesos que presentan los gráficos computacionales convierte TensorFlow en una solución muy flexible a la hora de repartir cargas de trabajo cuando se dispone de varios procesadores. Para paralelizar, simplemente habrá que crear dos ramas tal y como se puede observar en la imagen superior, y especificar qué componente será el encargado de ejecutar cada una de ellas.

KERAS

Keras, cuyo nombre proviene del griego κέρας, y significa bocina [47], es una librería de modelos de Redes Neuronales de alto nivel escrita en Python que puede correr sobre TensorFlow, CNTK y Theano, aunque su popularidad es mucho mayor con TensorFlow. Es la librería más fácil de usar de entre las que se presentan en este documento, lo cual ha hecho que gane popularidad debido a la simplicidad sintáctica y por facilitar un desarrollo rápido y sencillo en comparación con las otras opciones disponibles.

Keras ofrece al usuario desde capas que puede conectar fácilmente hasta modelos preentrenados y datasets disponibles para implementar con una línea de código. Esto es lo que ha hecho que se vuelva tan popular no solo entre los usuarios menos expertos, sino también en aquellos

usuarios que buscan una implementación rápida de un modelo gracias a estas herramientas que la plataforma ofrece.

Además, hay varias maneras en las que se puede trabajar en Keras, lo cual convierte a esta librería en una solución idónea para usuarios con niveles de experiencia muy distintos y que buscan niveles de personalización diferentes. A la hora de emplear Keras para construir modelos de Machine Learning en TensorFlow hay tres métodos diferentes que ofrecen soluciones distintas al usuario:

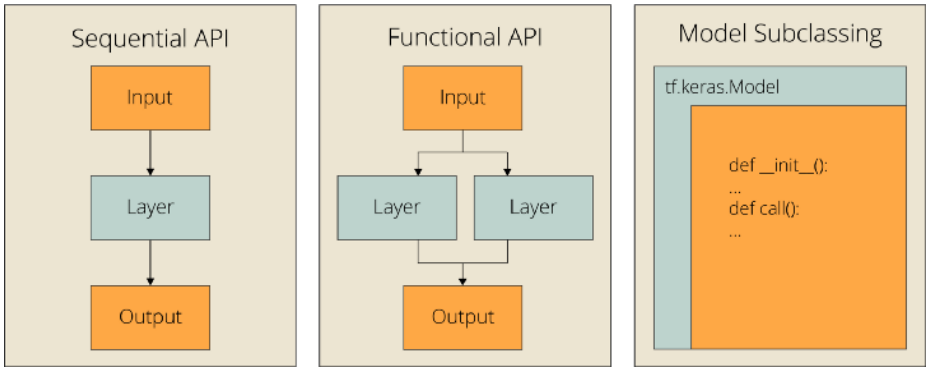


Figura 39. Métodos de Keras para la creación de modelos de Machine Learning [48]

A continuación, se explican cada una de estas APIs, ordenadas de menor a mayor dificultad de implementación y flexibilidad de la solución:

- **La API secuencial ("Sequential"):** este método es la mejor solución cuando el modelo que se quiere diseñar es simple, es decir, con una única entrada, una única salida y sin ramificaciones intermedias. Permite programar modelos de forma rápida y sencilla, reduciendo mucho el tiempo empleado en la programación del mismo, permitiendo que el usuario centre sus esfuerzos en la búsqueda de la arquitectura y los hiperparámetros óptimos. Esta solución es ideal para aquellos usuarios menos expertos.

- **La API funcional ("Functional"):** este método, tal y como se puede observar en la imagen, permite crear modelos más complejos que el módulo anterior (aunque con el también se pueden crear modelos simples como en el caso del secuencial), ya que permite introducir en el modelo varias entradas, obtener varias salidas y crear

ramificaciones. Es el módulo más popular de Keras, ya que permite gran flexibilidad manteniendo un nivel de simplicidad alto a la hora de programar, lo que permite crear de forma rápida y simple modelos muy personalizados. Esta solución es popular entre los usuarios profesionales que ya tienen experiencia previa en el diseño e implementación de modelos de Machine Learning.

- **Subclases de modelos ("Model Subclassing"):** este método es más complejo que los dos anteriores, y requiere de un nivel avanzado de experiencia para emplearlo a la hora de desarrollar modelos. Consiste en crear una clase del modelo que defina cada una de las características del mismo. Esta solución es la más empleada por investigadores con experiencia en el diseño y programación de modelos y que requieren de un alto grado de flexibilidad en el diseño de algoritmos.

En lo que a Visión Artificial se refiere, Keras ofrece muchas facilidades a los desarrolladores, tanto para comprender de forma sencilla cómo programar modelos en Keras (a través de múltiples tutoriales y códigos de ejemplo) como para acelerar el proceso de creación de modelos con diferentes módulos de alto nivel.

Además de disponer de gran cantidad de documentación online, se pone a disposición de los desarrolladores un gran repositorio de códigos de ejemplo [49] no solo para Visión Artificial sino para todas las ramas que el Machine Learning tiene en la actualidad. Los códigos de ejemplo para Visión por Ordenador son muy variados y cubren todos los grandes objetivos que se plantean para procesamiento de imágenes actualmente; ofreciendo códigos para clasificación de imágenes, segmentación de imágenes, detección de objetos, super resolución, etc.

Además, Keras proporciona una gran cantidad de modelos preentrenados [50] para aquellos usuarios que deseen hacer Transfer Learning, la cual es una técnica muy beneficiosa tal y como se comentará más adelante en este documento. Entre ellos podemos encontrar Xception, VGG, Resnet, Inception, DenseNet, etc. Todos estos modelos de tratamiento de imágenes pueden reducir mucho el tiempo de implementación, permitiendo al usuario utilizar estos modelos que han sido previamente entrenados e incluirlos en su pipeline con una sola línea de código.

Adicionalmente, esta librería dispone de una serie de pequeños datasets para que el usuario pueda emplearlos utilizando una única línea de código. El número de este datasets es reducido, así como la cantidad de datos que los forman. Keras debe aún desarrollar más esta parte en su plataforma, ya que solo se podrá hacer uso de ella en ocasiones muy concretas en las que la aplicación del proyecto coincida con los datos que ofrecen los limitados datasets disponibles.

PYTORCH

PyTorch es una librería escrita en Python que surgió como evolución de Torch, una librería basada en C. Fue desarrollado por Facebook e introdujo la aceleración mediante GPU, para la cual PyTorch utiliza internamente CUDA; una API desarrollada por NVIDIA que conecta la CPU con la GPU. Esta librería está gozando de mayor popularidad en las áreas de Visión Artificial y Procesamiento del Lenguaje Natural (NLP).

PyTorch es más reciente que TensorFlow, pero a pesar de ello, dispone de una gran cantidad de documentación y tutoriales disponibles para los usuarios, además de una comunidad de desarrolladores muy colaborativa.

La principal característica y ventaja de PyTorch es la gran flexibilidad que ofrece al usuario. A pesar de que esta flexibilidad se consiga a costa de ser una librería de bajo nivel (al igual que los niveles más bajos de TensorFlow), su programación es sustancialmente más fácil que en esta otra librería debido a que está totalmente basado en Python, y por tanto sigue el principio de facilidad en la lectura que defiende este lenguaje de programación, o como se dice al referirnos a esta característica, es muy “Pythonic”. Esto hace que a pesar de programar a nivel bajo los modelos de Machine Learning, no sea tan complejo para el usuario como en el caso de los niveles de TensorFlow que ofrecen una flexibilidad comparable.

AL igual que TensorFlow, PyTorch se basa en gráficos computacionales, pero, a diferencia de esta otra librería, PyTorch emplea gráficos computacionales dinámicos en vez de estáticos, lo que permite al usuario cambiar el comportamiento de la red neuronal en cualquier momento.

PyTorch es muy similar a la librería Numpy, lo cual lo hace intuitivo para aquellos usuarios que tienen experiencia desarrollando programas en los que se realizan cálculos matemáticos con

Python. La gran ventaja que tiene PyTorch sobre Numpy es la capacidad de ser ejecutado en una GPU, lo cual acelera mucho el entrenamiento de las redes neuronales.

A la hora de programar un modelo de Machine Learning en PyTorch, se debe de programar cada uno de los pasos que debe seguir el modelo, es decir, hay que programar el pipeline de forma detallada. Esto implica la necesidad de definir cada una de las capas que serán empleadas (incluido el tamaño del dato de entrada, hiperparámetros de la misma y el tamaño de salida), programar la forma en la que los datos fluyen entre cada una de las capas (arquitectura del modelo) y programar cada uno de los bucles necesarios para entrenar: el bucle de entrenamiento, de validación y de test, haciendo en cada uno de ellos las correspondientes llamadas a la pasada hacia delante por la red (forward propagation) y la pasada hacia atrás (backpropagation), así como las funciones necesarias para llevar a cabo el control del entrenamiento mediante la creación de KPIs.

Como se puede observar, la programación del modelo se realiza en mucho detalle, lo cual lleva a necesitar una cantidad importante de programación inicial para preparar el pipeline, pero aporta total flexibilidad en la creación del modelo y mucha facilidad en el análisis de cómo se está comportando el mismo, ya que podemos controlar de forma fácil cada una de las partes del pipeline diseñado.

En cuanto a las facilidades que ofrece PyTorch a la hora de desarrollar modelos de Visión Artificial con esta librería, cabe destacar la gran cantidad de herramientas que hay disponibles para el usuario. Es por ello por lo que actualmente es la solución más popular para el desarrollo de modelos de Visión Artificial. El módulo que presenta PyTorch y que supone la gran diferencia con respecto al resto de librerías de Machine Learning a la hora de desarrollar modelos de Visión Artificial es el denominado Torchvision [51].

Este módulo contiene a su vez diferentes partes:

- **Torchvision.datasets**: esta parte de la librería de Torchvision permite al usuario implementar de forma rápida y sencilla algunos de los datasets más empleados en la actualidad para el entrenamiento de modelos de Visión Artificial. No ofrece una llamada a la descarga de los datasets, sino que permite que, una vez sean descargados, el desarrollador pueda cargar los datos en el modelo con una sola línea de código.

Además, en esta carga de datos automatizada permite introducir varios parámetros para personalizar la carga, permitiendo elegir por ejemplo el número de fotogramas a usar en cada uno de los vídeos del dataset (en el caso de datasets de vídeo), transformaciones de los datos (como convertir los vídeos de color a escala de grises), etc. Esto permite ahorrar mucho tiempo a la hora de tratar con grandes datasets, los cuales tienen su propia forma de ser almacenados y tratados, evitando que el desarrollador tenga que crear un código para extraer los datos del mismo.

- **Torchvision.io**: este paquete de la librería de Torchvision ofrece diferentes funciones para realizar operaciones de entrada y salida de datos del modelo, es decir, permite adaptar los datos de entrada a los requerimientos de salida y también transformar los datos de salida. Por ejemplo, permiten convertir imágenes o vídeos del formato de origen a una variable de tipo tensor, la cual es la que necesita el modelo para ser entrenado. En cuanto a las transformaciones de salida, permite por ejemplo convertir una variable de tipo tensor a una imagen o vídeo.

- **Torchvision.models**: en Visión Artificial, hay una serie de modelos de gran popularidad, ya que son los que mejores resultados presentan para el problema que pretenden resolver. Algunos de estos modelos son AlexNet, ResNet o DenseNet. Torchvision ofrece a los usuarios algunos de estos modelos preentrenados y listos para ser implementados usando una sola línea de código. Se ofrece la posibilidad de emplearlos ya entrenados o simplemente la estructura sin entrenar.

- **Torchvision.ops**: esta parte de la librería de Torchvision ofrece al usuario una serie de operadores ya implementados específicos para Visión Artificial. Por ejemplo, se encuentra implementada la operación de non-maximum supression empleada en los algoritmos de detección de objetos en imágenes.

- **Torchvision.transforms**: consiste en una serie de operaciones de transformación de imágenes que facilitan mucho el proceso de aplicación de data augmentation, permitiendo por ejemplo hacer recortes aleatorios, cambiar el brillo, contraste y

saturación de las imágenes, convertir las imágenes a escala de grises o añadir un marco a las imágenes.

- **Torchvision.utils**: este módulo de Torchvision facilita algunas tareas más específicas de generación de imágenes. Permite en una sola línea de código crear una imagen compuesta por una matriz de imágenes, crear imágenes que contengan las "Bounding Boxes" que localizan los objetos en las mismas en el problema de detección de objetos y convertir de tensor a imagen y guardarlo.

PyTorch es la librería que se ha empleado a la hora de programar la solución de este proyecto la decisión se basó en la flexibilidad que esta librería aporta sin incurrir en una gran complejidad a la hora de programar, lo cual permitió crear modelos muy personalizados e iterar rápidamente entre las diferentes arquitecturas planteadas. Además, PyTorch es una gran solución para el diseño de modelos de Visión Artificial, debido a estos módulos especializados que se han comentado previamente.

COMPARATIVA

A continuación, se presenta una tabla comparativa con diferentes atributos, que puede servir para seleccionar la librería que más se adapta a los requerimientos de cada proyecto de Machine Learning: [52]

	TensorFlow	**Keras**	**PyTorch**
Nivel de API	Bajo y alto	Alto	Bajo
Velocidad de ejecución	Alta	Baja	Alta
Dificultad en uso	Baja o alta dependiendo del nivel de API usado	Baja	Alta
Facilidad de depuración	Difícil	Fácil	Fácil
Flexibilidad en el desarrollo de modelos	Media	Baja	Alta

Lenguajes de programación disponibles	C, C++, Python, Java, Go	Python	Python
Recomendado para	Datasets de gran tamaño y necesidad de alto rendimiento	Rápido prototipado y datasets pequeños	Necesidad de gran flexibilidad y de facilidad de depuración.

Tabla 1: Comparativa entre TensorFlow, Keras y PyTorch

6.2. CARGA DE DATOS CON PYTORCH

Una gran parte del tiempo de diseño de una solución de Machine Learning se dedica al tratamiento de datos. Tras completar la tarea de crear el dataset y llevar a cabo data augmentation en los casos en los que es necesario, se debe generar un flujo que permita introducir los datos en el modelo.

La forma de implementar este flujo dependerá de cómo se hayan guardado los datos en memoria, ya que se debe ser capaz de trabajar con la estructura de los mismos para poder acceder a ellos correctamente.

A la hora de diseñar un flujo de carga de datos, se debe tener en cuenta las capacidades y requerimientos del dispositivo que se emplea para ejecutar el modelo y de las necesidades de entrenamiento que hay. Esto se debe a que en PyTorch se puede personalizar totalmente este flujo, de forma que se permita tanto cargar todos los datos al comienzo del entrenamiento en memoria o cargar los lotes según se vayan necesitando directamente desde disco. Ambas opciones tienen sus ventajas e inconvenientes, como se comentará a continuación.

El primer caso comentado, el de realizar una carga de datos inicial volcando todo el dataset sobre la memoria RAM, es ideal para aquellos casos en los que el dataset sea de menor tamaño que el disponible en este elemento, es decir, para aquellos casos en los que sea posible contener todo el dataset dentro de la memoria RAM (Random Access Memory). Este tipo de memoria permite acceder a los datos de una manera más rápida que si se accediera a los datos del disco duro. Esto hace que la

carga de los lotes de entrenamiento, validación y test en el modelo se ejecuten de una forma rápida, y por tanto reduce en gran cantidad el tiempo de entrenamiento, ya que durante el mismo se realizan cientos de cargas de datos.

El segundo caso que se ha comentado, el de acceder únicamente a los datos necesarios directamente en disco, es la solución necesaria en aquellos casos en los que se dispone de un dataset de grandes dimensiones. En el caso de Visión Artificial, los datasets suelen ocupar una gran cantidad de gigabytes de memoria, lo cual hace imposible poder almacenar los datos en una memoria RAM a la hora de entrenar, por lo que es necesario acceder al lote de datos que se necesita en cada momento obteniéndolos directamente del disco, cargando así únicamente una pequeña cantidad de datos cada vez en la memoria RAM, que será la encargada de almacenar los datos temporales durante el entrenamiento. Para poder crear este flujo de carga de una forma más rápida y sencilla en algunos casos, PyTorch ofrece a los usuarios una serie de funciones ya implementadas para dicho flujo que se pueden emplear en algunos casos genéricos. Estas funciones se comentarán en detalle a continuación.

En PyTorch, el proceso de carga de datos pasa por definir dos elementos: un Dataset (no confundir con el conjunto de datos contenidos en disco) y un Dataloader.

TORCH DATASET

Un Dataset es una clase que representa nuestro conjunto de datos, y que nos permite acceder a estos de uno en uno. El diseño de la estructura del conjunto de datos (cómo son organizados en carpetas y cómo se etiquetan y donde se sitúan dichas etiquetas), se debe llevar a cabo teniendo en mente que se debe poder acceder a los datos y cargar los mismos de una forma rápida y sencilla, ya que se realizan cientos o miles de accesos a los datos durante el entrenamiento. Esto supone diseñar la estructura del conjunto de datos intentando facilitar todo lo posible la implementación de esta clase Dataset.

PyTorch ofrece una serie de clases ya implementadas que permiten implementar esta funcionalidad de forma rápida y en una sola línea de código. Cabe destacar dos grupos diferenciados dentro de estas clases prediseñadas:

- **Torch Dataset para conjuntos de datos populares:** estas implementaciones predefinidas permiten cargar los datos de ciertos conjuntos de datos con una sola línea de código, no teniendo que programar dicha tarea. Algo a destacar de PyTorch es la velocidad a la que integra los conjuntos de datos que comienzan a gozar de popularidad, de forma que el usuario encuentre una forma rápida de integrarlos en su solución de forma rápida y sencilla. Algunos de los conjuntos de datos que son una referencia en la implementación de soluciones de Visión Artificial y que disponen de funciones Dataset implementadas en PyTorch para su uso son COCO, MNIST, Kinetics-400, CIFAR y UCF-101. En la actualidad el usuario dispone de 25 de estas clases para los conjuntos de datos más famosos, ofreciendo esta característica para una variedad de problemas de Visión Artificial.

- **Torch Dataset genérico**: además de la implementación de una clase específica para ciertos conjuntos de datos, PyTorch ofrece una solución genérica para aquellos usuarios que emplean un conjunto de datos propio. Para poder implementarlos en el pipeline del proyecto, se debe emplear la misma estructura y nomenclatura de carpetas que se especifica en la documentación a la hora de organizar el conjunto de datos en disco. Dos ejemplos de esta solución son DatasetFolder y ImageFolder. Ambas soluciones son empleadas en aquellos problemas en los que los datos se guardan en forma de archivos, de forma que cada dato supone un archivo completo. Es por ello que son ampliamente utilizados en soluciones de Visión Artificial, siendo DatasetFolder una buena solución para aquellos proyectos en los que se trabaja con datos en formato vídeo e ImageFolder para aquellos proyectos en los que se trabaja con imágenes.

```
root/class_x/xxx.ext
root/class_x/xxy.ext
root/class_x/[...]/xxz.ext

root/class_y/123.ext
root/class_y/nsdf3.ext
root/class_y/[...]/asd932_.ext
```

Figura 40. Estructura de carpetas para DatasetFolder [53]

Además de emplear estas clases ya creadas para ejecutar en una sola línea, el usuario puede crear su propia clase personalizada, que permitirá resolver el problema de la carga de datos con gran flexibilidad, ofreciendo una solución a medida para el usuario. Cabe mencionar que al crear Datasets personalizados, se puede hacer uso de las funcionalidades de estas clases ya creadas por PyTorch, por lo que no es necesario implementar todo el código de cero, sino simplemente adaptar aquellas partes que interesen al usuario.

A la hora de desarrollar una clase Dataset personalizada, se debe tener en cuenta que esta clase tiene dos funciones principales: __len__, que sirve para especificar el tamaño del conjunto de datos, es decir, para especificar el número de datos del que se dispone, y __getitem__, que permite acceder a cada uno de estos datos para poder hacer uso de ellos. Además, si se quiere tener acceso a ciertas características que ofrecen los Datasets de PyTorch como la carga de datos en paralelo, se debe instanciar la clase base de Dataset ya implementada en esta librería.

Cabe mencionar que PyTorch pone a disposición del usuario documentación muy detallada acerca de cómo desarrollar esta clase Dataset de forma personalizada de forma sencilla, explicando aquellas cosas a tener en cuenta en cada una de las funciones clave a implementar para el correcto funcionamiento de la solución.

TORCH DATALOADER

Una vez se ha implementado el Dataset, el cual permite obtener los datos del conjunto de datos de uno en uno, se debe desarrollar el Dataloader, quien hará uso del Dataset para obtener los datos del disco y ofrecer un iterador sobre los datos.

Torch Dataloader es una API disponible para los usuarios que permite realizar la carga de datos en el modelo. Haciendo uso de la funcionalidad que le proporciona el Dataset para la obtención de datos, permite crear un iterador sobre los datos del conjunto para poder cargarlos de forma configurable en el modelo. Esta carga de datos permite generar lotes de entrenamiento con un tamaño seleccionado por el usuario, barajar los datos entre cada época para reducir el overfitting del modelo y hacer uso de las capacidades de multiproceso de PyTorch para acelerar la carga de datos.

Todo esto se puede hacer con tan solo una línea de código, lo cual no solo permite agilizar considerablemente el proceso de implementación de un sistema de carga de datos, sino que además reduce mucho la probabilidad de cometer errores en la carga de los mismos, lo cual llevaría a no alcanzar buenos resultados a la hora de entrenar.

Ahora que se han comentado las diferentes maneras que existen de cargar datos, cabe mencionar la forma en la que se han cargado en este proyecto.

Dado que el dataset del que se dispone es pequeño (tal y como se ha mencionado anteriormente), se ha optado por cargar los datos directamente en la memoria RAM, para garantizar un acceso más rápido a los mismos, reduciendo así los tiempos de ejecución a la hora de entrenar. Además, para reducir el tiempo de carga de datos, se ha optado por realizar un procesado de los mismos previo al entrenamiento.

Esta reducción de tiempos derivada del preprocesado se debe a la cantidad de veces que debe ser procesado un vídeo para convertirlo de formato vídeo a un formato con el que pueda trabajar el modelo. En el caso del preprocesado, solo se debe cambiar el formato del vídeo una única vez, mientras que, en el caso de no llevarlo a cabo, se debe procesar cada uno de los vídeos cada vez que se introduzca en el modelo. Teniendo en cuenta que en cada entrenamiento se hacen miles de llamadas a la obtención de datos (y con ello al procesado), el tiempo que se reduce con esta técnica es considerable.

Este procesado consiste en convertir los vídeos que conforman el dataset en matrices de tipo Numpy, y almacenarlas en un único archivo por cada set de datos (uno para entrenamiento, otro para validación y otro para test) de extensión npz. Esta extensión ha sido la elegida para el proyecto debido a que es la manera en la que se pueden almacenar este tipo de datos en un formato comprimido, de manera que el espacio en disco sea menor, lo cual ha permitido entrenar el modelo en el cloud reduciendo el espacio de almacenamiento requerido.

Este archivo solo se genera una vez para el formato de vídeo empleado, es decir, si se crea un archivo de este tipo con los vídeos con dimensiones 224x224x3 y con data augmentation en modo espejo e introducción de ruido, no será necesario volver a generar el archivo si se mantienen estos parámetros fijos, ya que se comprueba la existencia del mismo para los parámetros requeridos y se

carga sin necesidad de generar el archivo de nuevo. Por el contrario, si se varía por ejemplo el tamaño de los fotogramas, será necesario generar un archivo npz conteniendo los vídeos con el nuevo formato, pero no siendo necesario generarlo las próximas veces que se emplee dicho formato (siempre y cuando no se elimine el archivo).

Esta capacidad de comprobar la existencia del archivo para recudir el número de veces que se genera el mismo se consigue mediante el uso del nombre del archivo, en el que se especifican estos parámetros de los vídeos que contiene.

Una vez que se ha generado (o cargado) el archivo npz conteniendo los vídeos procesados, este se emplea para crear un Dataset de tipo TensorDataset, sobre el que se apoya el DataLoader para llevar a cabo la carga de datos del modelo. Este tipo de Dataset permite acceder a un conjunto de datos que está contenido en una variable de tipo tensor en vez de en un sistema de carpetas, permitiendo almacenar los datos en la memoria RAM para una mayor velocidad de carga, tal y como se ha comentado anteriormente.

6.3. TRANSFER LEARNING

Transfer Learning es una de las técnicas que consiste en emplear una red que ya ha sido entrenada para resolver un problema, y usarla o adaptarla para resolver un problema similar, es decir, pretende llevar a cabo una transferencia de conocimiento entre una red neuronal ya existente y una que está siendo diseñada para solventar un problema similar o que requiere de la tarea en la que se ha especializado la red ya entrenada. El Transfer Learning se ha convertido en una de las metodologías más importantes del Deep Learning, ya que permite reducir considerablemente no solo el tiempo de entrenamiento sino también la cantidad de datos necesarios para entrenar y el overfitting de la solución.

Para comprender bien lo que supone el Transfer Learning a la hora de desarrollar modelos de Deep Learning, analicemos un ejemplo. Imaginemos que queremos aprender a jugar a un deporte nuevo, por ejemplo, el baloncesto. Previamente a aprender a practicar este deporte, ya se parte de una base: sabemos andar, correr y mover los brazos de la forma deseada. Todo este conocimiento previo permite que el aprendizaje sea mucho más rápido, ya que no tendremos que centrarnos en aprender

todo lo previamente descrito, sino que podremos centrarnos en aprender a encestar una canasta. El Transfer Learning permite aprovechar este conocimiento previo, lo cual acelera mucho el aprendizaje de la red, ya que se puede hacer uso de redes que han sido entrenadas para tareas necesarias o similares a las empleadas para resolver el problema que se afronta.

Dos de los grandes problemas del Deep Learning que se pretenden afrontar con el uso del Transfer Learning son los largos tiempos de entrenamiento y la necesidad de grandes cantidades de datos. El hecho de poder emplear un modelo en el que ya se ha invertido gran cantidad de tiempo y que ha sido entrenado con gran cantidad de datos en vez de crear un modelo de cero y pasar por toda esa fase de procesado de datos y entrenado de modelos, permite a los desarrolladores no tener que volver a resolver un problema que ya ha sido exitosamente resuelto, pudiendo así construir sobre los modelos ya existentes de Deep Learning. Además, debido a que la red preentrenada ya es capaz de extraer características de los datos, se requerirá de un menor número de datos para entrenar el nuevo modelo que integra la red preentrenada que en el caso de diseñar un modelo de cero y tener que entrenarlo en su totalidad.

A continuación, se presenta una imagen en la que se muestra este nuevo enfoque que supone el Transfer Learning con respecto a los métodos de Machine Learning tradicionales:

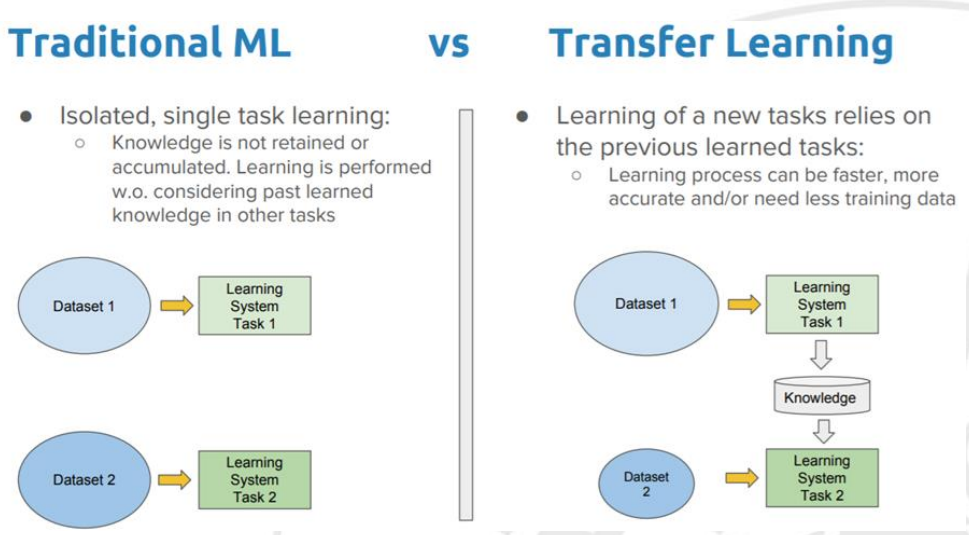


Figura 41. Transfer Learning [54]

Existen muchas tareas, especialmente en Visión Artificial, que son similares, y que por tanto permiten emplear esta técnica para adaptar una solución existente para una solución parecida. Por ejemplo, se puede reentrenar un modelo que ofrece buenos resultados en la detección de coches para que en su lugar detecte personas.

Existen diferentes tipos de Transfer Learning, que permiten aplicar esta técnica a problemas diferentes:

- **Domain adaptation**: consiste en emplear un modelo ya entrenado en un dominio distinto, es decir, cuando se pretende resolver el mismo problema, pero la distribución de los datos del dataset con el que el modelo fue entrenado originalmente es diferente a la del dataset que se empleará para resolver el nuevo problema. Un ejemplo de este tipo de Transfer Learning sería emplear un modelo que ha sido diseñado para detectar personas en centros comerciales (y que por tanto todos los datos de entrenamiento eran imágenes en centros comerciales) y emplearlo en una solución de detección de personas la calle (donde las imágenes del nuevo dataset serán de gente en exteriores). El hecho de que cambie la distribución de los datos implica la necesidad de ajustar un poco los parámetros del modelo para que la precisión se mantenga, ya que, si no se realiza ninguna modificación en este, los resultados obtenidos serán peores. Esta técnica de reentrenado se comentará más adelante en este apartado.

- **One-shot Learning**: una de las principales características de los modelos de Deep Learning es la necesidad de alimentar al modelo con una gran cantidad de datos, mientras que lo ideal sería que aprendieran como lo hacemos las personas, que vemos una ocurrencia de un objeto por ejemplo y sabemos identificar nuevas ocurrencias del mismo tipo de objeto. Esta necesidad de entrenamiento con elevadas cantidades de datos lleva a entrenamientos de larga duración, y puede suponer un problema en aquellas aplicaciones para las que no se dispone de un dataset de grandes dimensiones. Este problema se pretende resolver con este tipo de Transfer Learning, cuyo objetivo es conseguir que el modelo aprenda a clasificar los datos habiendo visto solo un ejemplo o unos pocos ejemplos de cada uno de ellos. Por ejemplo, el modelo sería capaz de detectar una naranja tras haber visto únicamente una imagen conteniendo una

naranja. Esto se consigue usando un modelo preentrenado para una tarea similar, de manera que sea capaz de extraer las características necesarias que le permitan clasificar datos nuevos viéndolos una única vez. En el ejemplo de la naranja, el modelo puede haber sido entrenado por ejemplo con cientos de imágenes de casas y coches, y por tanto está preparado para llevar a cabo esa clasificación. Al haber aprendido a extraer características para detectar estos dos objetos, se pretende que, al introducirle una única imagen de un objeto nuevo, sea capaz de aprender a clasificarlo al ver imágenes diferentes de ese mismo objeto (en este ejemplo, la naranja).

- **Zero-shot Learning**: este tipo de Transfer Learning es similar al anterior, pero en este caso pretende que un modelo consiga clasificar datos que no ha visto nunca. Esta técnica permite adaptar una red ya entrenada de forma que sea capaz de clasificar datos que no ha visto nunca. A diferencia de otros métodos de Transfer Learning, este no pretende llevar a cabo un nuevo entrenamiento de la red preentrenada, sino que tiene por objetivo sustituir este entrenamiento para que no sea necesario. Para ello, Zero-shot Learning propone modificar lo aprendido por la red preentrenada, de forma que enfoque este aprendizaje a la detección de nuevos datos, es decir, se pretende extraer los detectores de características que se generan en la red entrenada para detectar patrones que permitan clasificar datos nunca vistos antes por la red preentrenada. Por ejemplo, si se entrena una red que es capaz de detectar caballos y le hacemos saber que una cebra es un caballo con rayas blancas y negras, el modelo será capaz de detectar cebras. Zero-shot Learning es una gran solución para aquellos casos en los que no se disponga de un dataset específico para la nueva aplicación, permitiendo el reconocimiento de patrones sin ejemplos de entrenamiento previos. Esta técnica goza actualmente de gran popularidad, surgiendo nuevos métodos de Zero-shot Learning cada año.

Además, hay diferentes maneras en las que se puede aplicar Transfer Learning dependiendo del resultado final que se quiera obtener. En todos los casos, se partirá de un modelo preentrenado que se emplea como base para el nuevo modelo, y sobre el que se aplican ciertas modificaciones para adaptarlo totalmente a la nueva aplicación. Para ello se debe comprender que los modelos de Machine

Learning se pueden guardar y recrear de forma sencilla. Esto se realiza guardando tanto la arquitectura del modelo como los pesos que la componen, de manera que se pueda replicar el modelo y el resultado del entrenamiento que se ha realizado en el mismo, para poder así replicar los resultados obtenidos.

Las diferentes maneras en las que se puede emplear un modelo preentrenado para realizar Transfer Learning y así adaptarlo a una nueva aplicación son las siguientes:

- **Afinar un modelo preentrenado**: esta técnica consiste en desarrollar y entrenar un modelo con un tipo de datos (llamémoslos B) de los que se dispone en abundancia, para posteriormente partir de dicho modelo para entrenarlo sobre una serie de datos de los que no se dispone la cantidad necesaria para entrenar un modelo completo (llamémoslos datos A). De este modo, lo que esta técnica pretende conseguir es entrenar un modelo con datos que son fáciles de obtener, de forma que el modelo aprenda a reconocer patrones en los mismos de forma que pueda posteriormente adaptar este aprendizaje obtenido para necesitar menos entrenamiento para aprender a clasificar los datos A (ya que parte del aprendizaje que ya se ha llevado a cabo durante el primer entrenamiento). Cabe mencionar que para que esta metodología funcione adecuadamente, los datos A y B deben ser similares, ya que de lo contrario el aprendizaje de la red sobre los datos B no será un buen punto de partida para afinar el aprendizaje con los datos A. Este método es sencillo de implementar, ya que solo requiere de ser capaz de guardar el modelo preentrenado.

- **Congelación de capas**: esta aproximación consiste en emplear únicamente la parte que interesa de un modelo preentrenado, es decir, se pueden obtener las capas que contienen el aprendizaje que se busca y emplearlas en el nuevo modelo en modo de inferencia, de manera que permiten identificar patrones, pero no se entrenan con los nuevos datos. Esta idea de emplear únicamente ciertas capas a modo de extractores de características se presenta en la siguiente imagen:

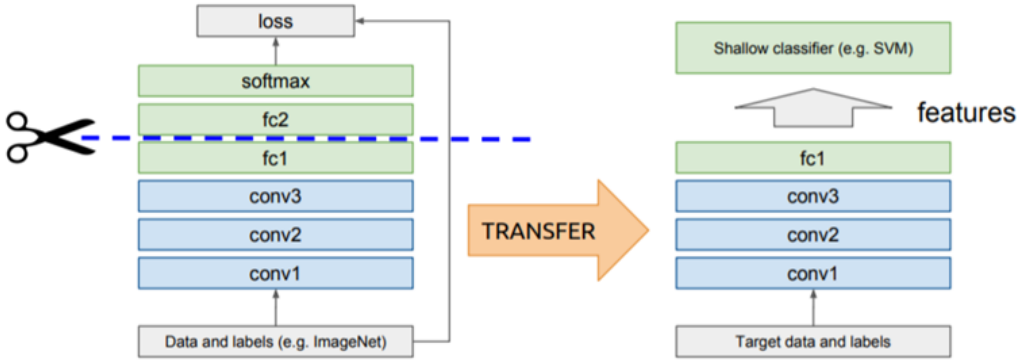


Figura 42. Congelación de capas [54]

En la imagen superior se puede observar cómo se ha extraído del modelo preentrenado la capa encargada de la clasificación. De este modo, se puede emplear el bloque convolucional para extraer características de los datos y crear una nueva capa de clasificación que se entrena con los nuevos datos. El bloque de capas que se extraen del modelo preentrenado no se vuelven a entrenar, o lo que es lo mismo, se congelan dichas capas, de forma que estas extraen la información de los datos para que la última capa (la nueva capa de clasificación) pueda ser entrenada con los nuevos datos para clasificar a partir de estos patrones extraídos por las capas preentrenadas. El hecho de disponer de ciertas capas preentrenadas que no requieren de nuevo entrenamiento, permite reducir la cantidad de datos necesarios para la solución, ya que se deben ajustar menos parámetros. Además, se reducirá el overfitting de la solución, ya que la parte de la red que se entrena es menos compleja, y por tanto se ajustará en menor medida a los datos, favoreciendo la generalización.

Esta última técnica mencionada se implementó en este proyecto tras comprobar que el modelo inicial diseñado sobreentrenaba a pesar de la exhaustiva regularización que se aplicó en el mismo. Con esta segunda implementación se pretendía paliar la falta de datos del dataset. Para ello se probaron varias arquitecturas de ResNet (ResNet18 y ResNet50), las cuales conformaban el bloque convolucional encargado de extraer características de cada uno de los fotogramas. A este modelo preentrenado se le añadió a la salida (sustituyendo la última capa densa) la capa LSTM y el bloque de clasificación formado por capas densas. Esto se llevó a cabo con PyTorch de la siguiente manera:

```
model = models.resnet18(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
model.fc = algorithm.CNN_LSTM(channels, nclass)
```

Figura 43. Código de Transfer Learning

Tal y como se puede observar, primero se declara el modelo preentrenado disponible en PyTorch, y posteriormente mediante el bucle for se congelan los pesos de dicho modelo, haciendo que no tengan un gradiente asociado y por tanto que no se lleve a cabo backpropagation. Posteriormente, se sustituye la última capa del modelo Resnet18 (la fully connected de la que dispone al final del modelo) por el modelo diseñado formado por la capa LSTM y las capas fully connected encargadas de la clasificación.

TRANSFER LEARNING CON PYTORCH

PyTorch ofrece no solo la posibilidad de llevar a cabo estos dos métodos de Transfer Learning que se han presentado en el apartado anterior, sino que además pone a disposición del usuario documentación detallada sobre cómo llevarlos a cabo de forma sencilla en unas pocas líneas de código.

Además de la posibilidad de implementar estas dos técnicas de Transfer Learning, PyTorch ofrece una gran cantidad de modelos preentrenados, tal y como se ha mencionado anteriormente. Estos modelos se pueden modificar de forma sencilla para congelar las capas deseadas o afinar el entrenamiento para los requerimientos de nuestra solución, pudiendo hacer todo esto sin necesidad de incurrir en largos códigos. Esto ha quedado reflejado en el apartado anterior donde se explicó cómo se llevó a cabo la implementación de Transfer Learning con PyTorch en este proyecto.

Cabe mencionar que PyTorch presenta una solución muy versátil a la hora de trabajar con los gradientes, permitiendo asociar o no un gradiente a cada peso con la función requires_grad incluida en cada uno de ellos. Esto es realmente útil para seleccionar qué capas del modelo se desean entrenar, aportando gran flexibilidad a la implementación de Transfer Learning.

Además, se puede implementar de forma sencilla el guardado y cargado de modelos, por lo que en el caso de querer implantar en el modelo Transfer Learning de un modelo preentrenado para posteriormente afinarlo, se puede realizar de forma sencilla partiendo de los parámetros iniciales deseados.

6.4. CONV2D + LSTM

En este apartado se pretende explicar el modelo que se ha desarrollado en este proyecto para clasificar vídeos mediante el uso de Visión Artificial. El modelo que se propone para ello es el Conv2D + LSTM, el cual consta de un bloque de capas convolucionales para extraer características de los fotogramas que conforman el vídeo y posteriormente una capa LSTM (Long Short-Term Memory) para realizar un análisis temporal de la secuencia de dichos fotogramas, es decir, para poder comprender la evolución de las imágenes en el tiempo. Finalmente, para poder llevar a cabo la clasificación, se sitúan tras la capa LSTM una serie de capas densas y una Softmax que clasifica entre las dos clases del dataset.

En la siguiente imagen, se presenta el modelo que se creó inicialmente, sobre el que se realizaron ciertas variaciones como se comentará a continuación. El diagrama de la imagen, los datos fluyen de izquierda a derecha:

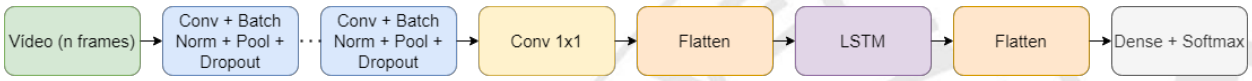


Figura 44. Estructura del modelo desarrollado

A continuación se explicará la funcionalidad de cada una de estas capas y su importancia en el modelo:

- **Vídeo**: el primer componente del diagrama no es una capa, sino que representa el dato de etrada. Estos vídeos que se introducen en el modelo han sido ya tratados empleando las técnicas de procesamiento de datos descritas previamente en este documento. Cabe

mencionar en este apartado la importancia de que las dimensiones de los datos de entrada sean constantes, es decir, que todos los vídeos tengan el mismo número de fotogramas, y cada fotograma el mismo número de píxeles de ancho y alto. Esto se debe a que las capas del modelo son configuradas para admitir un tamaño concreto de dato, y por tanto todos los vídeos deben seguir ese formato para poder ser procesados por la red. También se debe tener en cuenta que estos vídeos no se pueden introducir en formato vídeo en el modelo, sino que deben ser convertidos a tensores, ya que este es el tipo de variable que admiten las capas.

- **Bloque convolucional:** el bloque representado en azul en el diagrama es el convolucional, formado por una capa Convolucional, una capa de Batch Normalization, una capa de Pool y una capa de Dropout. Este bloque permite extraer información de cada uno de los fotogramas que conforman el vídeo. En este modelo hay varios de estos bloques conectados en cascada, de manera que cada imagen pasa por varios de estos bloques, ya que cada uno de ellos se especializa en la detección de unos patrones específicos. A continuación se desarrolla la importancia de cada una de las capas que conforman este bloque:
    - **Capa Convolucional:** esta capa es la encargada de realizar la operación de convolución para reconocer patrones en las imágenes. Su funcionamiento queda explicado en el apartado de estado del arte de este documento.
    - **Batch Normalization:** el objetivo de este tipo de capa es acotar los valores máximos y mínimos de cada dato de entrada de una capa convolucional. El funcionamiento de esta capa y sus beneficios se desarrollan más adelante en este documento.
    - **Pool:** este tipo de capa es ampliamente utilizado a la hora de desarrollar redes neuronales convolucionales. El objetivo de usar este tipo de capas es disminuir el tamaño (en ancho y alto) de cada uno de los mapas de características (“feature maps”) de salida de las capas convolucionales. Esto se consigue mediante la operación de Pooling, que puede ser de dos tipos distintos, llevando así a dos tipos de capa diferentes: Max Pooling y Average Pooling.

Max Pooling consiste en escoger el píxel de mayor valor de entre los píxeles de su alrededor y eliminar el resto. Por otro lado, el Average Pooling consiste en hacer una media de los píxeles cercanos y establecer dicho valor como valor de píxel y eliminar algunas posiciones. A continuación, se presentan estas dos operaciones y los resultados de aplicar las mismas en dos imágenes de ejemplo:

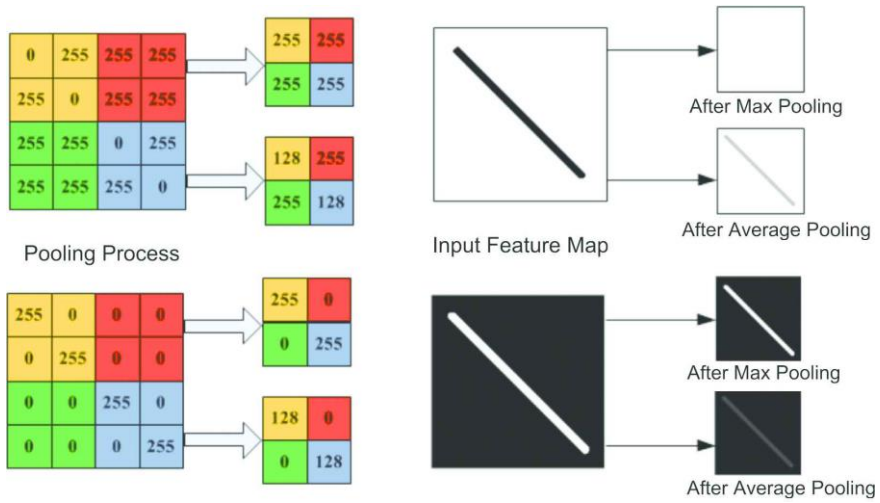


Figura 45. Max Pooling y Average Pooling [55]

Tras realizar varios entrenamientos y profundizar en el análisis de este tipo de capas, se optó por limitarlas en el modelo, pasando de disponer de ella en todos los bloques convolucionales a solo realizarlo en el primer bloque y en el último. Esta decisión se basó en los resultados adversos que pueden presentar. En el caso de Max Pooling, se podrían perder características importantes si se encuentran junto a un valor de píxel grande (tendiendo a blanco) tal y como se puede observar en la imagen superior, donde en el caso de disponer de una linea negra (cuyo valor de píxel es cercano a cero) rodeada de valores de píxel blanco (cuyo valor de píxel es cercano a 255) , se pierde por completo la línea negra. Por otro lado, el Average Pooling puede llevar a la pérdida de información relevante en el caso de activaciones de gran tamaño que pudieran proporcionar información, es decir, en aquellos casos en los que un valor muy distinto de

píxel pueda aportar información a la red, el Average Pooling eliminará dicha información por hacer la media de dicho valor con sus aledaños.

- **Dropout:** el objetivo de este tipo de capa es introducir un efecto regulador en el bloque convolucional, es decir, reducir el overfitting que ocurre en dicho bloque. La capa de Dropout consiste en prescindir de ciertos parámetros de forma aleatoria a la hora de entrenar, de forma que el resto de parámetros no se ajusten perfectamente a los datos, reduciendo así la cantidad de overfitting durante el entrenamiento. Los parámetros que se mantienen activos o inactivos varía durante el entrenamiento, de forma que en la capa de Dropout solo habrá que indicar la cantidad de parámetros que queramos descartar en cada pasada de datos por la red, por ejemplo, el 30% de ellos. Esto queda detallado en la siguiente imagen, donde se puede ver que al aplicar dropout se prescinde de ciertos valores del filtro convolucional:

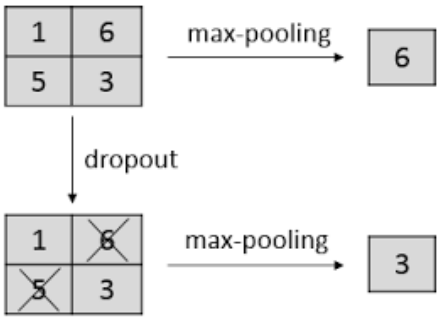


Figura 46. Dropout [56]

En versiones posteriores del modelo, se terminó por prescindir de esta capa de Dropout debido a su escaso efecto en las redes convolucionales tal y como se comentará más adelante en este documento en la sección que trata sobre regularización.

- **Conv1x1**: esta capa consiste en una capa convolucional cuyo tamaño de filtro es de dimensiones 1x1. Este tipo de capa se emplea en la mayoría de las redes convolucionales actuales, ya que permite reducir en gran cantidad el número de características con las que se debe tratar sin perder información en dicha reducción.

Lo que esta capa permite es reducir el número de canales mientras se introduce una no linealidad, y sin reducir el tamaño de cada uno de estos canales, es decir, manteniendo el alto y ancho de cada una de ellos. Es decir, esto permite por ejemplo reducir de una característica o salida de una capa de 112x112x128 a una característica de 112x112x60. Por otro lado, también permite aumentar el número de canales de la misma manera en la que los reduce, aunque este caso de uso es menos común.

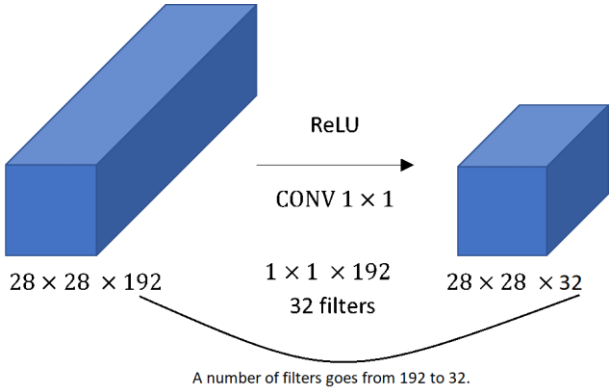


Figura 47. Conv 1x1 [34]

- **Flatten**: las capas de flatten permiten cambiar el formato de las características de la capa anterior, es decir, mantiene los mismos valores pero los distribuye de una manera diferente. En este modelo, los datos que salen del bloque convolucional siguen una estructura matricial del tipo NHxNWxNC, mientras que los datos de entrada que admite la capa LSTM son del tipo 1xN. Esto lleva a la necesidad de cambiar la estructura de los datos que llegan de los bloques convolucionales para adaptarlos a la capa LSTM, lo cual se lleva a cabo en esta capa flatten. Esto queda ilustrado en la siguiente imagen:

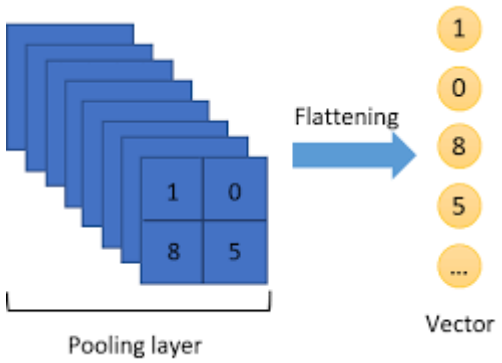


Figura 48. Flattening

En el caso de la segunda capa de Flatten presente en el modelo, es necesaria para introducir en la capa densa encargada de la clasificación los vectores de salida generados por la LSTM para cada uno de los fotogramas del vídeo.

- **LSTM**: esta capa es la encargada de llevar a cabo el análisis temporal de la información extraída de las imágenes por las capas anteriores. La ventaja que tiene esta capa frente a otras estructuras de Redes Neuronales Recurrentes es la capacidad de tratar con secuencias largas de datos. Esta capacidad no se ve presente en otras capas recurrentes, las cuales sufren del problema de "exploding and vanishing gradients", es decir, que los gradientes de los parámetros tiendan a infinito o a cero, lo cual ocurre en mayor medida cuando las secuencias son largas. Para poder solucionar este problema, se sustituye cada celda de la capa recurrente por una celda LSTM, dando lugar a una capa LSTM. Cada celda de este tipo tiene la siguiente estructura:

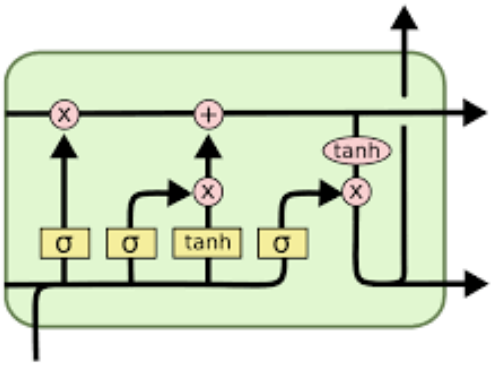


Figura 49. Celda LSTM [57]

El motivo por el que se ha empleado este tipo de capa recurrente frente a otros tipos es precisamente por esta capacidad mencionada de tratar con secuencias largas de datos, lo cual es necesario a la hora de tratar con vídeo. En este proyecto, los vídeos de entrada tienen una longitud de 60 fotogramas, lo cual presentaba problemas a la hora de entrenar con una red recurrente clásica, siendo necesario recurrir a una LSTM para evitar el problema de "exploding and vanishing gradients", que deteriora en gran medida el aprendizaje.

- **Dense + Softmax:** tras extraer la información temporal de los fotogramas con la LSTM y adaptar la salida de dicha capa con la capa flatten, se emplea una capa densa y una softmax para llevar a cabo la clasificación. Primero, los datos pasan por la capa densa, donde se aplica una no linealidad a los mismos mediante el uso de una función ReLu. Posteriormente, la salida de esta capa se conecta con la entrada de otra capa densa con una función Softmax, que será la encargada de proporcionar un vector de dos posiciones prediciendo la clase a la que pertenece el vídeo. Este vector contiene dos posiciones debido a que en este proyecto se está clasificando entre dos clases, pero en un caso genérico tendría como salida un vector de tantas posiciones como clases se quieran clasificar.

Esta es la estructura diseñada en este proyecto para afrontar el problema de clasificación de acciones en vídeo. Además de comprender esta estructura, cabe conocer cómo varía el tamaño de las características a lo largo de la red. Esto queda representado en la siguiente imagen:

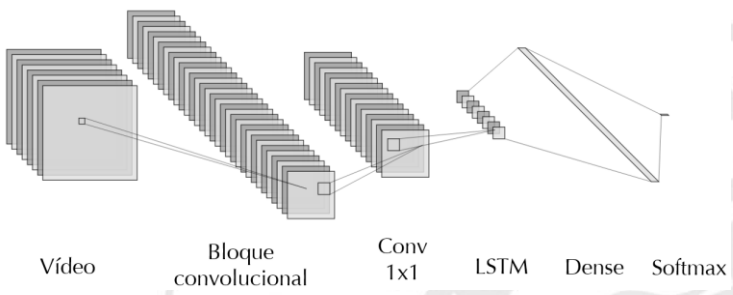


Figura 50. Tamaño de las características

A la izquierda del diagrama queda representado el vídeo de entrada. En este proyecto se ha decidido emplear 60 fotogramas por cada vídeo, manteniendo 10 fotogramas por segundo en cada uno de ellos para que la transición entre ellos sea consistente en el tiempo. Estos vídeos se introducen en el bloque funcional, en el cual experimentan una reducción en su tamaño a lo alto y ancho, mientras que se aumenta en gran medida el número de canales de la característica de salida. Debido a la gran cantidad de canales, se introduce la convolución 1x1, que permite reducir este número de canales para que los datos de entrada a la capa LSTM no sean demasiado grandes, mientras que se mantiene el ancho y alto de las características. Posteriormente, la LSTM proporciona un vector de características por cada uno de los fotogramas, los cuales se aplanan para introducirlos en la capa densa y posteriormente en la densa con Softmax, que da como salida del modelo un vector de dos posiciones con la probabilidad asociada a cada una de las clases a clasificar.

Son numerosas las modificaciones que se han introducido en esta estructura inicial con el objetivo de lograr mejores resulatdos. Algunas de las modificaciones más destacadas son la eliminación de gran parte de las capas Pool, tal y como se ha comentado anteriormente, la eliminación de las capas Dropout debido a su escaso efecto en la red y la implementación de un bloque convolucional que sigue una estructura basada en las redes residuales (ResNet).

Sin embargo, tras las numerosas modificaciones tanto en la estructura del modelo como en sus hiperparámetros y la introducción de múltiples métodos de regularización (los cuales se comentan más adelante), no se logró evitar el overfitting en el mismo sin alcanzar los resultados esperados en validación y test. Esto llevó a la decisión de llevar a cabo transfer learning para reducir este sobreaprendizaje que estaba experimentando el modelo. La manera en la que se ha aplicado transfer learning es introduciendo una red ResNet preentrenada como estimador, es decir, se sustituye el bloque convolucional por una ResNet que no se entrena en el modelo, sino que simplemente extrae las características de las imágenes y se las pasa a la LSTM. Para este modelo se realizaron entrenamientos tanto con una ResNet18 como con una ResNet50.

Este modelo que incluye la ResNet preentrenada sigue la siguiente estructura:



Figura 51. Estructura del modelo con ResNet

La manera en la que se ha integrado la ResNet para poder emplearla en este modelo es mediante la sustitución de las capas densas del final de la misma por el modelo compuesto por la LSTM y las capas de clasificación. Esto permite aprovechar el bloque convolucional de la ResNet (el cual no se vuelve a entrenar) para introducir las características que extraiga este en la LSTM y la posterior capa de clasificación. Cabe mencionar que en este caso no es necesaria la capa Flatten, ya que esta capa está integrada en la ResNet, siendo esta capa la que supone la conexión entre el resto de las capas de dicho modelo y la LSTM.

6.5. BATCH NORMALIZATION

Batch Normalization es una técnica de entrenamiento de Redes Neuronales Profundas que estandariza los datos de cada lote o batch a la hora de introducirlos en una capa del modelo. Esto permite estabilizar el entrenamiento y reducir drásticamente el número de épocas necesarias para entrenar un modelo.

Algo que hace muy complejo el entrenamiento de las redes profundas es el hecho de que constantemente están cambiando los parámetros de cada capa, lo cual hace que los valores de entrada a las capas sucesivas sean muy distintos en cada lote de entrenamiento (ya que tras cada lote se lleva a cabo el proceso de back propagation, en el que se aplica el descenso del gradiente para actualizar los pesos del modelo con el objetivo de minimizar la función de coste). Este cambio en los pesos de cada capa puede hacer que la magnitud de los valores de entrada a la siguiente capa cambie sustancialmente, lo cual puede ralentizar el entrenamiento.

El Batch Normalization pretende evitar este problema mediante la normalización de la salida de cada capa, de manera que los datos de entrada que reciben la siguiente capa estén estandarizados. Esto facilita mucho el proceso de actualización de los pesos de muchas capas simultáneamente,

reduciendo el efecto que tienen los cambios en los parámetros de una capa en el entrenamiento de las capas que la suceden.

La forma en la que funciona Batch Normalization es mediante el cálculo de la media y la deviación típica de los datos de un batch, de forma que todos los datos de cada batch estén normalizados con respecto al batch completo. Esto permite que los datos que recibe cada capa antes de actualizar los pesos tras hacer back propagation sigan una distribución normal.

6.6. OVERFITTING Y CÓMO EVITARLO

Uno de los grandes problemas a la hora de entrenar un modelo de Machine Learning es el denominado sobreaprendizaje u overfitting. El overfitting ocurre cuando el modelo se "aprende" (se ajusta mucho a) los datos del conjunto de entrenamiento, alcanzando una precisión muy alta en la predicción de dicho conjunto, pero no en el conjunto de validación o test. Lo que esto significa, es que el modelo no consigue generalizar adecuadamente, ya que no es capaz de predecir correctamente (o no con la precisión esperada) cuando no ha visto los datos previamente, o lo que es lo mismo, cuando no ha sido entrenado con dichos datos.

El hecho de que un modelo generalice correctamente es crucial a la hora de desarrollar una solución de Machine Learning ya que, de no ser así, el modelo no será capaz de funcionar correctamente en inferencia cuando se saque a producción, ya que no será capaz de predecir correctamente al ver los datos nuevos que se le introducen y con los que no ha sido entrenado en ningún momento.

El overfitting puede ocurrir por varios factores, de entre los que destacan la falta de datos para entrenamiento o una excesiva complejidad del modelo. Este efecto de sobreaprendizaje se puede detectar fácilmente a la hora de analizar las gráficas de entrenamiento, donde se puede apreciar una clara separación entre las precisiones de entrenamiento y validación (en la gráfica de precisión) o un aumento en el error en la curva de validación (en la gráfica de error) tal y como se puede observar en la siguiente imagen:

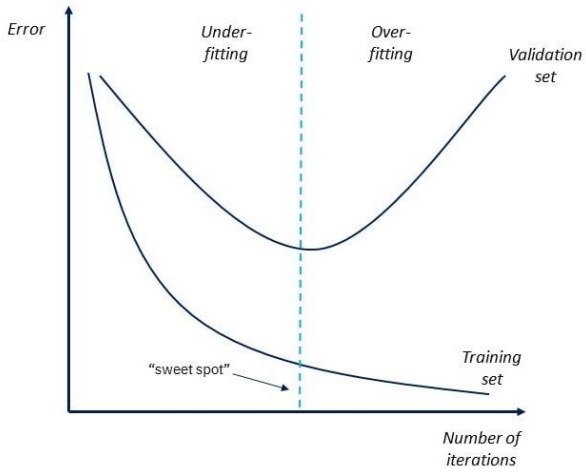


Figura 52. Overfitting [58]

Tal y como se puede observar, a partir de la línea discontinua el error en validación comienza a subir en vez de bajar, lo cual indica que a partir de ese momento del entrenamiento se está produciendo overfitting.

Ahora que se comprende la importancia de evitar el overfitting en los modelos de Machine Learning, cabe mencionar varias formas en las que se puede evitar o reducir este efecto:

- **Early stopping**: esta técnica consiste en detener el entrenamiento justo en el momento previo a que comience a sobreentrenar el modelo. En el caso de la imagen superior, consistiría en detener el entrenamiento en la línea discontinua, de forma que el error en validación y entrenamiento sea mínimo. Para conseguir esto, se deben guardar los pesos del modelo tras cada época, de forma que se pueda recuperar dicho estado del entrenamiento tras evaluar cuál es el punto de corte óptimo.

- **Aumentar el volumen de datos:** una segunda manera de reducir el overfitting del modelo es aumentar la cantidad de datos de la que se dispone para entrenar. Esta técnica funciona debido a que cuanto mayor sea la cantidad de datos, el modelo será capaz de extraer más características que le permitan generalizar, y será más complicado que se memorice los datos de entrenamiento (por el hecho de haber más). Para ello, se aplican las técnicas de Data Augmentation que se comentaron previamente en este documento.

- **Reducir la complejidad del modelo:** otra manera en la que se puede reducir el efecto de sobreaprendizaje en el modelo en algunos casos es mediante la simplificación del modelo. Esto se debe a que cuanto más complejo sea el modelo, más se adaptará a los datos de entrenamiento, dejando menos margen para la generalización debido a que se aprende el ruido presente en los datos (entendiendo por ruido aquellos datos o características que no representan realmente el dataset o el problema que se pretende resolver). De este modo, si se simplifica el modelo se puede perder algo de precisión en el conjunto de entrenamiento, pero se ganará en generalización ante nuevos datos. Esto se puede observar en la siguiente imagen, en la cual se presentan los resultados de predicción de tres modelos ante el conjunto de datos. La complejidad de los modelos aumenta de izquierda a derecha, siendo el modelo de la izquierda el más simple o de orden menor y el de la derecha el más complejo. El modelo de la izquierda no será capaz de extraer las características necesarias de los datos para predecir adecuadamente, obteniendo resultados más cercanos a los que se obtendrían si se intentara predecir de forma aleatoria. Por otro lado, el modelo de la derecha es de gran complejidad, ajustándose perfectamente a los datos de entrenamiento y aprendiéndose así los datos de entrenamiento, pero no siendo capaz de generalizar bien en validación y test. Por último, el modelo del centro es la mejor solución de las tres, ya que presenta buenos resultados en entrenamiento y presentará buenos resultados de generalización, ya que se puede observar que es capaz de comprender la tendencia presente en los datos sin ajustarse a cada uno de ellos individualmente.

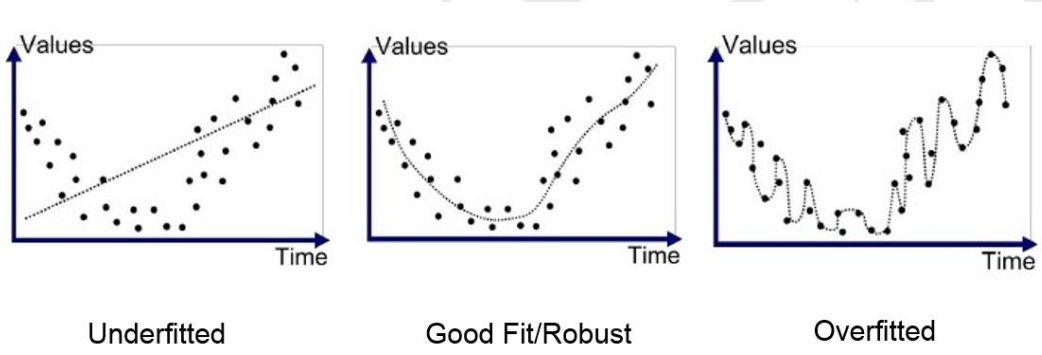


Figura 53. Ajuste a los datos según la complejidad del modelo [59]

Cabe mencionar que en algunos casos esta técnica de reducción de complejidad del modelo para evitar el overfitting no resulta útil, por ejemplo, en el caso en el que ya nos encontremos en un modelo como el del centro de la imagen superior, ya que el hecho de reducir la complejidad en este caso llevaría a perder la capacidad de extraer ciertas características fundamentales de los datos, lo cual provoca un descenso en la precisión del modelo.

- **Transfer Learning:** esta técnica, como se ha comentado anteriormente, permite reducir la cantidad de capas que se deben entrenar del modelo o reducir la cantidad de entrenamiento que requiere este, lo cual permite aprovechar más los datos disponibles para el entrenamiento, reduciendo así el overfitting para un dataset dado.

- **Regularización**: son una serie de métodos matemáticos que permiten reducir el overfitting. La idea en la que se basan es en incluir ciertos parámetros a la función de coste, de forma que se pueda limitar el entrenamiento de la red mediante la variación de los mismos. Hay dos métodos de regularización principales:
    - **Regularización L1**: es el método empleado en la regresión Lasso, y consiste en añadir la siguiente expresión (enmarcada en rojo) a la función de coste:

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n}|\theta_i|$$

Figura 54. Regularización L1 [60]

En esta nueva expresión, lambda es el parámetro de regularización que se varía manualmente para modificar el aprendizaje, y theta es el peso asociado a la capa sobre la que se está haciendo backpropagation. Tal y como se puede observar, el hecho de haber introducido este nuevo sumando a la ecuación hará que el coste inicial sea más elevado. Esta ecuación está diseñada para penalizar el tamaño de los pesos de la red, ya que, en la búsqueda de minimizar la función de coste, se forzará a que estos pesos sean más pequeños (y más pesos tengan valor cero, lo cual se suele decir que es una selección de variables) cuanto más

grande sea lambda. Esto deriva en que la red aprenda solo las características fundamentales, evitando activaciones por características no importantes para la predicción, favoreciendo así la generalización. Este tipo de regresión lleva a la creación de modelos dispersos, es decir, que muchos de sus pesos son cero, mientras que aquellos pesos que no lo son permiten extraer las características fundamentales de los datos. Se debe tener especial cuidado con este tipo de regularización, ya que en muchos casos podría bloquear el aprendizaje del modelo, llevando a underfitting.

- **Regularización L2**: es el método empleado en la regresión Ridge, y consiste en añadir el siguiente sumando a la función de coste:

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

Figura 55. Regularización L2 [60]

En la ecuación superior, lambda es el parámetro que se puede modificar manualmente para seleccionar el grado de regularización a aplicar, y theta es un peso de una capa. La regularización L2 busca el mismo efecto que la L1, que es reducir el tamaño de los pesos de la red mediante la introducción de un sumando en la función de coste con el que se penaliza el tamaño de los pesos de la red mediante el aumento del parámetro lambda. En este caso, la penalización de los pesos se realiza con el valor de los pesos al cuadrado, lo cual lleva a que todos los pesos se reduzcan en tamaño en un mismo orden de magnitud. Este tipo de regularización, y a diferencia del anterior, no deriva en modelos dispersos, ya que en este caso ningún peso tendrá un valor exactamente igual a cero, sino que tenderán a cero, pero sin llegar a dicho valor. Es por ello por lo que se puede decir que es un método de regularización menos agresivo, ya que mantendrá todas las características presentes, mientras que la

regularización L1 hará que los pesos menos significativos sean exactamente cero. Este es el tipo de regularización que se ha aplicado en el modelo.

- **Dropout**: tal y como se ha comentado previamente en este documento, el Dropout consiste en no incluir en el entrenamiento a ciertas neuronas del modelo de forma aleatoria en cada lote de datos. Esto permite que la red no se ajuste perfectamente a los datos, ya que parte de los parámetros de la misma no se actualizarán con los datos de entrada, y por tanto a la red le costará más ajustarse a los datos de entrenamiento. Esto, por tanto, favorece la generalización del modelo. El principal beneficio del Dropout es que evita que se disparen los valores de los parámetros de la red, y que las decisiones dependan de un solo nodo. Esta técnica es muy popular y beneficiosa en los modelos de Redes Neuronales Densas (dense o fully connected), pero sus beneficios no están tan claros en el caso de las Redes Convolucionales. Al no tener tantos parámetros como las capas densas, no requieren de tanta regularización como estas. Además, este método podría no ser efectivo por el hecho de aparecer parámetros correlacionados, por lo que solo se atrasaría el entrenamiento, pero no se prevendría la adaptación del modelo a los datos.

Es por este motivo por el que se decidió prescindir del Dropout en versiones más avanzadas del modelo diseñado en este proyecto, aunque en un primer diseño si que se implementó. Por otro lado, cabe mencionar que debido a la presencia de overfitting en el modelo, se implementaron todas las técnicas mencionadas en este apartado para intentar reducirlo.

Primero, se aumentó el volumen de datos mediante Data Augmentation, introduciendo vídeos en modo espejo y con ruido gaussiano. Posteriormente, se iteró sobre la estructura del modelo para reducir la complejidad del mismo, reduciendo así el número de parámetros totales. Además, se implementó regularización de tipo L2 y Transfer Learning mediante el uso de una ResNet como extractor de características. Por último, se estudiaron las gráficas de entrenamiento para evaluar cuál era el mejor punto de cese del entrenamiento para poder llevar a cabo Early Stopping.

6.7. ACELERACIÓN DEL ENTRENAMIENTO CON GPU

Una GPU (Graphics Processing Unit), al igual que la CPU (Central Processing Unit) es una pieza de hardware cuya funcionalidad es procesar datos. La principal diferencia entre ambas es que mientras que la CPU es una unidad de procesamiento de tipo genérico (vale para procesar cualquier tipo de datos), la GPU es de propósito específico y se especializa en el procesamiento de gráficos (imágenes y vídeos).

Lo que hace que la GPU sea mucho más rápida que la CPU es el número de núcleos de los que dispone. Mientras que las CPUs actuales llegan hasta los 8 núcleos, las GPUs disponen de cientos o miles de núcleos, fomentando el paralelismo. Esta estructura diferente se puede observar a continuación, donde cada cuadrado representa un núcleo:

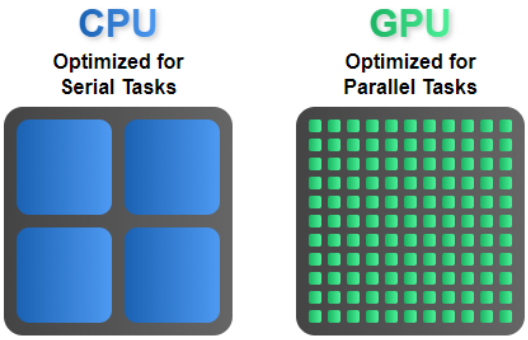


Figura 56. Estructura de una CPU y una GPU [61]

Teniendo en cuenta que cada núcleo procesa las órdenes de forma secuencial (a no ser que se trabaje en varios hilos), la cantidad de tareas que se pueden realizar en paralelo con una GPU supera con creces las que se pueden realizar en paralelo con una CPU debido a esta diferencia en el número de núcleos. En definitiva, mientras que al CPU está diseñada para procesar secuencialmente una gran variedad de procesos, la GPU está diseñada para ejecutar tareas en paralelo con gran velocidad.

Esta diferencia en la velocidad de computación queda representada en la siguiente imagen, donde la curva superior representa los modelos más punteros de GPU y la inferior los de CPU:

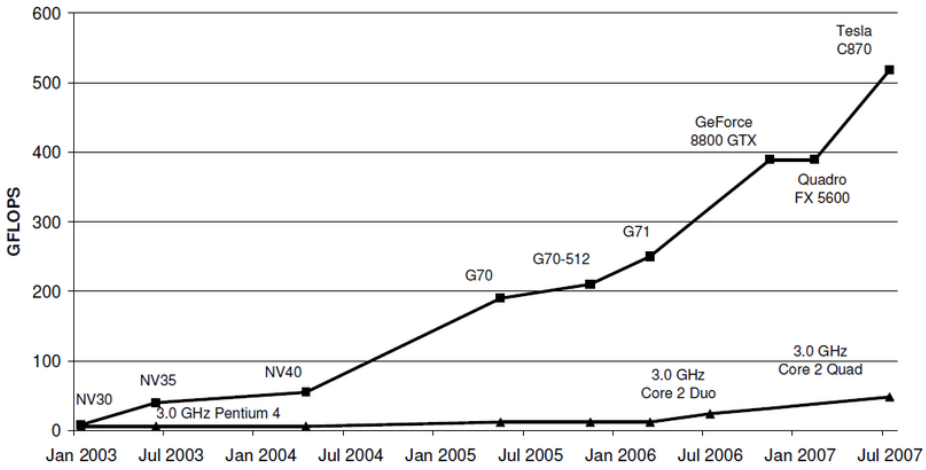


Figura 57. Comparativa de computación de CPU y GPU [62]

Ahora que se comprende la gran capacidad de paralelismo de las GPUs, cabe mencionar que, aunque las GPUs nacieron con la finalidad de ejecutar gráficos de forma rápida, en la actualidad se le dan otros usos varios que requieren de realizar tareas en paralelo, llevando al uso de GPUs para tareas de propósito general. Un ejemplo de esto es el uso de las GPUs para el entrenamiento de modelos de Machine Learning.

Esta técnica de emplear GPUs para el entrenamiento se denomina aceleración por hardware mediante GPU. La aceleración por hardware no consiste en sustituir la CPU por la GPU, sino que consiste en combinarlas para obtener un procesamiento más eficiente, de manera que las tareas que son computacionalmente más intensivas y que permiten paralelización son procesadas por la GPU, mientras que el resto queda en manos de la CPU.

Esto puede proporcionar una gran reducción del tiempo destinado a entrenar un modelo, lo cual es fundamental a la hora de desarrollar modelos de Machine Learning, ya que permite iterar entre diferentes arquitecturas de una forma rápida, pudiendo hacer muchas más pruebas para llegar a una solución óptima. Esta ha sido una de las claves por las que desde 2009, el campo del Deep Learning ha experimentado un "boom", ya que las GPUs han favorecido una mayor agilidad a la hora de diseñar y probar modelos. Cabe además mencionar que el uso de esta técnica de aceleración por hardware presenta resultados especialmente favorables cuando se trata de entrenar modelos de Visión Artificial, ya que estos trabajan con imágenes, que son datos cuyo procesamiento permite una gran

paralelización. A continuación, se presenta una imagen comparativa del tiempo de entrenamiento necesario al usar una CPU o una GPU (ambas del Google Colab) para entrenar un modelo basado en la ResNet50 destinado a la clasificación de imágenes de gatos y perros:

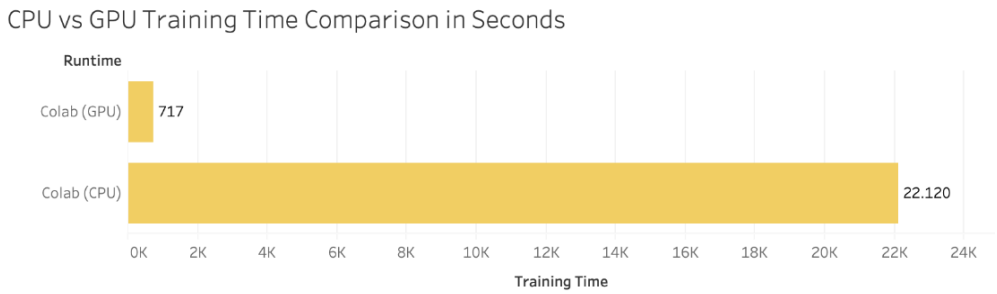


Figura 58. Tiempo de entrenamiento con CPU y GPU [63]

Para poder llevar a cabo procesamiento con la GPU, necesitamos hacer uso de los denominados tensores. Un tensor es una variable similar a un Numpy ndarray pero que puede ser procesado por una GPU. Este tipo de variables están totalmente integrados en todos las librerías de Machine Learning, como son PyTorch y TensorFlow, ya que se considera que son una parte fundamental para el entrenamiento de modelos en la actualidad.

Además de los tensores, cabe hablar de CUDA a la hora de integrar la GPU en las tareas de procesamiento. CUDA es una plataforma desarrollada por NVIDIA que permite de forma sencilla la integración de GPU y CPU para llevar a cabo las tareas de procesamiento. Esta plataforma gestiona la transferencia de datos entre estas dos unidades de forma automatizada, asignando a la GPU aquellas tareas computacionalmente más intensivas y que permiten una mayor paralelización, de forma que el usuario no tiene que hacerse cargo de dicha tarea de gestión del hardware. Esto facilita mucho la tarea de los desarrolladores, quienes pueden hacer uso de sus GPUs de forma sencilla y con unas pocas líneas de código, aprovechando así todos los beneficios de la aceleración por hardware sin complicar el desarrollo de la solución.

7. ANÁLISIS DEL ENTRENAMIENTO

Una de las grandes fases a completar en el proceso iterativo de diseño de un modelo de Machine Learning, es el de análisis del entrenamiento. Esta fase permitirá comprender los cambios que necesita el modelo para alcanzar una precisión mayor, ya que en los resultados de su entrenamiento se podrán observar diferentes tendencias que indiquen cómo se está comportando el mismo.

En este apartado, se desarrollarán diferentes metodologías como la selección y análisis de métricas de evaluación, el estudio de las gráficas de entrenamiento y cómo pueden ayudarnos los mapas de activación a comprender las características que extraen las capas convolucionales.

7.1. MÉTRICAS DE EVALUACIÓN

Las métricas de evaluación son los indicadores con los que podremos saber la calidad de predicción del modelo. Dependiendo del problema que se quiera resolver, se deben emplear unas métricas u otras, ya que cada está enfocada a un análisis distinto.

Por ejemplo, en el caso en el que el problema a resolver sea un problema de regresión, las métricas a emplear serán diferentes que si el problema es de clasificación. Además, según qué métricas se emplee, se podrá enfocar el análisis a destacar unas u otras características de los resultados del modelo.

En cuanto las métricas destinadas a evaluar modelos de clasificación, cabe destacar las que se muestran a continuación. Sin embargo, antes de desarrollar cada una de ellas, se deben comprender bien algunos conceptos en los que se basan estas. Estos son los “true positives”, “true negatives”, “false positives” y “false negatives”:

- **True positives (TP)**: en una clasificación binaria, es la cantidad de veces que el modelo ha etiquetado como positivo la clase positiva (1), es decir, que ha predicho correctamente la clase positiva.

- **True negatives (TP)**: en una clasificación binaria, es la cantidad de veces que el modelo ha etiquetado como negativo la clase negativa (0), es decir, que ha predicho correctamente que el dato no corresponde a la clase positiva.
- **False positives (FP)**: en una clasificación binaria, es la cantidad de veces que el modelo ha predicho que el dato corresponde a la clase positiva cuando en realidad corresponde a la negativa, es decir, ha predicho de forma errónea la clase negativa, etiquetando como valor 1 cuando realmente el valor de la etiqueta es el 0.
- **False negatives (FN)**: en una clasificación binaria, es la cantidad de veces que el modelo predice que el dato corresponde a la clase negativa cuando realmente corresponde a la positiva, es decir, ha predicho de forma errónea la clase positiva, etiquetándola como negativa.

Conociendo estos conceptos, se comprenderán mejor las principales métricas de evaluación empleadas en problemas de clasificación:

- **"Accuracy"**: esta métrica representa la fracción de datos que el modelo ha clasificado correctamente sobre el total de todos los datos predichos. Para ello, se hace uso de los conceptos explicados anteriormente, los cuales permiten calcular la precisión de la siguiente fórmula:

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

Figura 59. Fórmula de Accuracy

Esta métrica es empleada de forma extendida en todos los modelos de clasificación por ser muy intuitiva para controlar la evolución del modelo durante el entrenamiento, ya que se imprime por pantalla tras cada época de entrenamiento para conocer cuál es el estado del entrenamiento. Además, es la métrica que se emplea en las gráficas que se analizan tras el entrenamiento para comprender los resultados obtenidos durante el

entrenamiento. Proporciona un valor entre 0 y 1, y será mejor cuanto más cercano a uno sea. Algo a tener en cuenta a la hora de analizar la precisión, es que se debe conocer cómo están representadas las clases en el dataset. Por ejemplo, si se dispone de un dataset no balanceado en el que el 70% de los datos corresponden a la clase A y el 30% a la clase B, la precisión del modelo en el caso de dar como salida siempre la clase A será del 70%, lo cual puede parecer un buen resultado, mientras que el modelo no sabe predecir.

- **"Precision"**: es una métrica que representa el ratio entre las ocurrencias que se han clasificado correctamente como positivas (clase 1) y todas las ocurrencias que han sido clasificadas como positivas en total:

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

Figura 60. Fórmula de "Precision"

- **Matriz de confusión**: en algunos casos, la precisión no es suficiente para analizar los resultados del modelo, ya que en algunas ocasiones el coste de clasificar de forma errónea una clase no es igual para todas las clases. Por ejemplo, si se están detectando personas que puedan tener una enfermedad contagiosa, supone un error más grave no detectar a una persona que sí que tiene la enfermedad, que etiquetar mal a una persona que no la tiene diciendo que si la tiene. Es por ello que debemos comprender cuántos aciertos y fallos tiene el modelo en cada una de las clases. Esto es lo que se pretende expresar en la matriz de confusión:

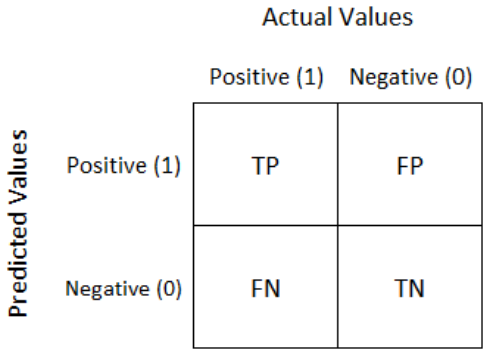


Figura 61. Matriz de confusión [64]

Esta matriz, donde podemos ver los valores true positive (TP), true negative (TN), false positive (FP) y false negative (FN) permite conocer si el modelo tiene algún tipo de tendencia a la hora de decantarse por una clase u otra, y nos permitirá introducir elementos penalizadores en el modelo en el caso en el que clasificar correctamente una clase sea más crítico que otra, pudiendo comprender cuántos errores se cometen por cada clase. En esta matriz, cuanto mayores sean los valores de la diagonal principal y menores los valores del resto de posiciones, mejores resultados estará presentando el modelo.

- **Sensibilidad (recall)**: es una métrica que representa el ratio entre la cantidad de ocurrencias que se han clasificado como pertenecientes a la clase positiva frente a todas las ocurrencias que realmente pertenecían a dicha clase:

$$\text{Sensitivity} = \frac{TP}{(TP + FN)}$$

Figura 62. Sensibilidad

- **Especificidad**: la especificidad sigue la misma idea que la sensibilidad, pero en este caso para la clase negativa (0). La fórmula se muestra a continuación:

$$\text{Specificity} = \frac{TN}{(TN + FP)}$$

Figura 63. Especificidad

- **F1-Score**: esta métrica es especialmente útil cuando se dispone de un dataset no balanceado, ya que permite cuantificar el efecto de los falsos positivos y falsos negativos a la hora de clasificar:

$$\text{F1-score} = \frac{2 \times (\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})}$$

Figura 64. F1-Score

  Esta métrica genera un valor entre 0 y 1, siendo mejor resultado cuanto más cercano a 1. Cuanto más pequeño sea el valor de F1, habrá menos falsos positivos y falsos negativos en la predicción.

De entre estos parámetros, en el proyecto se empleó el "accuracy" durante el entrenamiento para conocer tras cada época la precisión del modelo tanto en el conjunto de entrenamiento como en el de validación, y tras el entrenamiento en test. Además, se analizó la matriz de confusión para conocer las tendencias del modelo a la hora de predecir, y poder así reducir el posible sesgo hacia una de las clases.

7.2. GRÁFICAS DE ENTRENAMIENTO

Las gráficas sobre cómo ha transcurrido el entrenamiento son cruciales para comprender cómo ha evolucionado este durante las diferentes épocas. Estas curvas permiten identificar ciertas tendencias negativas en el modelo, como puede ser el overfitting, el underfitting, la presencia de un

conjunto de datos no representativo… permitiendo obtener una mejor idea sobre cuáles son las modificaciones necesarias en el modelo para mejorar la predicción.

Las dos gráficas que se analizan generalmente son la de precisión y la de la función de pérdidas. Ambas gráficas presentan dos curvas: una que representa los resultados en el conjunto de entrenamiento y otra para los de validación, tal y como se muestra a continuación:

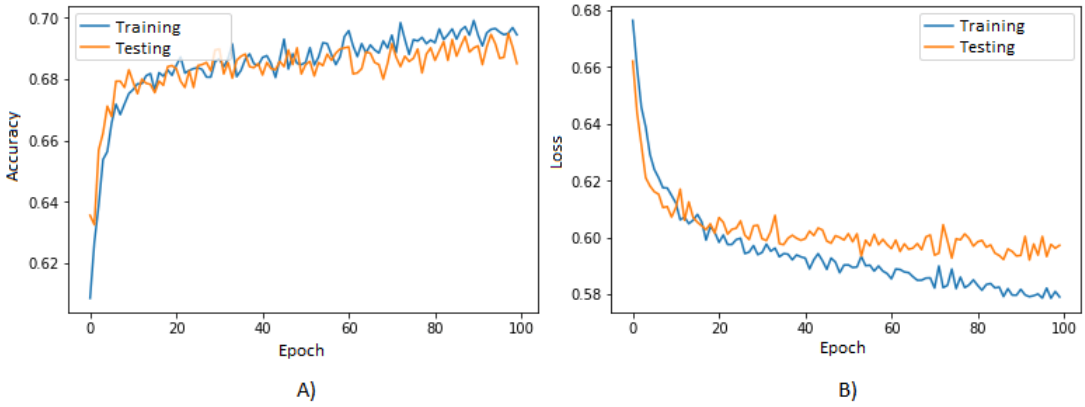


Figura 65. Curvas de precisión y pérdidas [65]

La gráfica de precisión (izquierda) muestra para cada época los resultados de "accuracy" del modelo, es decir, cuántos datos ha clasificado correctamente frente a los totales que se han introducido durante dicha época. Estas curvas tendrán tendencia ascendente, lo cual representa que el modelo va aprendiendo a lo largo del entrenamiento.

Por otro lado, la gráfica de pérdidas ("loss") de la derecha, representa el resultado de la función de pérdidas en las diferentes épocas del entrenamiento. La función de pérdidas es una ecuación que cuantifica la diferencia entre la predicción del modelo y las etiquetas reales de los datos, es decir, cuánto se asemejan las predicciones del modelo a la solución perfecta. Esta función es la empleada a la hora de aplicar backpropagation, donde se calcula el gradiente de cada uno de los pesos de la red con respecto a la función de pérdidas, aplicando el descenso del gradiente con dicho valor. En la gráfica, la función de pérdidas tendrá una tendencia descendente, lo cual indica que el modelo va reduciendo su error a la hora de predecir, o lo que es lo mismo, que su predicción se acerca cada vez más a las etiquetas reales de los datos.

Algo a tener muy presente a la hora de analizar estas gráficas es que mientras que las pérdidas tienen en cuenta la confianza con la que el modelo ha predicho una clase, la precisión solo tiene en cuenta si la clase ha sido predicha correctamente según cierto umbral, es decir, si por ejemplo el modelo predice que un dato corresponde a la clase A con una confianza del 5%, se tomará como un error para la precisión y tendrá cierto valor en la función de pérdidas, y si lo predice con una confianza del 40%, en la precisión se seguirá tomando como un dato mal etiquetado por el modelo (si el umbral de confianza es el 50%) y sin embargo la función de pérdidas tendrá un valor menor que en el caso anterior.

Otra cosa a tener en cuenta en estas gráficas es que el hecho de disponer de dos curvas en cada gráfica (una para el conjunto de entrenamiento y otra para el conjunto de validación) es crucial para poder comprender totalmente la evolución del modelo. La comparativa de una con otra nos permitirá saber no solo si el modelo está aprendiendo a reconocer patrones, sino además a comprobar que el modelo es capaz de generalizar adecuadamente y que su comportamiento en producción será el adecuado.

Tal y como se ha comentado, estas gráficas permiten identificar diferentes situaciones en las que el modelo no esté obteniendo los resultados óptimos. A continuación, se pretenden explicar estas situaciones y cómo se pueden identificar gracias a las curvas de entrenamiento, así como diferentes técnicas para paliar dichos efectos negativos en el entrenamiento:

- **Overfitting**: una de las principales preocupaciones y quebraderos de cabeza de los desarrolladores de modelos de Machine Learning es el overfitting. Estas gráficas permiten detectarlo para poder buscar una solución con las técnicas de prevención que se presentaron anteriormente en este documento. Es por esta capacidad de detectar el overfitting por lo que estas gráficas son ampliamente utilizadas en el desarrollo de modelos. El overfitting se puede observar claramente en la gráfica de precisión, donde se observará un aumento de la precisión del conjunto de entrenamiento si su correspondiente ascenso en el conjunto de validación. También se puede detectar fácilmente con la curva de pérdidas, donde se observará que las pérdidas en validación aumentan o se mantienen constantes mientras que las de entrenamiento siguen en

descenso a partir de cierto punto (tras el cual comienza el overfitting). Esto se puede observar claramente en la siguiente gráfica:

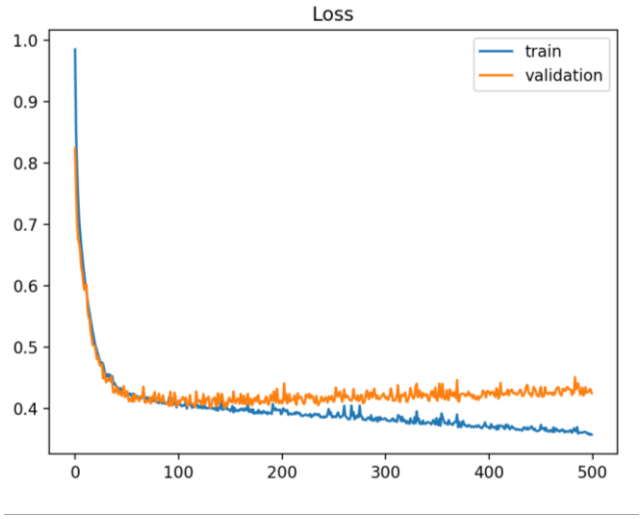


Figura 66. Overfitting [33]

- **Underfitting**: el underfitting ocurre cuando el modelo no es capaz de extraer suficientes características de los datos que le permitan predecir de forma adecuada la salida, tanto en el conjunto de entrenamiento como en el de validación. Dentro del underfitting podemos encontrar dos situaciones diferentes. La primera situación se da cuando el modelo es demasiado simple como para extraer las características necesarias de los datos para predecir mejor que de forma aleatoria o cuando se ha aplicado excesiva regularización:

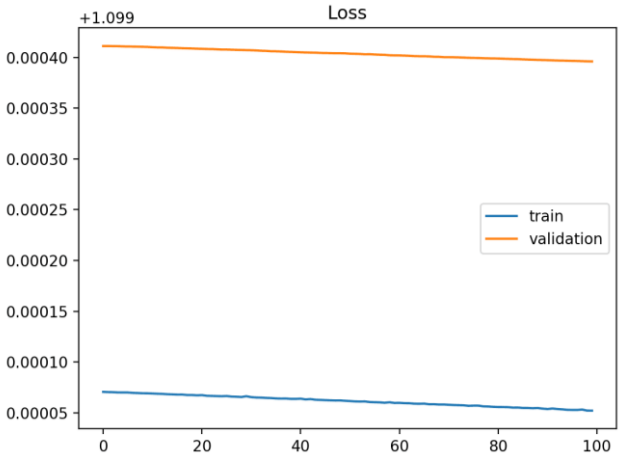


Figura 67. Underfitting [33]

Esto se puede solucionar aumentando la complejidad del modelo, de forma que tenga mayor capacidad de adaptarse a los datos y extraer características de los mismos.

La segunda situación que se puede dar es que el modelo sea capaz de aprender pero que le falte entrenamiento, es decir, que puede alcanzar valores de precisión y pérdidas mejores de los que está alcanzando actualmente. Esto en las gráficas se puede identificar cuando al final del entrenamiento, la función de pérdidas no tiene una tendencia horizontal, sino que sigue en descenso, lo cual indica que el modelo aún puede conseguir mejorar sus predicciones (ya que la función de pérdidas puede ser menor de lo que es actualmente, ya que de lo contrario tendería a ser horizontal, indicando que el modelo no es capaz de extraer más características de los datos que le permitan realizar una mejor predicción.

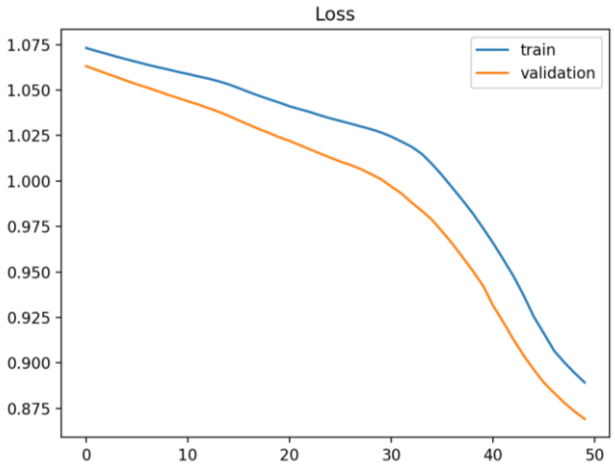


Figura 68. Underfitting [33]

Esto se puede solucionar entrenando el modelo durante más épocas, lo cual permitirá llegar al punto en el que las curvas tienen una tendencia horizontal, indicando que el modelo prácticamente no está aprendiendo más características nuevas.

- **Conjunto de entrenamiento no representativo**: la escasez de datos es uno de los principales problemas que se afrontan a la hora de desarrollar un proyecto de Machine Learning. Los datos son lo que alimenta el modelo, y por tanto una parte fundamental de su entrenamiento. Estas gráficas permiten detectar la escasez de datos, lo cual queda representado en la función de pérdidas como una oscilación en el conjunto de entrenamiento:

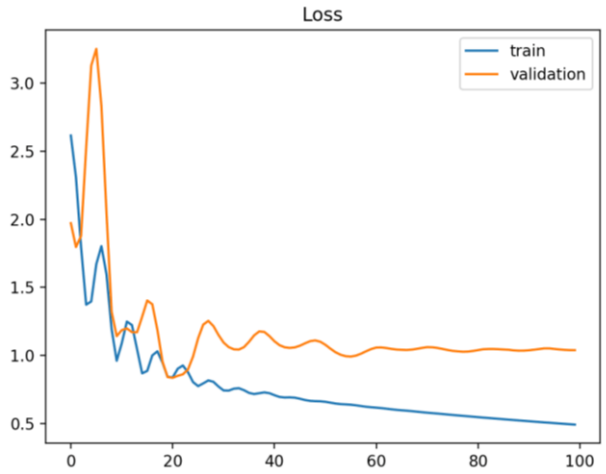


Figura 69. Conjunto de entrenamiento no representativo [33]

La solución a este problema pasa por aumentar el número de datos presente en el dataset, para lo cual se puede llevar a cabo técnicas de data augmentation. Además, se debe tener cuidado con que todas las clases queden representadas, lo cual se puede comprobar analizando la matriz de confusión y viendo la precisión del modelo al predecir cada una de las clases.

- **Conjunto de validación no representativo**: algo que puede ocurrir a la hora de realizar la partición de datos entre entrenamiento y validación, es que no se destinen suficientes datos a este último conjunto. Esto llevaría a no ser capaces de evaluar realmente si el modelo es capaz de generalizar, o lo que es lo mismo, no sería posible comprender la capacidad de predicción del modelo cuando salga a producción. Esto se puede identificar de dos maneras distintas. La primera es que la curva de pérdidas de validación será oscilante respecto de un valor de pérdidas que no aumenta ni decrece mucho, mientras que el de entrenamiento si desciende. Otra forma en la que se puede identificar es por tener un valor de pérdidas menor en validación que entrenamiento. Lo que esto indica es que el conjunto de validación es más fácil de predecir que el de entrenamiento debido a la escasez de datos. Ambos casos quedan representados en las siguientes gráficas de pérdidas:

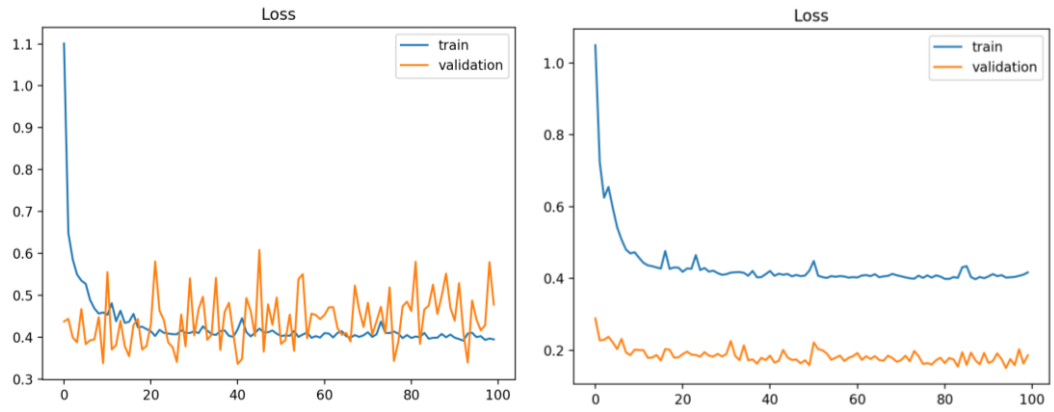


Figura 70. Conjunto de validación no representativo [33]

- **Falta de normalización de los datos**: en los modelos de Deep Learning, es crucial normalizar los datos a la salida de cada capa de la red neuronal, ya que de lo contrario la modificación de los pesos de una capa (producto del aprendizaje de la misma) llevaría a que los datos de entrada de las capas sucesivas fueran muy dispares antes y después de llevar a cabo el backpropagation. La necesidad de normalizar los datos se puede observar en la gráfica de precisión, donde se observará una línea horizontal sin oscilaciones:

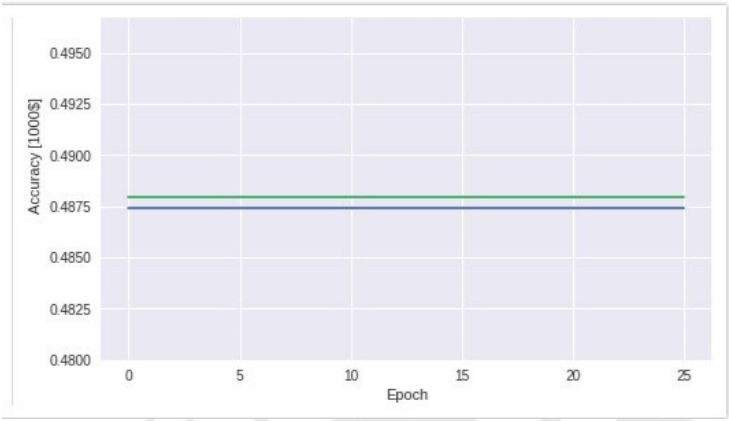


Figura 71. Efecto de normalizar los datos [41]

Esto se puede solucionar normalizando los datos, lo cual lleva a un entrenamiento más estable y se podrá observar que las curvas de entrenamiento y validación comienzan a presentar mejoras con el transcurso del entrenamiento.

7.3. MAPAS DE ACTIVACIÓN

Las capas convolucionales están compuestas por una serie de filtros que se aplican a las imágenes para extraer información de las mismas. Cada uno de los múltiples filtros que contiene cada una de estas capas se especializa en extraer una característica diferente de las imágenes, y es la combinación de todos ellos la que permite que una red neuronal convolucional pueda comprender el contenido incluido en una fotografía.

Un gran problema a la hora de entrenar un modelo de Machine Learning es que este se puede convertir en una caja negra, dentro de la cual entendemos lo que está pasando, pero no sabemos en qué basa sus decisiones ni qué características está extrayendo de los datos. A la hora de desarrollar un modelo de visión artificial exitoso, se debe comprender en qué se está fijando el modelo en cada imagen para poder clasificarla. Esto se puede conseguir con los mapas de activación.

Los mapas de activación ("activation maps" o "feature maps") son representaciones de las salidas de las capas convolucionales. Permiten comprender en qué están basando el modelo sus predicciones, o lo que es lo mismo, qué características están extrayendo de los datos este tipo de capas. A continuación se presenta un ejemplo de estos mapas de activación (izquierda) junto con las imágenes a las que corresponden (derecha).

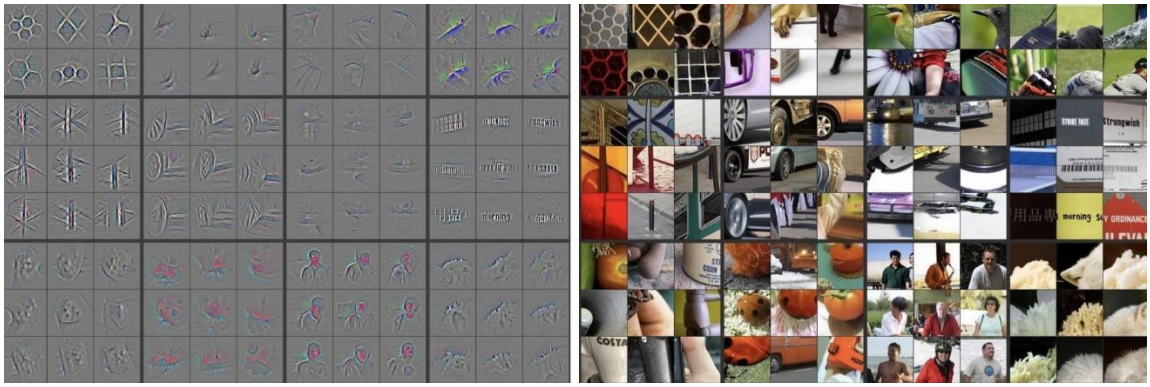


Figura 72. Mapas de activación [66]

Estos mapas de activación serán más abstractos en las primeras capas de la red neuronal, ya que cada filtro se especializa en extraer características simples de la misma, como puede ser en detectar bordes horizontales, bordes verticales o en detectar un color en concreto. Sin embargo, según se va avanzando en la red, estos mapas se vuelven más complejos, ya que se va concatenando la información extraída por los filtros de las capas anteriores para poder extraer características más complejas. Es por ello que mientras que al comienzo de la red solo veremos mapas de activación abstractos, en las capas del final de la red podremos identificar figuras geométricas e incluso objetos, como es el caso de la imagen superior.

Estos mapas de activación son fundamentales para poder asegurar que el modelo está comprendiendo los datos y que ha extraído características relevantes y que no llevarán a un sesgo cuando el modelo trabaje en inferencia. Por ejemplo, si en un modelo se busca detectar si hay personas en una imagen y en los mapas de activación se aprecia que para ello se fija en si hay ojos en la imagen, nos permite comprender que en el caso en el que el modelo no detecte los ojos de una persona no será capaz de detectarla. Esto llevaría a la necesidad de volver a entrenar el modelo con imágenes de personas de espaldas donde no se les vean los ojos para que el modelo aprenda a extraer otras características distintas y no "se fije en los ojos".

Cada vez se está viendo más la importancia de ser capaz de explicar en qué basan sus decisiones los modelos, ya que esto es crucial para desarrollar una Inteligencia Artificial ética y responsable. Los mapas de activación suponen un gran avance en este sentido a la hora de desarrollar

modelos de Visión Artificial, permitiendo al desarrollador comprender las decisiones que toma el modelo y poder reentrenarlo si las características extraídas llevaran a posibles sesgos.

8. OPTIMIZACIÓN DEL ENTRENAMIENTO

Una vez que se ha llevado un análisis del entrenamiento, se deben tomar decisiones sobre cómo modificarlo para que el siguiente entrenamiento presente mejores resultados que los del entrenamiento que se acaba de completar. Este paso del proceso iterativo del entrenamiento de un modelo de Machine Learning permite aumentar la precisión o reducir el tiempo de ejecución de un modelo mediante la modificación de ciertas partes del código o de los hiperparámetros o estructura de la red neuronal empleada.

En este apartado se tratarán dos partes diferentes: una primera en la que se explicarán los parámetros que se pueden modificar para que la precisión del modelo aumente, y una segunda en la que se explicará una técnica que permitirá optimizar los tiempos de ejecución y los requerimientos de computación a la hora de entrenar el modelo, denominada Mixed Precission.

8.1. OPTIMIZACIÓN DE HIPERPARÁMETROS

Los hiperparámetros son aquellas variables que determinan tanto la estructura de la red (como por ejemplo la función de activación empleada o el tamaño y número de filtros en una capa convolucional) como las variables que determinan la manera en la que entrenará el modelo (como por ejemplo el learning rate). Estas variables se modifican tras cada entrenamiento para alcanzar resultados de precisión mejores que en la iteración anterior sobre el modelo.

La búsqueda de los hiperparámetros adecuados es una tarea iterativa y que consume mucho tiempo a la hora de desarrollar un proyecto de Machine Learning. Encontrar la combinación adecuada de hiperparámetros es crucial para una buena

Además del número de capas y del tamaño y número de filtros o neuronas de cada una de ellas (lo cual no se desarrollará en este apartado), cabe destacar los siguientes hiperparámetros:

- **Learning Rate**: es sin duda uno de los hiperparámetros más importantes. El Learning Rate define la velocidad a la que se actualizarán los pesos de la red, es decir, la velocidad a la que aprenderá el modelo. Siempre es un valor positivo, y suele tener un valor pequeño, normalmente menor que 1. Este parámetro multiplica en la ecuación de cálculo de los nuevos pesos tras realizar el descenso del gradiente, por lo que un valor mayor de Learning Rate llevará a que los pesos se actualicen de manera más rápida cada vez que se lleve a cabo backpropagation. Tanto un Learning Rate pequeño como uno grande son perjudiciales para el entrenamiento, tal y como se puede observar en la siguiente imagen, donde se representa la búsqueda del punto óptimo y las diferentes actualizaciones mediante el descenso del gradiente:



Figura 73. Learning Rate

Tal y como se puede observar, un Learning Rate demasiado pequeño puede alargar mucho el proceso de búsqueda del punto óptimo de los pesos, ya que el valor de cada peso de la red cambiará poco respecto del batch anterior. Por otro lado, si el valor de este hiperparámetro es demasiado elevado, podría impedirse la convergencia del modelo, impidiendo al modelo llegar a la solución óptima.

Una técnica empleada a la hora de asignar un valor a este hiperparámetro consiste en asignarle un valor algo elevado al principio e ir reduciéndolo según avanza el entrenamiento. A esto se le denomina Learning Rate Adaptativo. Lo que permite esta técnica de variación del hiperparámetro a lo largo del entrenamiento es fomentar el cambio rápido de los pesos de la red durante las primeras épocas para "explorar" el espacio y poder acercarse a la región óptima para, posteriormente, con el Learning

Rate más pequeño buscar el punto óptimo dentro de la región candidata a tener dicho punto, mediante cambios más suaves en los pesos.

- **Número de épocas**: se denomina época al proceso de pasar todos los datos del conjunto hacia delante y hacia atrás (backpropagation) por la red. Por tanto, el número de épocas define cuántas veces se empleará todo el conjunto de datos para entrenar el modelo. Aunque este hiperparámetro pueda parecer trivial y de poca importancia, la longitud del entrenamiento quedará definido por este número de épocas, ya que el entrenamiento del modelo no se dará por completado hasta que se hayan completado todas las épocas especificadas. El hecho de seleccionar el número de épocas adecuado implica detener el entrenamiento en un momento en el que el modelo haya aprendido a extraer las características necesarias obteniendo buenos resultados tanto en entrenamiento como en validación, pasando el punto de underfitting (momentos en los que el modelo no es capaz de predecir adecuadamente) pero sin llegar al punto de overfitting (momento a partir del cual el modelo se ajusta demasiado a los datos y pierde capacidad de generalización. Una técnica empleada para que no sea necesario calcular el número exacto de épocas que se necesitarán en el entrenamiento es hacer uso del denominado Early Stopping, el cual se ha desarrollado anteriormente en este documento. Hacer uso de esta técnica permitiría poder asignar un número de épocas mayor al necesario para entrenar, de forma que entremos claramente en la región de overfitting (pero sin alargar excesivamente el entrenamiento) y poder recuperar el estado del modelo en la época en la que aún no ha comenzado el overfitting. De este modo, evitaremos el caso en el que el modelo no llegue a aprender, pudiendo asignar un número de épocas mayor al necesario y escoger a posteriori el número correcto de épocas para el entrenamiento, guardando los pesos de la red de dicho momento.

- **Tamaño de lote ("batch size")**: a la hora de entrenar una red neuronal, se suelen dividir los datos en lotes, es decir, en pequeños grupos de datos, de modo que se introduce cada uno de estos lotes en la red para el entrenamiento, y tras pasar todos los datos correspondientes a un lote, se lleva a cabo el proceso de backpropagation. Por tanto, el número de datos que conformen cada lote influirá en el comportamiento de la

red, ya que indicará cada cuántos datos se actualizan los pesos de la misma, y cuantos datos se tienen en cuenta a la hora de llevar a cabo dicha actualización. El hecho de emplear un tamaño de lote mayor, favorecerá la capacidad del modelo de generalizar, ya que los pesos de la red se actualizarán teniendo en cuenta un mayor número de datos. Por otro lado, el hecho de tener un valor de tamaño de lote pequeño hará que los pesos se actualicen más veces, haciendo que oscilen más sus valores. Existe la opción de usar un único batch, introduciendo todo el conjunto de datos previamente a realizar backpropagation, lo cual lleva al denominado "Batch Gradient Descend". Por otro lado, se puede seleccionar un tamaño de lote igual a 1, lo cual lleva al denominado "Stochastic Gradient Descent". Por último, si el tamaño de lote se encuentra entre 1 y el total de los datos, se da lugar al llamado "Minibatch Gradient Descent"

- **Función de activación**: la función de activación es una función que se aplica a la salida de una neurona en una red neuronal, y que permite aplicar una no-linealidad al cálculo realizado en la neurona. El hecho de aplicar una función de activación que introduzca esta no-linealidad es crucial para el desarrollo de redes neuronales, ya que de lo contrario la red colapsaría en una regresión lineal. Existen diferentes funciones de activación, siendo cada una de ellas más apropiada para cierto tipo de soluciones, tal y como se comentará a continuación:
    - **Sigmoide**: la función sigmoide se presenta a continuación:

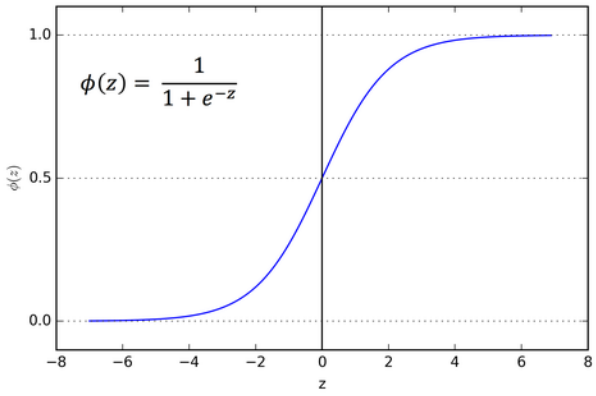


Figura 74. Función Sigmoide [67]

Tal y como se puede observar, esta función va de 0 a 1, y es por ello por lo que es ideal para aquellos casos en los que se busque que la salida sea una probabilidad. Algo a tener en cuenta, es que esta función presentará una ralentización del entrenamiento en sus extremos, ya que la derivada (el gradiente) es cercana a cero, por lo que los pesos no experimentarán a penas actualización a la hora de hacer backpropagation en caso de caer en esas partes de la curva, mientras que esto no ocurrirá en la parte central de la curva (valores de x entre -2 y 2). Este problema es el denominado "vanishing gradients". Algo a destacar de esta función es que, al estar acotada entre 0 y 1, las activaciones no explotarán (no podrán tener un valor infinito o excesivamente grande), sino que estarán acotadas entre 0 y 1. Cabe nombrar también la función **Softmax**, la cual permite llevar a cabo una clasificación multivariable, ya que permite dar como salida varias probabilidades (una por cada clase) cuya suma es igual a 1.

- **Tangente Hiperbólica (tanh):** esta función de activación es similar a la anterior, pero en este caso presenta valores entre -1 y 1:

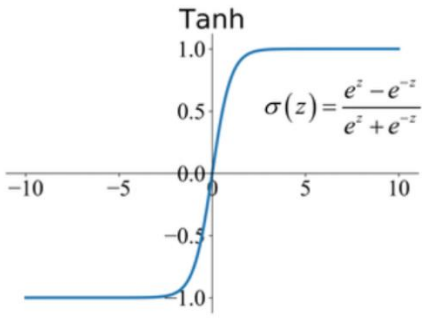


Figura 75. Función Tangente Hiperbólica [68]

Esta función presenta los mismos beneficios que la función anterior; el entrenamiento se ralentizará en esta función en el caso de que los valores de la misma se encuentren en sus extremos, y sus valores están acotados también, por lo que las activaciones no explotarán, quedando acotadas entre -1 y 1. Algo

que la diferencia de la Sigmoide, es que presenta un gradiente mayor en la parte central (valores de x entre -2 y 2), lo cual lleva a un mayor aprendizaje de la red cuando se emplea esta función frente al uso de la función anterior.

- **ReLU ("Rectified Linear Unit"):** esta función es sustancialmente diferente a las dos anteriores, y fue diseñada para paliar los problemas de aprendizaje que presentan estas ("vanishing gradients"). Esta función se presenta a continuación:

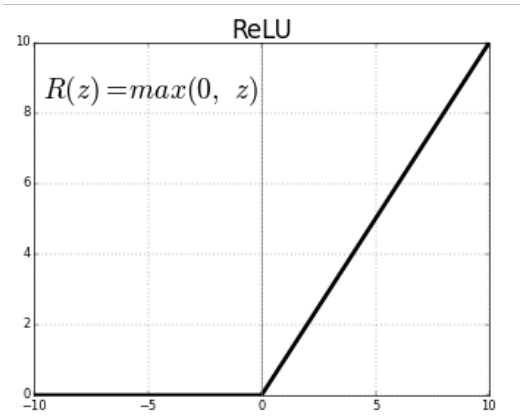


Figura 76. Función ReLU [67]

Tal y como se puede observar, esta función devolverá cero para todos los valores de entrada menores de cero, y devolverá como valor de salida el mismo que la entrada para aquellos valores mayores de 0. Esto presenta una gran ventaja en el aprendizaje frente a las activaciones anteriores para el caso de los valores mayores que 0, pero no presenta aprendizaje alguno para los valores negativos. Algo muy beneficioso que presenta esta función frente a las anteriores es la facilidad de calcular tanto su valor como su gradiente, lo cual reduce mucho la computación a la hora de llevar a cabo tanto el cálculo de la salida como el backpropagation.

- **Leaky ReLU**: esta función nació para resolver este problema que presenta la ReLU de no aprendizaje si los valores de entrada son negativos. Presenta una

versión modificada de la ReLU, en la que la función tiene cierta pendiente positiva para los valores menores de cero:

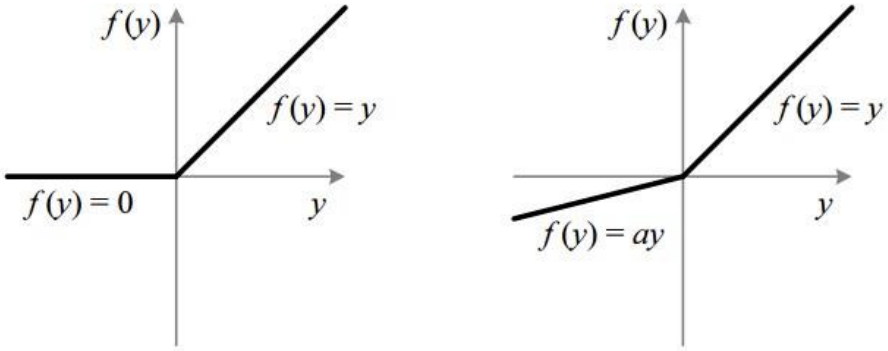


Figura 77. ReLU vs Leaky ReLU [67]

Tal y como se puede observar en la ecuación, hay un parámetro configurable a. este parámetro suele tener un valor cercano a 0.01. la modificación de este parámetro permite establecer el ratio de aprendizaje de la capa en los caso en los que el resultado de la función sea negativo, de manera que un valor más grande de a dará lugar a mayor modificación de los pesos, mientras que un valor más pequeño de a dará lugar a una menor modificación, y por tanto un aprendizaje más lento.

Ahora que se han comentado los principales hiperparámetros, cabe desarrollar las diferentes técnicas que se pueden emplear para encontrar cuál es el valor óptimo de cada uno de estos hiperparámetros (excepto la función de activación, que debe de ser seleccionada de forma manual):

- **Manual Search**: este método de búsqueda de los hiperparámetros óptimos es el más sencillo, pero a la vez el que llevará más tiempo. Consiste en realizar una búsqueda manual de los hiperparámetros, cambiándolos tras cada entrenamiento según se considere oportuno.

- **Grid Search**: es una técnica fácil de implementar, que consiste en programar el conjunto de valores que se probarán para cada hiperparámetro. Una vez programados, se realizará un entrenamiento para cada una de las combinaciones posibles, y tras los entrenamientos se debe llevar a cabo un análisis para comprender qué combinación es la que presenta mejores resultados (que será la que lleve a que el modelo genere mejores predicciones). En definitiva, es una técnica basada en la fuerza bruta, y se podría entender como una manera de automatizar la búsqueda manual. Debido a los múltiples entrenamientos que se deben llevar a cabo, es un método muy poco eficiente, que requiere de mucho tiempo de computación.

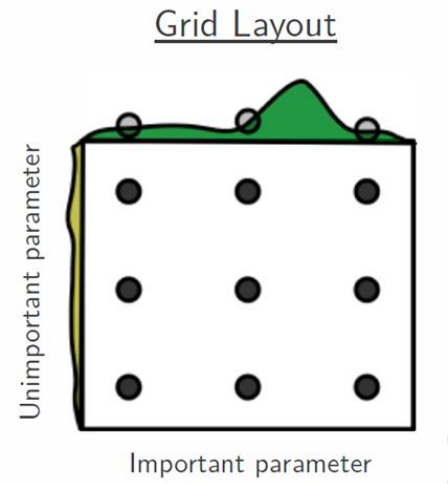


Figura 78. Grid Search [69]

- **Random Search**: esta técnica es similar a la anterior, pero en este caso, y tal y como indica su nombre, la selección de los valores de los hiperparámetros no se escogerá y programará, sino que será elegida de forma aleatoria. Este método presenta mejores resultados que el Grid Serch, pero también consume muchos recursos de computación para poder llevar a cabo todos los entrenamientos necesarios para probar las diferentes combinaciones de hiperparámetros.

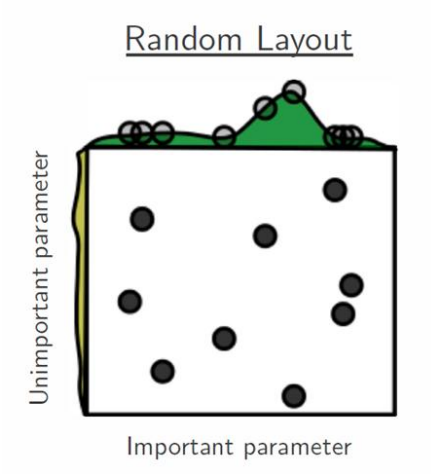


Figura 79. Random Search [69]

- **Bayesian Optimization**: en este caso, y a diferencia de los dos anteriores, se hace uso de un algoritmo de optimización para encontrar la mejor combinación de hiperparámetros. El algoritmo empleado en este método es un algoritmo de optimización Bayesiano. Las dos grandes elecciones que se deben llevar a cabo a la hora de definir este método son la selección de la función a priori que expresa la creencia sobre cómo es la función a optimizar y la selección de la denominada función de adquisición, la cual es la encargada de definir cómo se debe explorar el espacio de los hiperparámetros durante la optimización Bayesiana. Esta técnica presenta mejores resultados y requerirá de un menor número de entrenamientos que las dos anteriores para encontrar una combinación óptima de hiperparámetros, pero es la más compleja de implementar.

8.2. MIXED PRECISSION

A la hora de entrenar empleando aceleración por hardware mediante el uso de una GPU, además de disfrutar de los beneficios en tiempos de ejecución que esta técnica aporta, permite implementar otras metodologías como es el Mixed Precission, que permiten acelerar aún más el entrenamiento y reducir la carga computacional, proporcionando hasta 3 veces mayor velocidad de entrenamiento en las GPUs modernas [70].

El Mixed Precission es una técnica que permite llevar a cabo el entrenamiento de una red neuronal mediante GPU empleando tensores tanto de 32 bits como de 16 bits de tipo float. Es decir, permite usar dos longitudes de dato diferente en los pesos de la red neuronal según sea conveniente en cada caso.

A la hora de entrenar una red neuronal, el tipo de dato que se suele emplear es float32. Esto se debe a que algunos parámetros de la red pueden llegar a ser muy pequeños, por lo que si se limitara a emplear float16 se podrían perder gran capacidad de entrenamiento, o incluso no alcanzar una solución debido a una no convergencia. Esto se debe a que, aquellos pesos que tienen una magnitud menor de 2^{-24} ($\approx 5{,}96*10^{-8}$) se convierten en cero en float16 [71], ya que este es el mínimo valor que se puede alcanzar con este tipo de dato. Esto llevaría a que queden inutilizados, pudiendo llevar a que el modelo no converja.

Sin embargo, el hecho de emplear float32 como tipo de dato para los pesos de la red frente a float16 hace que los tiempos de ejecución del entrenamiento sean mayores, y que los requerimientos de memoria también lo sean.

Es por ello que el Mixed Precission es una técnica muy interesante, ya que permite optimizar la computación mientras se mantiene la estabilidad y convergencia del modelo. La forma en la que funciona esta técnica es evaluando según la dimensión de cada peso cuál es el tipo de dato que más conviene usar. En aquellos casos en los que el peso sea de pequeña magnitud y que por tanto se pierda precisión al usar tan solo float16 (o incluso que float16 no tenga suficiente alcance como para representar dicho peso sin que se tome como si fuera un cero) se empleará float32. Por otro lado, en aquellos casos en los que el parámetro quede correctamente representado con float16, se empleará este tipo de dato para reducir el tiempo de computación y el espacio en memoria que se requiere, acelerando así el entrenamiento.

En este proyecto, se ha implementado esta técnica, lo cual ha permitido reducir considerablemente los tiempos de entrenamiento. Cabe mencionar, que NVIDIA ha desarrollado un módulo que permite implementar Mixed Precission con unas pocas líneas de código, ofreciendo este framework tanto en TensorFlow como en PyTorch. Cabe destacar, que uno de los objetivos de NVIDIA a la hora de desarrollar este módulo es que el cambio en el código fuera mínimo, de forma

que se integra perfectamente con el código previo a su implementación, haciendo que los desarrolladores no tengan que realizar grandes cambios en el programa.

La manera en la que se implementa Mixed Precission en PyTorch consiste en implementar las siguientes líneas de código, que es muy similar al empleado en un entrenamiento típico implementado en PyTorch como se comentará a continuación:

```
scaler = GradScaler()

with autocast():
    output = model(input)
    loss = loss_fn(output, target)

scaler.scale(loss).backward()

scaler.step(optimizer)

scaler.update()
```

Figura 80. Código de Mixed Precission [72]

La única diferencia entre este código y la manera en la que se programaría esta parte de un modelo en PyTorch es que hay que declarar el scaler y que tanto el output como las pérdidas se calculan haciendo uso de la función autocast, pero los pasos seguidos en el código son idénticos, de forma que solo cambia la sintaxis empleada.

CAPÍTULO V: RESULTADOS

Los resultados obtenidos en este proyecto no se limitan a los resultados del modelo, ya que para obtenerlos, se ha desarrollado un pipeline completo de Visión Artificial. Es por ello que en este apartado cabe hacer mención también a los desarrollos que se han completado en cada una de las partes del pipeline, los cuales han permitido llevar a la implementación de un modelo con capacidad de extraer características de vídeos.

En primer lugar, se ha generado una base de datos con vídeos encontrados en internet. Para ello, no solo se ha realizado una búsqueda exhaustiva que ha permitido recolectar 210 vídeos de un tipo de acción con escasos datos en internet, sino que, para agilizar la descarga, se ha implementado un código que permite descargar tanto vídeos individuales como listas de YouTube de forma automática.

Una vez que se recolectaron todos los datos, se implementó un código que permite dividir los vídeos para extraer los fotogramas en los que se desarrolla la acción de interés, y que etiqueta el vídeo según si aparece dicha acción o no.

A continuación, teniendo los vídeos clasificados, se implementaron con éxito dos técnicas de Data Augmentation (modo espejo y ruido gaussiano), que permitieron aumentar el tamaño del dataset en un 300%, hasta tener un total de 630 vídeos. Además, se implementaron funciones que permitieron formatear cada uno de los vídeos para conseguir unas dimensiones iguales en cada uno de ellos, de forma que quedaran totalmente preparados para ser procesados por la red.

En lo que al modelo se refiere, se desarrollaron tres modelos diferentes. El primer modelo se desarrolló haciendo uso de la librería Keras. Este modelo se terminó por descartar, y se realizó una migración a PyTorch, donde se desarrolló un modelo similar pero más complejo. Este modelo de Conv2D + LSTM presenta los siguientes resultados:

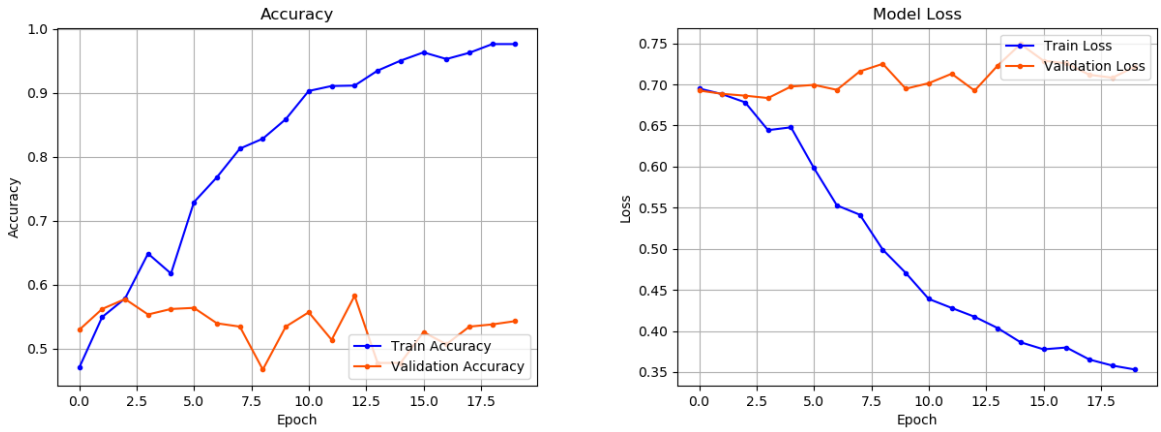


Figura 81. Resultados del modelo Conv2D + LSTM

Tal y como se puede observar, hay un gran overfitting en el modelo a pesar de las diversas técnicas que se han empleado para evitarlo. En la búsqueda de reducción de este overfitting, se ha introducido Dropout, implementado Regularización L2, aumentar el volumen de datos (lo que llevó a implementar las técnicas de Data Augmentation mencionadas anteriormente) y reducción de la complejidad del modelo. Dado que la implementación de estas técnicas no favoreció la capacidad de generalización, se decidió implementar Transfer Learning para intentar reducir este overfitting.

Esta decisión de hacer uso de Transfer Learning es lo que llevó al desarrollo del tercer y último modelo que se ha implementado en este proyecto. Este modelo es un ResNet18 + LSTM. De esta manera, se sustituyó el bloque convolucional por una ResNet18 preentrenada, lo cual permitió reducir sustancialmente el número de parámetros de la red que se debían entrenar, ya que la ResNet18 se emplea como un estimador, de forma que extrae las características de los fotogramas, pero no se entrena con dichos datos. Los resultados se pueden observar a continuación:

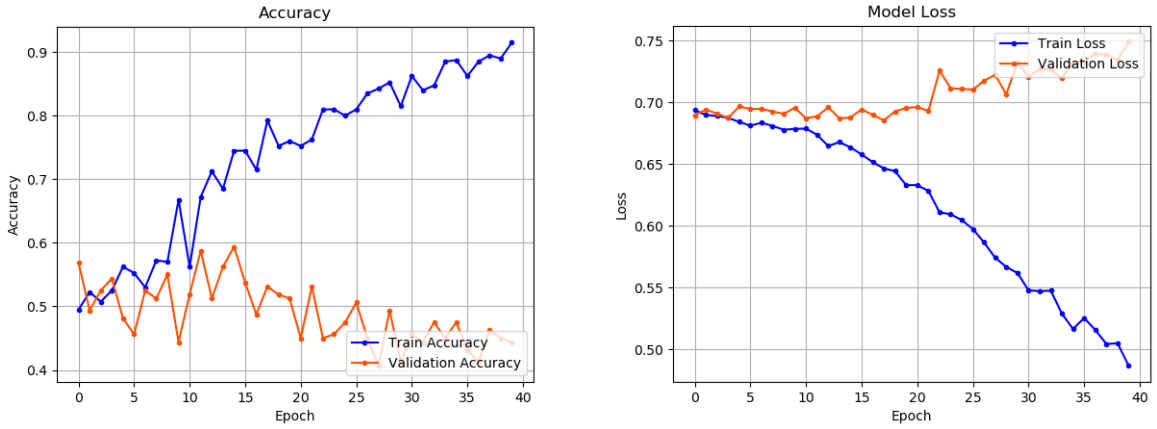


Figura 82. Resultados del modelo ResNet18 + LSTM

Tal y como se puede observar, a pesar de que el comienzo del entrenamiento presenta mayores valores de precisión en validación, no se ha logrado evitar el overfitting que también presentaba el modelo anterior. En este modelo se han aplicado las mismas técnicas que en el anterior para intentar evitar este sobreaprendizaje, pero a pesar de su implementación no se ha conseguido resolver el problema.

A pesar de este problema en los resultados, cabe destacar que el hecho de que el modelo haga overfitting, indica que es un modelo funcional, lo cual supone un punto de partida para continuar con el desarrollo en la búsqueda de mejores resultados de predicción. Esto además muestra que los pasos previos desarrollados en el pipeline funcionan correctamente, y permiten que el modelo aprenda a extraer características de los datos.

Cabe mencionar, por último, que además de todo esto mencionado, en este proyecto se ha desarrollado con éxito técnicas de optimización del entrenamiento como la aceleración por hardware, que ha permitido reducir en gran medida los tiempos de entrenamiento mediante el uso de GPU y Mixed Precission, que ha permitido reducir los tiempos de entrenamiento y los requerimientos de computación.

CAPÍTULO VI: CONCLUSIONES

Son millones las cámaras de videovigilancia presentes en el mundo para garantizar la seguridad tanto en las calles como en interiores. El análisis de todos estos vídeos es una tarea que requiere de la presencia de una persona controlando varias de estas cámaras simultáneamente, lo cual es una tarea ardua y sujeta a posibles errores. Es por ello por lo que cada vez hay un mayor interés en automatizar esta labor, empleando tecnología que apoye a la persona encargada de tal actividad.

Con el objetivo de resolver parte del problema, en este proyecto se desarrolla un modelo de Visión Artificial que pretende detectar una acción en concreto en cámaras de videovigilancia. Para ello, se han estudiado diferentes arquitecturas y se ha optado por implementar un modelo de Conv2D+LSTM.

En este proyecto no solo se ha implementado el modelo, sino que se ha desarrollado de forma completa un pipeline de Visión Artificial, llevando a cabo desde la recolección de datos para la formación del dataset hasta la optimización del modelo, pasando por el procesado de los vídeos, el diseño de la arquitectura del modelo, análisis del entrenamiento…

A la hora de analizar los resultados del modelo, se puede observar que es capaz de extraer características, lo cual indica que el pipeline desarrollado funciona adecuadamente. A pesar de ello, no se ha conseguido evitar el overfitting en el modelo pese a las numerosas técnicas aplicadas para intentar solventar este problema tan común en los modelos de Machine Learning. Por otro lado, estos resultados llevan a pensar que pueda haber un error en los datos (bien en la calidad, la cantidad o en la manera de clasificarlos) o que el modelo diseñado no es el adecuado para este tipo de problema, no siendo capaz de extraer las características necesarias a pesar de las múltiples modificaciones que se han realizado en sus hiperparámetros. Esta segunda idea es la que lleva a que en los futuros desarrollos de este proyecto se planteen diferentes estructuras de red para poder afrontar el problema con un enfoque distinto al actual.

Una de las grandes problemáticas y retos de este proyecto ha sido la creación del dataset. Esto se debe a la escasez de datos presentes en internet y a la imposibilidad de generar dichos datos. Por otro lado, la calidad de los vídeos disponibles en internet es muy baja, no superando en muchos casos una calidad de 240 píxeles. Esto ha podido limitar en gran medida la capacidad de aprendizaje del

modelo, ya que, si los datos con los que se entrena no son de calidad, tampoco pueden serlo sus predicciones.

El desarrollo de algoritmos de Machine Learning es complejo y largo, y no siempre resulta exitoso, pero siempre permite aprender cuáles pueden ser los siguientes pasos a seguir para lograr una implementación exitosa.

CAPÍTULO VII: FUTUROS DESARROLLOS

En cuanto a los futuros desarrollos para este proyecto, abarcan desde el dataset hasta la optimización del modelo, pasando por la estructura del mismo.

Con respecto al dataset, es necesario aumentar la cantidad de datos presentes, tarea que no se puede llevar a cabo actualmente debido a no disponer de más vídeos que contengan la acción en concreto, por lo que esta tarea quedaría pausada hasta el momento en el que se compartan en internet más vídeos de este tipo. Por otro lado, se plantea mejorar la calidad del dataset presente, aumentando la calidad de los vídeos mediante técnicas de edición de imágenes para disponer de vídeos de mejor calidad para el entrenamiento.

Además, para poder aprovechar los escasos datos del dataset, se propone implementar validación cruzada, lo cual permitirá no tener que separar los datos en entrenamiento y validación, sino poder aprovechar todos para entrenar mientras se mantiene la funcionalidad de llevar a cabo una validación, pero sin la necesidad de crear un conjunto de datos separado.

En lo que se refiere al modelo, hay varias variaciones que se quieren implementar para aumentar su precisión. Por un lado, se plantea emplear un modelo preentrenado capaz de extraer los puntos clave de las personas (articulaciones) para posteriormente estudiar su evolución en el tiempo con el objetivo de detectar acciones con dicho. Esto se planteó como opción a la hora de desarrollar el proyecto, pero debido a la baja calidad de los vídeos presentes en el dataset, el modelo preentrenado de detección de puntos clave presentaba poca precisión, lo cual limitaría mucho la precisión del modelo final.

Una segunda mejora que se plantea para el modelo es la implementación de Optical Flow. Esto permitiría comprender el movimiento de las personas en la escena, pudiendo hacer un análisis más exhaustivo de las escenas.

Además, se pretende implementar modelos de atención que permitan al modelo centrarse en la persona y no verse influenciado por el entorno, de manera que la toma de decisiones se vea más enfocada a las personas presentes en el vídeo. Se espera que estos modelos de atención tengan impactos positivos en la precisión del modelo.

Por último, se plantea automatizar la búsqueda de los hiperparámetros óptimos, la cual actualmente se lleva a cabo de forma manual. La implementación de un optimizador Bayesiano para la búsqueda de dichos valores óptimos reduciría sustancialmente el número de entrenamientos para llegar a unos resultados de calidad en la predicción.

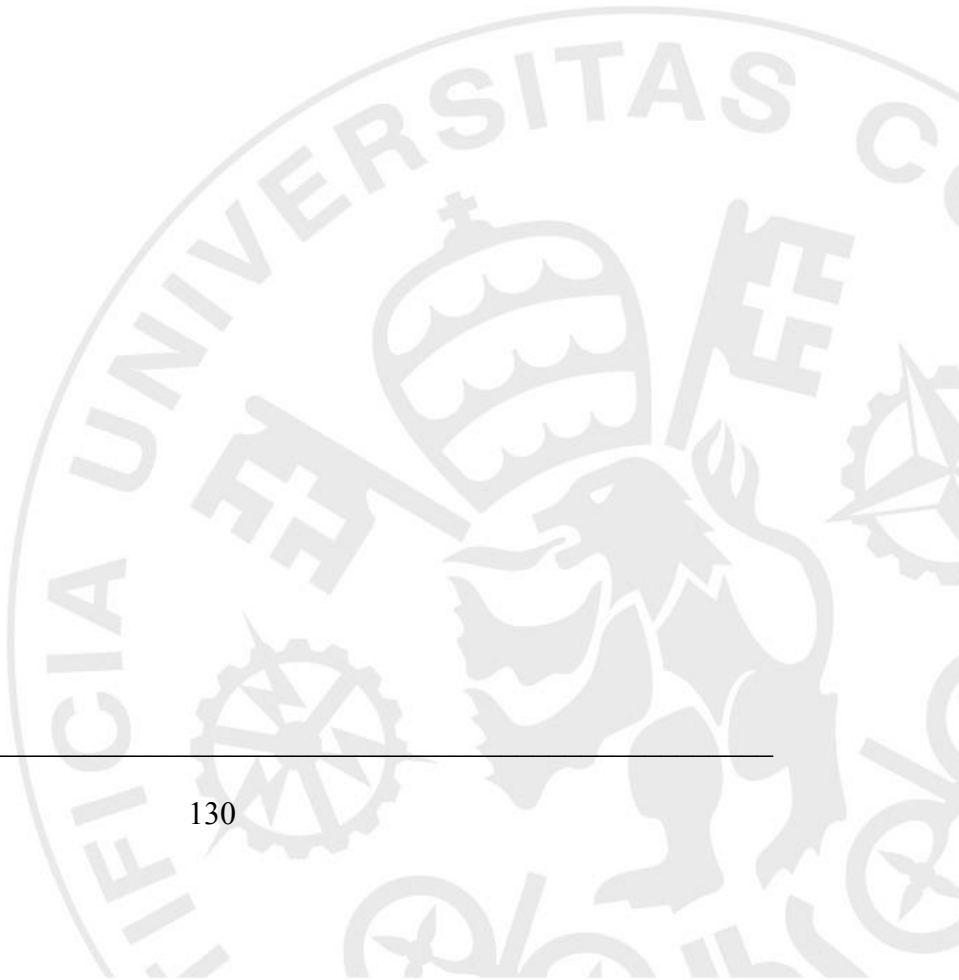
Con todos estos desarrollos, se busca que el modelo presente mejores resultados en la predicción de la acción en cuestión, y que además se agilice el proceso iterativo de entrenamiento.

CAPÍTULO VIII: ALINEACIÓN CON LOS ODS

Vivimos en un mundo globalizado y con una creciente población. El hecho de tener una sociedad con un mayor número de personas implica que debemos explotar más recursos para poder mantener la calidad de vida de todas ellas. Esto ha incrementado la preocupación por el modo en el que hacemos uso de dichos recursos a la hora de desarrollar un nuevo proyecto, ya que no debemos hacer uso de los recursos del mundo a una velocidad mayor de la que estos se pueden regenerar. Es por este motivo por el que el desarrollo sostenible ha pasado a un primer plano en la sociedad actual, marcando el ritmo y la manera en la que muchas empresas crecen y desarrollan sus proyectos.

Siguiendo esta tendencia del mundo actual, este proyecto está alineado con los objetivos de desarrollo sostenible, y concretamente con dos de ellos:

- Industria, Innovación e Infraestructuras: este proyecto fomenta la innovación y participa en la mejora de la capacidad tecnológica ofreciendo una solución para el análisis de acciones humanas grabadas desde cámaras de videovigilancia. Supone una solución disruptiva, ya que presenta una automatización a un trabajo que siempre ha sido manual, requiriendo la total atención de la persona encargada de la seguridad en dichas cámaras, y que con este desarrollo verá facilitado su trabajo.
- Paz, justicia e instituciones sólidas: la creación de un modelo que detecte acciones humanas de forma autónoma podrá aumentar la seguridad en aquellos lugares en los que haya instaladas cámaras de seguridad, pudiendo detectar acciones delictivas o en las que alguien quede dañado. De esta manera, este proyecto presenta una herramienta contra la violencia y la criminalidad, fomentando así la paz, la justicia y la seguridad.



CAPÍTULO IX: REFERENCIAS

[1] «Historia de las Redes Neuronales,» Itnuevolaredo, [En línea]. Available: http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/RNA/Redes%20Neuronales2.pdf.

[2] M. R. a. H. Ritchie, «Technological Progress,» Our World in Data, [En línea]. Available: https://ourworldindata.org/technological-progress.

[3] N. Babich, «What Is Computer Vision & How Does it Work? An Introduction,» Adobe, 28 Julio 2020. [En línea]. Available: https://xd.adobe.com/ideas/principles/emerging-technology/what-is-computer-vision-how-does-it-work/.

[4] L. Hulstraert, «A Beginner's Guide to Object Detection,» Datacamp, 19 Abril 2018. [En línea]. Available: https://www.datacamp.com/community/tutorials/object-detection-guide.

[5] P. Sharma, «Computer Vision Tutorial: A Step-by-Step Introduction to Image Segmentation Techniques,» Analytics Vidhya, 01 Abril 2019. [En línea]. Available: https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/.

[6] M. Rovai, «Realtime Multiple Person 2D Pose Estimation using TensorFlow2.x,» Towards Data Science, 03 Agosto 2020. [En línea]. Available: https://towardsdatascience.com/realtime-multiple-person-2d-pose-estimation-using-tensorflow2-x-93e4c156d45f.

[7] «Historia del Deep Learning,» Datahack, 08 Enero 2019. [En línea]. Available: https://www.datahack.es/blog/big-data/historia-deep-learning-etapas/.

[8] C. M. Learning, «Red Convolucional en PyTorch,» Medium, 09 Enero 2019. [En línea]. Available: https://medium.com/@cleverpysolutions/red-convolucional-en-pytorch-c499ae8af6ca.

[9] S. Verma, «Understanding 1D and 3D Convolution Neural Network | Keras,» Towards Data Science, 20 Septiembre 2019. [En línea]. Available: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610.

[10] V. S. Bawa, «Basic Architecture of RNN and LSTM,» PyDeepLearning, 18 Enero 2017. [En línea]. Available: https://pydeeplearning.weebly.com/blog/basic-architecture-of-rnn-and-lstm.

[11] «Machine Learning & Deep Learning Fundamentals,» Deeplizard, [En línea]. Available: https://deeplizard.com/learn/video/FK77zZxaBoI.

[12] A. Anwar, «Difference between AlexNet, VGGNet, ResNet, and Inception,» Towards Data Science, 07 Junio 2019. [En línea]. Available: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96.

[13] C. S. Y. J. Wei Liu, «Going deeper with convolutions,» Arxiv, 17 Sptiembre 2014. [En línea]. Available: https://arxiv.org/pdf/1409.4842v1.pdf.

[14] V. Feng, «An Overview of ResNet and its Variants,» Towards Data Science, 05 Julio 2017. [En línea]. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035.

[15] X. Z. S. R. y. J. S. Kaiming He, «Deep Residual Learning for Image Recognition,» Arxiv, 10 Diciembre 2015. [En línea]. Available: https://arxiv.org/pdf/1512.03385.pdf.

[16] M. Chablani, «YOLO - You only look once, real time object detection explained,» Towards Data Science, 21 Agosto 2017. [En línea]. Available: https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006.

[17] D. C.-e. Lin, «Tutorial: Build your custom real-time object classifier,» Towards Data Science, 30 Septiembre 2018. [En línea]. Available: https://towardsdatascience.com/tutorial-build-an-object-detection-system-using-yolo-9a930513643a.

[18] A. Rosebrock, «Intersection over Union (IoU) for object detection,» Pyimagesearch, 07 Noviembre 2016. [En línea]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[19] B. Raj, «An Overview of Human Pose Estimation with Deep Learning,» Medium, 28 Abril 2019. [En línea]. Available: https://medium.com/beyondminds/an-overview-of-human-pose-estimation-with-deep-learning-d49eb656739b.

[20] S. H. y. K. Niimura, «Optical Flow for Real-Time Human Detection and Action Recognition Based on CNN Classifiers,» Fuji Technology Press, 20 Julio 2019. [En línea]. Available: https://www.fujipress.jp/jaciii/jc/jacii002300040735/.

[21] e. a. Jinwoo Choi, «Why Can't I Dance in the Mall? Learning to Mitigate Scene Bias in Action Recognition,» Arxiv, 11 Diciembre 2019. [En línea]. Available: https://arxiv.org/pdf/1912.05534.pdf.

[22] C.-B. Jin, «Real-Time Action Detection in Video Surveillance using Sub-Action,» Arxiv, [En línea]. Available: https://arxiv.org/ftp/arxiv/papers/1710/1710.03383.pdf.

[23] J. C. y. A. Zisserman, «Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset,» Arxiv, 12 Febrero 2018. [En línea]. Available: https://arxiv.org/pdf/1705.07750.pdf.

[24] «Optical Flow,» CommonLounge, [En línea]. Available: https://www.commonlounge.com/discussion/1c2eaa85265f47a3a0a8ff1ac5fbce51.

[25] «Dataset UCF,» UCF, [En línea]. Available: https://www.crcv.ucf.edu/data/UCF101.php.

[26] «A Large-Scale Video Benchmark for Human Activity Understanding,» ActivityNet, [En línea]. Available: http://activity-net.org/.

[27] «HMDB: a large human motion database,» Serre Lab, [En línea]. Available: https://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/.

[28] «Dataset Something-Something,» Twentybn, [En línea]. Available: https://20bn.com/datasets/something-something.

[29] «The Sports-1M Dataset,» Github, [En línea]. Available: https://github.com/gtoderici/sports-1m-dataset/blob/wiki/ProjectHome.md.

[30] «AVA Datasets,» AVA, [En línea]. Available: http://research.google.com/ava/.

[31] «Dataset YouTube-8M,» YouTube-8M, [En línea]. Available: https://research.google.com/youtube8m/.

[32] «Kinetics Dataset,» DeepMind, [En línea]. Available: https://deepmind.com/research/open-source/kinetics.

[33] J. Brownlee, «How to use Learning Curves to Diagnose Machine Learning Model Performance,» Machine Learning Mastery, 27 Febrero 2019. [En línea]. Available: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/.

[34] «CNN Data Augmentation,» datahackers, 21 Noviembre 2018. [En línea]. Available: http://datahacker.rs/deep-learning-data-augmentation/.

[35] «Python OpenCV: Flipping an image,» Techtutorialsx, [En línea]. Available: https://techtutorialsx.com/2019/04/21/python-opencv-flipping-an-image/.

[36] «Data augmentation Techniques,» IQ Opengenus, [En línea]. Available: https://iq.opengenus.org/data-augmentation/.

[37] «Random Erasing Data Augmentation,» Awesome Open Source, [En línea]. Available: https://awesomeopensource.com/project/zhunzhong07/Random-Erasing.

[38] S. Karagiannakos, «Data Augmentation Techniques for Larger Image Datasets,» Lionbridge AI, 06 Abril 2020. [En línea]. Available: https://lionbridge.ai/articles/data-augmentation-with-machine-learning-an-overview/.

[39] «Image Data Augmentation for Maching Learning with Python,» Codebox, 16 Octubre 2016. [En línea]. Available: https://codebox.net/pages/image-augmentation-with-python.

[40] B. Dickson, «What is Adversarial Machine Learning?,» TechTalks, 15 Julio 2020. [En línea]. Available: https://bdtechtalks.com/2020/07/15/machine-learning-adversarial-examples/.

[41] U. Jaitley, «Why Data Normalization is necessary for Machine Learning models,» Medium, 08 Octubre 2018. [En línea]. Available: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029.

[42] M. Sharma, «Understanding Images with skimage-Python,» Towards Data Science, 31 Octubre 2019. [En línea]. Available: https://towardsdatascience.com/understanding-images-with-skimage-python-b94d210afd23.

[43] V. Tabora, «JPEG Image Scaling Algorithms,» Medium, 03 Septiembre 2019. [En línea]. Available: https://medium.com/hd-pro/jpeg-image-scaling-algorithms-913987c9d588.

[44] «Image Processing without OpenCV | Python,» Geeks for Geeks, 03 Abril 2019. [En línea]. Available: https://www.geeksforgeeks.org/image-processing-without-opencv-python/.

[45] J. D. Cook, «Three algorithms for converting color to grayscale,» John D. Cook Consulting, 24 Agosto 2009. [En línea]. Available: https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/.

[46] «TensorFlow Seminar,» TensorFlow, [En línea]. Available: https://ekababisong.org/gcp-ml-seminar/tensorflow/.

[47] «About Keras,» Keras, [En línea]. Available: https://keras.io/about/.

[48] «Tres formas de construir redes neuronales en TensorFlow con la API de Keras,» ICHI.PRO, [En línea]. Available: https://ichi.pro/es/tres-formas-de-construir-redes-neuronales-en-tensorflow-con-la-api-de-keras-141738974600062.

[49] «Keras code examples,» Keras, [En línea]. Available: https://keras.io/examples/.

[50] «Keras available models,» Keras, [En línea]. Available: https://keras.io/api/applications/.

[51] «PyTorch Torchvision,» PyTorch, [En línea]. Available: https://pytorch.org/vision/stable/index.html.

[52] P. Pedamkar, «Keras vs TensorFlow vs PyTorch,» EDUCBA, [En línea]. Available: https://www.educba.com/keras-vs-tensorflow-vs-pytorch/.

[53] «PyTorch Torchvision Datasets,» PyTorch, [En línea]. Available: https://pytorch.org/vision/stable/datasets.html.

[54] D. Sarkar, «A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning,» Towards Data Science, 14 Noviembre 2018. [En línea]. Available: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

[55] H. J. J. y. P. Wanda, «A Dynamic Pooling Layer for Convolution Neural Network,» Atlantis Press, Enero 2020. [En línea]. Available: https://www.atlantis-press.com/journals/ijcis/125932844/view.

[56] H. W. a. X. Gu, «Towards Dropout Training for Convolutional Neural Networks,» Arxiv, [En línea]. Available: https://arxiv.org/ftp/arxiv/papers/1512/1512.00242.pdf.

[57] «Understanding LSTM Networks,» Github, 27 Agosto 2015. [En línea]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[58] «Overfitting,» IBM Cloud Education, [En línea]. Available: https://www.ibm.com/cloud/learn/overfitting.

[59] K. Hoffman, «Machine Learning: How to Prevent Overfitting,» Medium, 30 Junio 2021. [En línea]. Available: https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9.

[60] A. O. Ahsan, «Convolutional Neural Network and Regularization Techniques with TensorFlow and Keras,» Medium, 05 Enero 2020. [En línea]. Available: https://medium.com/intelligentmachines/convolutional-neural-network-and-regularization-techniques-with-tensorflow-and-keras-5a09e6e65dc7.

[61] J. López, «¿Por qué las GPUs consiguen más rendimiento entre generaciones que las CPUs?,» HardZone, 10 Diciembre 2020. [En línea]. Available: https://hardzone.es/tutoriales/rendimiento/gpus-rendimiento-cpus-escalabilidad/.

[62] «Performance of GPU vs. CPU,» Researchgate, [En línea]. Available: https://www.researchgate.net/figure/Performance-of-GPU-vs-CPU-

Enlarging-performance-gap-between-GPUs-and-CPUs-Figure-from_fig3_292146956.

[63] D. Radecic, «PyTorch: Switching to the GPU,» Towards Data Science, 03 Mayo 2020. [En línea]. Available: https://towardsdatascience.com/pytorch-switching-to-the-gpu-a7c0b21e8a99.

[64] «Métricas de Machine Learning,» Breakdown, [En línea]. Available: https://bookdown.org/f_izco/BDC-POC/metricas.html.

[65] «Accuracy and Loss graphs,» ResearchGate, [En línea]. Available: https://www.researchgate.net/figure/Graphs-of-the-performance-of-the-accuracy-A-and-the-loss-function-B-of-the-ANN-in_fig5_325829191.

[66] C. Kevin, «Feature Maps,» Medium, 11 Mayo 2018. [En línea]. Available: https://medium.com/@chriskevin_80184/feature-maps-ee8e11a71f9e.

[67] S. Sharma, «Activation Functions in Neural Networks,» Towards Data Science, 06 Septiembre 2017. [En línea]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[68] «Tanh Activation,» Papers with Code, [En línea]. Available: https://paperswithcode.com/method/tanh-activation.

[69] Prabhu, «Understanding Hyperparameters and its Optimisation techniques,» Towards Data Science, 03 Julio 2018. [En línea]. Available: https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debba07568.

[70] «Precisión mixta,» TensorFlow, [En línea]. Available: https://www.tensorflow.org/guide/mixed_precision.

[71] S. N. e. al, «Mixed Precission Training,» Arxiv, 15 Febrero 2018. [En línea]. Available: https://arxiv.org/pdf/1710.03740.pdf.

[72] «Automatic Mixed Precission,» NVIDIA, [En línea]. Available: https://developer.nvidia.com/automatic-mixed-precision.

[73] «Universidad de Stanford: procesamiento del lenguaje natural y aprendizaje profundo (CS224n) Notas Lección 13 Red neuronal convolucional (CNN) en PNL,» ProgrammerClick, [En línea]. Available: https://programmerclick.com/article/4916832766/.

[74] R. Alake, «Deep Learning: GoogLeNet Explained,» Towards Data Science, 23 Diciembre 2020. [En línea]. Available: https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765.

[75] Sayatini, «Keras vs TensorFlow vs PyTorch : Comparison of the Deep Learning Frameworks,» Edureka, 24 Junio 2020. [En línea]. Available: https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/.

[76] S. Sharma, «Explained: Deep Learning in Tensorflow,» Towards Data Science, 02 Diciembre 2019. [En línea]. Available: https://towardsdatascience.com/explained-deep-learning-in-tensorflow-chapter-1-9ab389fe90a1.