Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación de clasificadores basados en Deep Learning para el reconocimiento de acciones en videos.

Autor: Jose Carlos García Arnaiz

Tutor: Sergio Toral Marín

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Aplicación de clasificadores basados en Deep Learning para el reconocimiento de acciones en videos.

Autor:
Jose Carlos García Arnaiz

Tutor:
Sergio Toral Marín
Catedrático

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Grado: Aplicación de clasificadores basados en Deep Learning para el reconocimiento de acciones en videos.

Autor: Jose Carlos García Arnaiz

Tutor: Sergio Toral Marín

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2020

A mi familia

A mis maestros

Agradecimientos

Este trabajo pone fin a la etapa que sin duda más me ha marcado en mi vida. He adquirido conocimientos de gran valor no solo académicos sino también personales. Por esto me gustaría agradecer a todos los profesores que me han acompañado durante estos años de aprendizaje.

Me gustaría agradecer también a mi tutor Sergio el haberme mostrado este mundo de la inteligencia artificial y el haberme guiado y aconsejado durante la realización de este trabajo que tan interesante ha sido.

También quería agradecer a mis compañeros por el apoyo en estos años. En especial a Javier que siempre ha estado ahí.

Y no podía olvidarme de agradecer a mi familia, mi novia y su familia por su apoyo y acompañarme durante el esfuerzo que han supuesto todos estos años.

Jose Carlos García Arnaiz

Sevilla, 2020

Resumen

Nos encontramos en una época donde las redes sociales, y el contenido de entretenimiento por streaming está totalmente asentado e incrustado en nuestras vidas. El procesamiento de imágenes se convierte en un campo de grandes posibilidades. El marketing es sin duda uno de los campos donde más se hace uso de estas técnicas, pero la detección en imágenes o vídeos mediante el uso de una inteligencia artificial nos abre un amplio abanico de posibilidades. Como por ejemplo un sistema de recomendaciones de videos o series similares a lo que estás actualmente viendo, la detección en tiempo real de objetos en el campo de la seguridad, la detección de enfermedades y otros muchos campos.

En este trabajo vamos a introducirnos en la creación de unos clasificadores de videos creados con redes neuronales, para ello tendremos que entrenar un modelo de redes neuronales para que sea capaz de distinguir distintas clases de contenidos en videos. En concreto vamos a usar la base de datos de videos conocida como UCF101 que contiene un total de 101 clases diferentes de videos.

Nuestra intención es usar modelos ya existentes, que ya están entrenados. La técnica de transfer learning nos permitirá usar estos modelos ya entrenados para clasificar otro tipo de videos diferentes. Esto no solo nos permite entrenar un modelo mucho más rápido, sino que también nos permite entrenar el modelo con un ordenador menos potente sin reducir su porcentaje de acierto clasificando.

Los modelos pre entrenados normalmente están entrenados para un número de cientos de clases diferentes, esto como muestra de rendimiento es interesante, pero la realidad es que normalmente sólo nos interesará reconocer ciertas acciones, por ejemplo, si queremos contabilizar el tiempo que se dedica en los telediarios a cada deporte, no es necesario que intentemos clasificar con todos los deportes existentes, sino con los mayoritarios o los que a nosotros nos interesa detectar.

En concreto entrenaremos un modelo llamado C3D y otro llamado I3D, donde descubriremos que, aunque ambos modelos podrían ser utilizables para las clases escogidas, cada modelo tiene sus propios puntos fuertes frente al otro.

Abstract

We are in an age where the social networks and the streaming entertainment content have entered our lives. The image processing has become an area of great possibilities. Marketing is the area where these techniques are being used the most, but the image detection and video detection using artificial intelligence open a wide range of possibilities. Like a recommendation system of videos or series similar to what you are currently watching, live objects detection in the security area, disease detection in the health area, an many other areas.

We are going to use some pre-trained models. The transfer learning technique allow us use these models to classify other different classes of video. This will help us to train the model much faster and to train the model with a less powerful personal computer without bring down the accuracy of the model.

Pre-trained models are usually trained to classify hundreds of different classes of videos, this shows the power of the model, but actually we are going to classify a few of these classes. For example, if we want to count the time that TV news show something about different sports, we don't need to detect all the different sports, we can only detect the most played sports or the sports we are interested in.

In short, we are going to train a model known as C3D, and another model known as I3D. We will be found that both models are usable to classify properly, each model will have its advantages over the other model.

Índice

1 INTRODUCCIÓN

Some people call this artificial intelligence, but the reality is this technology will enhance us. So instead of artificial intelligence, I think we'll augment our intelligence.

Ginni Rometty, CEO IBM

1.1 Motivación del trabajo

La inteligencia artificial se conoce desde hace bastantes años, pero no ha sido hasta hace relativamente poco que los ordenadores han sufrido un incremento de su capacidad computacional exponencial tan grande que ahora es posible hacer uso de ella desde casa sin un superordenador para tareas sencillas de inteligencia artificial.

Este crecimiento de la capacidad de computación junto a la miniaturización de los componentes ha propiciado que el consumo de terminales electrónicos inteligentes se haya introducido en todas y cada una de las facetas de nuestra vida.

Otro aspecto que ha ayudado a la popularización de este campo es la aparición de las criptomonedas, ya que para conseguir minar una de estas necesitas un ordenador con la suficiente potencia para realizar una serie de cálculos complejos. Además, con la industria de los videojuegos en alza, empresas como AMD o NVIDIA han invertido e investigado en la mejora de estos procesos de cálculos, lo que ha propiciado no solo una mayor capacidad computacional en las GPUs sino también a una reducción de los precios.

La creación de paquetes especializados en Deep learning como Pandas, Tensorflow, Keras, Pytorch para el uso en el lenguaje de Python, ha hecho más accesible el proceso de creación de redes neuronales, su entrenamiento y las predicciones de estas. Esto ha ayudado a la popularización de uso.

Estas tecnologías de aprendizaje para máquinas se encuentran en plena ebullición y las empresas cada día encuentran nuevas formas de hacer uso de ellas y de aumentar su eficiencia y mejorar sus predicciones.

El uso de la inteligencia artificial se está utilizando en campos muy diversos, como en la fabricación de productos, automóviles inteligentes, en medicina como para la detección de enfermedades. En nuestro caso nos vamos a centrar en la detección de acciones en videos. Esto puede usarse con distintos objetivos como la recomendación de videos y series similares, mejora de sistemas de seguridad, recomendación de canales de televisión donde estén emitiéndose algún deporte en concreto, control parental para evitar a niños el consumo de material inadecuado para su edad y otros tantos usos.

Los avances en el campo de la inteligencia artificial nos han permitido conocer las diferentes técnicas de Deep Learning, que al contrario que con la inteligencia artificial y las otras técnicas de machine learning no es necesario que tengamos conocimientos tan avanzados de las matemáticas para la creación de un algoritmo que consiga dar a un programa la capacidad de aprender a realizar tareas tan complejas para un programa informático.

Con el Deep Learning, usaremos las llamadas neuronas artificiales que tratan de simular el funcionamiento de las neuronas humanas, que se interconectan entre ellas mandando señales eléctricas, al hecho este de que una neurona le mande una señal eléctrica a otra se le conoce por el nombre de activación. Una neurona artificial se

marcará como activada cuando emita un valor de 1. Para dar más valor a las neuronas que se activen correctamente durante el entramiento, a cada neurona les asignaremos un peso, que conforme mayor sea este, más importancia tendrá en el resultado final de la clasificación.

De esta forma las neuronas por su cuenta irán aprendiendo a diferenciar distintos patrones en las imágenes para conseguir clasificarlas, sin que el programador le indique qué es lo que debe aprender a las neuronas, si no que estas de forma autónoma, conociendo los datos de entrada con los que vamos a entrenar y el resultado a la salida que deberíamos obtener con estos datos de entrada, irá aprendiendo y corrigiéndose.

1.2 Objetivos del trabajo

En la actualidad han proliferado los datasets, bases de datos, de imágenes. Estos datasets separan las imágenes en distintas clases para que puedan ser usadas para entrenar clasificadores. También existen, pero algo más escasos, los datasets de videos.

Con la llegada de los datasets comenzaron a llegar los concursos de clasificadores, donde los participantes deben entrenar sus modelos para clasificar de la forma más acertada posible los datos pertenecientes a un dataset concreto. En nuestro trabajo vamos a usar el dataset de videos UCF101 que está compuesto por 101 clases de videos de acciones humanas en diferentes.

Esto, junto a la cultura del open software, fomentó la aparición de modelos entrenados por estos participantes. Pero estos modelos se encontraban entrenados para clasificar ciertas clases concretas de vídeos o imágenes. Para clasificar un nuevo deporte, animal, o tipo de flor, era necesario entrenar un modelo nuevo desde cero.

Aquí es donde aparece la técnica llamada transfer learning, que nos permite reutilizar modelos ya entrenados para clasificar cosas diferentes o solo alguna de ellas. El objetivo del trabajo es clasificar un conjunto reducido de secuencias de videos a partir de modelos pre-entrenados con bases datos y utilizando el concepto de transfer learnig.

Esto nos permite alcanzar grandes porcentajes de acierto sin tener que entrenar el modelo entero, dependiendo nuestro fin y del modelo nos bastará con colocar sólo una capa que adapte la salida al objetivo que queramos conseguir.

Como normalmente tampoco es necesario crear un clasificador de cientos de clases, nosotros nos vamos a centrar en un número reducido de clases que han sido seleccionadas aleatoriamente. Concretamente vamos a clasificar videos de bebes gateando, patinaje artístico y skateboard. Pero por ejemplo en la situación de una cámara de seguridad podríamos detectar por ejemplo actos de violencia, signos de vandalismo o de robos, y activar una alarma, enviar una notificación por móvil, o encender alguna luz de aviso.

Tras realizar el transfer learning compararemos el rendimiento en dos modelos distintos, que veremos que, aunque los dos clasifican bien las clases, tienen sus ventajas frente al otro modelo.

Otro tema que vamos a abarcar es la creación de una clase nueva de vídeos. Esta nueva clase serán carreras de coches y los 232 videos con los que vamos a entrenar y evaluar el modelo haciendo transfer learning se han obtenido de la plataforma de YouTube. Comprobaremos que como el modelo está construido correctamente, a pesar de tener que clasificar 4 modelos, entre ellos uno creado por nosotros. Su eficiencia en cuanto a porcentaje de acierto se ve resentida pero ligeramente.

Durante el desarrollo del trabajo no hemos usado solo el ordenador personal, también hemos usado la herramienta de Google Colab. Google Colab, con algunas limitaciones, te permite ejecutar en la nube de forma gratuita programas escritos con el lenguaje de Python. Este entorno de ejecución ya cuenta con algunos de los paquetes más usados como Tensorflow, Keras, Pandas, Matplotlib, etc. De esta forma no consumiremos recursos de nuestro ordenador mientras entrenamos los modelos a los que hacemos referencia en el trabajo, sino que se ejecutarán en la nube.

2 ESTADO DEL ARTE

Antes de avanzar con el trabajo es necesario introducir ciertos conceptos que son básicos para el entendimiento del funcionamiento de las redes neuronales y el deep learning.

2.1 Redes neuronales

Las técnicas de Deep Learning basan su algoritmo en el uso de redes neuronales artificiales. Las redes neuronales artificiales son un modelo computacional basado en un conjunto de unidades más simples y básicas a las que llamamos neuronas. Estas neuronas artificiales tratan de similar el comportamiento que se observa en los axones de las neuronas de algunos animales y en el cerebro humano.

En la siguiente imagen podemos ver cómo serán las neuronas aproximadamente.

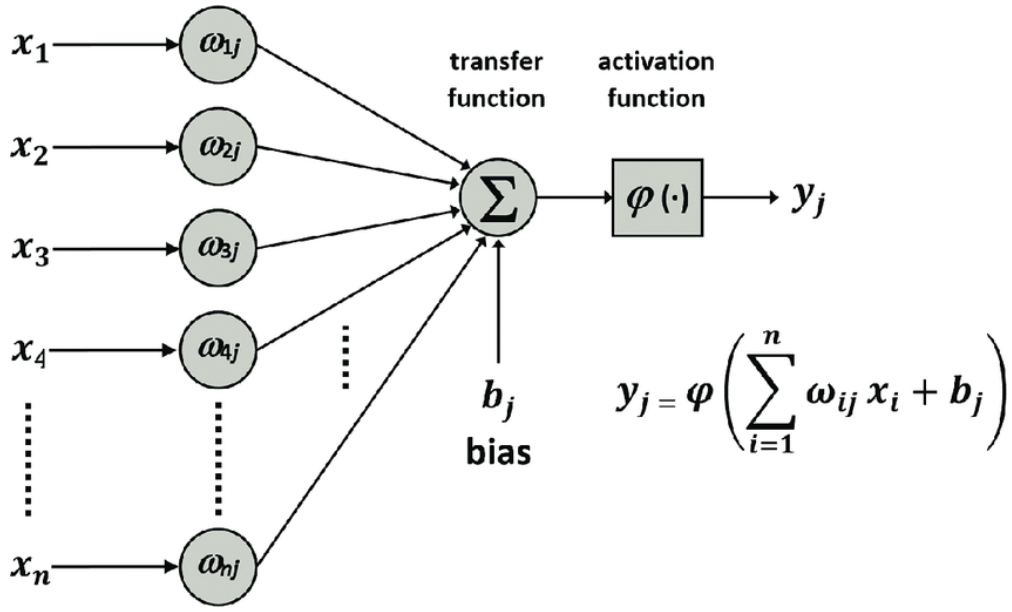

Figura 2-1: Esquema de una neurona artificial [1]

Cómo podemos ver en la imagen tenemos diferentes componentes en la neurona. Los valores de x_1, x_2, …, x_n son los datos de entrada en la neurona. Estos datos pueden ser también procedentes de otra neurona anterior.

Las variables ω_0, ω_1, ω_2, …, ω_n son los pesos relativos de cada valor de entrada.

El valor b es el valor de sesgo o bias. Este es un valor que viene de una neurona especial que tendremos en cada capa de la red neuronal que simplemente tendrá un valor de 1. Sin esta neurona es posible que una entrada que tiene todos sus parámetros a 0, nunca podría llegar por ejemplo a la salida correcta si es distinto de cero. Con tener al menos esta neurona activa, los pesos se modificarán lo necesario para ajustarse. El peso ω_0 sería el correspondiente a al valor de bias.

La función φ es la llamada función de activación. Esta función es la salida de la neurona y es lo que da tanta flexibilidad a las redes neuronales y hacer relaciones no lineales de datos. Esta función puede ser una función sigmoidal, una función tangente o una función ReLU entre otras.

Conforme el problema que queramos resolver sea más complicado, deberemos agregar más neuronas y más capas de neuronas. Pero no hay que olvidar que esto aumentará el número de cálculos que deberá realizar el programa durante el entrenamiento y por tanto necesitará más tiempo y potencia del ordenador.

Un ejemplo de red neuronal podría ser el siguiente:

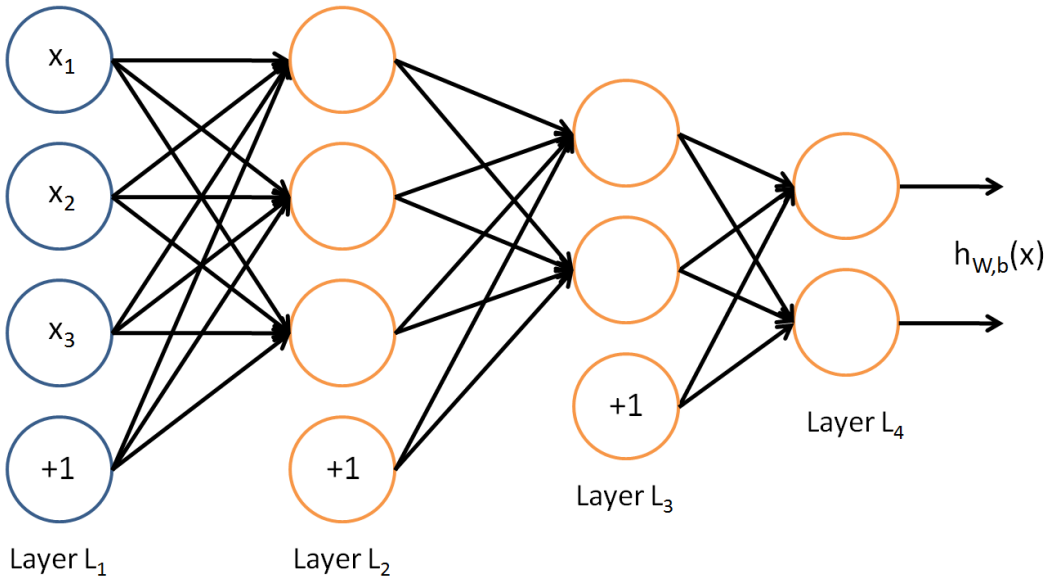

Figura 2-2: Ejemplo de red neuronal artificial[2]

En la figura anterior podemos distinguir 3 tipos distintos de capas.

La capa de entrada (o input layer en inglés), la capa L1 en el dibujo, es la capa que contiene las neuronas las que le daremos los datos de entrada del problema.

La capa de salida (u output layer) que es la capa L4, es la capa con las neuronas que nos darán los resultados del paso de los datos de entrada por toda la red neuronal.

El resto de capas, L2 y L3, son conocidas como capas ocultas (o hidden layers) son las capas que realizarán los cálculos necesarios para ajustar los pesos y obtener el resultado deseado.

El back propagation o propagación hacia atrás es un algoritmo que determina el error en la salida y lo propaga hacia atrás por toda la red. Con este algoritmo es con el que las neuronas ajustarán sus pesos y, en definitiva, el algoritmo que permite aprender a las redes neuronales.

2.2 Arquitecturas

2.2.1 CNN

Nuestro cerebro cuando visualiza algo trata de buscar ciertas propiedades y dependiendo de las propiedades que detectamos solemos categorizarlas. Las redes convolucionales (CNN) tratan de simular este comportamiento. Estas redes neuronales están especialmente pensadas pues para la visión artificial y el reconocimiento de imágenes.

Este supuesto de que las entradas son imágenes nos permite reconocer ciertas propiedades que nos permitirán reconocer elementos concretos en las imágenes. La idea es que cada capa de la red neuronal se especializará en detectar algún tipo de estructura diferenciadora.

La principal ventaja de las capas convolucionales frente a las capas densas es que mientras las capas densas aprenden patrones globales, las capas convolucionales aprenden patrones locales más pequeños.

En el caso de las imágenes normalmente usaremos las llamadas capas convolucionales 2D. Aunque la realidad es que las imágenes se almacenan en matrices de 3 dimensiones, los dos ejes espaciales (altura, anchura) y el eje de canal. En caso de que estemos tratando una imagen RGB tendremos 3 canales diferentes, mientras que en imágenes en escala de grises solo tendremos 1 canal.

En vídeo deberemos añadir una dimensión más que será la variable temporal. Como nos encontramos en un entorno digital, en lugar del tiempo esta dimensión extra será el número de frames correspondiente.

El funcionamiento de las capas convolucionales es muy sencillo. Se basan en el uso de la matriz de la imagen,

a la cual le realizaremos la operación de convolución con sus pesos. Aunque en las imágenes los pesos serán entre 0 y 255, en la siguiente imagen podemos verlo con pesos de 0 y 1 para ilustrarlo fácilmente.

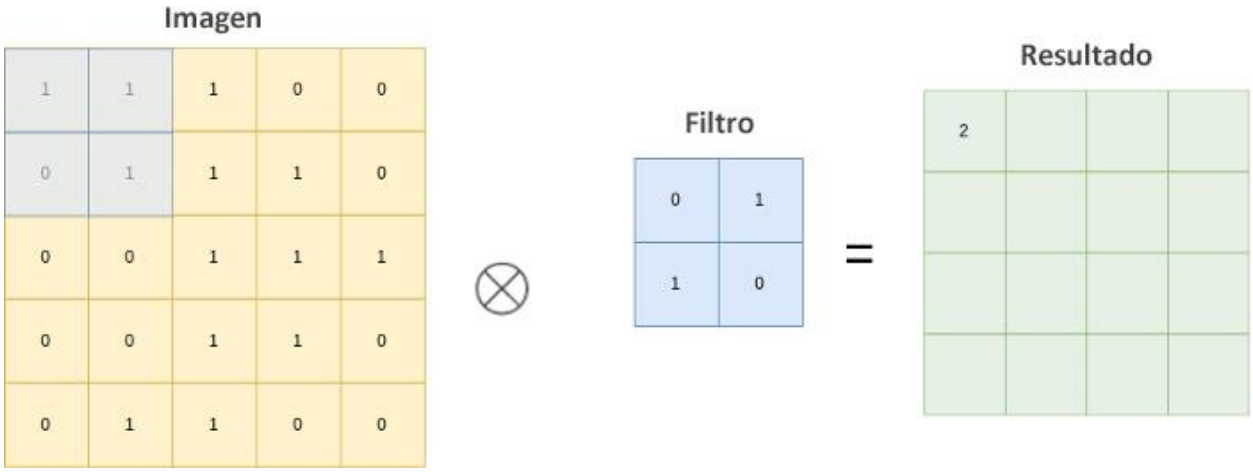

Figura 2-3: Ejemplo de la operación de convolución

Un parámetro muy importante de estas capas es el stride. Por defecto cuando el filtro o kernel realiza la convolución se desplazará un valor a la derecha, pero si ponemos un stride=2, el filtro se moverá dos lugares a la derecha, lo que hará que la matriz resultante será más pequeña que con un stride=1.

Aunque en la imagen veamos que la imagen es de 2 dimensiones, normalmente tendrá 3 como hemos comentado, e igualmente el filtro tendrá 2 dimensiones para las imágenes y 3 para el caso de video.

Igualmente, la explicación se ha realizado con 1 sólo filtro, pero en las redes convolucionales tendremos decenas o incluso cientos de filtros por capa y muchos diferentes.

Un tema importante son los pesos que se le asignan al propio filtro. Estos pesos normalmente serán calculados por Keras en nuestro caso como parte del entrenamiento de la red neuronal. Pero existen miles de filtros ya creados para conseguir distintos efectos en la imagen. Como un efecto de blur[3], de detección de bordes, enfocar la imagen, y muchos más.

2.2.2 RNN

En redes neuronales recurrentes las neuronas se alimentarán ellas mismas un número N de veces, es decir que su salida la vuelve a poner en su entrada.

Esta auto realimentación permite que tengan cierta memoria a corto plazo, por lo que la neurona recordará lo que pasó atrás en el tiempo, esto les permite pasarse ellas mismas información en el futuro y analizarlas.

Hay tres grandes arquitecturas en el caso de RNN:

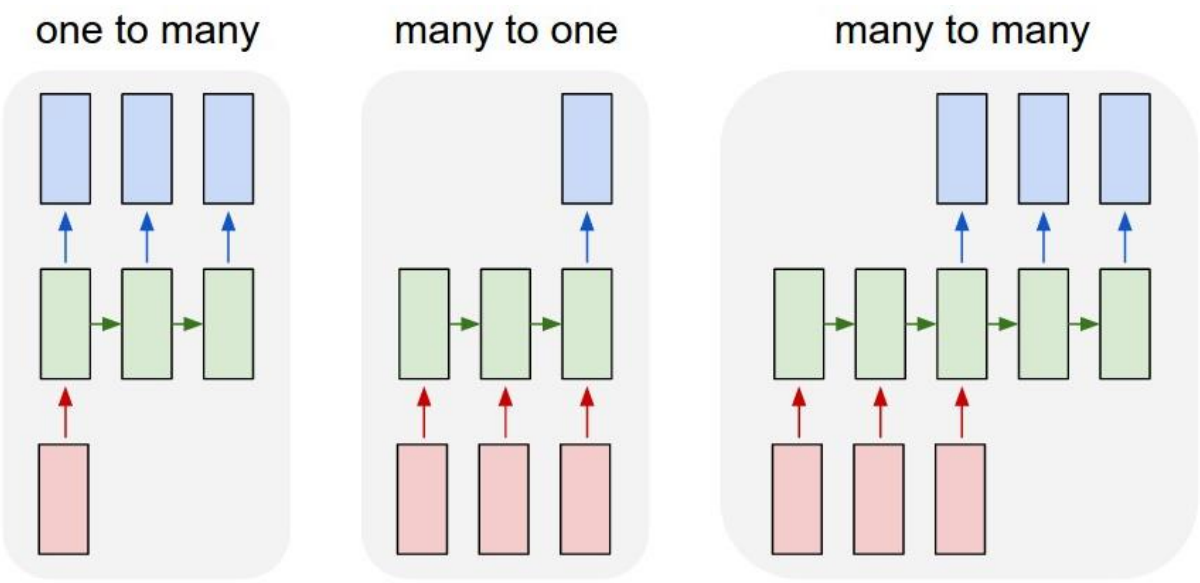

Figura 2-4:Arquitecturas principales de RNN[4]

- One to many: Tenemos una entrada y varias salidas. La entrada será una imagen y la salida una descripción de la imagen. Normalmente suele usarse primero una red convolucional antes de pasar por la red neuronal recurrente
- Many to one: Varias entradas pero una salida. Un ejemplo sería cuando analizamos si una reseña sobre un producto es positiva o negativa, que tenemos que detectar varias palabras claves.
- Many to many: Varias entradas y varias salidas. Puede usarse como ejemplo cuando realizamos la traducción de una frase, que las entradas son varias palabras y la salida también varias palabras.

Otra arquitectura muy usada es la arquitectura LSTM, que básicamente mediante unas neuronas más complejas consigue que el tiempo que la neurona recuerda sus entradas es bastante más largo.

2.3 Transfer Learning

Hasta ahora todo lo que hemos visto sobre el Deep Learning parece de ensueño. Por lo que no entenderíamos cómo es posible que no esté mucho más extendido el uso del Deep Learning para ciertos problemas.

Para entrenar una red neuronal de forma satisfactoria que sea compleja vamos a necesitar 3 cosas fundamentales. La primera es un gran set de ejemplos para entrenar la red neuronal, el segundo va relacionado a esto mismo, y es que vamos a necesitar un ordenador potente para poder procesar las imágenes y poder entrenar a la red neuronal. Y, por último, tiempo, entrenar una red neuronal puede llevar bastante tiempo.

El problema de necesitar un ordenador potente estaba siendo un problema para el desarrollo del trabajo, pero finalmente haciendo uso de la herramienta Google Colab pudimos hacer algunos entrenamientos, pero los recursos de Google Colab no son infinitos así que nos hemos tenido que limitar en el número de clases diferentes que vamos a emplear.

Para solucionar el problema del tiempo y un poco también la capacidad necesaria de proceso, haremos uso del Transfer Learning.

El Transfer Learning es un método de Deep Learning que nos permite reusar un modelo ya entrenado, como punto de partida para el problema que queremos solucionar con nuestra red neuronal. Esto sería como coger una red neuronal que está entrenada para distinguir perros de gatos y reutilizarla para distinguir distintos pájaros.

Especialmente cuando vamos a hacer una clasificación de imágenes suele ser recomendable hacer uso de modelos ya pre entrenados, aunque nos parezca que el modelo está pre entrenado para un problema que no está relacionado para nada con el modelo que deseas, es aconsejable intentarlo, pues las capas interiores que han aprendido por ejemplo a detectar los bordes y otros patrones, podrían ayudarnos.

Existen muchos datasets típicos que a su vez ya suelen tener también modelos pre entrenados que normalmente son modelos que han ganado concursos usando como entrenamientos estos datasets. En nuestro caso vamos a hacer uso de la biblioteca de Keras que es una biblioteca extensión de la biblioteca de Tensorflow. En la propia biblioteca de Keras se incluyen varios de los modelos pre entrenados más usados para clasificación de imágenes.

En la siguiente tabla se puede ver una selección de alguno de los modelos más conocidos.

Modelo	Tamaño	Acierto Top-1	Acierto Top-5	Parámetros	Capas
VGG16	528 MB	0,713	0,901	138.357.544	23
ResNet50	98 MB	0,749	0,921	25.636.712	50
InceptionV3	92 MB	0,779	0,937	23.851.784	159
Xception	88 MB	0,790	0,945	22,910,480	126
DenseNet121	33 MB	0,750	0,923	8.062.504	121

Tabla 2-1: Datos de modelos preentrenados conocidos de Keras

Normalmente en el Transfer Learning agregaremos algunas capas nuevas, normalmente en el caso de un clasificador como es nuestro caso, agregaremos algunas nuevas capas densas para ayudar al modelo a mejor clasificación de los datos de la entrada.

2.4 Overfitting

Uno de los mayores problemas que nos encontraremos en Deep Learning al entrenar un modelo es el overfitting. Esto ocurre porque el modelo que estamos entrenando se intenta ajustar demasiado bien los datos con los que estamos entrenando, y tiene un resultado muy pobre para otros datos que debería clasificar sin problemas.

Como el modelo ya estará pre entrenado, hacer Transfer Learning también nos ayudará a evitar encontrarnos overfitting por falta de datos de entrada para el entrenamiento.

El overfitting podemos detectarlo de varias formas. Si al entrenar nuestro modelo detectamos que la curva del error de entrenamiento (Training error) difiere mucho del error de pruebas (Testing error).

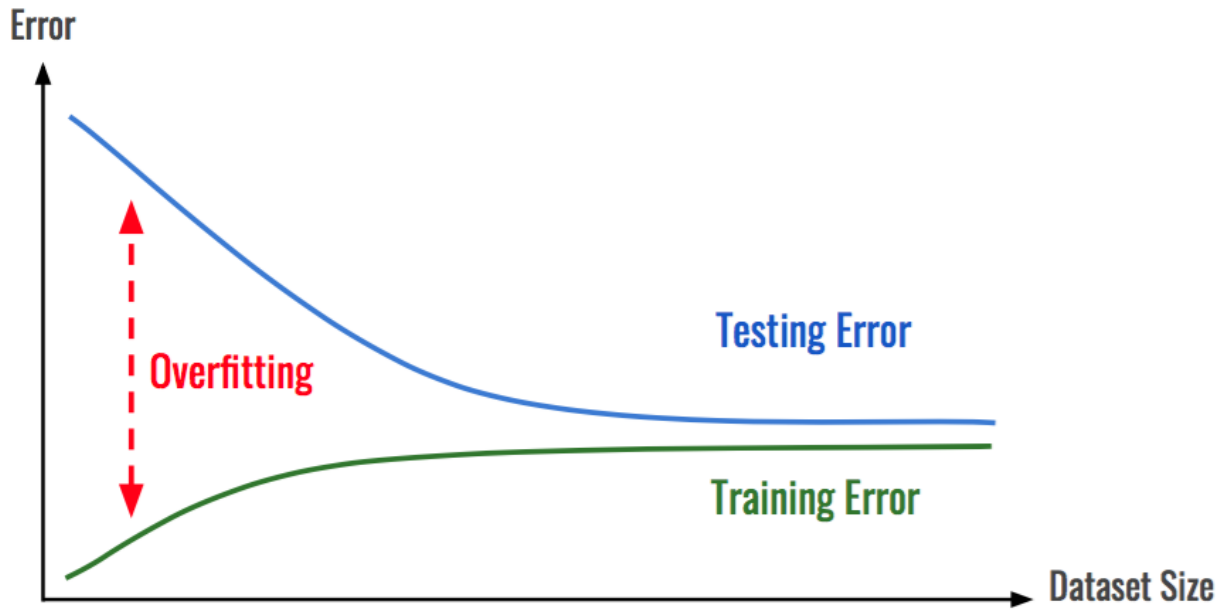

Figura 2-5: Ejemplo de overfitting[5]

Algunas medidas que podemos tomar para evitar el overfitting son:

- Usar capas dropout, explicaremos posteriormente qué hacen, pero es importante no usarlas tras la última capa.
- Agregar más datos de entrenamiento, en caso de que no nos sea posible deberemos hacer uso de técnicas de data augmentation. Estas técnicas se basan en entrenar la red varias veces con la misma imagen, pero con zoom, o volteada, o girada, o desenfocada, en definitiva, modificada, y esto ayudará también a entrenarse a la red.
- Normalizar los datos de entrada también suele ser una buena opción para que la red consiga entrenarse de manera óptima.
- Otra de las técnicas que podemos usar es la regularización, consiste en ponerle una penalización a la función de coste, de esta forma lo que conseguimos es un modelo más simple que generalizará mejor. Los modelos complejos son los que suelen tender a tener overfitting. Las regularizaciones más usadas son la regularización Lasso (también llamada L1) y la regularización Ridge (L2).
- Es importante también elegir un buen número para el learning rate, si lo ajustamos muy alto, tendremos overfitting casi con toda probabilidad, pero tampoco podremos ponerlo muy bajo porque entonces el modelo no conseguirá aprender al ritmo necesario. La forma más sencilla para observar si hemos seleccionado un learning rate correcto sería a modo de prueba/error. La idea no es entrenar el modelo por completo, sino observar el comportamiento en las primeras épocas del entrenamiento para detectar si hay overfitting, y en caso de haberlo parar el entramiento y usar otro valor de learning rate.
- Por último, otra técnica que suele usarse para evitar el overfitting es la parada temprana (o early stopping). Para ello debemos atender al error de validación y al error del entrenamiento. Normalmente estos valores irán decrementándose durante el entrenamiento, pero cuando tenemos overfitting el error de validación se empezará a incrementar. Con la parada temprana el entrenamiento del modelo se parará por dos motivos, uno porque ha detectado que el error de validación se está incrementando, y dos parará cuando detecte que a pesar de estar entrenando la mejora es insignificante.

2.5 Capas de Keras usadas

La biblioteca de Keras nos proporciona muchas capas diferentes para crear nuestras redes neuronales. A continuación, vamos a ver una selección de las capas que hemos usado y cuál es su fin.

2.5.1 Conv3D

La capa de convolución 3D es una de las capas básicas cuando queremos hacer tratamiento de videos como matrices de datos.

Esta capa suele ser la primera capa que nos encontraremos en algunas de las redes neuronales creadas para ser entrenadas con videos. Esta capa creará un filtro de 3 dimensiones que realizará la operación de convolución por todos los datos de entrada.

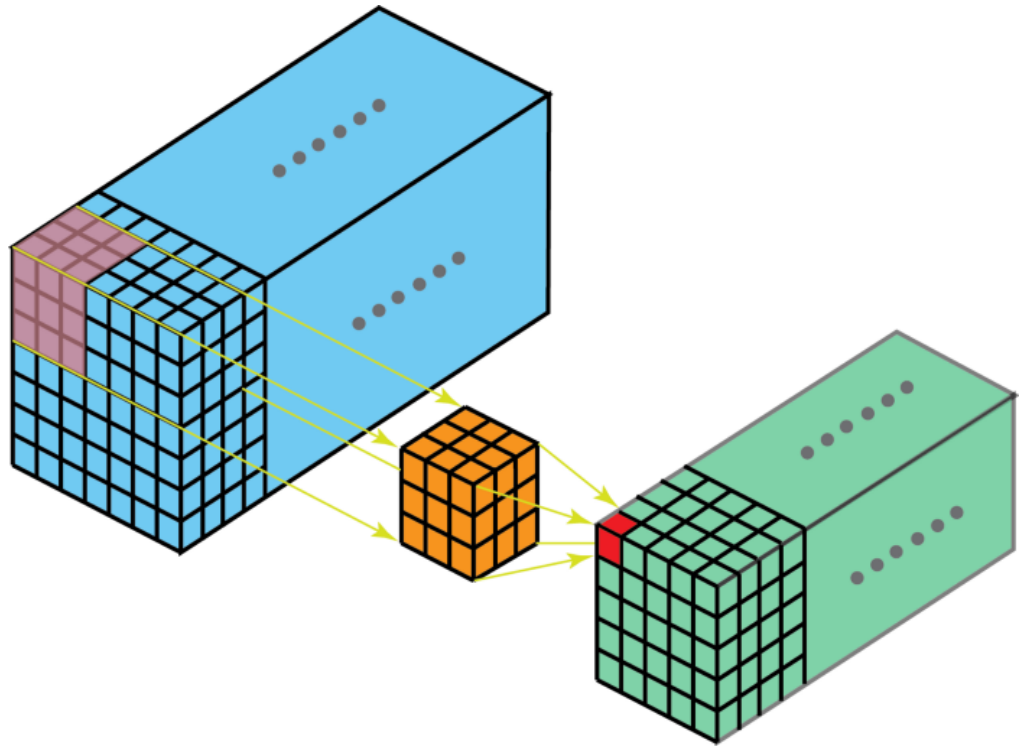

Figura 2-6: Esquema de una convolución 3D[6]

Un parámetro importante es el input_shape. Para aprender cómo configurarlo debemos entender que cuando vamos a entrenar una red neuronal con videos, normalmente tendremos matrices de 5 dimensiones. Tener tantas dimensiones puede llegar a sorprendernos. Las dimensiones serán normalmente en este orden (muestra, frames, altura, anchura, canales).

De esta forma si vamos a entrenar con 15 videos, los que tomaremos 20 frames de 500x400 pixeles y lo haremos con escala de grises, tendremos una matriz con las siguientes dimensiones (15,20,500,400,1).

Con el input_shape imponemos cómo queremos que sean los datos de entrada, pero lógicamente no podemos imponer con cuántos vídeos vamos a entrenar la red neuronal, por lo que el input_shape deberá tener 4 dimensiones (frames, altura, anchura, canales).

2.5.1 MaxPooling3D

Normalmente cuando usamos una capa de convolución lo hacemos con más de un filtro a la vez, por lo que aumenta mucho el número de parámetros a entrenar, es por esto por lo que en muchas ocasiones tras una capa de convolución haremos uso de capas de pooling para reducir el número de neuronas y parámetros necesarios.

Con la capa de MaxPooling configuraremos un volumen de datos y de entre todos ellos nos quedaremos solo con el valor máximo. Al igual que en la capa de convolución este filtro irá desplazándose por los datos y así reduciremos el número de parámetros y neuronas necesarios.

Para ejemplificar el concepto, vamos a visualizarlo en la siguiente imagen para el caso de dos dimensiones.

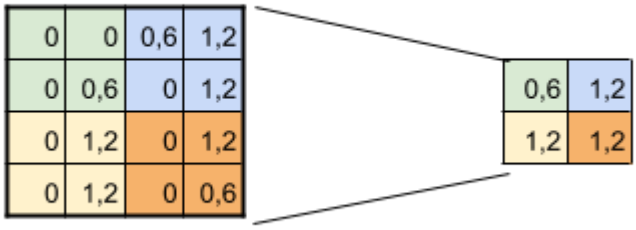

Figura 2-7: Ejemplo de MaxPooling2D

En el ejemplo anterior podemos observar cómo es el funcionamiento en 2D para un MaxPooling configurado para un filtro 2x2 y con un stride de 2.

2.5.2 Flatten

La capa flatten convierte los elementos de la matriz de imágenes en un array o vector de una sola dimensión. Esta capa suele usarse cuando ya no vamos a usar más capas de convolución o pooling y queremos acercarnos a las etapas finales del clasificador.

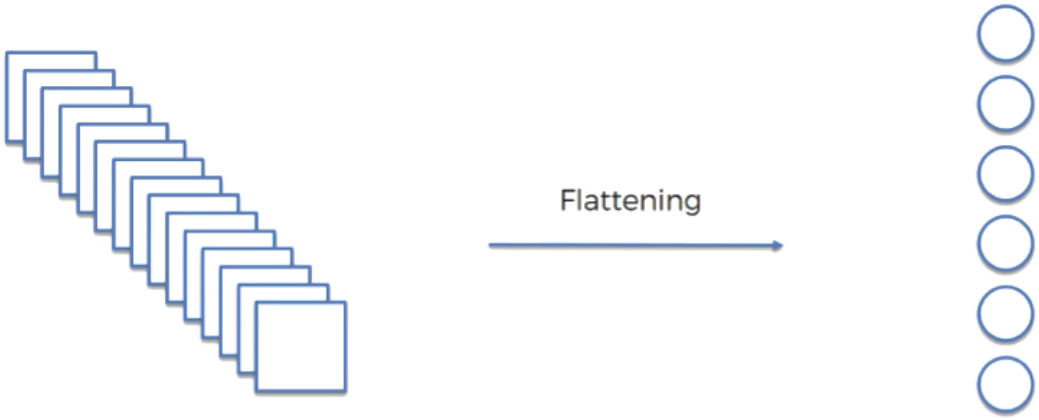

Figura 2-8: Esquema del funcionamiento de la capa Flatten[7]

2.5.3 Dense

Las capas densas son las capas más conocidas en Deep Learning, también se consideran la arquitectura más simple de una red neuronal, en estas capas las neuronas se conectan, por defecto, con todas las neuronas de la siguiente capa. De esta forma una neurona sabría todas las neuronas que se han activado en la capa anterior.

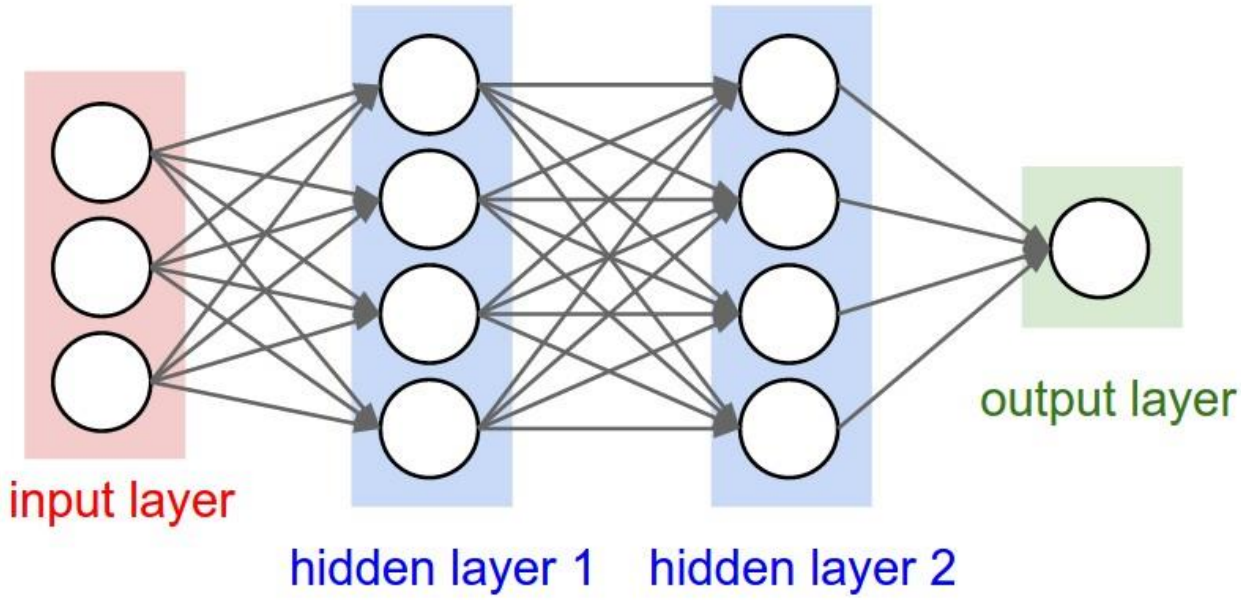

Figura 2-9: Ejemplo de una red neuronal que hace uso de capas densas[8]

La función que establece cuándo se activa una neurona, conocida como función de activación, es lo que usamos para propagar hacia delante la salida de una neurona. Esta función incluirá la no linealidad en el modelado de la red, es importante para que el modelo se puede ajustar al problema que le pidamos resolver.

Existe una gran variedad de funciones de activación, las más conocidas y las que vamos a usar en el trabajo son las ReLU (rectified linear unit). Esta función solo activará la salida de la neurona si la entrada está pro encima de cierto umbral. Concretamente cuando la entrada es negativa, la salida será cero, mientras que cuando pasa de cero sigue la función f(x) = x.

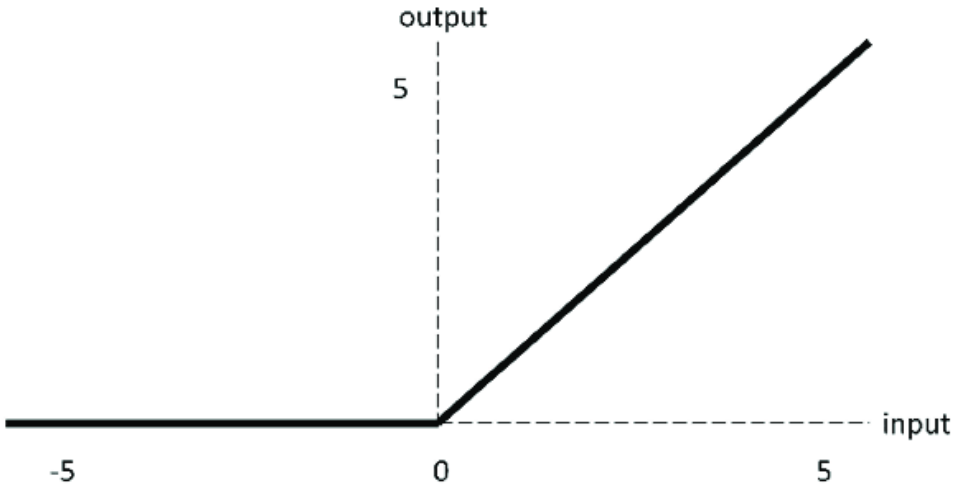

Figura 2-10: Representación de la función de activación ReLU[9]

La otra función de activación muy importante es la función de activación Softmax. Esta función genera a su salida un vector de probabilidades decimales, correspondiendo cada número del vector a una clase diferente. Estas probabilidades decimales si las sumamos deben sumar 1.

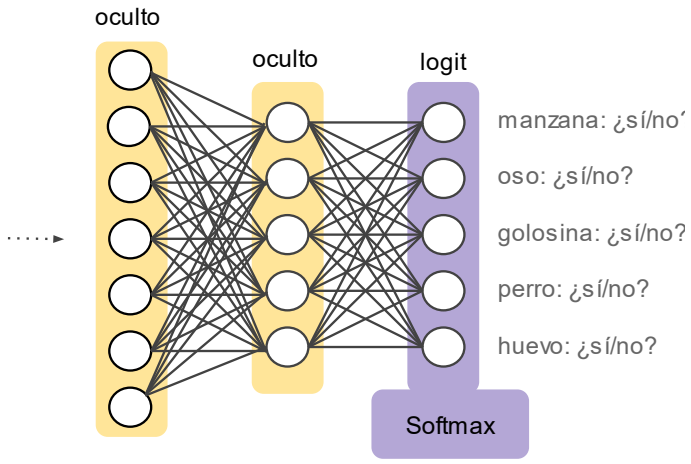

Figura 2-11: Ejemplo de uso de la función de activación softmax[10]

2.6 Datasets de acciones

Los datasets o bases de datos para la clasificación de imágenes han tenido un gran auge en los últimos años, especialmente con un gran número de concursos donde los participantes consiguen obtener el mejor porcentaje de acierto creando sus propios modelos.

Normalmente antes de comenzar el concurso se les aporta a los participantes una colección de clases e imágenes, que son los que generan esta base de datos sobre la que los participantes correrán sus modelos.

Algo más escaso son los datasets de vídeos para el entrenamiento de acciones, principalmente debido a que su peso suele ser mucho mayor que el de imágenes. A continuación, vamos a mostrar algunos de los datasets de videos que existen actualmente.

Dataset	Tipo	Peso	No. Videos	Clases	Año
HMDB51[11]	Acciones	2 GB	6.466	51	2011
UCF101[12]	Acciones	6.5 GB	1.320	101	2012
Sports-1M[13]	Deportes	-	1.100.000	487	2014
ActivityNet[14]	Acciones	-	28.000	203	2015
Kinetics[15]	Acciones	-	500.000	400	2017
Youtube-8M[16]	General	-	8.000.000	4716	2016

Tabla 2-2: Datasets de clasificación de videos

En la tabla anterior podemos comprobar que muchos de ellos no tienen dato conocido para el peso del mismo. Esto se debe a que estos datasets en realidad son enlaces a vídeos de YouTube dónde normalmente también te indican el comienzo y el fin del vídeo que se querrá clasificar.

Esto se suele aportar de esta forma porque el descargar esta cantidad de videos supondrían mucha capacidad de almacenamiento, por eso se suele entrenar con parte de estos datasets y no se emplean al completo de forma amateur.

2.7 Redes neuronales para clasificación de videos

Como hemos visto anteriormente el tratamiento la clasificación de vídeos siempre se realiza teniendo en cuenta que un vídeo nos más que una sucesión de imágenes.

Es por esto por lo que hay dos grandes vertientes a la hora de intentar clasificar un dataset de videos. Clasificando cada uno de los frames del propio video y, por tanto, la entrada al modelo que queremos entrenar sería de 4 dimensiones, como hemos explicado anteriormente, o clasificando el video completo como una sucesión donde todas las imágenes de cada video sólo ejemplifican una acción concreta.

Concretamente durante este trabajo nos vamos a centrar en los modelos que usan las convoluciones 3D, aunque posteriormente no sea para abarcar todas sus dimensiones con los filtros. De esta forma en todos los modelos, todos los frames serán tratados como de un mismo ejemplo.

2.7.1 Modelos propuestos

De forma resumida vamos a ver algunos de los modelos que se han propuesto durante este tiempo para la creación de diferentes clasificadores de acciones humanas.

Existen una multitud de modelos creados para la clasificación de vídeos, a continuación, vamos a ver los grandes grupos de modelos

2.7.1.1 C3D

En estos modelos trataremos la entrada como un volumen de datos de 5 dimensiones como hemos citado anteriormente.

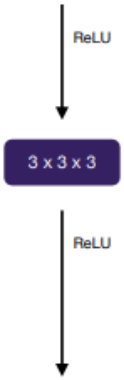

Figura 2-12: Esquema de una unidad para los clasificadores C3D[17]

El modelo por tanto constará de simples capas de convolución de 3 dimensiones, donde los filtros por tanto tendrán 3 dimensiones. En las imágenes se omite la dimensión de los canales para tener una visión más clara.

2.7.1.2 (2+1)D

En este caso vamos a usar una convolución de 3 dimensiones, pero el filtro no actuará en todas ellas. Primero tendremos un filtro que actuará sobre las dimensiones espaciales de altura y anchura y posteriormente tendremos un filtro que solo actuará en la dimensión temporal que indica los frames[18].

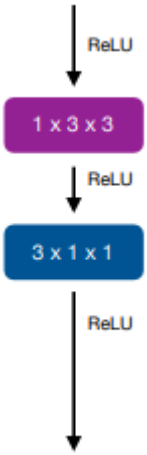

Figura 2-13: Esquema de una unidad para los clasificadores C(2+1)D

Con este modelo la cantidad de parámetro normalmente se verá reducida respecto a los modelos de C3D. Sin embargo, necesitaremos más memorias pues ahora tendremos el doble de convoluciones.

2.7.1.3 P3D

El nombre proviene de pseudo convoluciones 3D, son variaciones de nuevo del modelo de convoluciones 3D. De hecho, tenemos 3 tipos de pseudo convoluciones y los modelos P3D-A[19] son muy similares al modelo de (2+1)D.

Los modelos P3D-B son parecidos al (2+1)D, pero en este caso los filtros espaciales y temporales se hacen en paralelo.

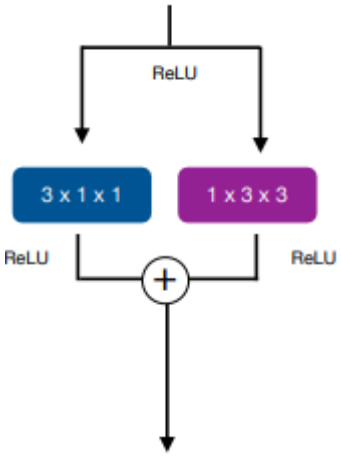

Figura 2-14: Esquema de un modelo P3D-B

La principal ventaja es que la arquitectura es más diversa. De esta forma la red se centrará por separado en obtener la información de cada dimensión por separado.

En el caso de los P3D-C son un camino intermedio entre los dos casos anteriores.

2.7.1.4 FAST3D

El modelo de FAST3D es una mezcla de tres convoluciones[20]. Al contrario que en (2+1)D tras una convolución espacial, no vamos a pasar los datos por un filtro temporal, si no que usaremos dos convoluciones espacio-temporales que podemos llamar XT e YT.

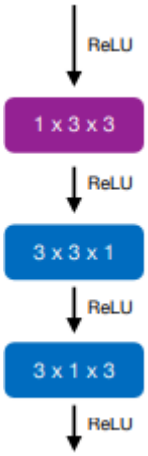

Figura 2-15: Esquema de un modelo FAST3D

Esto nos permite enfocarnos y localizar patrones de movimientos horizontales y verticales. Como los filtros son complementarios también podremos detectar acciones complejas que pueden ser modeladas con el uso de los dos filtros juntos.

2.7.2 Resultados de estos modelos

En la tabla que nos encontramos a continuación podemos ver una comparación del comportamiento de los modelos. En este caso han sido entrenados con el dataset Sport-1M y se han probado con el dataset HMDB51 que contiene 51 clases diferentes y UCF101 de 101 clases. No se han podido conseguir todos los datos porque no todos los reportes incluían estos datos.

Modelo	Acierto	Capas	Parámetros	Acierto HMDB51
C3D	81,14	137	40.600.000	43,9
(2+1)D	81,75	147	25.840.000	45,8
P3D	82,8	152	-	45,1
FAST3D	83,82	157	43.480.000	45,9
I3D	84,5	192	91.704.000	49,8

Tabla 2-3~~2 3~~: Comparativa de los modelos existentes

3 IMPLEMENTACIÓN DEL TRANSFER LEARNING

Habiendo presentado ya las ventajas de las técnicas de transfer learning, parece lógico que su uso será indispensable para el fin del trabajo.

Como ya avanzamos anteriormente aun haciendo uso del transfer learning, la capacidad de computación necesaria para entrenar los modelos con todos los ejemplos y todas las clases de los datasets es demasiado alta. Para el trabajo hemos seleccionado dos modelos diferentes, pero ambos, para ser comparados.

3.1 Modelos seleccionados

Hay modelos donde el programa lo primero que se encarga es de detectar la persona y se centra en lo que ocurre junto a ella. En nuestro caso, para darle mayor simplicidad a los modelos trataremos los videos al completo.

3.1.1 C3D

Las convoluciones 3D son más eficientes detectando patrones y propiedades espacio temporales que las convoluciones 2D. Este es un modelo simple, pero bastante efectivo.

En el 2D ya estaba bastante extendido como un estándar tras muchas pruebas que el mejor tamaño para la clasificación de imágenes era 3x3, del mismo modo, se ha obtenido los mejores resultados con capas convolucionales de filtro 3x3x3 para el caso de las 3 dimensiones.

Layer (type)	Output Shape	Param #
conv1 (Conv3D)	(None, 16, 112, 112, 64)	5248
pool1 (MaxPooling3D)	(None, 16, 56, 56, 64)	0
conv2 (Conv3D)	(None, 16, 56, 56, 128)	221312
pool2 (MaxPooling3D)	(None, 8, 28, 28, 128)	0
conv3a (Conv3D)	(None, 8, 28, 28, 256)	884992
conv3b (Conv3D)	(None, 8, 28, 28, 256)	1769728
pool3 (MaxPooling3D)	(None, 4, 14, 14, 256)	0
conv4a (Conv3D)	(None, 4, 14, 14, 512)	3539456
conv4b (Conv3D)	(None, 4, 14, 14, 512)	7078400
pool4 (MaxPooling3D)	(None, 2, 7, 7, 512)	0
conv5a (Conv3D)	(None, 2, 7, 7, 512)	7078400
conv5b (Conv3D)	(None, 2, 7, 7, 512)	7078400
zeropad5 (ZeroPadding3D)	(None, 2, 8, 8, 512)	0
pool5 (MaxPooling3D)	(None, 1, 4, 4, 512)	0
flatten_1 (Flatten)	(None, 8192)	0
fc6 (Dense)	(None, 4096)	33558528
dropout_2 (Dropout)	(None, 4096)	0
fc7 (Dense)	(None, 4096)	16781312
dropout_3 (Dropout)	(None, 4096)	0
fc8 (Dense)	(None, 487)	1995239

Total params: 79,991,015

Figura 3-1: Arquitectura del modelo C3D

Tras cada capa de convolución tendremos una capa de MaxPooling para reducir el número de parámetros, excepto en las capas 3,4 y 5 donde tendremos dos capas convolucionales y posteriormente la capa de MaxPooling.

Los MaxPooling son de 2x2x2, excepto el primero que es de 1x2x2 con la idea de no unificar el tiempo demasiado pronto.

La entrada para este modelo deberá ser de videos RGB, es decir con 3 canales, y con una altura y anchura de 112x122 píxeles.

3.1.2 Inflated Inception 3D

Este modelo, también llamado por las siglas I3D, está basado en el modelo diseñado para capas convolucionales de 2 dimensiones presentado para los concursos de ImageNet, en particular se seguirá la estructura del modelo conocido como Inception v1 [21].

La estructura de esta arquitectura es más compleja y por tanto necesita más capacidad de computación, pero el rendimiento es mejor en cuanto al acierto a la hora de clasificar los datos de entrada.

El principal problema que nos encontramos con esta estructura es que está pensada para el concurso de ImageNet, es decir, que se usa para clasificación de imágenes, es decir de matrices con una dimensión menor a la que nosotros deseamos. Por esto se hace necesario "inflar" el modelo.

La idea es usar el modelo pre entrenado para 2D, el primer problema que nos encontramos es que tenemos filtros cuadrados de NxN y para usarlos en un modelo 3D los vamos a hacer cúbicos de NxNxN. Otro tema por solucionar es que nosotros necesitamos un cierto número de frames porque trabajamos con video y no con imágenes, para ello lo que se hace es crear algo llamado "boring video" que en español podría traducirse como "video aburrido", donde lo que hacemos es duplicar la imagen el número necesario de veces para tener el número de frames requeridos. Pero de esta forma seguiríamos sin poder usar los pesos ya calculados para el modelo, para hacerlo lo que hay que hacer es repetir los pesos de cada capa N veces, pero reescalándolos todos dividiéndolos por N, de esta forma.

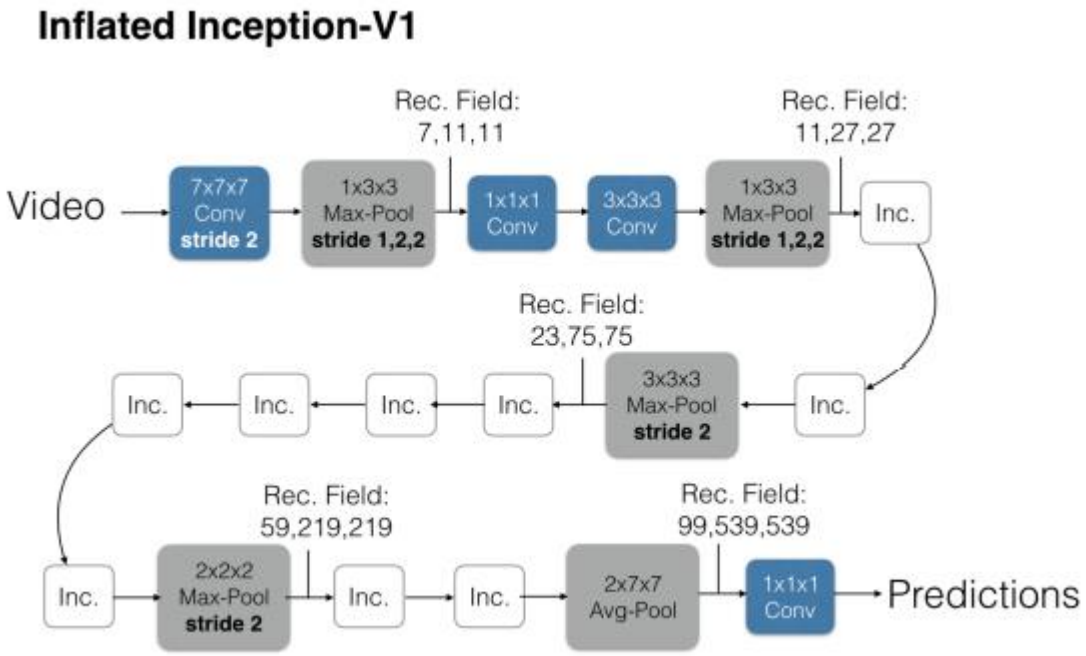

Figura 3-2: Esquema general de la arquitectura Inflated Inception[22]

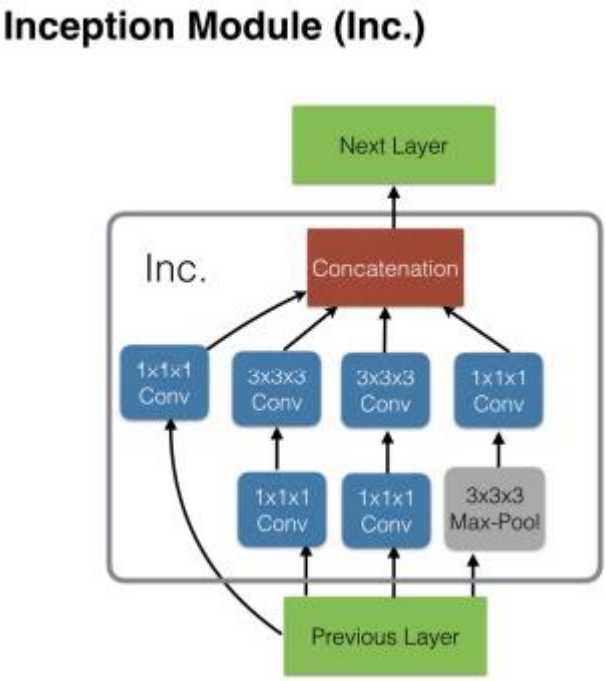

Figura 3-3:Subesquema de los bloques Inc. de la arquitectura

Se van a usar filtros en su mayoría de 1x1x1 o 3x3x3 para evitar que a la hora de analizar localmente parte de las imágenes tengamos problemas de alineamiento. Las capas de convolución de 1x1x1 se usan para reducir la capacidad de computación necesaria antes de hacer las convoluciones de 3x3x3 que son más costosas. Además, también tienen el propósito de hacer más lineal la activación de las neuronas.

Usaremos también MaxPooling para reducir el tamaño de la red neuronal para que la capacidad de computación necesaria no se incremente hasta unos números inabarcables. El stride de estos pooling serán de 2, exceptos en las primeras capas en las que para no perder los datos de la dimensión temporal los filtros serán de 1x3x3 y el stride de 1x2x2.

Antes de pasar la arquitectura por las dos capas fully connected finales, tendremos otra capa de pooling, pero esta vez será una de 2x7x7 y AvgPooling.

Tras cargar este modelo pre entrenado inflado de 2 dimensiones a 3 dimensiones, el modelo que vamos a usar fue entrenado con el datasets de vídeos de acciones humanas llamado Kinetics400.

La entrada para este modelo serán 16 frames de 224x224 en formato RGB, es decir, con los tres canales. Además, se normalizarán los frames de 0 a 1 dividiendo el valor de cada píxel entre 255.

A pesar de todas las reducciones, para el entrenamiento de este modelo fueron necesarias 32 GPUs para la mayoría de del modelo, excepto para las convoluciones de 3x3x3 que fueron necesarias 64 GPUs. Las GPU usadas para este entrenamiento fueron las Tesla K40 de Nvidia.

3.2 Selección de capas entrenables

3.2.1 C3D

En primera instancia tras cargar el modelo pre entrenado[23], para ello tendremos que cargar un fichero de extensión h5 que incluirá los pesos del modelo, la arquitectura debemos haberla generado nosotros antes con las indicaciones del autor de este modelo pre entrenado.

En este caso el modelo fue entrenado con el dataset Sport-1M, sólo le agregaremos una capa densa con activación softmax que nos permitirá tener una salida con un vector del mismo tamaño que el número de clases que queremos clasificar y lo reentrenaremos usando el dataset UCF101.

Durante el transfer learning, se procederá a entrenar sólo la capa que hemos agregado y dejar las capas del

modelo pre entrenado congeladas. Este entrenamiento lo haremos de 20 épocas. En este caso como los datos de entrada no van a estar normalizados, vamos a usar la técnica de gradiente descendente con un learning rate alto de 0,1.

Una vez ya entrenado el modelo, procederemos a descongelar las tres últimas capas densas del modelo, y realizaremos un entrenamiento más rápido de 5 épocas. A este procedimiento también se le conoce con el nombre de fine tuning.

3.2.2 I3D

Ahora la carga la haremos a través de la biblioteca de Tensorflow Hub[24], haciendo uso de las instrucciones que nos aporta esta biblioteca, podremos obtener el modelo con los pesos cargados. Este modelo fue entrenado con el dataset Kinetics400 y lo reentrenaremos usando el dataset UCF101.

En este caso le vamos a agregar al final del modelo una capa flatten y posteriormente una capa densa del tamaño del número de clases que vamos a clasificar. El entrenamiento lo volveremos a hacer con la técnica de gradiente descendente, pero esta vez, como los datos de entrada sí están normalizados el learning rate que hemos usado es de 10^{-4} con una duración de 20 epochs para compararlo con el anterior modelo.

Para este modelo, como los datos de entrada son de 224x224, siguen siendo RGB, y la arquitectura es más compleja, con los recursos que tenemos no hemos podido realizarle fine tuning.

4 RESULTADOS

4.1 Resultados con clases UCF101

Con el límite de la capacidad de computación necesaria marcado por el modelo I3D, finalmente decidimos hacer el entrenamiento con 3 clases diferentes. Concretamente las clases elegidas son las que muestran videos de un bebe gateando, patinaje artístico sobre hielo y de skateboard. Aunque en nuestro caso estas clases fueron seleccionadas al azar, es posible que para algún caso solo queramos elegir ciertas clases para detectar un contenido en concreto.

A continuación, vamos a mostrar algunas imágenes de videos aleatorios de cada clase para ejemplificar.

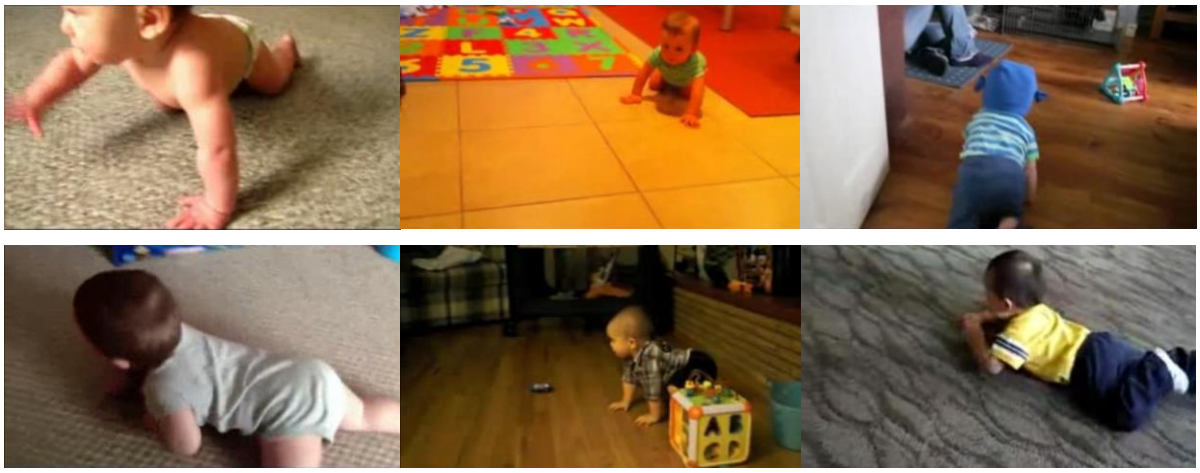

Figura 4-1: Muestras de la clase de bebés gateando

Figura 4-2: Muestras de la clase de patinaje artístico sobre hielo

Figura 4-3: Muestras de la clase de skateboard

Para comparar los dos modelos los hemos entrenado durante 20 epochs. Los learning rate, debido a que en el caso del C3D no están normalizados, son diferentes, 0.1 para el caso del C3D y 10^{-4} para el caso del I3D.

Los videos se han divido entre videos de entrenamiento y evaluación y de los videos de validación, hemos cogido el 20% para usarlos como datos de validación.

Para evitar problemas a la hora de entrenar y de comparar los datos de validación, una vez hemos leído y procesado todos los datos de entrada, se ha procedido a realizar una mezcla de los datos usando el comando shuffle de la librería Sklearn que nos permite mezclar los datos de forma totalmente aleatoria. Para no equivocar al clasificador durante el entrenamiento y la validación, se realiza exactamente la misma mezcla a los datos que a las etiquetas, que nos dicen a qué clase pertenecen.

Otro dato a tener en cuenta es que los datos de entrada para el caso del I3D son de 224x224 píxeles, mientras que para el caso del C3D es de 112x112px.

Durante el entrenamiento, mantendremos el modelo base congelado, para que sus pesos no se vean modificados durante el entrenamiento. Así evitaremos que el entrenamiento tenga que realizar el cálculo de todos estos pesos.

Al modelo base le vamos a agregar una capa densa con activación softmax, esta capa tendrá un tamaño igual al número de clases, en nuestro caso 3. Así la salida que obtendremos al predecir la clase a la que pertenece un video, obtendremos un vector con tres valores del 0 al 1 que representa la probabilidad de que el vídeo pertenezca a cada una de las clases.

En nuestro caso mostraremos los tres porcentajes y qué clase pertenece cada uno y diremos que la clase a la que pertenece el video es a la clase que tenga un mayor porcentaje.

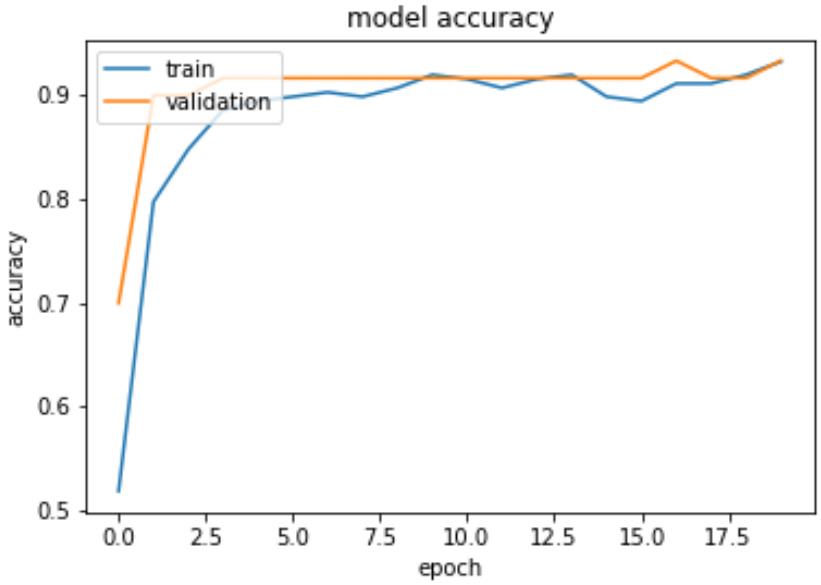

Figura 4-4: Porcentaje de acierto de los datos de entrenamiento y validación para el modelo C3D

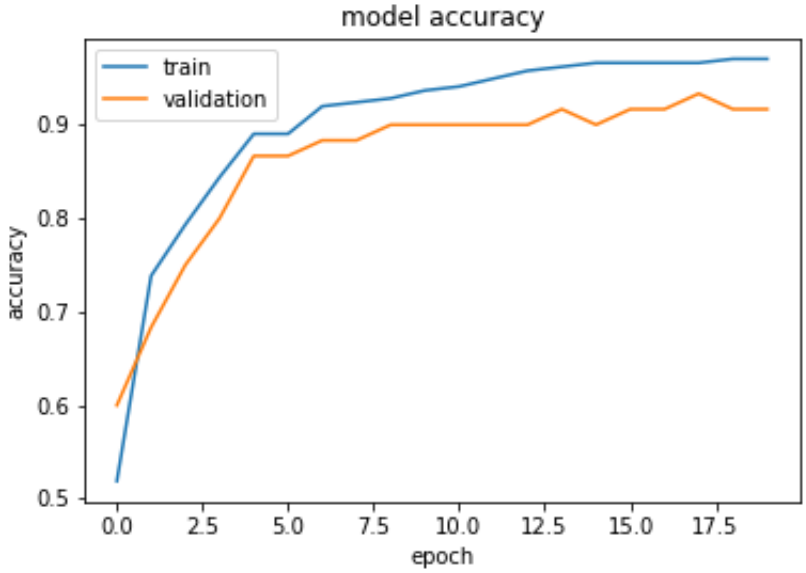

Figura 4-5: Porcentaje de acierto de los datos de entrenamiento y validación para el modelo I3D

Podemos observar que al comienzo en ambos casos el acierto de los casos de entrenamiento está entorno al 50%, aunque el de los de validación en el caso del C3D comienza un poco mejor. A pesar de que gracias al alto learning rate del C3D llega a un alto porcentaje de acierto más rápido y aunque acaba un poco menos estable, acaba con un valor de validación mayor que el de I3D. A pesar de ello el I3D también consigue un buen resultado con los datos de validación.

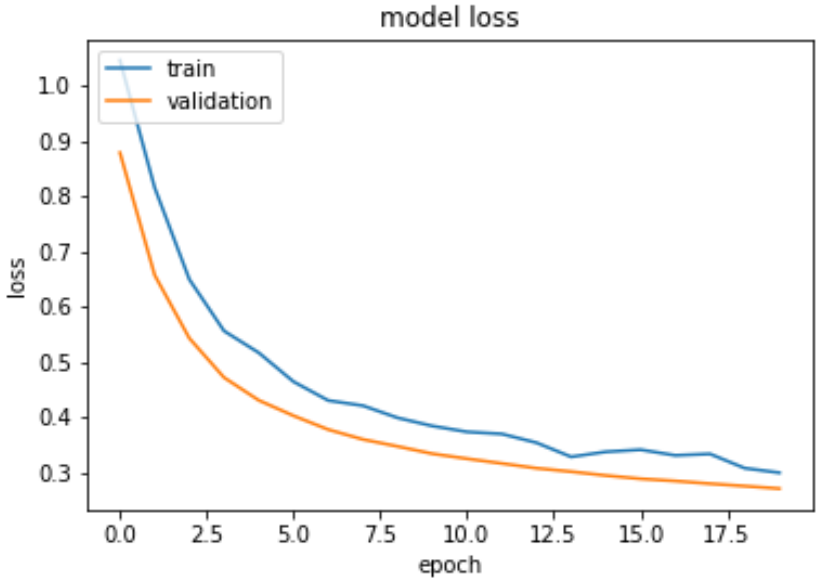

Figura 4-6: Pérdidas del modelo para C3D

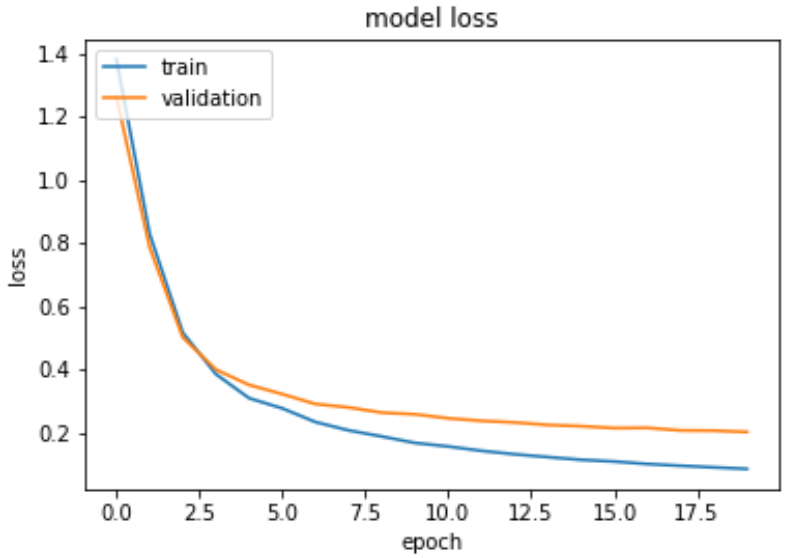

Figura 4-7: Pérdidas del modelo para I3D

Comparando las gráficas podemos ver que, aunque las pérdidas al inicio son menores en el caso de C3D, es el modelo I3D el que más rápido y mejor consigue reducir las pérdidas.

El entrenamiento y validación nos dejan tres datos más. Para el caso C3D.

Test loss	0,4569634795188904
Test accuracy	0,9692307710647583
Time	9387,860940694809

Tabla 4-1: Resultado para el modelo C3D

Mientras que para I3D.

Test loss	0,10128648579120636
Test accuracy	0,9646017551422119
Time	11914,16603398323

Tabla 4-2: Resultado para el modelo I3D

Confirma que, aunque tenemos un porcentaje muy similar en ambos casos, las pérdidas resultan ser mayores en el caso del modelo C3D. Además, vemos que debido a la cantidad de datos de entrada y la complejidad del modelo, el modelo I3D es más lento de entrenar.

Ahora vamos a ver el reporte que obtenemos de los modelos cuando los analizamos con los datos de evaluación.

	Precision	**Recall**	**F1-score**	**Support**
BabyCrawling	0,83	1,00	0,91	35
IceDancing	1,00	0,91	0,95	46
SkateBoarding	1,00	0,81	0,90	32

Tabla 4-3: Tabla de resultados para C3D

	Precision	**Recall**	**F1-score**	**Support**
BabyCrawling	0,95	1,00	0,97	35
IceDancing	0,96	1,00	0,98	46
SkateBoarding	1,00	0,81	0,90	32

Tabla 4-4: Tabla de resultados para I3D

Tal y como vimos con los datos de validación podemos observar una pequeña ventaja para el modelo I3D.

Como la arquitectura del modelo C3D nos permite exprimir el modelo un poco más debido a su mayor simpleza, y a unos datos de entrada menos pesados, vamos a aplicarle al modelo la técnica de fine tuning.

La técnica conocida como fine tuning consiste en, tras entrenar el modelo que sólo tendrá descongelada la última capa densa con activación softmax, volver a realizar otro entrenamiento más corto descongelando las últimas capas del modelo base.

Como el modelo base ya estaba pre entrenado y la capa con softmax ya está también entrenada, necesitaremos menos epochs para acabar de afinar el modelo. También es buena idea usar pocas epochs porque ahora vamos a entrenar muchos más parámetros que antes y el modelo tendrá que calcular los nuevos pesos para estos parámetros. Concretamente vamos a entrenar ahora durante 5 epochs más.

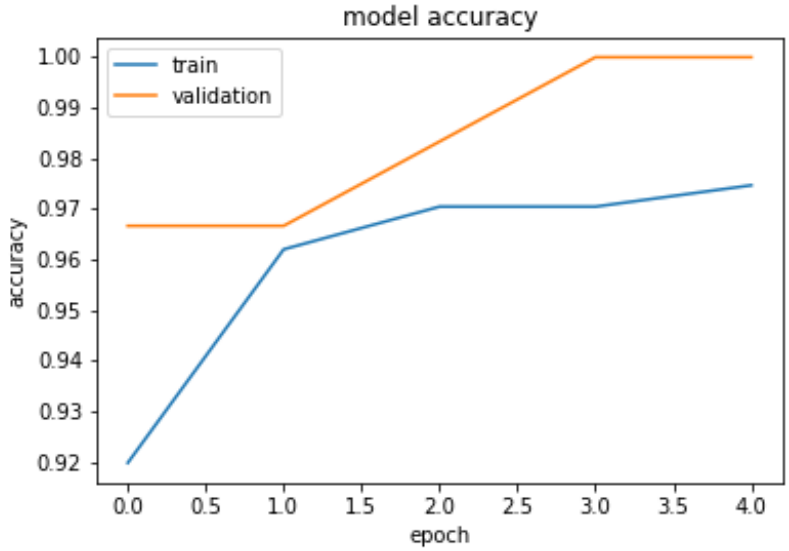

Figura 4-8: Porcentaje de acierto del modelo C3D tras el fine tuning

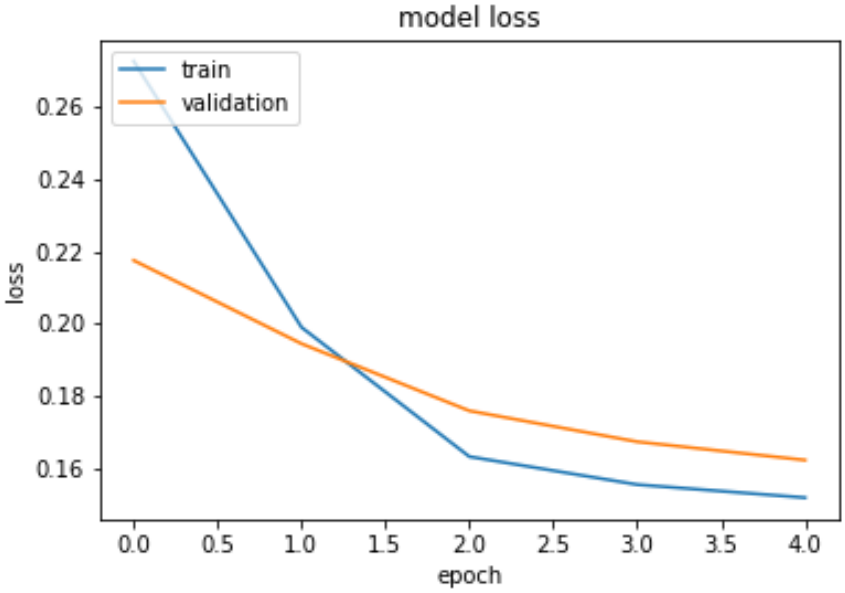

Figura 4-9: Pérdidas del modelo para C3D tras fine tuning

Test loss	0,15761710703372955
Test accuracy	0,9734513163566589
Time	2323,6329607963562

Tabla 4-5: Resultados del modelo C3D tras fine tuning

Podemos observar que, aunque la mejora en el porcentaje de acierto no ha sido muy grande, en parte también porque estamos en un porcentaje de acierto ya muy alto. El fine tuning ha conseguido reducir de forma más que evidente los datos de error del modelo.

Como hemos entrenado con un cuarto de los epochs del entrenamiento inicial que hicimos con el transfer learning, el tiempo que hemos tardado ahora es en torno al cuarto de tiempo también.

Por tanto, cuando ya veamos que estamos satisfechos con el porcentaje de acierto de nuestro modelo y veamos que el error del modelo ya apenas consigue reducirse, tendremos que pensar en aplicarle fine tuning al modelo para conseguir terminar de afinar nuestro modelo.

4.2 Resultado con una clase propia

Con motivo de no hacer uso solo de la base de datos del UCF101 y de explorar las posibilidades de los clasificadores. Vamos a introducir en el modelo C3D una nueva clase a entrenar además de las tres anteriores que ya usamos.

Esta clase será de carreras de coches, los videos que vamos a emplear para entrenar con esta clase son videos obtenidos de la plataforma de videos de YouTube.

Las siguientes imágenes son frames de alguno de los videos con los que hemos entrenado, o evaluado la clasificación del modelo.

Figura 4-10: Muestras de la clase de carreras de coches

Vamos a usar un total de 232 videos para esta clase. Un total de 160 videos aleatorios serán seleccionados para el entrenamiento y validación, de estos el 20% serán usados para validación y el 80% restante para el entrenamiento del modelo. Los otros 72 videos serán empleados para evaluación del modelo entrenado.

De nuevo, los videos serán de 112x112px con los tres canales RGB y cogeremos 16 frames centrales de los videos.

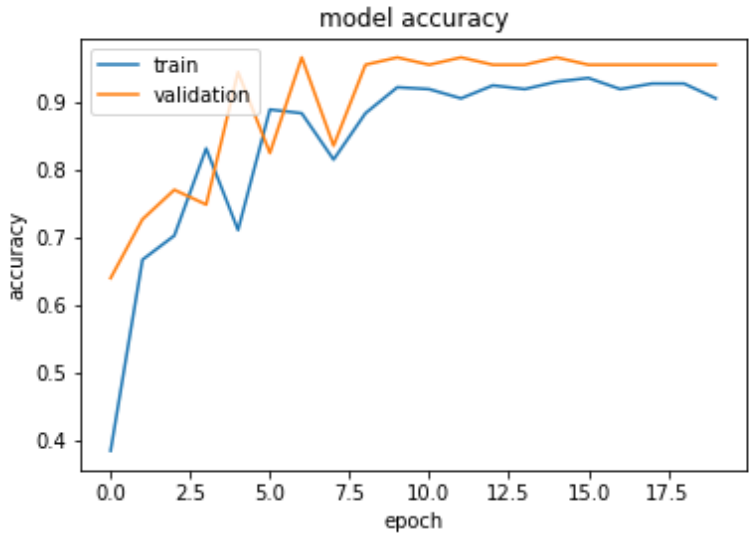

Figura 4-11: Porcentaje de acierto del modelo C3D haciendo uso de una clase custom

Si atendemos al porcentaje de acierto vemos que hemos acabado tras las 20 epochs en un buen nivel. Vemos que durante el entrenamiento tenemos unos picos de bajada y subida, esto se debe a que el learning rate que estamos usando es alto, pero es necesario para este modelo, ya que, al no estar normalizado, con un learning rate más bajo no conseguiría aprender a una velocidad necesaria para nosotros. Es cierto que la línea del acierto en el entrenamiento parece errática hacia el final porque baja, pero es normal porque ya estamos en unos niveles altos del porcentaje de acierto.

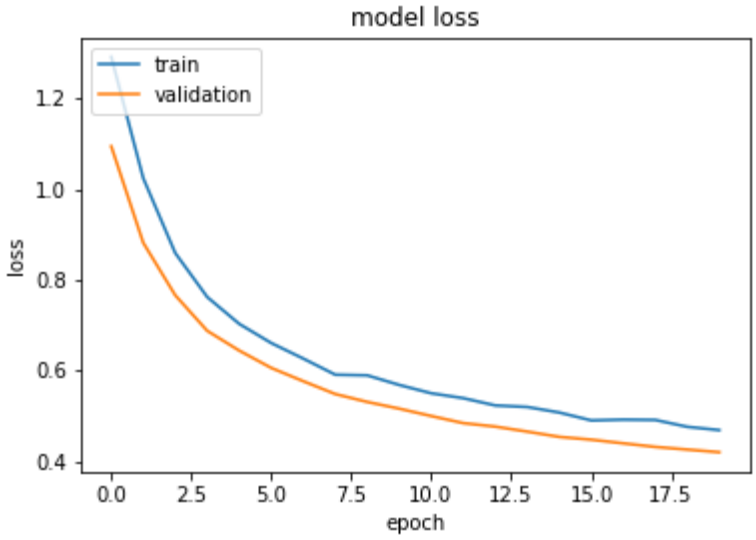

Figura 4-12: Pérdidas del modelo C3D con una clase custom

En el caso de la gráfica de las pérdidas del modelo podemos ver cómo desciende correctamente durante todo el entrenamiento y acaba en unos niveles un poco superior al anterior caso de C3D, pero teniendo en cuenta que ahora estamos entrenando 4 clases es un resultado esperable.

	Precision	Recall	F1-score	Support
BabyCrawling	0,82	1,00	0,86	35
CarRacing	1,00	0,96	0,98	72
IceDancing	1,00	0,93	0,97	46
SkateBoarding	1,00	0,84	0,92	32

Tabla 4-6: Resultados de clasificación del modelo C3D con categoría custom

Test loss	0,43553152680397034
Test accuracy	0,9243243336677551
Time	11880,16603398323

Tabla 4-7: Datos del entrenamiento del modelo C3D con categoría custom

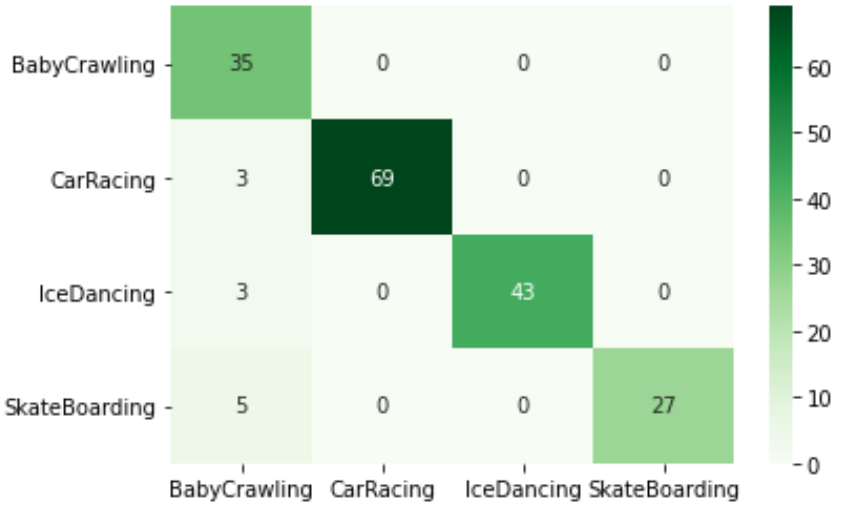

Figura 4-13: Matriz de confusión al entrenar con transfer learning

Observamos que, aunque hemos conseguido de nuevo un alto porcentaje de acierto, los errores del modelo siguen siendo un poco grande. Por eso vamos a realizarle fine tuning al modelo para intentar mejorar en este aspecto.

En esta ocasión en vez de realizar 5 epochs de fine tuning, como el error es alto y el modelo más complejo, vamos a hacer 10 epochs de fine tuning en su lugar.

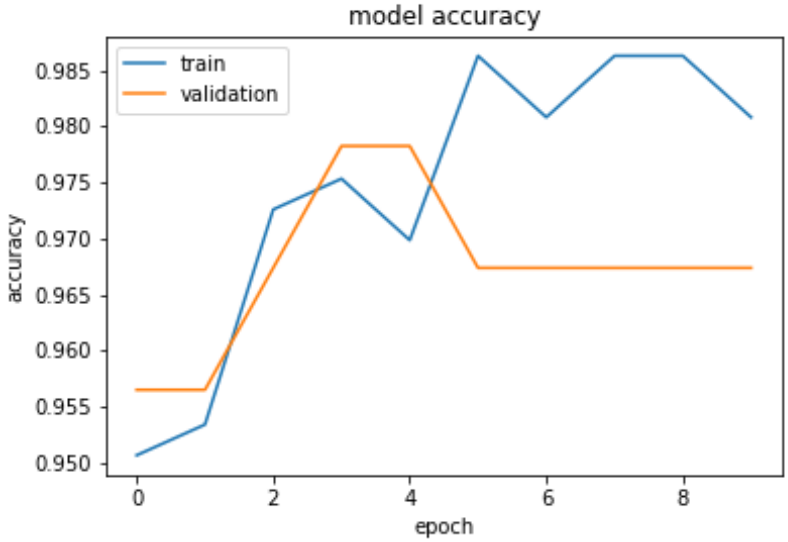

Figura 4-14: Porcentaje de acierto del modelo C3D tras hacer fine tuning

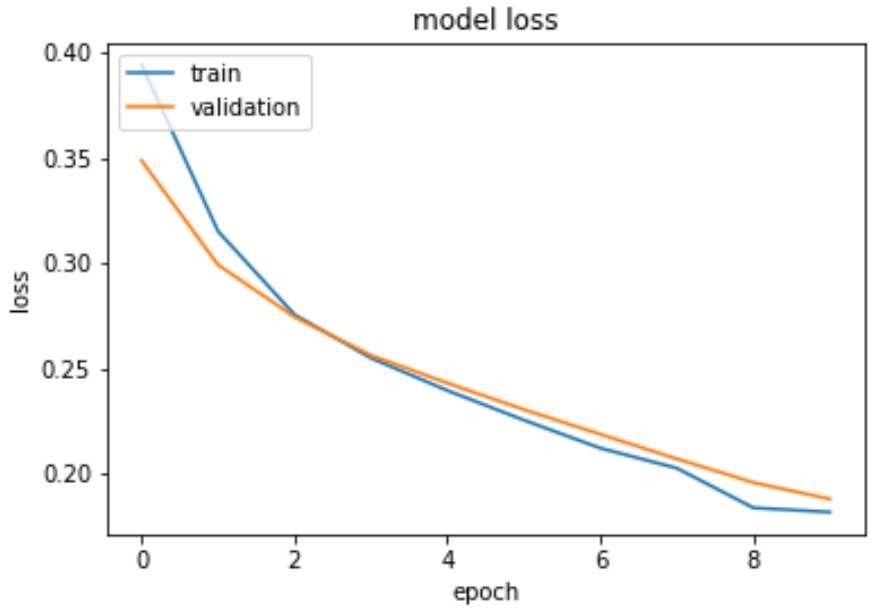

Figura 4-15: Pérdidas del modelo C3D tras realizar fine tuning

El porcentaje de acierto como está ya aun nivel bastante alto sube de forma errática. Sin embargo, la gráfica de las pérdidas del modelo nos muestra cómo va mejorando con el entrenamiento del mismo. Vamos a fijarnos en el resto de los resultados para ver si esta mejoría se ve reflejada.

	Precision	**Recall**	**F1-score**	**Support**
BabyCrawling	0,95	1,00	0,97	35
CarRacing	1,00	0,99	0,99	72
IceDancing	1,00	1,00	1,00	46
SkateBoarding	1,00	0,97	0,98	32

Tabla 4-8: Resultados tras realizar fine tuning

Test loss	0,18726404011249542
Test accuracy	0, 9891892075538635
Time	5937, 456963479518

Tabla 4-9: Datos del entrenamiento con fine tuning

Observamos que la eficiencia del modelo clasificando las cuatro clases, incluida la que ha sido generada por nosotros, ha aumentado de forma notable.

En este caso vamos a mostrar la matriz de confusión para ver gráficamente el comportamiento del clasificador con los datos que hemos seleccionado de evaluación.

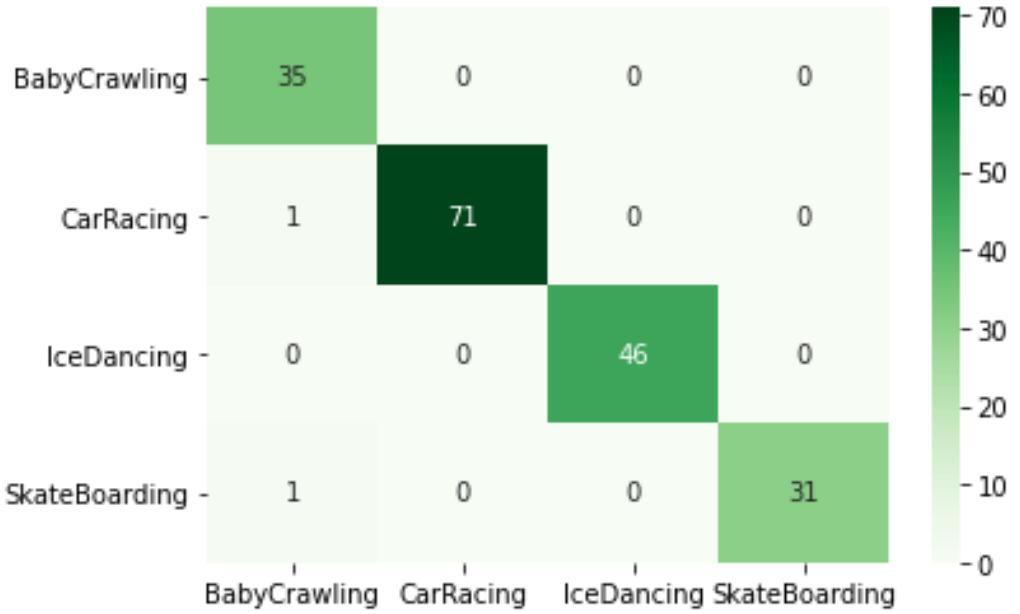

Figura 4-16: Matriz de confusión, calculada con datos de evaluación, tras realizar fine tuning

Tal y como reflejaban los resultados anteriores, podemos observar en la matriz de confusión que el clasificador está trabajando correctamente y solo dos videos de entrada de la colección de videos de evaluación han sido clasificados de forma errónea.

En la matriz de confusión en las filas tenemos las clases reales de cada video y en las columnas tenemos la clase que le asignó el clasificador. Por tanto, hay un video de Skateboard y otro de carrera de coches, que han sido clasificados erróneamente en la clase de un bebé gateando.

5 CONCLUSIONES Y LÍNEAS FUTURAS

La principal conclusión que podemos sacar tras realizar este trabajo es que por muy sencillo que vaya a ser nuestro clasificador de video, es indispensable partir desde un modelo ya pre entrenado haciendo uso de las técnicas de transfer learning, no solo por ahorrar capacidad de procesamiento, sino también en la reducción del tiempo necesario para entrenar el modelo hasta un porcentaje de acierto aceptable.

Atendiendo a los resultados que hemos obtenido con los dos modelos ya pre entrenados, podemos usar uno de los dos modelos atendiendo a nuestro fin y nuestros recursos. Si no disponemos de una gran potencia en cuanto a ordenador o recursos en la nube y podemos permitirnos cierto margen de fallo, una gran opción es usar el modelo pre entrenado con las clases de Sport1M de C3D, ya que cuando entrenemos con este modelo no vamos a necesitar normalizar la entrada, lo cual requiere bastante potencia, y además hace uso de unos datos de entrada la mitad de grande.

Sin embargo, si queremos tener un margen de fallo más pequeño y disponemos de una buena potencia de cálculo, ya sea física o en la nube, usaremos el modelo de I3D, este modelo fue pre entrenado con las clases de Kinetics400. La necesidad de una mayor capacidad de computación viene de que en este caso hacemos uso de datos de entrada de 224x224, el doble que el modelo anterior, y además normalizamos los valores de cada píxel de entrada.

Para dar continuidad a este trabajo podría realizarse un estudio de la plataforma AWS de Amazon, donde podremos alquilar servidores en la nube mucho más potentes a cambio de una tarifa por cada hora de procesamiento. Esta plataforma es un entorno nuevo y por este motivo junto al ser una plataforma de pago no ha sido incluido en el trabajo.

Sería también interesante no solo buscar más modelos pre entrenados para compararlos con los dos más usados que hemos encontrado, si no investigar el uso de redes RNN pre entrenadas para hacer clasificadores. El hecho de ser redes con cierta memoria, si le pasamos frame a frame quizá consigamos unos buenos rendimientos. En el trabajo como nos hemos centrado en el uso de videos como datos de entrada, en vez de tener entradas frame a frame, no se ha incluido en el mismo.

Otra línea por donde podríamos expandir el alcance de este trabajo sería en el entrenamiento de un clasificador que detecte acciones gore, violentas y demás contenido sensible, de esta forma una plataforma de series y películas en streaming, si hemos configurado el control parental, podría censurar las escenas donde detecte este tipo de contenido.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

REFERENCIAS

[1] D. Alvarez *et al.*, "Usefulness of Artificial Neural Networks in the Diagnosis and Treatment of Sleep Apnea-Hypopnea Syndrome," in *Sleep Apnea - Recent Updates*, InTech, 2017.

[2] Standford University, "Neural Network model." [Online]. Available: http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/.

[3] Gimp, "Convolution Matrix image." [Online]. Available: https://docs.gimp.org/2.8/en/plug-in-convmatrix.html.

[4] A. Karpathy, "Recurrent Neural Networks image," 2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[5] D. M. Hawkins, "The Problem of Overfitting," *Journal of Chemical Information and Computer Sciences*, vol. 44, no. 1. American Chemical Society , pp. 1–12, Jan-2004.

[6] B. Kunlun, "3D filter matrix image," 2019. [Online]. Available: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215.

[7] SuperDataScience, "Flatten layer image," 2018. [Online]. Available: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening.

[8] cs231n, "Fully-connected layer image." [Online]. Available: https://cs231n.github.io/neural-networks-1/.

[9] H. H. Sultan, N. M. Salem, and W. Al-Atabany, "Multi-Classification of Brain Tumor Images Using Deep Neural Network," *IEEE Access*, vol. 7, pp. 69215–69225, 2019.

[10] Google, "Softmax activation image," 2020. [Online]. Available: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax?hl=es-419.

[11] S. Lab, "HMDB51 dataset." [Online]. Available: https://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/.

[12] M. S. Khurram Soomro, Amir Roshan Zamir, "UCF101 dataset," 2012. [Online]. Available: https://www.crcv.ucf.edu/data/UCF101.php.

[13] L. F.-F. Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, "Sport-1M dataset," 2014. [Online]. Available: https://cs.stanford.edu/people/karpathy/deepvideo/.

[14] B. G. Fabian Caba Heilbron Victor Escorcia and J. C. Niebles, "ActivityNet: A Large-Scale Video Benchmark for Human Activity Understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 961–970.

[15] A. Z. Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, "Kinetics 400," 2017. [Online]. Available: https://deepmind.com/research/open-source/kinetics.

[16] S.-I. Y. Sami Abu-El-Haija, Anja Hauth, Lu Jiang, Nisarg Kothari, Joonseok Lee, Hanhan Li, Paul Natsev, Joe Ng, Sobhan Naderi Parizi, George Toderici, Balakrishnan Varadarajan, Sudheendra Vijayanarasimhan, "YouTube-8M dataset," 2018. [Online]. Available: http://research.google.com/youtube8m/index.html.

[17] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning Spatiotemporal Features with 3D Convolutional Networks," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 4489–4497.

[18] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 6450–6459, Nov. 2017.

[19] Z. Qiu, T. Yao, and T. Mei, "Learning Spatio-Temporal Representation with Pseudo-3D Residual Networks," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-October, pp. 5534–5542, Nov. 2017.

[20] A. Stergiou and R. Poppe, "Spatio-Temporal FAST 3D Convolutions for Human Action Recognition," *Proc. - 18th IEEE Int. Conf. Mach. Learn. Appl. ICMLA 2019*, pp. 183–190, Sep. 2019.

[21] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07-12-June-2015, pp. 1–9.

[22] J. Carreira and A. Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 4724–4733, May 2017.

[23] A. Montes, "C3D Model pre-trained," 2017. [Online]. Available: https://gist.github.com/albertomontesg/d8b21a179c1e6cca0480ebdf292c34d2.

[24] Tensorflow, "Tensorflow Hub repository." [Online]. Available: https://www.tensorflow.org/hub.