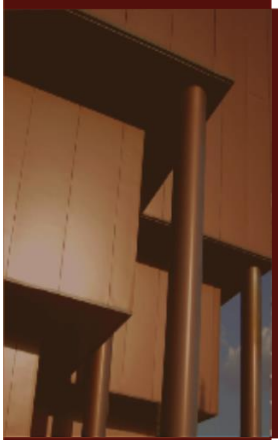
Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Etiquetado automático de vídeo utilizando aprendizaje máquina

Autor: Fernando Vega Zájara
Tutor: Francisco José Simois Tirado



Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

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

Etiquetado automático de vídeo utilizando aprendizaje máquina

Autor:
Fernando Vega Zájara

Tutor:
Francisco José Simois Tirado
Profesor Contratado Doctor

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Etiquetado automático de vídeo utilizando aprendizaje máquina

Autor: Fernando Vega Zájara

Tutor: Francisco José Simois Tirado

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis amigos

Agradecimientos

Finalizar el grado universitario presenta una oportunidad para echar una vista atrás en el tiempo y es una gran satisfacción personal haber llegado hasta este punto. Me gustaría primero agradecer a mi tutor, Francisco José Simois, por abrirme las puertas a un mundo tan apasionante y novedoso para mí como es el *Deep Learning*. Junto a él me gustaría agradecer a los profesores de la escuela, especialmente a aquellos que consiguen hacer que los estudiantes nos apasionemos por sus asignaturas gracias a su cariño y ganas.

Me gustaría agradecer al Colegio Mayor Guadaira y a todas las personas que conocí en él, ya que me apoyaron mucho durante los primero años de carrera y por todas las experiencias vividas junto a ellos, de dónde más que colegiales me llevo a amigos y donde sé que siempre tendré una casa.

A todos mis compañeros de facultad. Sin vosotros, vuestro apoyo con apuntes, quedadas en las salas de estudio y fuera de ellas y ayudas con las prácticas y exámenes jamás habría podido acabar. Me habéis inculcado valores de compañerismo que espero llevar siempre conmigo.

A mis amigos de San Fernando, gracias por haberme aguantado, en las buenas y en las malas. Por haber escuchado mis quejas, por haberme apoyado y por estar siempre ahí. Me habéis visto crecer desde antes de entrar en la carrera y salir de ella, y espero estar siempre con vosotros.

Por último y por encima de todo me gustaría agradecer a mi familia. Especialmente a mis padres, que son las personas que siempre me han apoyado durante toda mi vida. Me habéis inculcado unos valores de responsabilidad y cariño por los que nunca podré agradeceros lo suficiente. Habéis creído en mí cuando ni yo mismo lo hacía. Gracias Papá, gracias Mamá.

Muchas gracias por todo.

Fernando Vega Zájara

Sevilla, 2021

Resumen

En los últimos años, el aprendizaje máquina se ha venido utilizando intensamente para la clasificación de imágenes y vídeo. El objetivo de este trabajo es etiquetar adecuadamente las tareas observadas en una amplia base de datos de vídeos de *YouTube*. Este es un problema de gran complejidad y variedad que lo hace especialmente difícil de analizar usando técnicas clásicas de procesado pero que supone desafíos apropiados para los altos niveles de abstracción que se pueden conseguir con las técnicas de aprendizaje máquina. El objetivo es conseguir un clasificador para 10 clases de vídeos de la base de datos. Para ello se expone una base teórica sobre redes neuronales necesaria para la compresión del problema y se proponen diferentes modelos basados en arquitecturas de redes neuronales densamente conectadas, consiguiendo un alto porcentaje de acierto.

Abstract

In the last years, machine learning has been widely used to classify images and videos. The aim of this research is to properly label the tasks observed in a wide *YouTube* videos data base. This is a really complex problem that makes it difficult to analyze using classic processing technics, although is pose the necessary challenges according to the high levels of abstraction that we can achieve with machine learning techniques. The objective is to obtain a classifier for 10 classes of videos in the database. For this purpose, a theoretical basis on neural networks necessary for the understanding of the problem is presented and different models based on densely connected neural network architectures are proposed, achieving a high percentage of success.

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

ÍNDICE DE ECUACIONES

1 INTRODUCCIÓN

1.1 Motivación y objetivos del trabajo

En las últimas décadas hemos sido espectadores del increíble crecimiento e importancia que ha ido obteniendo la Inteligencia Artificial. Gracias a los avances tecnológicos en tarjetas gráficas de mano de empresas como NVIDIA o AMD, la creación de enormes bases de datos debido al desarrollo de Internet y la introducción en este campo de gigantescas empresas como *Google* o *Facebook*, es una rama que se ha ido desarrollando transversalmente en prácticamente todas las áreas del conocimiento.

Se han conseguido auténticas proezas utilizando algoritmos de Aprendizaje Profundo, tales como conducción autónoma, habilidades sobrehumanas para distintos juegos, traducción automática o publicidad dirigida, entre muchas otras. Pero, de entre las tareas que dieron comienzo a la base de lo que hoy conocemos como Inteligencia Artificial, cabe destacar la clasificación, es decir, entrenar un sistema para que a partir de unos datos de entrada pueda determinar ciertas etiquetas correspondientes.

Los paquetes especializados en Aprendizaje Máquina y Aprendizaje Profundo tales como *Keras*, *TensorFlow*, *Pytorch*, *Pandas* o *Numpy*, todos accesibles mediante el potente pero sencillo lenguaje de programación *Python*, han permitido que personas de todo el mundo puedan acceder de manera gratuita a recursos del campo de la Inteligencia Artificial que permiten que, sin tener estudios específicos en matemáticas o ingeniería, se puedan desarrollar proyectos en este campo.

Es además la visión por ordenador una de las especializaciones de la Inteligencia Artificial gracias a la cual ha cobrado la importancia que tiene a día de hoy, con avances como las redes convolucionales o las redes recurrentes. Si bien se han conseguido increíbles resultados para clasificar imágenes o texto manuscrito, esto ha sido posible solamente gracias a la existencia de enormes bases de datos, algo con lo que por desgracia hasta hace unos años no se contaba para vídeo. Varias bases de datos famosas de vídeo como *UCF101* o *Sports-1M*, estaban restringidas por el bajo número de ejemplos y etiquetas, pero en 2016 *Google* presentó *Youtube-8M*, una enorme base de datos comparable a *ImageNet*, la más famosa de imágenes, para vídeo. Con ello se buscaba conseguir una mejora en el campo de la visión por ordenador, ya que todos los avances en dicho ámbito se han relacionado con la existencia de grandes bases de datos, durante muchos años solamente de imágenes.

El presente trabajo se ha basado en la famosa página de competiciones de Aprendizaje Máquina, *kaggle*, en la cual *Google* durante varios años ha propuesto competiciones sobre la clasificación de vídeo utilizando su base de datos, *Youtube-8M* [1]. Para ello, ya que en el grado de Telecomunicaciones no hay ninguna asignatura específica sobre esta tecnología, primero se busca crear una base teórica necesaria, y después aprender mediante un lenguaje de programación, en este caso *Python*, a utilizar las funciones que ofrecen las distintas librerías para crear redes neuronales, y más específicamente clasificadores.

El objetivo es, a partir de las características de RGB (rojo, verde y azul) y audio de vídeos sacados de *YouTube*, conseguir que una red neuronal las relacione con una de entre una serie de etiquetas posibles. Para ello se ha considerado que las etiquetas posibles sean visualmente reconocibles por el ser humano, como por ejemplo “Videojuego” o “Guitarra”. El etiquetado de vídeos es importante en plataformas como la misma *YouTube*, ya que permite poder adaptar las recomendaciones a los usuarios que estén viendo una clase de vídeo determinada.

Para resolver el problema se ha hecho uso de un ordenador personal y de las herramientas ya mencionadas como *Keras* o *TensorFlow*.

1.2 Estructura de la memoria del trabajo

La memoria del trabajo se ha dividido en los apartados siguientes:

1. **Introducción:** Es el capítulo actual donde se ha expuesto la motivación y objetivo del trabajo, que se irá desarrollando a lo largo del resto de capítulos.
2. **Redes Neuronales:** En este capítulo se realiza una introducción a la rama de la Inteligencia Artificial, y se van desarrollando teóricamente todos los conceptos necesarios para la compresión del presente trabajo.
3. **Metodología propuesta:** En este capítulo se realiza una explicación de la base de datos utilizada en el trabajo, del material, tanto hardware como software utilizado, y de las medidas utilizadas para solucionar el problema.
4. **Entrenamiento y resultados:** En este capítulo se expone el proceso llevado a cabo para el entrenamiento de la red neuronal y sus resultados, con la explicación de los distintos cambios llevados hasta llegar a la arquitectura final. Se presentan fragmentos de códigos que ayudarán a entender mejor las estructuras de las redes creadas.
5. **Conclusiones y líneas futuras:** En este capítulo final se concluye el trabajo con una opinión sobre el problema basada en los resultados conseguidos, y una serie de posibles enfoques que si bien no son desarrollados en el trabajo, forman parte del futuro de la clasificación de vídeo.

2 REDES NEURONALES

En el primer capítulo se ha hecho una introducción al objetivo del trabajo, el cual se resume en clasificar de manera efectiva vídeos utilizando para ello una red neuronal. En este capítulo se va a hacer una explicación teórica de conceptos necesarios para poder comprender la red creada y todos sus componentes. Primero se explicará el contexto de surgimiento de conceptos como Inteligencia Artificial y sus ramas, y tras ello se tratan conceptos más específicos como son las neuronas, redes o funciones de activación.

2.1 Inteligencia Artificial y Aprendizaje Máquina

Si bien «Inteligencia Artificial» (en adelante IA) es un término en auge que hace referencia a toda la rama de la ciencia de datos que simula el aprendizaje humano, tiene más de medio siglo de uso, ya que surgió en la década de los 50, y podría definirse como "la disciplina académica relacionada con la teoría de la computación cuyo objetivo es emular algunas de las facultades intelectuales humanas en sistemas artificiales" [2]. Dicha definición tan abstracta implica que la IA puede referirse a tareas muy diferentes, desde jugar al ajedrez a la adquisición del lenguaje humano.

Cualquier sistema de IA requiere de herramientas pertenecientes a diferentes campos como el cálculo, la estadística, la teoría de la señal o el control automático, entre muchas otras. Pero, pese a ser considerada una rama de la informática, beneficia a muchas otras.

IA es un término que engloba otros campos como «Aprendizaje Máquina» o «Aprendizaje Profundo» y otros muchos, basados principalmente en el aprendizaje. Pero no todos los sistemas de IA se basan en aprender, ya que "emular" las facultades humanas puede referirse a tener una serie de reglas más o menos complejas que el sistema debe seguir. Este enfoque según el cual se puede crear un sistema inteligente a partir de ir sumando cada vez más reglas es lo que se denominó "Inteligencia Artificial simbólica", y fue la idea predominante hasta los años 80 [3].

Tras la IA simbólica surgió el Aprendizaje Máquina donde, en vez de programar a la máquina para una tarea específica, se introducen los datos y sus respuestas esperadas para que la propia máquina aprenda las reglas y pueda hacer predicciones ante los datos no utilizados durante el proceso denominado entrenamiento.

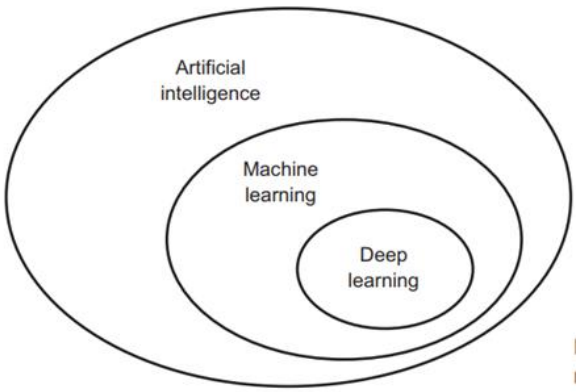


Figura 2-1. Mapa conceptual de ramas de la IA [3]

2.1.1 Ramas del Aprendizaje Máquina

Realmente, dentro del Aprendizaje Máquina se pueden tener distintos enfoques para solucionar un problema, entre los que se distinguen principalmente las tres ramas [3]:

1. **Aprendizaje supervisado**: Consiste en que el sistema aprende a asignar a datos de entradas unas etiquetas conocidas previamente. Una vez aprendido, se puede usar el sistema para predecir a partir de nuevos datos de entrada unas etiquetas de salida, de entre las posibles. Estas etiquetas han sido previamente relacionadas con sus correspondientes datos normalmente por un ser humano, de ahí el nombre supervisado. Además, a cada conjunto de entradas puede corresponderle una sola etiqueta, o varias, es decir, sistema multietiqueta. Dentro de esta rama se encuentran la mayoría de problemas y algoritmos utilizados en el Aprendizaje Máquina. Por ejemplo: clasificación de imágenes o reconocimiento de voz.
2. **Aprendizaje no supervisado**: En este caso el sistema recibe unos datos de entrada pero no conoce las salidas correspondientes. Se busca que el sistema realice una serie de transformaciones sobre estos datos, sin un objetivo claro. Es un tipo de aprendizaje muy utilizado en la analítica de datos. Si bien es una rama mucho menos avanzada que la supervisada, muchos expertos apuntan a que podría ser el futuro del campo. Ejemplos de su uso: reducción de ruido o compresión de datos.
3. **Aprendizaje por refuerzo**: En este tipo de aprendizaje, un "agente" recibe información de su entorno y aprende a elegir acciones que maximizan algún tipo de recompensa. Es con diferencia el área del Aprendizaje Máquina con menos avance, ya que más allá de los juegos no se ha conseguido resultados eficaces, pero se piensa que podrá tener una gran utilidad en la conducción autónoma o la robótica. Ejemplos de su uso: un videojuego donde la red toma acciones para ir maximizando la puntuación con cada simulación.

El objetivo de este trabajo es la clasificación o etiquetado de vídeo, por lo que entra dentro de la rama de aprendizaje supervisado.

2.2 La neurona

Dentro del campo de la IA, la unidad básica de transmisión de información recibe el nombre de neurona. Es el concepto básico a partir del cual se ha podido construir una enorme variedad de arquitecturas más complejas que permiten la resolución de multitud de problemas.

2.2.1 El perceptrón

Las neuronas biológicas son células que, al recibir cierto estímulo, participan en una compleja red de transmisión y procesado de señales eléctricas y químicas. Frank Rosenblatt [4] propuso un modelo basado en la idea de que una neurona puede verse como una puerta lógica con salida binaria, a la cual le llegan múltiples entradas que, si superan cierto umbral, activa dicha salida. Rosenblatt propuso en su modelo un algoritmo de aprendizaje basado en ciertos coeficientes cuyos valores son variables. Estos coeficientes o pesos son los que se multiplican por las entradas de la neurona y permiten calcular si la neurona se activa.

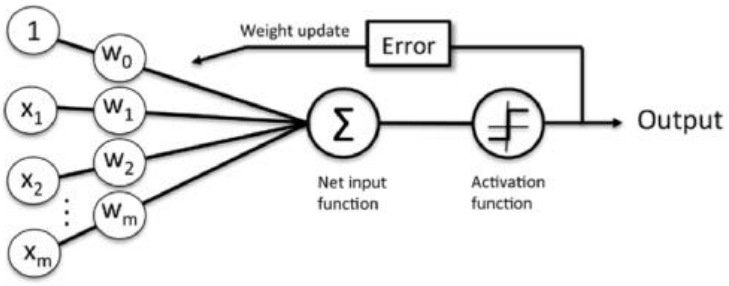


Figura 2-2. Esquema del perceptrón [5]

Este tipo de neurona básica o perceptrón es capaz de realizar una clasificación binaria de dos clases linealmente separables. Su funcionamiento se divide en dos operaciones lineales, suma y multiplicación, cuyo resultado se pasa por una función de activación para producir una salida.

Definiendo el vector de entrada x, y el de pesos w, la entrada a la neurona se define como:

$$Z = w_0 + \sum_{i=0}^{N} w_i x_i \tag{2.1}$$

Donde la w_0 corresponde con el sesgo, cuya entrada correspondiente es siempre 1. Esta entrada de la neurona se pasa después por la función de activación, que en el perceptrón corresponde con el escalón, que da la salida y de la neurona, de esta forma, es posible separar dos clases, la correspondiente al '0' y al '1':

$$y = \delta(Z) = \begin{cases} 1 \; si \; Z \geq 0 \\ 0 \; si \; Z < 0 \end{cases} \tag{2.2}$$

Rosenblatt propuso una regla para el aprendizaje del perceptrón que se basa en primero inicializar los pesos a 0 o números aleatorios muy pequeños, y después, para las distintas muestras de entrenamiento, calcular la salida de la neurona $\hat{y}$. Con la salida calculada y la verdadera, se calcula el error y se actualizan los pesos. Este tipo de aprendizaje corresponde a supervisado ya que cuenta previamente con los datos etiquetados según una de las dos posibles clases. Según la regla del perceptrón los pesos se actualizan de la manera:

$$w_i := w_i + \alpha(y - \hat{y})x_i \tag{2.3}$$

En dicha ecuación y corresponde con la salida esperada, $\hat{y}$ con la que proporciona la neurona, w_i es cada uno de los pesos correspondientes a la entrada x_i. Es importante recalcar que todos los pesos se actualizan de manera simultánea antes de volver a calcular la siguiente salida. El factor α corresponde con un parámetro variable llamado el factor de aprendizaje, que amortigua el cambio de los pesos, cuyo valor suele estar entre 0 y 1.

2.2.2 Funciones de activación no lineales

El perceptrón inicial era un modelo capaz de clasificar dos clases linealmente separables pero demostró ser un modelo muy simple al no poder clasificar una puerta XOR [6]. De esta manera las transformaciones realizadas sobre los datos son todas lineales, una suma y una multiplicación, por lo que el espacio de hipótesis está restringido a uno lineal, y si se unen varias capas de neuronas, que fue la solución que se dio entonces, una pila de capas de neuronas no ampliará dicho espacio, es decir, será como si hubiera una sola gran neurona.

Para ello una vez realizadas las primeras dos operaciones se utiliza una función para introducir no linealidad y poder obtener representaciones más profundas. Entre ellas las más utilizadas en el campo del Aprendizaje Máquina son:

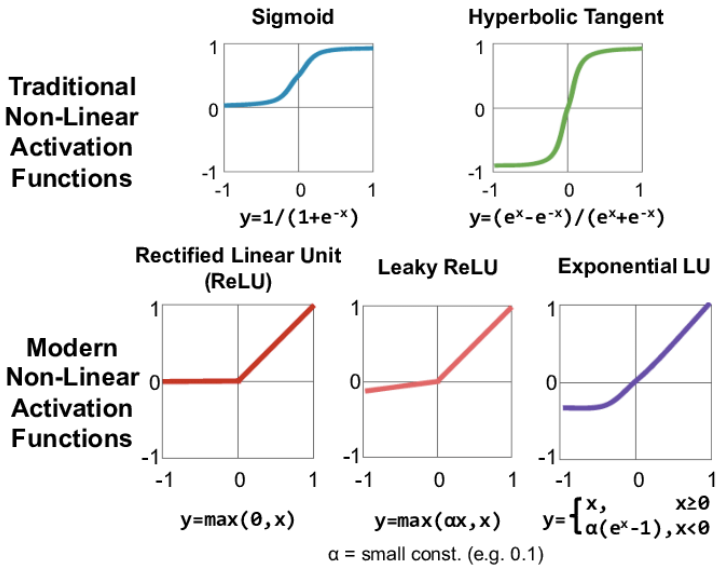


Figura 2-3. Funciones de activación no lineales [7]

2.3 Redes densamente conectadas

Las redes densamente conectadas son la base del Aprendizaje Profundo, el campo que más ha crecido en las últimas décadas dentro del Aprendizaje Máquina. Consisten en ir apilando bloques denominados capas que contienen cierto número variable de neuronas, que va cambiando de capa en capa, uniendo una capa de entrada y una de salida de la red.

Dentro de una red, las distintas neuronas de cada capa van aprendiendo a codificar información de los datos de entrada, que será utilizada por las siguientes capas para ir codificando cada vez información más útil. Las capas que hay entre la de entrada y salida se denominan capas ocultas y son las que aprenden nuevas representaciones de los datos de entrada y relaciones complejas y abstractas entre los datos de entrada y las etiquetas. Conforme más profunda son las redes, es decir, conforme más capas ocultas tengan, mayor capacidad tendrán de detectar información [8].

Las neuronas de la red están todas conectadas, recibiendo de la capa anterior conexiones con todas las neuronas de dicha capa como entradas, y conectándose su salida con las entradas de todas las neuronas de la siguiente capa. Es por ello que este tipo de arquitectura recibe nombre de red densamente conectada.

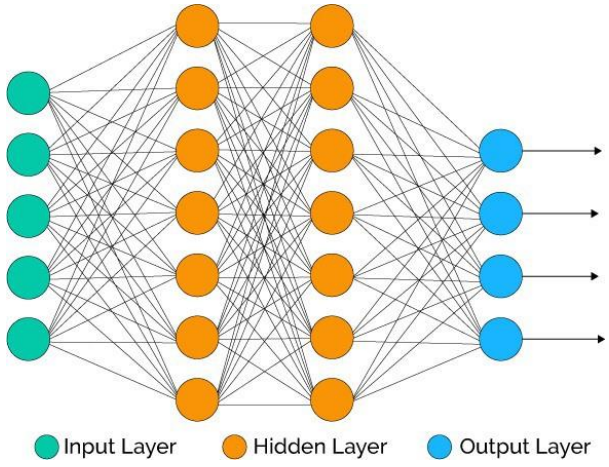


Figura 2-4. Ejemplo de una red densamente conectada [9]

En la Figura 2-4 de ejemplo puede verse que esta red tiene dos capas ocultas. Toda la red puede verse como una función $f(x)$, donde la x representa el vector de entrada y que da como resultado la salida de la red. Cada capa puede verse asimismo como una función que toma como entrada la salida de la anterior. De esta manera podría verse la red completa como:

$$f(x) = f_4(f_3(f_2(f_1(x)))) \tag{2.4}$$

Donde cada subíndice indica el número de la capa, siendo el 1 la de entrada y el 4 la de salida.

2.4 Retroprogagación del error y descenso del gradiente

Una vez presentadas las neuronas y como estas se conectan para formar una red, el problema recae en conseguir que, a partir de unas muestras de datos, dicha red pueda aprender. Para ello el algoritmo de retropropropagación marcó completamente las investigaciones dentro del campo de la IA.

2.4.1 La retropropagación

El algoritmo de retropropagación, presentado en 1986, fue el que permitió que las redes neuronales aprendieran de manera eficiente [8]. Este algoritmo indica que tras hacer pasar por la red una entrada y calcular esta su salida correspondiente, se compara la salida esperada con la obtenida. Si estas coinciden, no se hace nada, pero si no coinciden se calcula el error y los pesos deben ser ajustados para que dicho error sea disminuido.

La diferencia entre este algoritmo y los estudios anteriores a este radica en que en vez de buscar una o varias neuronas culpables del error, varía todas las neuronas de todas las capas gradualmente, atribuyendo una contribución del error a todas ellas. Este ajuste no busca eliminar el error por completo, sino una minimización gradual a través del entrenamiento.

Se denomina retropropagación porque el algoritmo identifica el error en su capa final de salida, y va propagando dicho error a través de toda la red por las diferentes capas ocultas, hasta llegar a la capa de entrada, es decir, va en sentido opuesto a la propagación de una entrada por la red.

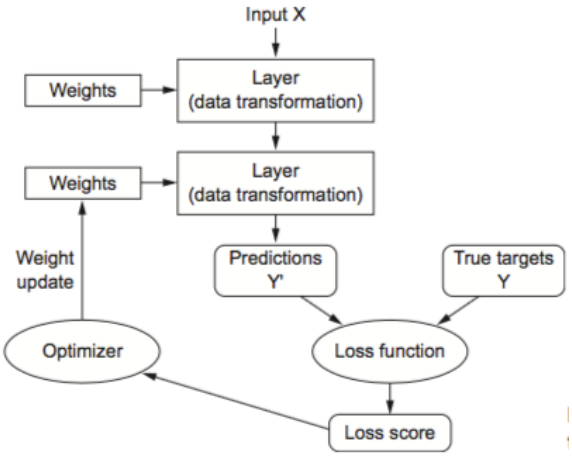


Figura 2-5. Funcionamiento de algoritmo de retropropagación [8]

La retropropagación calcula las derivadas parciales de la función de coste, que cuantifica el error, para poder calcular en cada capa cuanto de ese error corresponde haciendo la derivada parcial del coste de la capa de salida con respecto a cada uno de los pesos de dicha capa. Una vez calculada la de la capa final se calcula de la

justo anterior y así hasta la primera. Esta es una manera eficiente de cálculo de derivadas ya que previamente a este algoritmo había que calcular cada uno de los posibles caminos desde la capa de entrada a la de salida, con una enorme carga computacional.

2.4.2 El descenso del gradiente

Este algoritmo de retropropagación necesita de una función de coste para cuantificar el error y poder minimizarlo. Después se hace uso de un algoritmo de optimización, de entre los cuales el descenso del gradiente es el más conocido.

El descenso del gradiente es un algoritmo que busca mínimos de una función de manera iterativa, analizando el gradiente de la función de error en cierto punto. El objetivo de este algoritmo es ir variando los pesos de las neuronas para minimizar el error.

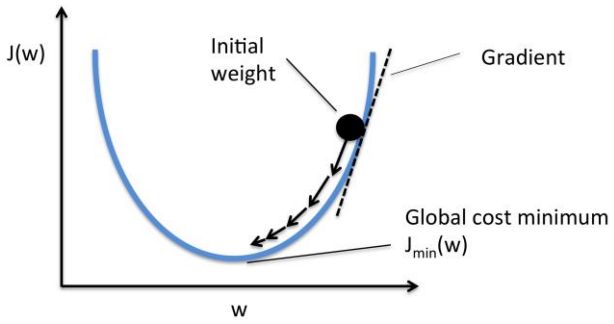


Figura 2-6. Descenso del gradiente [5]

En la Figura 2-6, $J(w)$ corresponde con el coste, que mide el error a minimizar y que depende de los pesos de todas las neuronas.

El proceso de aprendizaje en este caso es iterativo. Normalmente se dividen los datos en tres grupos, diferenciado entre entrenamiento, validación y test. El set de entrenamiento es el que se usa para, durante varias épocas, ir pasando sus datos por la red e ir calculando el error. Ese error es el que usa el descenso del gradiente para ir variando a su vez los pesos de la red en cada iteración hasta llegar al mínimo.

El algoritmo depende del factor de aprendizaje α que, al igual que en el perceptrón, se utiliza para que la variación de pesos no sea demasiado grande y es un parámetro a ajustar en la red. Si es demasiado pequeño el algoritmo tardará muchas épocas en llegar al óptimo, y si es demasiado grande se pueden dar oscilaciones hasta incluso aumentar el coste.

Figura 2-7. Efecto de distintos valores del factor de aprendizaje [10]

2.5 Sobreajuste y subajuste

El objetivo de una red neuronal es generalizar el aprendizaje conseguido durante el entrenamiento, es decir, una vez aprendido un problema poder obtener una salida correcta ante nuevos datos desconocidos.

Para poder medir como de buena es una red ya se ha mencionado la división en tres grupos de los datos. El set de entrenamiento y el de validación son los que se usan durante el entrenamiento de la red. El set de test se usa una vez es entrenada la red para medir en realidad cómo de bien funciona la red. Normalmente de los datos usados para el entrenamiento, para el set de entrenamiento se usa un 80% y para validación un 20%, aunque estas cantidades pueden variar.

La red tras el entrenamiento puede dar tres tipos de resultados dependiendo de cómo de generalizado es el aprendizaje. Para poder medir dicho resultado se hace uso de la función de pérdida, que es la que se va minimizando durante el entrenamiento. En este trabajo se hará uso de la función *categorical-crossentropy*, que se explicará más adelante, como función de pérdida a minimizar.

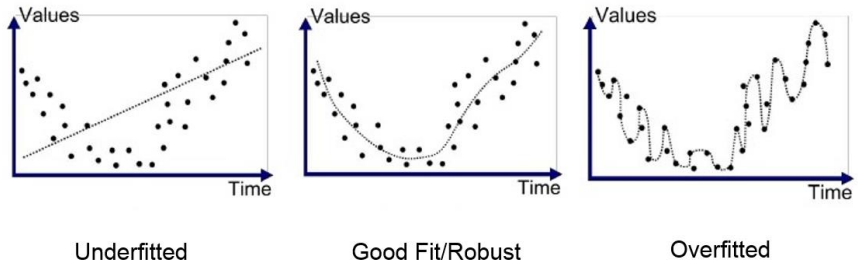


Figura 2-8. Posibles resultados en una red [11]

Si la red no es capaz de adaptarse a los datos de entrenamiento se produce un subajuste, que normalmente suele suceder porque la red es demasiado simple para la complejidad de los datos del problema. Normalmente si se van añadiendo capas o neuronas se soluciona este problema, y al ir realizando más iteraciones en el entrenamiento.

Si la red se adapta demasiado a los datos de entrenamiento pierde la generalización y se produce un sobreajuste, donde la red memoriza para los datos de entrenamiento sus correspondientes salidas. Suele darse cuando se entrena demasiadas épocas a la red y es uno de los problemas más típicos dentro del Aprendizaje Máquina.

El punto medio entre el subajuste y el sobreajuste es el que se busca en una red de este tipo, donde se consigue generalizar el conocimiento adquirido. Estos tres casos posibles pueden verse durante el entrenamiento si se observa la función de error, que es el objetivo a minimizar. Cuando se produce subajuste tanto el error de entrenamiento como el de validación es elevado. Normalmente si la red es lo suficientemente compleja y se sigue iterando, se va minimizando ambos errores. Pero llega un punto normalmente en que el error de entrenamiento continúa disminuyendo pero el de validación empieza a crecer. Se está produciendo un sobreajuste, ya que se está perdiendo la generalización y la red se está adaptando demasiado a los datos de entrenamiento.

Una posible solución al problema sería ir graficando el error de entrenamiento y validación. Al principio empezará bajando superando la fase de subajuste si la red es lo suficientemente compleja. Si llega el punto en que el error de entrenamiento sigue disminuyendo pero el de validación ha llegado a un mínimo y empieza a crecer, se puede parar de entrenar en dicha época, ya que significa que es el punto óptimo para la red.

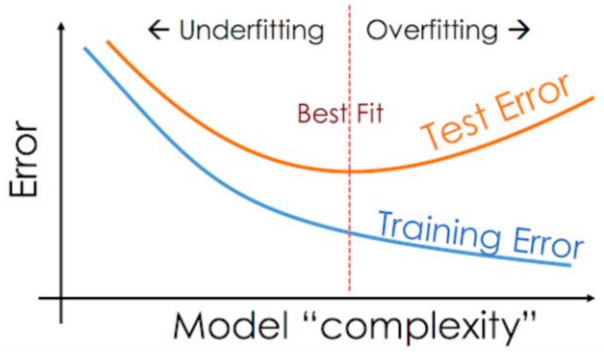


Figura 2-9. Sobreajuste y subajuste en el error [12]

2.5.1 Dropout

Otra posible solución al problema del sobreajuste es el *Dropout* [13], una técnica desarrollada en la universidad de Toronto, que se basa en, al aplicarla en una capa, desactivar de manera aleatoria un cierto número de neuronas, lo que implica poner sus entradas a cero durante el entrenamiento. El número de neuronas desactivadas es dependiente de la tasa de *Dropout* que se use. Normalmente se usa 0.5, que indica que de manera aleatoria la mitad de las neuronas de esa capa se desactiva. Esta técnica reduce el sobreajuste en la red ya que se demuestra que limita la memorización de los datos de entrenamiento. En este trabajo se empleará esta técnica.

2.6 Redes convolucionales

Las redes convolucionales (CNN en inglés) son una familia de modelos que se inspiran en el funcionamiento de la corteza visual del cerebro humano para el reconocimiento de objetos. Surgen en los años noventa cuando Lecun [14] y varios colegas propusieron un nuevo tipo de arquitectura de red neuronal para la clasificación de dígitos manuscritos a partir de imágenes, basada en buscar ciertos patrones. Por ejemplo en la imagen de un coche se buscan patrones simples, como líneas o curvas, para construir un patrón más complejo como una rueda o una puerta.

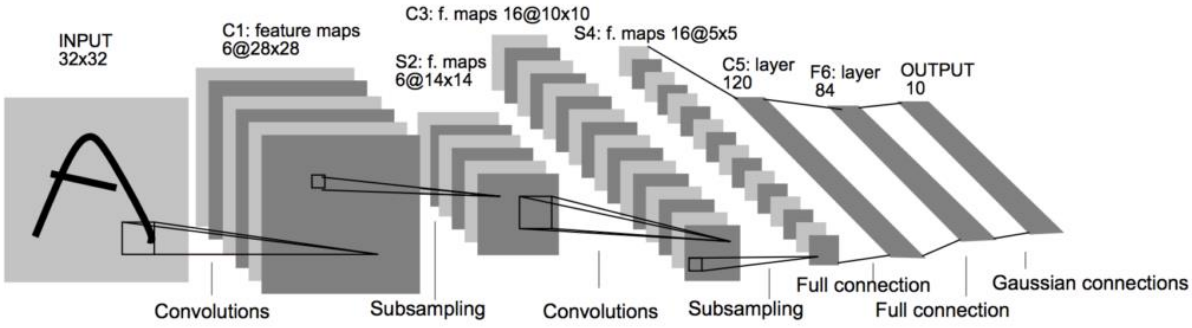


Figura 2-10. Red propuesta para clasificación de caracteres manuscritos usando CNN [14]

Este tipo de redes obtiene un rendimiento excepcional para clasificación de imágenes, lo que ha hecho que adquieran mucha importancia en el campo del Aprendizaje Profundo en las últimas dos décadas.

Las CNN en imágenes se basan en calcular un mapa de características a partir de la imagen de entrada. Normalmente estas redes se componen de varias capas convolucionales y submuestreo seguidas, y al final se hace uso de una red densamente conectada para la tarea de clasificación. Además de las dos principales capas de convolución y submuestreo, se realiza un aplanado de características para poder usar como entrada un vector en la red densamente conectada.

Las CNN se aprovechan de una propiedad que tienen las imágenes, su estructura espacial. Antes de estas redes para poder introducir una imagen en una red neuronal, cada pixel de esta imagen era pasada a la red como una variable independiente, recibiendo la imagen como un único vector plano. Pero de esta forma se perdía la importancia que tiene la posición de ese pixel dentro de la imagen, donde sabemos que normalmente el valor de un pixel está muy relacionado con los píxeles que tiene alrededor, permitiendo la construcción de patrones.

2.6.1 Capa convolucional

Este tipo de capa se caracteriza por aplicar sobre la entrada, que normalmente es una imagen por lo que tiene dos dimensiones, la operación matemática de la convolución. Esta operación puede verse como la aplicación de un filtro sobre estos datos de entrada.

Estos filtros son una matriz de números, de dimensiones normalmente mucho más pequeñas que la entrada, que se pone sobre la imagen original, y después se multiplican sus valores por los valores de los píxeles de la imagen de entrada. De esta manera se consiguen unos nuevos valores haciendo un sumatorio de la multiplicación. Tras hacer esta multiplicación, el filtro se va desplazando por toda la imagen, hasta conseguir una nueva imagen.

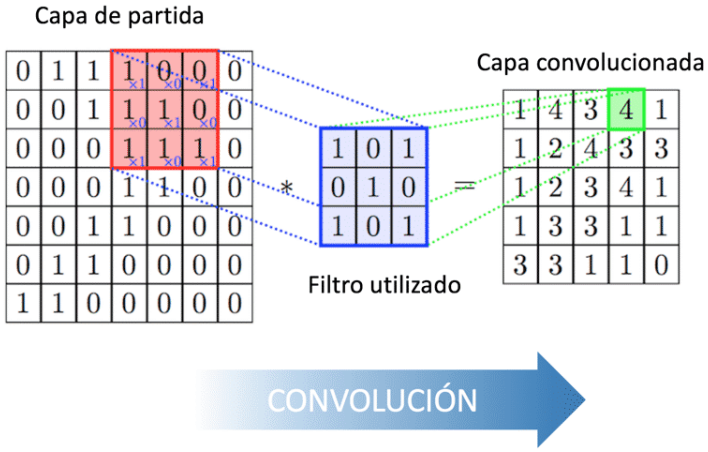


Figura 2-11. Operación de la convolución [15]

Esta imagen resultado dependerá de los valores que tenga el filtro, existiendo diferentes filtros como algunos especializados en buscar diferencias de contraste, mientras que otros pueden ser usados para desenfocar. Son estos valores del filtro los que durante el entrenamiento la propia red aprenderá para detectar los patrones. El resultado de aplicar el filtro es lo que, como ya se ha mencionado, se denomina mapa de características.

La potencia de las CNN radica en que realmente lo que se produce es un conjunto de convoluciones, donde la entrada de cada una es la salida de la anterior. De esta manera cada vez será más profunda la red y más potente, ya que podrá detectar más información útil. Cuando ya se tiene un enorme conjunto de mapas de características con los patrones necesarios para poder detectar dicha información es cuando se pasa por una red densamente conectada.

Las imágenes realmente pueden verse con sus dos dimensiones espaciales, pero dependiendo de si es a color tiene tres canales (RGB) o en blanco y negro, un canal. Para cada uno de los canales se crea un mapa de características diferente al aplicar la convolución.

2.6.2 Capa de pooling

En las CNN, al realizar las convoluciones, se aplica también una capa denominada *pooling*, ya que en cada capa convolucional no se aplica un solo filtro, sino un conjunto de filtros, lo que da como resultado que cada vez haya más capas de características al ir profundizando en la red, por lo que computacionalmente tendría un coste enorme.

Para ello se hace una reducción del tamaño de las imágenes filtradas o mapas de características. Hay diversos tipos de capas de *pooling*, entre los cuales el más usado es el de *Max-pooling*. Se resume en aplicar también un filtro, por ejemplo un 2x2, donde en este caso se toma el máximo valor de la entrada por cada aplicación del filtro en la imagen. De esta manera se reduce a la mitad sus dimensiones.

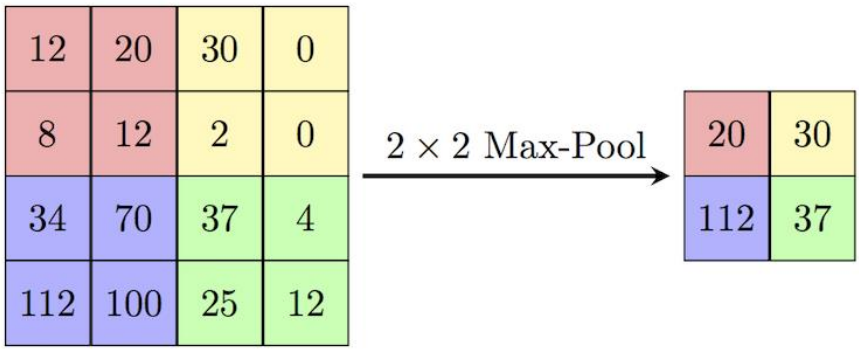


Figura 2-12. Aplicación de filtro *Max-pooling* 2x2 [16]

Otra capa de *pooling* importante es el *Avarage-pooling*, que toma el valor medio, pero la de *Max-pooling* es la más utilizada.

2.6.3 Capa Flatten

Las CNN terminan haciendo uso de una red densamente conectada, pero este tipo de redes funciona recibiendo como entrada un solo vector. Es necesario pues una capa previa a esta red, una vez finalizadas todas las capas convolucionales y de *pooling*, que se denomina *Flatten* o aplanado.

La tarea de esta capa es convertir los últimos mapas de características de la CNN es un vector e introducírselo a la red neuronal multicapa densamente conectada. Es necesaria para poder combinar estos dos tipos de redes distintas.

3 METODOLOGÍA PROPUESTA

En este capítulo se va a hacer una presentación de la base de datos y los materiales utilizados, tanto desde el punto de vista software como hardware. Primero se explicará la potencia y utilidad que tiene la base de datos *YouTube-8M* y cómo extrae las características de los vídeos. De la misma forma se comentarán las librerías de software que se usarán, junto a algunas de sus funciones más importantes.

Tras ver los materiales se explicarán las herramientas de medida que se usarán durante el entrenamiento y los objetivos que se buscan a la hora de ir eligiendo los parámetros de la red neuronal.

3.1 Base de datos utilizada: Youtube-8M

Para el presente trabajo se ha decidido hacer uso de entre todas las bases de datos de vídeo disponibles la de *YouTube-8M*, proporcionada por *Google*. Su gran tamaño y sus diferentes etiquetas hacen de esta una base de datos interesante de analizar.

A diferencia de otras bases de datos anteriores a ella como *ActivityNet* o *Sports-1M*, *YouTube-8M* no se restringe solo a vídeos de acciones o deportes. De hecho en su página oficial [17] se ofrece un buscador para todo su vocabulario, agrupando los vídeos en 24 grupos principales denominados "verticales".

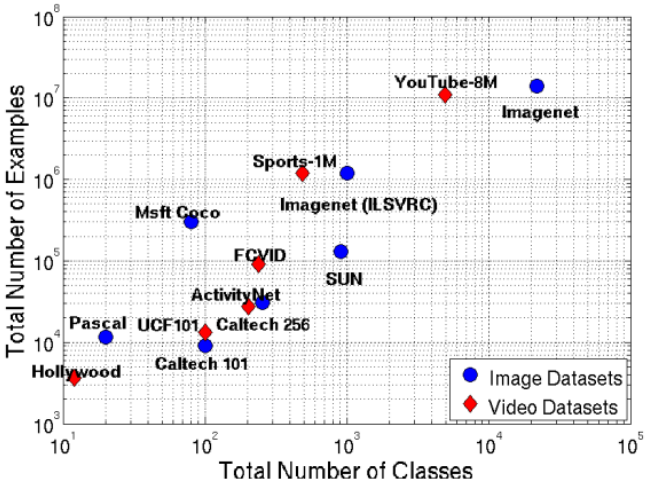


Figura 3-1. Bases de datos para imagen y vídeo [18]

La base de datos fue construida a partir de vídeos de la plataforma *YouTube*, ya que esta era una buena fuente de vídeos de muy distintas categorías. Se eligieron para cada vídeo una serie de etiquetas que fueran visualmente reconocibles, de las cuales muchas las ponía el propio algoritmo de la plataforma. Pero la base de datos original tenía millones de vídeos que se traducían en cientos de Terabytes de información, lo que la hacía ineficaz para su uso en entrenamiento de redes neuronales.

3.1.1 Red Inception

Es por la gran cantidad de espacio necesario que los creadores de *YouTube-8M* decidieron hacer uso de una red ampliamente conocida y públicamente accesible, *Inception* [19], que ha sido entrenada haciendo uso de la base de datos de imágenes *ImageNet*. Se procesaron los vídeos de *YouTube*, decodificándolos a 1 *frame*/segundo, y se introdujeron estos *frames* en la red *Inception*, utilizando una función de activación *ReLu* en su última capa oculta, antes de la capa de clasificación final. El resultado fue un vector de dimensión 2048

por segundo de vídeo [18]. Después se hizo una serie de reducciones hasta conseguir un vector de características de dimensión 1024 por segundo de vídeo. Además de las características de vídeo, de igual manera *Youtube-8M* ofrece las de audio por cada vídeo.

Cabe destacar que la red *Inception* es una compleja red convolucional, por lo que como solución al problema de clasificación de vídeos, ya que *YouTube-8M* previamente ha procesado estos vídeos mediante dicha red, lo que se va a realizar en este trabajo es una red densamente conectada, que tomará estas características y aprenderá de los patrones.

ImageNet es una base de datos que surge en el 2009 [20], y en sus inicios recogía más de 3 millones de imágenes. Fue propuesta en el campo del Aprendizaje Máquina, específicamente de la visión por ordenador, con el principal objetivo de ser un recurso en el entrenamiento de redes y para los investigadores.

Previamente a esta base de datos, la mayoría de redes de visión por ordenador se centraban en un pequeño número de objetos comunes, como coches, peatones y caras, debido a que las bases de datos disponibles eran de pequeña escala y únicamente disponías de estas pocas clases. *ImageNet* en cambio, además del gran número de imágenes por clase, permite acceder a objetos menos comunes. Aporta una gran calidad, diversidad y escala, propiedades que hicieron que, más allá de convertirse en una de las bases de datos más famosas de la última década, se hayan creado potentes redes para analizar relaciones complejas y jerarquías entre ellas, y se haya avanzado más en el entendimiento del sistema visual humano.

Además de la clasificación, en la que se centra este trabajo, hay muchas otras aplicaciones de conjuntos de imágenes y vídeos, como detección de objetos, redes generativas adversarias, conducción autónoma, reconocimiento facial y muchas otras.

Inception surge justo como solución vencedora, seguido por *VGGNet*, en la competición de Reconocimiento Visual a Gran Escala de *ImageNet* 2014 (ILSVRC14). Se trataba en su primera versión, también conocida como *GoogLeNet*, de una red convolucional profunda, formada por 22 capas cuyo principal uso era la clasificación y la detección de imágenes. La principal característica de esta arquitectura es la utilización mejorada de los recursos informáticos dentro de la red, mediante un diseño cuidadosamente elaborado de los mismos, aumentamos la profundidad y la anchura de la red, manteniendo a su vez constante el coste computacional.

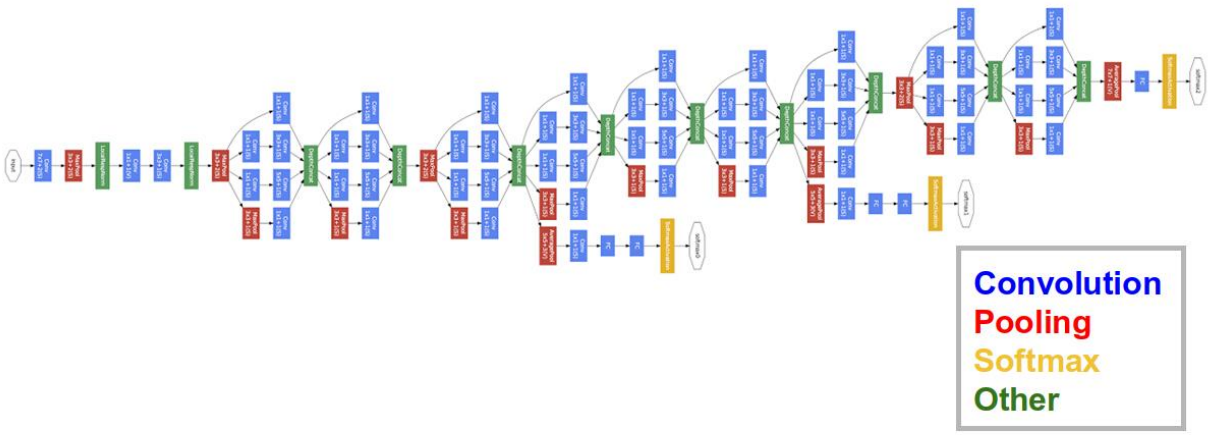
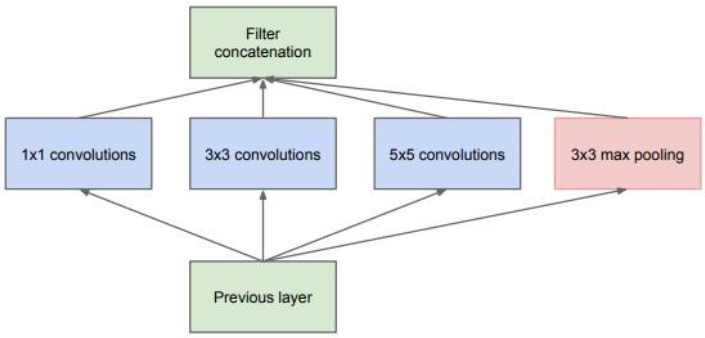


Figura 3-2. Red convolucional *Inception* [19]

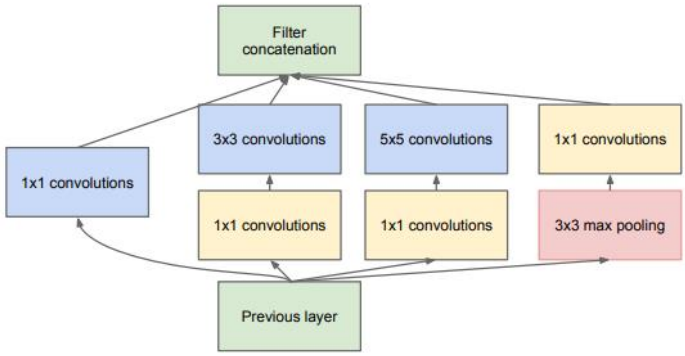
La red *Inception*, como otras que surgieron estos años en estas competiciones, fue un gran avance para el campo del Aprendizaje Profundo porque demostró que con una mejora mínima del hardware, y usando las mismas bases de datos, se conseguían grandes mejoras en el rendimiento de clasificación. Lo importante es que estas mejores se conseguían únicamente mediante mejores algoritmos y arquitectura, logrando por ejemplo en 2014 utilizar 12 veces menos parámetros que la versión ganadora de dos años antes, aun aumentando el rendimiento.

Las redes convoluciones en los años anteriores seguían la regla de ir haciéndose cada vez más profundas para lograr representaciones mejores, de manera secuencial, reduciendo las dimensiones entre capas usando antes de la siguiente capa convolucional una de *pooling*. Pero la estructura básica de la red *GoogLeNet* es el bloque o módulo *Inception* que se basa en usar varias capas en paralelo [21]. Para ello una capa toma la salida de la capa anterior, aplica diferente capas de convoluciones y concatena los resultados para la capa posterior. El módulo básico consta de una capa convolucional con filtros de tamaño 1x1, una con 3x3, una con 5x5 y una de *Max-Pooling* con 3x3. Además ofrece otra versión posible de esta red utilizando capa convolucionales 1x1 extras para aplicar una reducción de características.

La utilización de diferentes filtros en paralelo permite extraer diferentes tipos de características de una misma entrada. Si se opta por la opción de aplicar los filtros 1x1 antes de las capas convoluciones con filtros 3x3 y 5x5, permite que el número de parámetros se reduzca y sea más eficiente computacionalmente, consiguiendo además la mejora del rendimiento.



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figura 3-3. Módulo *Inception* sin y con reducción de dimensiones [19]

3.1.2 Uso de YouTube-8M

YouTube-8M ofrece para cada uno de los vídeos una media de 3 etiquetas/vídeo. En este trabajo se ha decidido utilizar una sola etiqueta por vídeo para que sea más asequible desde un punto de vista computacional. Para ello se ha escogido de manera aleatoria 725 archivos que contienen las características procesadas de más de 300.000 vídeos, que se han dividido en 70% para el set de entrenamiento y 30% para validación, además de casi 50.000 vídeos para el set de test.

En estos vídeos se han decidido escoger 10 etiquetas de entre todas las que ofrece la base de datos, ya que permitirá que la red se entrene en menos de una hora por entrenamiento, además de hacer más visibles los resultados mediante la matriz de confusión, que se explicará más adelante.

Es importante señalar que en esta base de datos el número de veces que aparecen las etiquetas va descendiendo según aumenta su identificador, por lo que se han escogido etiquetas que tienen identificadores bajos. Más adelante se trabajará alimentando redes cumpliendo con esta proporción como viene en la base de datos, o con

el mismo número de vídeos por cada etiqueta, para evitar cualquier tipo de memorización de la red con las etiquetas que tienen mayor probabilidad de aparecer, viendo las ventajas o desventajas de cada caso.

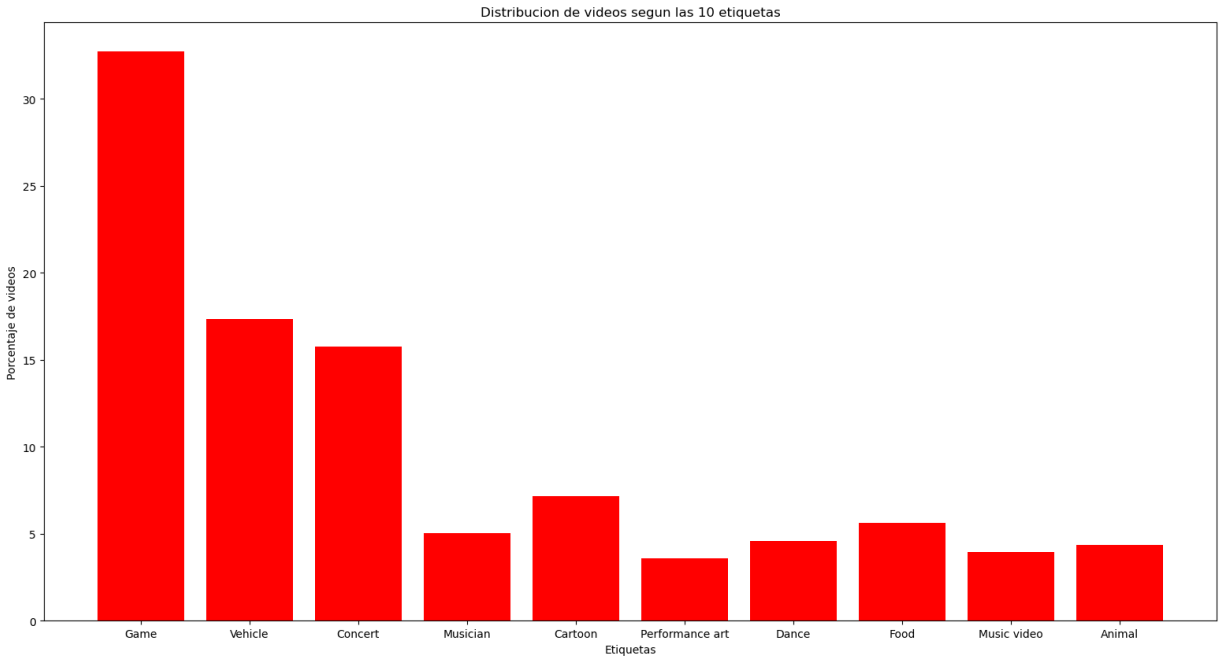


Figura 3-4. Distribución de las etiquetas para el entrenamiento (%)

Las características de los vídeos, junto a sus identificadores, vienen dados por *YouTube-8M* en un formato denominado *tensorflow.Example*, un protocolo que se guarda en archivos con extensión *.tfrecord*, haciendo uso de la librería *TensorFlow*, que se explicará más adelante. En la siguiente imagen se muestra un ejemplo de cómo es la estructura interna de estos archivos.

```
features: {
  feature: {
    key  : "id"
    value: {
      bytes_list: {
        value: (Video id)
      }
    }
  }
  feature: {
    key  : "labels"
    value: {
      int64_list: {
        value: [1, 522, 11, 172]  # label list
      }
    }
  }
  feature: {
    # Average of all 'rgb' features for the video
    key  : "mean_rgb"
    value: {
      float_list: {
        value: [1024 float features]
      }
    }
  }
  feature: {
    # Average of all 'audio' features for the video
    key  : "mean_audio"
    value: {
      float_list: {
        value: [128 float features]
      }
    }
  }
}
```

Figura 3-5. Ejemplo de estructura de cada vídeo en *YouTube-8M* [17]

De esta forma, se dan las etiquetas asignadas a cada vídeo como una lista de enteros de tamaño variable, y las características de vídeo y audio como una lista de flotantes, de tamaños fijos 1024 y 128 respectivamente.

Haciendo uso del vocabulario ofrecido en la página oficial de la base de datos, en la siguiente tabla se muestra por cada una de las etiqueta el número de vídeos que se ha usado en el entrenamiento.

Etiqueta	Número de vídeos	Etiqueta	Número de vídeos
Game	102.216	Performance Art	11.202
Vehicle	54.225	Dance	13.978
Concert	49.153	Food	17.514
Musician	15.551	Music Video	12.401
Cartoon	22.328	Animal	13.596

Tabla 3-1. Número de vídeos por etiqueta

3.2 Material

3.2.1 Hardware

Todo el campo de la IA, especialmente el Aprendizaje Máquina y el Aprendizaje Profundo, ha conseguido grandes resultados gracias al enorme avance de la tecnología en las últimas décadas. Un modelo es una estructura que consta de hasta miles o millones de variables, y su funcionamiento se basa principalmente en multiplicación de matrices, por lo que hacen falta materiales específicos para ello.

Durante muchos años las mejoras fueron principalmente de la mano de la unidad central de procesamiento, CPU, que permite hacer operaciones aritméticas y lógicas lo suficientemente potentes a día de hoy para que cualquiera en su portátil pueda entrenar pequeños modelos.

Pero a la vez, empresas como NVIDIA o AMD han invertido millones de euros en el desarrollo de las unidades de procesamiento gráfico, GPU, destinados a mejorar la calidad de los videojuegos, siendo este un mercado en alza. Las GPUs se diferencian de las CPUs en que las segundas son de propósito general, mientras que las GPUs cumplen tareas más específicas como reproducir escenas 3D en las pantallas de los ordenadores. NVIDIA lanzó además en 2007 CUDA [22], una interfaz de programación para sus GPUs, de esta manera pudo ser utilizada por la comunidad científica. Muchos investigadores comenzaron entonces a escribir implementaciones de CUDA para redes neuronales.

Desde los últimos años, NVIDIA ha invertido mucho dinero en el campo de la Inteligencia Artificial, incluso desarrollando GPUs específicas para estas tareas, usadas principalmente para problemas más complejos como visión por ordenador o reconocimiento de texto.

Otra empresa que ha invertido mucho en este campo es *Google*, que en 2016 presentó la unidad de procesamiento de tensores, TPU [23], un proyecto diseñado específicamente para ejecutar redes neuronales profundas, y que puede llegar a ser mucho más eficiente y consumir menos que las mejores GPU.

Además *Google* ofrece *Google Cloud*, una plataforma que permite el acceso a las GPUs y TPUs de la compañía de manera virtual. Ofrece una versión gratuita mediante la cual se puede hacer uso de la tecnología que ofrece la compañía pero dentro de ciertos límites, además de que en cualquier momento si la compañía requiere de estos recursos puede reducirlos o dejar de darlos.

En este trabajo se ha decidido optar por la opción de trabajar con el portátil personal, ya que si bien no tiene una GPU especializada en Aprendizaje Máquina, al ser personal se puede hacer uso de él en cualquier momento e incluso dejarlo entrenando todo el tiempo necesario, sin costes adicionales ni cortes. Se trata de un portátil ASUS cuyas características se presentan en la siguiente tabla.

Sistema Operativo	Linux Mint 18.3 Cinnamon 64-bit
CPU	Intel Core i7-6700HQ @2.60GHz x4
GPU	NVIDIA GeForce GTX 950M

Tabla 3-2. Especificaciones del ordenador personal utilizado para el entrenamiento

3.2.2 Software: Lenguaje de Programación

Dentro del apartado del *software*, lo primero es la elección del lenguaje de programación. En el campo de la IA hay una amplia variedad de opciones como *Java*, *R*, *C++* y muchas otras. De entre los posibles candidatos se ha elegido *Python* por sus características, siendo un lenguaje muy simple y fácil de aprender, pero a la vez de una gran potencia.

Dentro de la ciencia de datos *Python* se ha convertido en el lenguaje más popular justo por su curva de aprendizaje, y porque su simplicidad permite al usuario olvidarse de la parte más tediosa de la programación y volcar mejor sus ideas. En el grado de Telecomunicaciones de la Universidad de Sevilla no hay ninguna asignatura específica sobre *Python*, si bien es cierto que se está implantando en las prácticas de diversas asignaturas, lo que ha facilitado aún más su asimilación, ya que cada vez es más demandado en el mercado laboral.

Python además aporta una enorme cantidad de librerías especializadas en el Aprendizaje Máquina y manejo de datos. Es de código abierto, gratuito y tiene una gran comunidad por detrás en foros que sirven como ayuda al nuevo usuario.

3.2.3 Software: Librerías de Python

Dentro de las librerías de *Python*, se ha hecho uso de las siguientes:

1. *Numpy* [24]: Librería que da soporte para vectores y matrices de grandes dimensiones. Es de software libre, siendo la principal librería usado por la comunidad científica para tareas de computación.
2. *Matplotlib* [25]: Principal librería de visualización en *Python.* Permite crear visualizaciones estáticas, dinámicas o interactivas.
3. *Pandas* [26]: Librería de software libre para el análisis de datos de manera flexible, rápida, práctica y potente.

3.2.4 Software: TensorFlow y Keras

TensorFlow [27] es una plataforma creada por Google, especializada para el Aprendizaje Máquina, enfocada principalmente a *Python*. Dota de una arquitectura flexible a la construcción de redes neuronales para poder ser ejecutadas en CPUs, GPUs y TPUs, además de incluir varias librerías y herramientas. Permite la operación sencilla con tensores, e implementar derivación automática. Se convirtió en una de las librerías más utilizadas, al ser multiplataforma, y permitir su uso en otros lenguajes como *C++*.

Keras [28], es una librería creada por el ingeniero de *Google* François Chollet [3] en 2015, al principio para uso personal, pero más adelante fue adoptada por *TensorFlow* integrándola en la versión 1.1.0, ejecutándose por encima de este. Su gran sencillez permitió que se convirtiera en la estructura más elegida a la hora de iniciarse en la creación de estructuras de redes neuronales.

Si bien hay varias otras opciones disponibles en el mercado, tales como *PyTorch* o *Caffe*, cada una con sus ventajas e inconvenientes, se ha decidido hacer uso de *Keras* sobre *TensorFlow* justo por la sencillez, y rápido aprendizaje que permiten. Además la base de datos requiere el uso de *TensorFlow* para la lectura de sus datos.

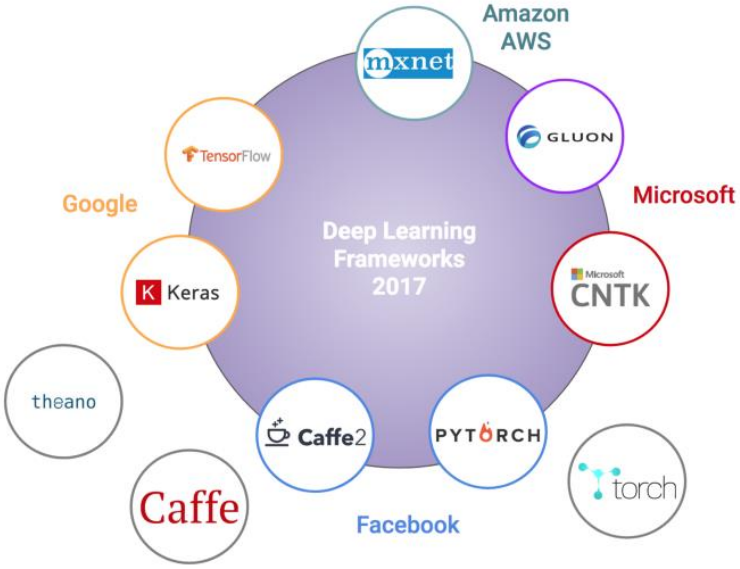


Figura 3-6. Posibles librerías de *Deep Learning* en el mercado [29]

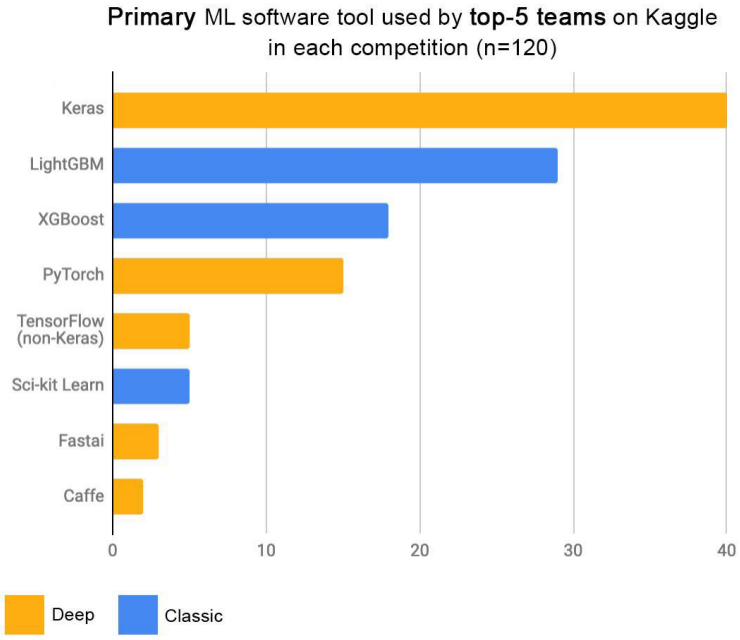


Figura 3-7. Software utilizado por los ganadores de competiciones de *kaggle* [28]

Son además librerías líderes en las competiciones de *kaggle*, donde participan competidores de todo el mundo, avalando su gran potencia para tareas de Aprendizaje Máquina y Aprendizaje Profundo como en este trabajo donde se ha hecho uso principalmente de estas dos librerías además de las de *Python* ya mencionadas. Las funciones de *TensorFlow* se han usado para sacar los datos de los archivos de la base de datos *YouTube-8M,* y las funciones de *Keras* para la construcción de la red neuronal.

3.3 Capas utilizadas

Para la construcción de la red se han utilizado las funciones de *Keras* que permiten ir construyendo cada capa y después ir pasando las capas anteriores como entradas de las siguientes. Se hace una pequeña explicación de cada una de ellas.

3.3.1 Capa Input

Las primeras capas de las redes que se han implementado han sido *Input* que lo que hace es crear una instancia de tensores, pasando como argumento el tamaño de dicho tensor. En este caso las entradas serán las características de audio y vídeo, de tamaños 128 y 1024 respectivamente.

Esta capa permite asegurar que durante tanto el entrenamiento como el posterior uso de la red ya entrenada, las entradas sean siempre de los tamaños correctos indicados.

3.3.2 Capa Dense

Es la capa que más se utiliza en los modelos implementados. Hace referencia a una capa de neuronas densamente conectadas que, por defecto, se conectan a todas las neuronas de la capa posterior.

La función toma como parámetros el número de neuronas de la capa y la función de activación. Son parámetros que se han ido variando durante el entrenamiento para buscar un mejor resultado, al igual que el número de capas *Dense* utilizadas. La entrada de cada capa es la capa anterior, excepto la primera capa oculta que toma directamente como entrada la *Input*, y la última capa, que tendrá como función de activación siempre *softmax*, como se explicará en la implementación de la red, y que será la capa de salida.

3.3.3 Capa Dropout

Esta capa se utiliza para introducir en la red la técnica del *Dropout* ya explicada, cuyo objetivo es la reducción del sobreajuste de la red.

Toma como entrada una capa *Dense*, y aplicará la tasa de *Dropout* sobre ella. Esta tasa, que indica el número de neuronas inactivas de manera aleatoria durante el entrenamiento, es el parámetro de entrada que se pasa a la capa.

3.3.4 Capa concatenate

Esta capa sencillamente toma como entradas un número de capas anteriores, y las une formando como salida un tensor de tamaño la suma de los tamaños de sus entradas.

Se ha utilizado porque, como hay dos tipos de entradas distintas (audio y vídeo), primero para cada entrada se han utilizado varias capas *Dense*, y posteriormente se han unificado las dos ramas de la red en una sola, gracias a esta capa *concatenate*.

3.4 Medidas del resultado

Para poder saber si un modelo propuesto funciona correctamente hace falta hacer medidas tanto del error como del acierto. Para ello se propone en este caso hacer uso de dos funciones, usadas durante el entrenamiento de la red neuronal para poder optimizar sus parámetros, y después del entrenamiento para poder visualizar si el resultado es el esperado.

Durante el entrenamiento, para el cálculo del error, la función que se ha usado para la optimización ha sido *categorical-crossentropy*, junto al *accuracy* una vez terminada la iteración para ver el porcentaje de acierto.

Después se ha hecho uso de la matriz de confusión, una herramienta visual que permite una vez entrenada la red, sobre un conjunto de muestras no usadas durante el entrenamiento, el conjunto de test, visualizar para cada una de las etiquetas el resultado obtenido.

3.4.1 Categorical-crossentropy

Esta función es usada como medida de la distancia entre distribuciones de probabilidad. La capa de salida de la red neuronal tendrá como función de activación una *softmax*. Esta función da como resultado para cada una de las posibles etiquetas la probabilidad entre 0 y 1 de que, para unos datos de entradas, cada una de las etiquetas sea la salida correcta. La ecuación de la función *softmax* es:

$$\hat{y}_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \qquad j = 0,1,2,\dots,N \tag{3.1}$$

Esta función se utiliza en la capa final de la red neuronal, donde por cada neurona, su salida $\hat{y}_i$ se calcula utilizando los valores finales calculados por todas las capas ocultas de la red, siendo x_i el valor que corresponde a dicha neurona final antes de aplicar la función *softmax*. De esta forma, el valor de $\hat{y}_i$ se puede interpretar como la probabilidad de que esa clase sea la verdadera. Además la capa de salida tiene un número de neuronas igual al número de clases posibles, y cada una de esas neuronas se asociará a una clase.

Se suele suponer que en una red neuronal, aquellas neuronas que reciban mayor entrada neta se convierten rápidamente en las más activas, mientras que las actividades del resto se suprimen, por lo que existe una especie de competencia entre neuronas. El comportamiento *Winner-Take-All* corresponde a la situación donde solo la neurona de salida que recibe mayor entrada se activa mientras el resto se anulan. Sin embargo se puede argumentar que la perspectiva *Winner-Gets-More*, donde se permite que más neuronas estén activas para cada entrada y si los valores que toman están entre 0 y 1, se consigue una codificación más eficaz [30], por ello el uso de *softmax* se convierte en una gran alternativa.

Las redes neuronales utilizan la función *categorical-crossentropy* (CE) cuando el número de clases posibles es mayor que dos, en este caso 10.

$$CE = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{3.2}$$

Donde y_i es cada uno de los componentes del vector de clase verdadera, mientras que $\hat{y}_i$ es cada componente del vector a la salida de la capa *softmax*, por lo que corresponderá con las probabilidades de que para una entrada dada sea cada una de las clases.

El número de clases en este problema es $N = 10$, pero lo interesante de esta función es que solo utiliza el valor de la clase verdadera, ya que será la que tenga $y_i = 1$, mientras que el resto se anularán, siendo computacionalmente muy eficaz. Esto es así porque los vectores de etiquetas vendrán dados en un formato denominado *one-hot*, donde cada uno de estos vectores será todo cero, menos la posición correspondiente a la clase verdadera, que tendrá valor uno.

En principio, si la red funciona correctamente, la salida correspondiente a la clase verdadera tendrá un valor muy cercano a 1, mientras que el resto serán valores cercanos a 0, sumando las 10 salidas un total de 1, es decir, la probabilidad total [31].

3.4.2 Accuracy

En redes neuronales cuyo fin es la clasificación, el acierto es una medida que permite entender con facilidad si el funcionamiento de la red es el esperado o no, especialmente para los clasificadores de problemas de visión por ordenador.

El acierto calcula como de bien en media, unas medidas concuerdan con los valores que la red predice, es decir, da como resultado una proporción. Por simplicidad se va a formular para un clasificador binario, donde las clases posibles son 0 (negativa) y 1 (positiva). Para el cálculo de esta medida hace faltan unos parámetros:

- Verdadero positivo (TP): Cuando el clasificador predice 1, siendo la clase verdadera 1.
- Falso positivo (FP): Cuando el clasificador predice 1, siendo la clase verdadera 0.
- Verdadero negativo (TN): Cuando el clasificador predice 0, siendo la clase verdadera 0.
- Falso negativo (FN): Cuando el clasificador predice 0, siendo la clase verdadera 1.

De esta manera la ecuación del acierto para la clasificación binaria queda:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

Es importante señalar una verdad que aun siendo sencilla a menudo se pasa por alto: Es posible clasificar un conjunto de datos finito prácticamente sin errores, especialmente durante el entrenamiento pero el verdadero problema es diseñar un clasificador que garantice un rendimiento adecuado en los datos futuros [32]. Es por ello que si bien el acierto de entrenamiento es interesante para ver como la red se va adaptando a los datos, el valor que interesa para poder medir de manera fiable al clasificador es el acierto sobre el conjunto de datos de test.

3.4.3 Matriz de confusión

La matriz de confusión es una herramienta visual muy utilizada en los problemas de clasificación. Hace uso de los parámetros definidos en el apartado anterior: TP, TN, FP y FN.

Una matriz de confusión resume el rendimiento de clasificación de un clasificador con respecto a unos datos de prueba. Es una matriz bidimensional, indexada en una dimensión por la clase verdadera de un objeto y en la otra por la clase que el clasificador asigna [33].

Por simplicidad, se vuelve a ejemplificar el caso binario, tal y como se puede visualizar en la Figura 3-8. Lo que se consigue es una matriz cuyas filas hacen referencia a las clases verdaderas y sus columnas a las clases asignadas por el clasificador. Se formula tal que:

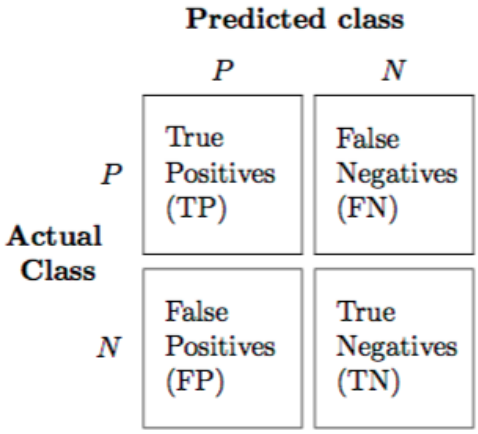


Figura 3-8. Matriz de confusión para un clasificador binario [34]

Esta matriz de confusión se puede extender para el número de clases totales del clasificador, en este caso 10 clases. De esta forma, visualmente se puede comprobar en qué proporción cada clase se clasifica correctamente, lo que corresponderá con la diagonal de la matriz. Además lo interesante es que muestra la

proporción de ejemplos mal clasificados, y para cada una de las clases verdaderas qué clases erróneas han sido asignadas por el clasificador del conjunto de ejemplos.

3.5 Optimizador

Ya se ha mencionado el algoritmo de descenso del gradiente como optimizador usado por la red neuronal para minimizar la función de coste, buscando de manera iterativa el mínimo de dicha función. El uso del descenso del gradiente fue un gran avance en el campo del Aprendizaje Máquina, pero hay algoritmos de optimización más avanzados. Este se trata de un algoritmo lento y con varios problemas entre los que cabe destacar cuando el optimizador se queda en un mínimo local de la función en funciones no convexas.

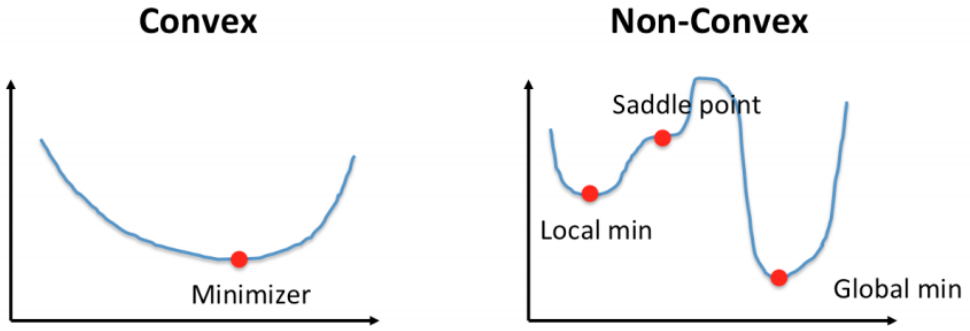


Figura 3-9. Problema de mínimos locales en funciones no convexas [8]

Este problema depende del punto inicial de minimización, que normalmente es aleatorio. La librería *Keras* ofrece una amplia variedad de clasificadores como funciones, entre los que cabe destacar: *SGD* (el descenso del gradiente), *Adam* y *RMSprop*. Para todos ellos además pueden especificarse distintos parámetros como, entre otros, el factor de aprendizaje.

Para distintos problemas supervisados de clasificación en el campo del Aprendizaje Máquina se han hecho uso de los algoritmos *Adam* y *RMSprop* con buenos resultados. En varias pruebas en famosas bases de datos como *MNIST*, *Cifar10*, *Cifar100*, *FashionMNIST*, el análisis de los resultados muestra que el algoritmo de optimización *Adam* funciona mejor que los demás en la fase de prueba y *RMSprop* y *Adam* en la fase de entrenamiento [35]. En este trabajo se han hecho pruebas con estos dos últimos clasificadores, pero al final se ha optado por *RMSprop* porque daba mejores resultados.

3.6 Arquitectura de red

Como se ha comentado previamente, la red a crear toma como entradas unos vectores de características que vienen como resultado de vídeos que han sido procesados previamente por una compleja red convolucional, por lo que se debe crear una red densamente conectada que tome y aprenda las características extraídas por las capas convolucionales hasta aprender a clasificar entre las 10 posibles clases.

Para decidir los hiperparámetros que conforman a la red, en este trabajo de clasificación de vídeo se han tomado principalmente dos enfoques diferentes a la hora de realizar dicha arquitectura de red densamente conectada. El primero objetivo es maximizar el porcentaje de acierto del clasificador, buscando para ello la arquitectura más completa, independientemente de la complejidad de la misma. Para ello se han ido realizando distintas pruebas variando el número de capas y de neuronas por cada una de ellas, además de ir probando con distintas funciones de activación no lineales.

Al recibir la red como entradas dos tipos diferentes de características, audio y vídeo, se plantea que la arquitectura tenga primero una serie de capas que se especialicen en cada tipo de entrada distinto, para después concatenarlas en una sola capa, e ir profundizando en la red hasta la capa final *softmax*. También se han hecho pruebas concatenando ambos tipos de entradas como una sola entrada, recibiendo la red de esta manera por vídeo un vector de dimensión 1152.

El otro objetivo ha sido minimizar la complejidad computacional, logrando que la red pueda entrenarse en

menos tiempo, pero consiguiendo aun así un buen porcentaje de acierto. Se ha encontrado que el mejor enfoque pasa primero por ir creando una red simple, que de buenos resultados y a partir de ahí ir añadiendo más capas y neuronas, haciendo la red más compleja buscando si hay una mejora del resultado.

Como también se ha mencionado al explicar la base de datos utilizada, hay etiquetas que tienen un porcentaje de vídeos muy alto del total utilizado, en comparación con otras. Por ello se han hecho pruebas tanto usando esta proporción de vídeos como utilizando el mismo número de vídeos por clase, usando para ellos el número de vídeos de la etiqueta que menos tiene. De esta manera se consiguen dos cosas, primero al usar menos vídeos la red se entrena más rápido y requiere menor carga computacional, y además se comprueba si en el caso de dejar la proporción inicial la red no "memoriza" y asigna mayor proporción de salidas correctas a las clases con mayores probabilidades sin realmente aprender de sus características.

4 ENTRENAMIENTO DE LA RED Y RESULTADOS

En este capítulo se irán explicando los distintos pasos dados y los resultados obtenidos. Se irán viendo las redes creadas hasta llegar a la que mejores resultados han obtenido, buscando los objetivos fijados de minimizar la carga computacional y tiempo de entrenamiento, pero maximizando el porcentaje de acierto del clasificador.

Para el entendimiento del trabajo se van a ir explicando mediante fragmentos de código los pasos tomados y los cambios. Se ha decidido minimizar el uso de código en la memoria a solo unos fragmentos para el entendimiento de las librerías usadas y el procedimiento a seguir, en general, para la construcción de cualquier algoritmo de red neuronal. Se dejará en un anexo el código completo de la arquitectura final.

Este procedimiento puede resumirse en los siguientes tres pasos: Primero se hace un procesamiento de los datos, preparándolos como vectores que puedan ser utilizados como entradas a la red. Segundo se crea la estructura de la red, haciendo uso de librerías como *Keras* o *TensorFlow*, que facilitan el procedimiento, indicando el número de capas, dimensiones de los vectores de entrada, funciones de activación y procedimientos para mitigar el sobreajuste. Tras esto se produce el entrenamiento, indicando el optimizador y funciones de pérdida. Durante el entrenamiento de la red se visualizarán las pérdidas y el acierto y, si la red brinda un resultado interesante, se podrá probar sobre datos no usados en el entrenamiento.

Al no tener experiencia previa en el campo del Aprendizaje Máquina, antes de abordar el problema del trabajo, se utilizaron las librerías *Pandas*, *Numpy* y *Matplotlib* para probar distintas redes con problemas más sencillos, como primero problemas de regresión lineal, quizás los más sencillos. Tras ello se avanzó tratando el conocido problema de clasificación de caracteres de la base de datos *MNIST*, lo que permitió sentar un punto de partida para poder entender el problema de clasificación de vídeo.

Como herramientas para iniciarse en el mundo del Aprendizaje Máquina y Aprendizaje Profundo, cabe destacar por su sencillez y su atractivo didáctico, el libro escrito por el creador de la librería *Keras*, François Chollet [3], famoso ingeniero de *Google* dentro del campo del Aprendizaje Máquina. Siguiendo su libro, haciendo uso de la librería creada por él, permite a cualquier persona sin formación previa en matemáticas, ingeniería ni programación, un primer acercamiento a este campo. Además también dentro de la comunidad hispanohablante, existe un canal de *YouTube*, que ha servido para asentar una base tanto teórica como práctica de este campo, además de hacerlo aún más atractivo, denominado DotCSV [36]. Fue creado por el ingeniero informático Carlos Santana, divulgador de Inteligencia Artificial, y tiene varios vídeos dedicados a explicar conceptos como el de la neurona, redes neuronales, redes convolucionales y muchos otros que se relacionan con este trabajo. Además acompaña los vídeos teóricos de otros explicando cómo llevar la teoría al código, siempre de una manera sencilla pero potente.

4.1 Procesamiento de datos

Los primeros pasos para la realización de cualquier proyecto de Aprendizaje Máquina tras determinar el problema y herramientas de programación a utilizar es escoger la plataforma para escribir y probar el código. En este caso al utilizar un sistema Linux, se ha optado por programar en *Python* utilizando el editor de textos *Sublime Text*, por la comodidad que aporta. Para la ejecución del código, una vez instalado *Python* y todas las librerías a usar, se ha hecho uso de la terminal Linux de manera que se ha desestimado instalar otras aplicaciones adicionales como entorno de desarrollo.

Todas las pruebas requieren de la importación de las distintas librerías de *Python* (Código 4-1), y el uso de funciones de *TensorFlow* y *Keras*.

```
import os
import numpy as np
import tensorflow as tf
from keras import models
from keras import layers
import keras
from keras import Input
from keras.models import Model
import matplotlib.pyplot as plt
```

Código 4-1. Importación de librerías

YouTube-8M es una enorme y compleja base de datos, y para las distintas competiciones que *Google* ha propuesto haciendo uso de dicha base de datos, la compañía ha aportado los distintos vídeos almacenados en formato .tfrecord, haciendo uso para ello de funciones de la librería *TensorFlow* (Código 4-2) para la extracción de los datos.

Este tipo de archivo almacena en un formato simple una secuencia de registros binarios. Se trata de un búfer de protocolos para la serialización eficiente de datos estructurados. Cada mensaje representa una asignación de valores a unos identificadores.

Se han descargado los vídeos y se han almacenado en el directorio de trabajo según la siguiente estructura:

-/TFG/data/train: Archivos para entrenamiento

-/TFG/data/validate: Archivos para validación

-/TFG/data/test: Archivos para test

-/TFG/tfg.py: Cada modelo ha sido guardado en un fichero *Python* para su ejecución

```
for i in ficheros:
   record = './data/train/'+i
   labels = []
   mean_rgb = []
   mean_audio = []

   for example in tf.compat.v1.python_io.tf_record_iterator(record):
       result = tf.train.Example.FromString(example)
       labels.append(result.features.feature['labels'].int64_list.value)
       mean_rgb.append(result.features.feature['mean_rgb'].float_list.value)
       mean_audio.append(result.features.feature['mean_audio'].float_list.value)
```

Código 4-2. Extracción de datos de archivos .tfrecord

De esta manera se extraen y se guardan en listas para todos los ficheros utilizados los tres tipos de datos distintos: etiquetas, características de vídeo y audio, respectivamente almacenados como enteros y flotantes. La dimensión del vector de etiquetas por cada vídeo dentro de cada fichero es variable, mientras que los vectores de vídeo y audio se corresponden siempre con dimensiones 1024 y 128.

Lo que se consigue mediante el Código 4-2 es extraer de los archivos los datos, pero estos deben ser procesados antes de poder ser introducidos a la red neuronal. Las etiquetas deben estar en formato *one-hot*, ya que es la manera en que las redes puedan procesarlas. De esta forma en el Código 4-3 se muestra como por ejemplo para el vídeo que tenga asignado la etiqueta con valor 0, hay que asignarle un vector de ceros de dimensión 10 pero con un 1 en la primera posición. De la misma manera se procede para cada etiqueta en cada vídeo, donde se van guardando en la lista etiquetas los vectores de etiqueta correspondientes a cada vídeo. Además en el mismo orden se guarda en otras listas las características de vídeo y audio como listas Se asigna además un variable por cada tipo de etiqueta, en el ejemplo num_0, para contabilizar el número total de vídeos que se utiliza por clase.

```
etiquetas = []
audio = []
video = []

for i in range(len(labels)):
     if(labels[i][0] == 0):
          num_0 +=1
          etiquetas.append([1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
          audio.append(list(mean_audio[i])
          video.append(list(mean_rgb[i]))
```

Código 4-3. Procesado de entradas

Todo este procedimiento se hace por cada fichero, y al final se guardan los conjuntos de datos conseguidos en unas listas para entrenamiento (X_train_video, X_train_audio, Y_train) y validación (X_validate_video, X_validate_audio, Y_validate), donde las X corresponden con las entradas a la red y las Y con las etiquetas de salida.

4.2 Creación de las redes

Una vez se tienen almacenados en memoria las entradas y salidas esperadas de una manera que puedan ser utilizadas por una red neuronal, se procede a la creación de la misma. Para ello se usa la librería *Keras*, donde el proceso para todas las redes creadas se repite. Primero en todas las redes se definen las capas de entradas que son de tipo *Input*, donde se puede tomar la opción de una entrada para cada tipo de característica o una sola entrada que sea la unión de ambas. Después se utilizan las funciones *Dense* para ir apilando capas ocultas, donde el número de capas y neuronas son los principales parámetros a definir y lo que diferencia las arquitecturas propuestas. Además se pueden usar diferentes funciones de activación que aporta *Keras* directamente, aunque la capa de salida debe ser siempre una de tipo *softmax* para realizar la labor de clasificación.

Keras permite guardar una imagen de la arquitectura creada, que será de gran utilidad para ver la diferencia entre ellas sin tener que enseñar el código de cada una, ya que es bastante repetitivo.

Tras la arquitectura se define el optimizador junto a la medida que se usará, el acierto, y se procede al uso de la función *fit*, que es la que realiza el entrenamiento de la red, a la vez que va mostrando el progreso y los valores de los parámetros al terminar cada época. En principio se van a usar 20 épocas en cada entrenamiento, para una mejor visualización de los resultados.

```
video_input = Input(shape=(1152,))
hidden_1_video = layers.Dense(108, activation='relu')(video_input)
salida = layers.Dense(10,activation='softmax')(hidden_2_video)

model = Model(video_input,salida, name="tfg_v1")
print(model.summary())
keras.utils.plot_model(model, 'tfg_v1.png', show_shapes=True)

model.compile(optimizer="rmsprop", loss="categorical_crossentropy",
metrics=["acc"])
historia = model.fit(X_train, Y_train, epochs=20, batch_size=256,
validation_data=(X_validate,Y_validate))
model.save('tfg_v1.h5')
```

Código 4-4. Creación de arquitectura de red neuronal densamente conectada

El ejemplo del Código 4-4 fue el primer modelo de arquitectura que se probó. Se empezó con uno sencillo de solo una capa oculta *Dense* de 108 neuronas, con función de activación *ReLu*, ya que es de las más utilizadas.

Para cada tipo de arquitectura se han ido probando los distintos hiperparámetros para buscar el mejor resultado. Estos son el número de neuronas en cada capa, el factor de aprendizaje, el número de capas ocultas, la función de activación de cada capa o el uso de *Dropout*. El número de pruebas hechas ha sido enorme, ya que hay una gran cantidad de distintas combinaciones entre estos hiperparámetros, pero en la memoria del trabajo se van a reflejar por simplicidad la evolución que se ha seguido, y la mejor arquitectura conseguida para cada tipo.

4.3 Visualización de resultados

Para visualizar los resultados en cada arquitectura, se hace uso de la librería *Matplotlib*, que permite realizar gráficas. Se visualizará el progreso tanto de la pérdida como del acierto en cada arquitectura durante el entrenamiento.

En principio se visualizará solo la pérdida, ya que nos permitirá ver si se produce subajuste o sobreajuste, y si los resultados son lo suficientemente buenos se visualizará también el acierto y se utilizará el modelo entrenado, cuyos parámetros se guardan en un fichero .h5, para utilizarlo en el set de datos de test, es decir, valores de vídeos que la red no ha visto durante el entrenamiento, para visualizar su matriz de confusión.

```
historia_dict = historia.history
acc = historia_dict['acc']
val_acc = historia_dict['val_acc']
val_loss = historia_dict['val_loss']
loss = historia_dict['loss']
epochs = range(1,len(acc)+1)
plt.plot(epochs, acc, 'g', label='Acierto en entrenamiento')
plt.plot(epochs, val_acc, 'b', label='Acierto en validación')
plt.title('Exactitud en entrenamiento')
plt.xlabel('Epocas')
plt.ylabel('Exactitud')
plt.legend()
plt.show()
plt.plot(epochs, loss, 'g', label='Perdida en entrenamiento')
plt.plot(epochs, val_loss, 'b', label='Perdida en validación')
plt.title('Perdida en entrenamiento')
plt.xlabel('Epocas')
plt.ylabel('Perdida')
plt.legend()
plt.show()
```

Código 4-5. Visualización del rendimiento de la red

En cada gráfica diferenciará por el color verde a los resultados correspondientes con el entrenamiento y en azul al correspondiente con la validación.

4.4 Evolución de las arquitecturas

La primera arquitectura que se crea es de una sola capa oculta, y con pocas neuronas. Toma como entrada directamente por cada ejemplo todas sus características, tanto de vídeo como audio, y las introduce en la red. Se ha tenido en cuenta que en este tipo de arquitectura el número de neuronas en una capa debe ser mayor que el número de neuronas de la salida, ya que de no ser así, se produciría una pérdida de información. Además, es una buena práctica intentar que la arquitectura sea lo más simple posible, ya que conforme más parámetros tiene, mayor carga computacional será necesaria. Además, cuantas menos neuronas se utilicen, estas, para conseguir clasificar correctamente los distintos vídeos, tienen que generalizar más diferentes conceptos. Si a una arquitectura se van añadiendo más neuronas, cada una de ella va especializándose en conceptos más específicos, por lo que se pierde cierta generalidad, y quizás funcione mejor con el conjunto de datos de entrenamiento, cuya salidas la red va conociendo y se va adaptando a ella, que con un conjunto de muestras cuyas salidas no conozca.

El tiempo que necesita el ordenador para procesar los datos de los archivos y guardarlos para poder ser introducido en la red es de unos 20 minutos, y ya conlleva una gran carga computacional, por lo que es aún más importante trabajando con unos datos de esta naturaleza buscar la simplicidad de la red para el entrenamiento. Se utilizan unos 312.000 vídeos para entrenamiento y unos 140.000 para validación.

Los resultados en la pérdida de esta primera arquitectura (Modelo 1) demuestran que se produce rápidamente un sobreajuste enorme, ya a partir de prácticamente la tercera época la pérdida de la validación crece a un elevado ritmo. Por ello se procede al cambio de otros de los hiperparámetros, la función de activación, probando con otra función no lineal muy importante en el campo del Aprendizaje Máquina, la tangente hiperbólica (*tanh*). Se puede ver bastante mejora en los valores de la pérdida, por lo que se seguirá usando esta función en vez de la *ReLu*, pero se sigue cumpliendo el mismo comportamiento.

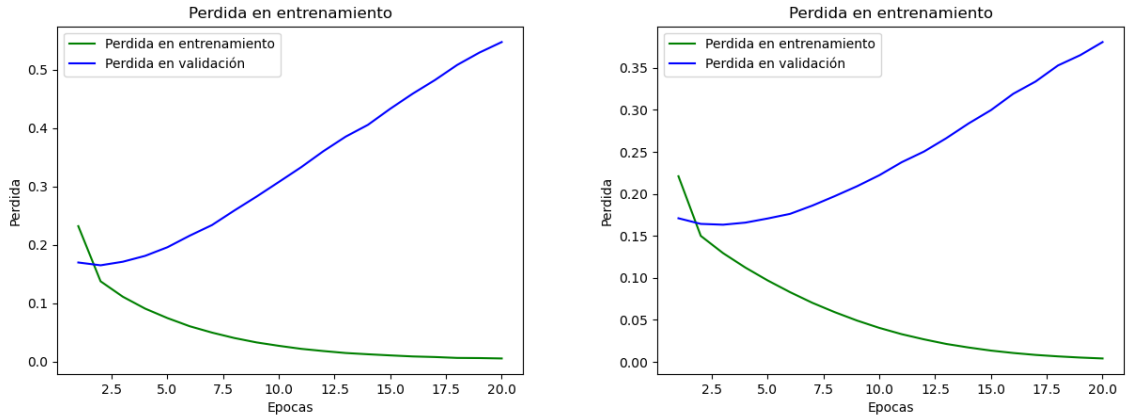


Figura 4-1. Pérdida en el Modelo 1 con una sola capa oculta con *ReLu* y con *tanh*

Por ello se introduce la técnica de *Dropout* con una tasa de 0.5, de manera que con la misma única capa oculta y el mismo número de neuronas se consigue una gran mejora, ya que se suaviza la curva de la pérdida de validación.

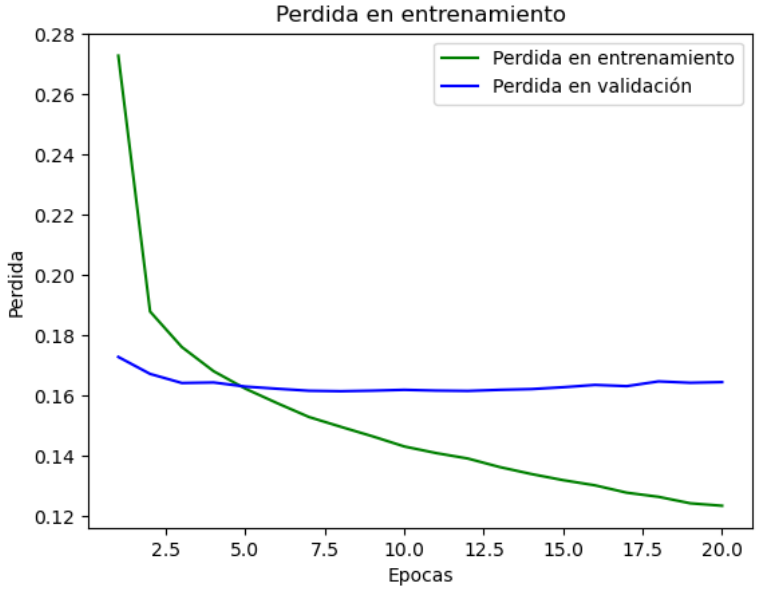


Figura 4-2. Pérdida en Modelo 2 de una sola capa con la introducción de *Dropout*

Este modelo (Modelo 2), aun siendo muy simple, consigue en la clasificación un 92% de acierto en validación, porcentaje bastante bueno para la poca carga computacional consumida. Acaba prácticamente con el sobreajuste, y tarda únicamente 6 épocas en estabilizarse del todo.

Tras el primer modelo y su mejora, se decide tener en cuenta la naturaleza de las entradas, ya que son de diferente tipo. Se sigue teniendo una arquitectura con una sola capa oculta, pero esta vez una para los datos de vídeo y otra distinta para los datos de audio (Modelo 3). Después de capa *Dense* desde el primer modelo siempre se utiliza la técnica *Dropout* para evitar el sobreajuste.

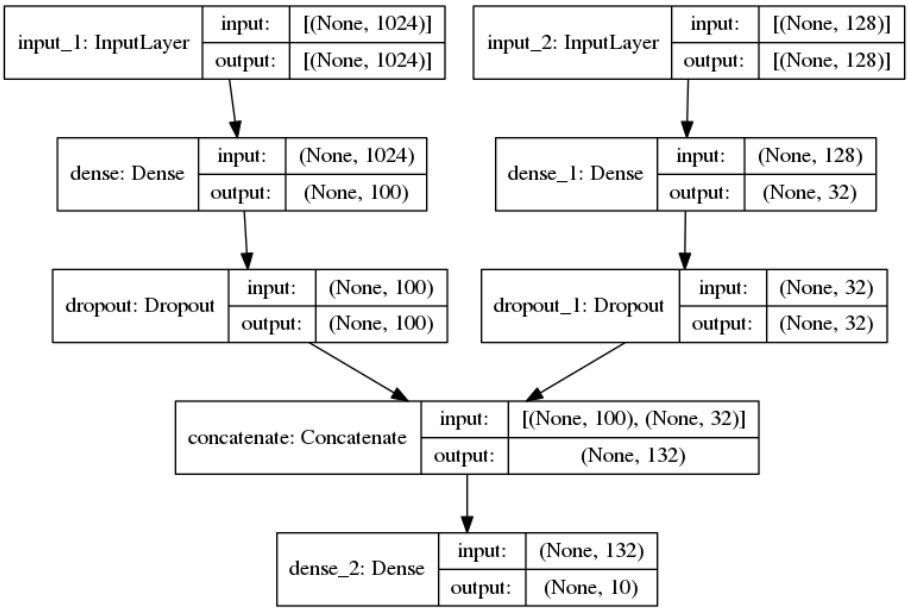


Figura 4-3. Modelo 3 con una capa oculta para cada tipo de entrada

La idea es que quizás cada una de las capas podría especializarse en características diferentes según la naturaleza de la entrada, y de esa forma lograr mejores representaciones y aumentar el porcentaje de acierto en la clasificación. El resultado es que si bien la pérdida no sufre de sobreajuste, aumenta mínimamente su valor.

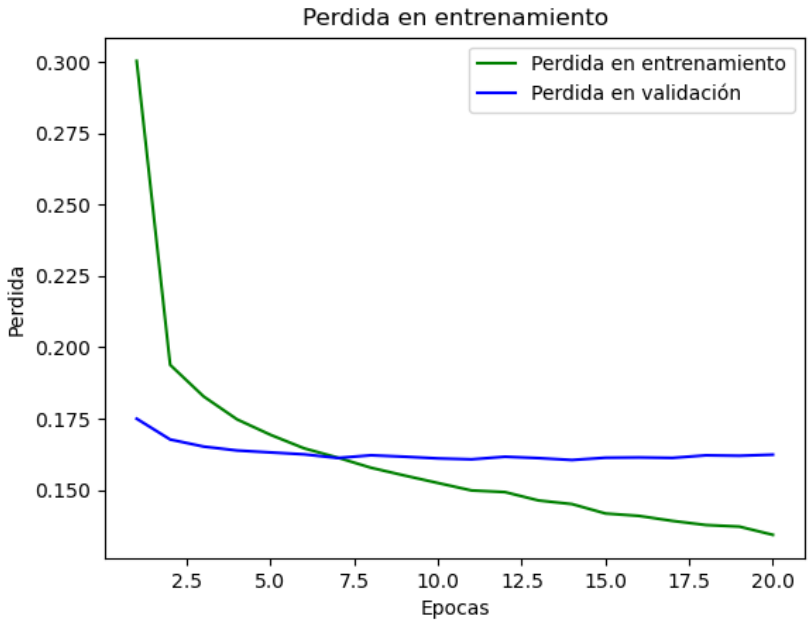


Figura 4-4. Pérdida en el Modelo 3

Si bien no se logra el objetivo que se busca, sigue siendo una buena posibilidad, por lo que se hace uso de otro hiperparámetro, el número de capas ocultas de la red. El siguiente paso es pues tomar dos capas ocultas para la red. De nuevo se han probado numerosas combinaciones de número de neuronas por capa hasta llegar al mejor resultado posible.

Hay dos posibilidades a la hora de usar una segunda capa oculta. La primera sería una segunda capa oculta por cada una de las entradas (Modelo 4), mientras que la otra posibilidad es usar una capa oculta después de concatenar las primeras capas ocultas de cada entrada (Modelo 5).

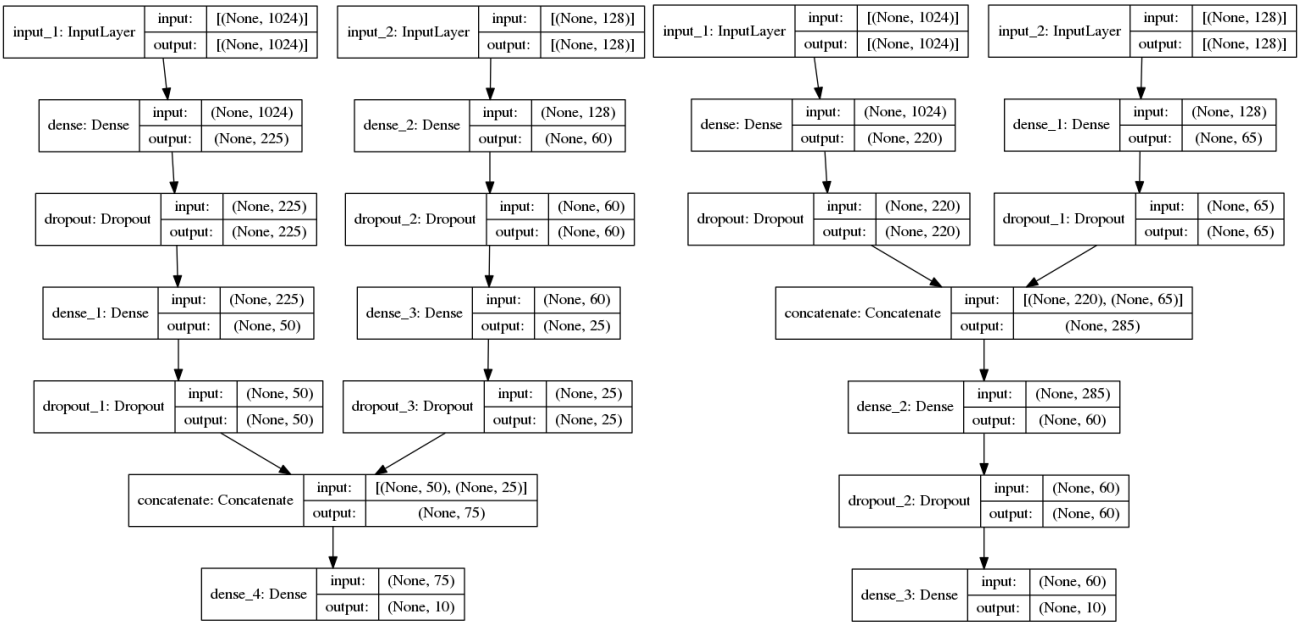


Figura 4-5. Modelos 4 y 5 al añadir una segunda capa oculta

Los resultados se muestran en la Figura 4-6 en el mismo orden que los modelos de la anterior.

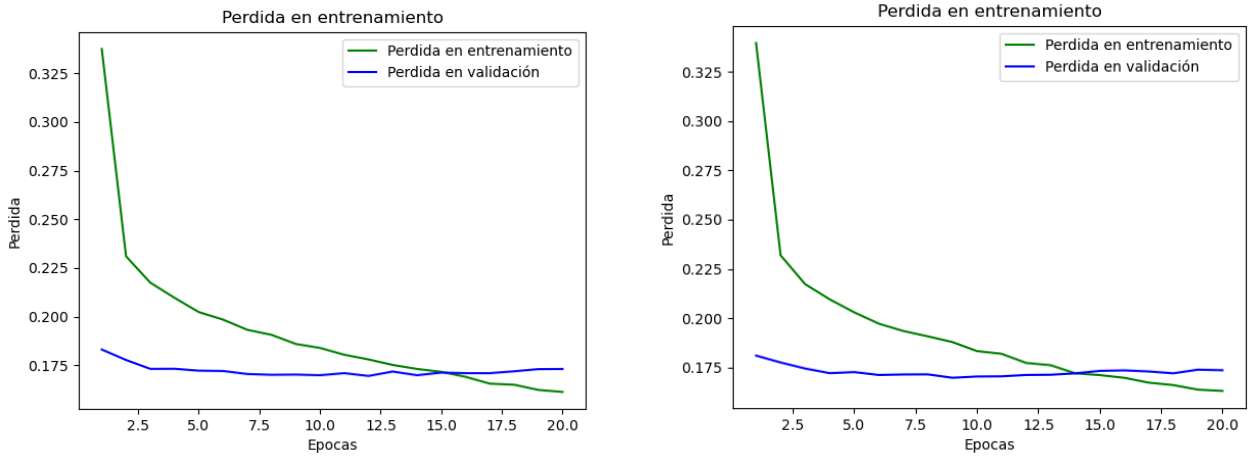


Figura 4-6. Resultados de los Modelos 4 y 5 con dos capas ocultas

Ambos resultados son muy similares y demuestran que aun añadiendo otras capas ocultas la pérdida se incrementa. Pero a la hora de visualizar el acierto, este ha aumentado a más del 94%, una mejora de un 2% con respecto al modelo de una sola capa oculta. El mejor resultado se da, por muy poco, en el caso en el que la segunda capa oculta se introduce tras la concatenación de las primeras, caso que se muestra en la Figura 4-7.

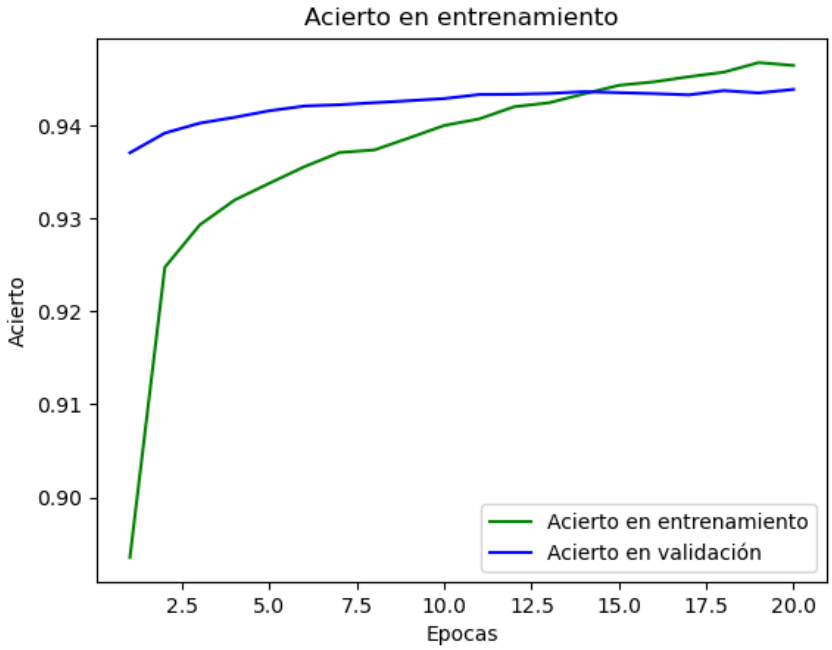


Figura 4-7. Acierto en el modelo con una segunda capa oculta tras concatenar las primeras

En el resultado se ve que el acierto de la validación es rápidamente muy alto, aunque hasta la época 12 no se termina de estabilizar. En la gráfica de la pérdida es en la época 12 donde se da a su vez el mínimo, aunque después crece levemente.

Para finalizar con el aumento de capas, se comprueba qué sucede al añadir una tercera capa oculta (Modelo 6), pero tras varias el número de neuronas en varias casos, el resultado es siempre un aumento enorme del error en comparación con el modelo anterior, un disminución del acierto a menos del 93%, y encima, al tener más parámetros, un aumento considerable de la carga computacional.

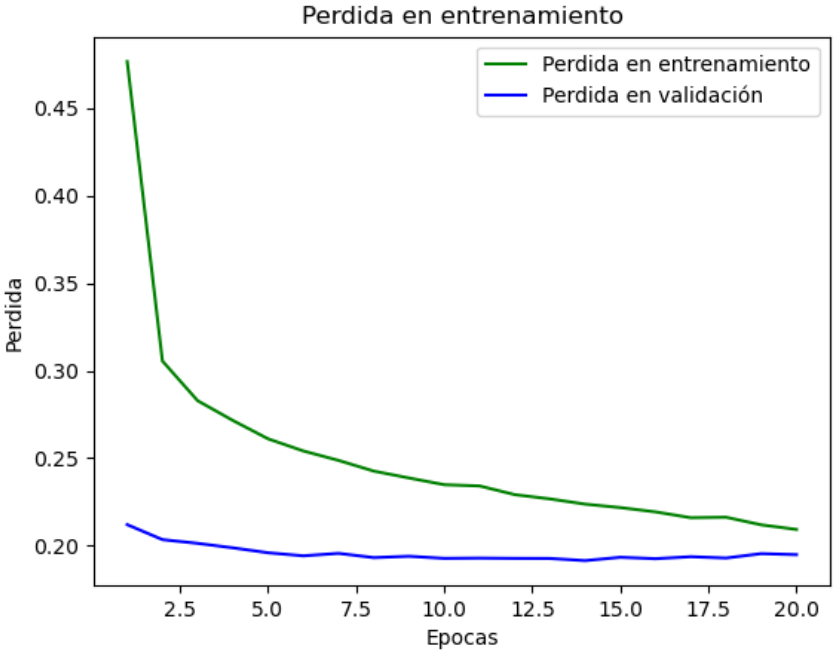


Figura 4-8. Pérdida en el Modelo 6 con tres capas ocultas

Por último se intenta tener en cuenta, para el mejor modelo conseguido (Modelo 5), tener en cuenta la naturaleza de la base de datos. Como se ha mencionado en la explicación de la misma, hay etiquetas que, en los conjuntos de vídeos utilizados para el entrenamiento y validación, aparecen en una mayor proporción que otras. La diferencia es notable ya que la clase que más veces aparece lo hace en un 34% del total aproximadamente, mientras que la que menos aparece, en un 4%.

Lo que se propone es comprobar si al tener más probabilidades de aparecer unas clases con respecto a otras, la red de alguna manera aprenda que es más probable asignar las salidas a estas clases sin realmente aprender de las características de todas ellas.

Por ello se pone como límite por clase el número total de vídeos de la clase que menos aparece, para tener en todas las clases el mismo número de ejemplos. De esta manera se pasa a aproximadamente 111.000 vídeos para entrenamiento y 51.100 para validación. Esto permite que al utilizar menos ejemplos para entrenamiento y validación la red se entrene mucho más rápido, dando los resultados ofrecidos en la Figura 4-10.

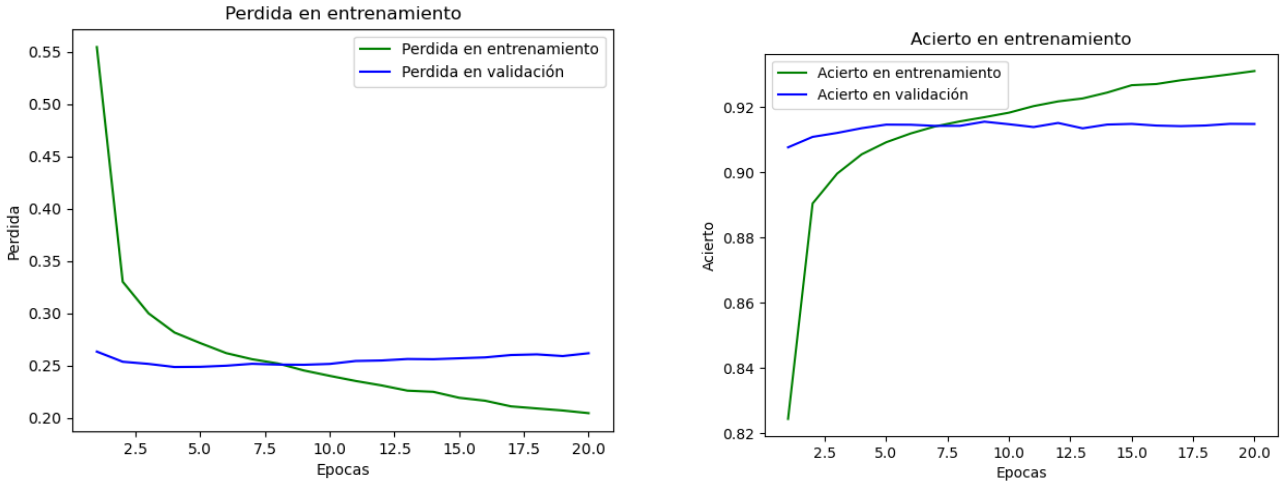


Figura 4-9. Pérdida y validación del mejor Modelo 7 con mismo número de vídeos por etiqueta

De esta forma se visualiza que la pérdida crece mucho, pasando de 0.175 aproximadamente a 0.25 en la mejor época. Además el acierto pasa de más de un 94% a estar por debajo del 92%, todo esto con el mismo número de capas, neuronas por capas y función de activación.

Caso de uso	Mejor Valor Pérdida	Porcentaje de acierto	Época
Modelo 5	0.168	94.2%	9
Modelo 7	0.25	91.6%	5

Tabla 4-1. Resultado al igualar el número de vídeos por etiqueta

Si bien la arquitectura de los modelos 5 y 7 es la misma respecto a número de capas y neuronas, se toman como modelos distintos porque al ser entrenados con distintos conjuntos de datos, los parámetros que las redes aprenden son diferentes, por lo que a la hora de usarse con datos nuevos, darán resultados diferentes.

Pueden utilizarse los parámetros guardados de los modelos de ambos casos, que están como ficheros .h5, para realizar sobre el conjunto de test pruebas para ver como los modelos se comportan en la realidad al recibir como entradas datos que nunca han visto.

4.5 Matrices de confusión

El conjunto de test se compone por unos 46.000 vídeos y los resultados se pueden apreciar mediante el uso de la herramienta visual de la matriz de confusión.

```
modelo = keras.models.load_model('modelo5.h5')

y_predict = modelo.predict([X_train_video,X_train_audio])
y_test = Y_train
y_test_non_category = [ np.argmax(t) for t in y_test ]
y_predict_non_category = [ np.argmax(t) for t in y_predict ]

from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test_non_category, y_predict_non_category)

plt.title("10 labels Confusion Matrix")
sns.heatmap(conf_mat,annot=True,fmt="d")
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.show()

resultados = modelo.evaluate([X_train_video,X_train_audio],Y_train)
```

Código 4-6. Matriz de confusión y evaluación sobre conjunto de test

Se han representado dos matrices de confusión para el Modelo 5 y dos para el Modelo 7. Para cada modelo primero se ha representado la matriz utilizando el conjunto de test con igual número de vídeos por etiqueta, y la segunda con el número de vídeos por etiqueta que originalmente se encuentra en la base de datos, de manera que habrá clases con muchos más vídeos que otras.

Los resultados respecto al porcentaje de acierto se encuentran en la siguiente tabla:

	Modelo 5	**Modelo 7**
Igual número de vídeos	90.65	91.34
Distinto número de vídeos	94.26	93.03

Tabla 4-2. Resultado sobre el conjunto de test en modelos 5 y 7

Es decir, el modelo 5, empeora bastante al igualar el número de vídeos por etiqueta, pero aunque el modelo 7 da mejor resultados en este caso la diferencia es solo de 0.69%, mientras que si se usase en el caso de distinto número de vídeos por etiqueta, la diferencia entre ambos modelos sube al 1.23%, siendo mejor el Modelo 5 por lo que probablemente sería mejor opción utilizar el Modelo 5 en ambos casos.

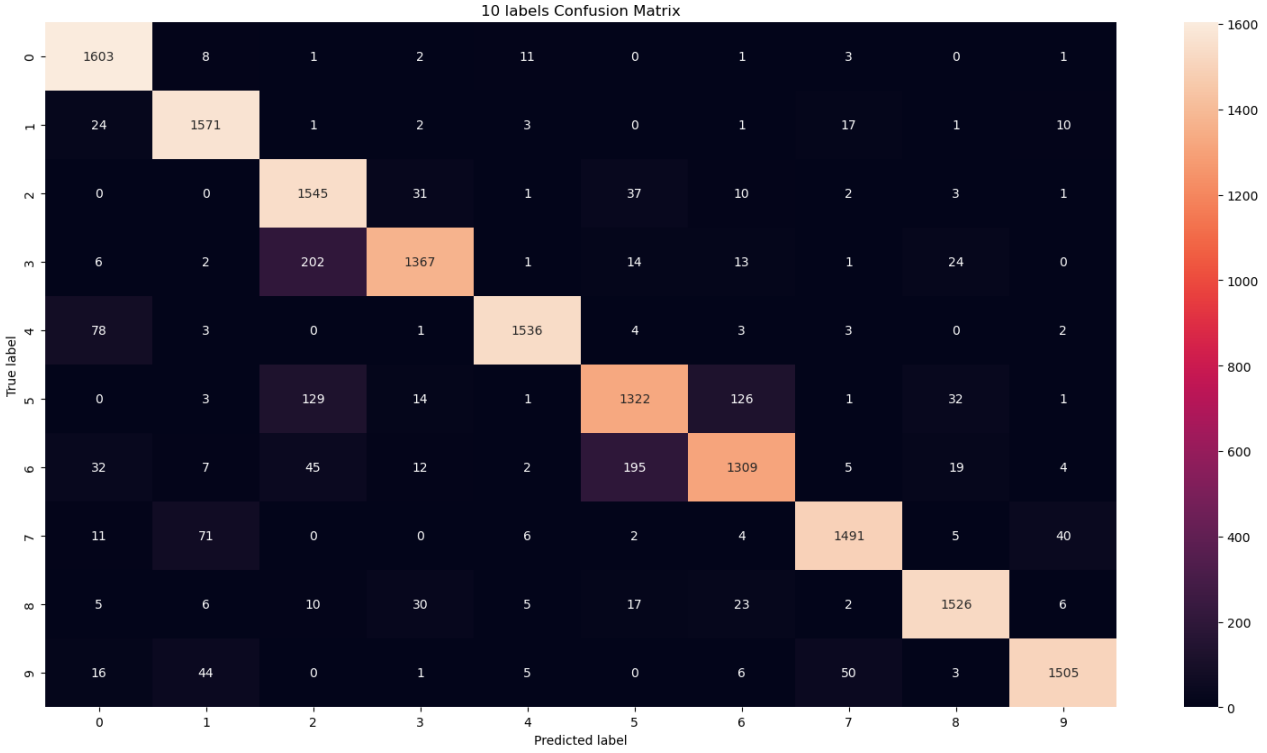


Figura 4-10. Matriz de confusión del Modelo 5 con igual número de vídeos por etiqueta

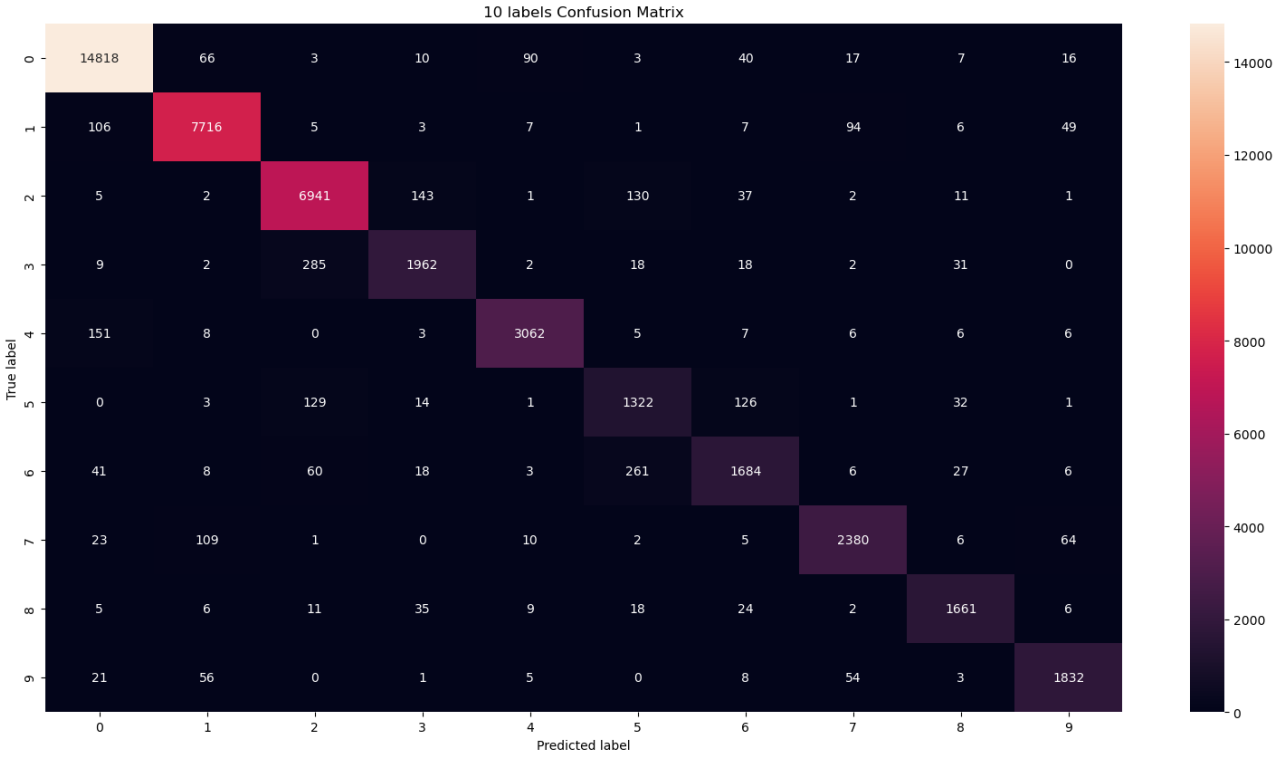


Figura 4-11. Matriz de confusión del Modelo 5 con distinto número de vídeos por etiqueta

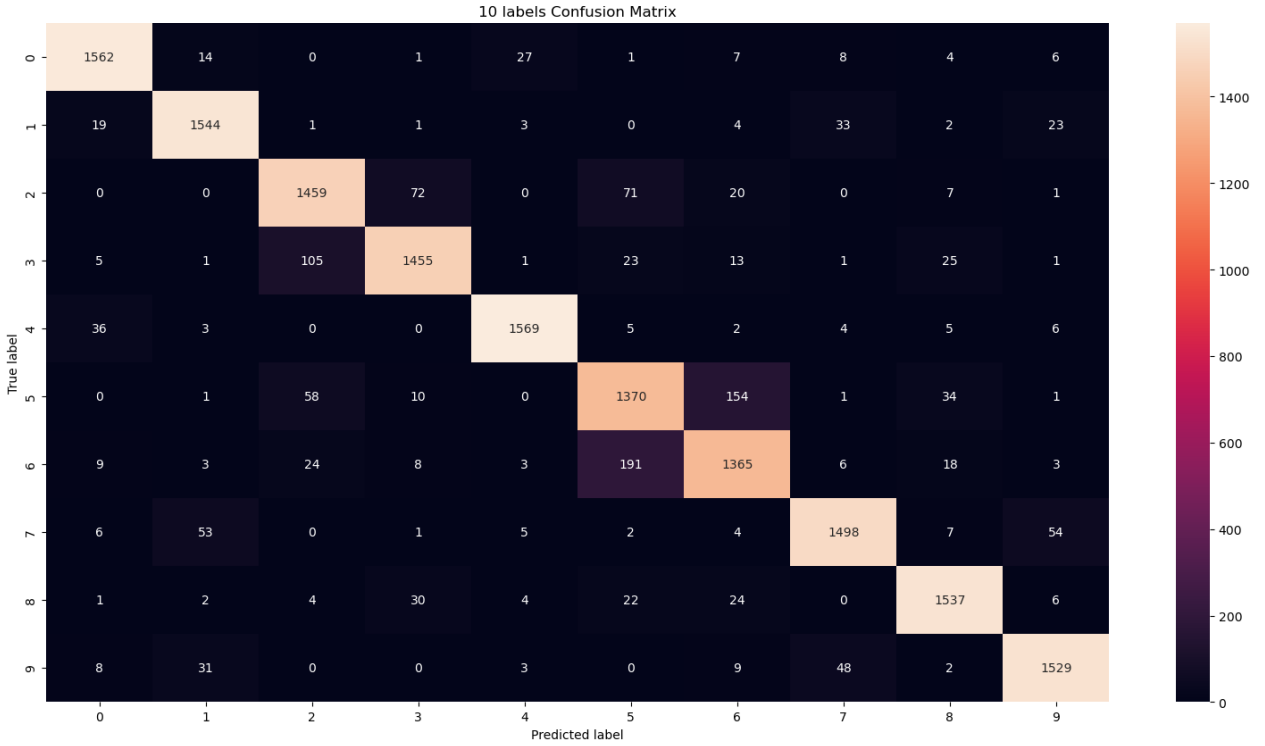


Figura 4-12. Matriz de confusión del Modelo 7 con igual número de vídeos por etiqueta

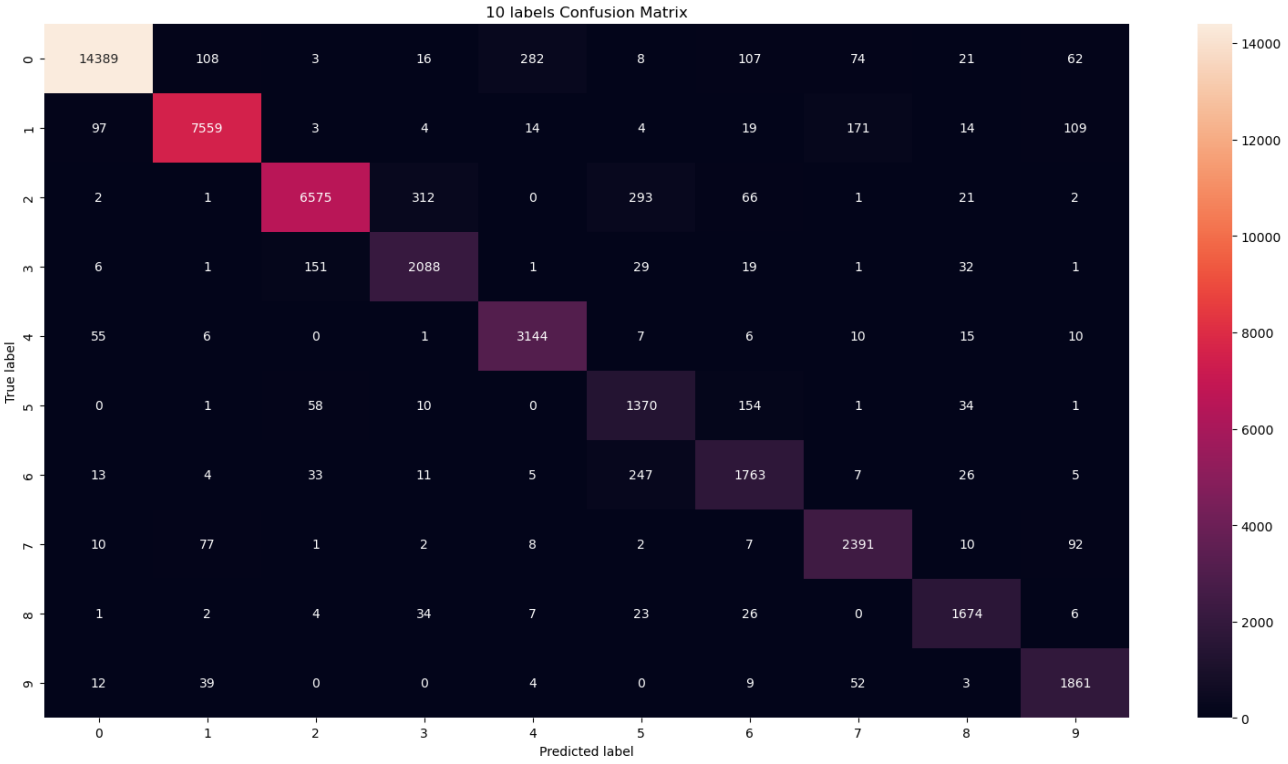


Figura 4-13. Matriz de confusión del Modelo 5 con distinto número de vídeos por etiqueta

En las matrices en general se puede observar un correcto funcionamiento de los modelos, más allá de que uno sea mejor que el otro. La diagonal indica los vídeos que han sido correctamente etiquetados por los clasificadores, y es con diferencia la que tiene la mayor cantidad de vídeos. Quizás es más fácil de verlo en el caso en que el número de vídeos por etiqueta es el mismo, que aunque el porcentaje de acierto empeora, al tener el mismo color por tener parecido número de etiquetas correctas, se visualiza mejor el comportamiento de la diagonal.

Además la matriz de confusión permite ver para cada clase cuál es la clase con la que el clasificador más se equivoca. Por ejemplo con la etiqueta 1, que corresponde con la clase "Vehicle", el clasificador se equivoca más clasificándolo con "Game". Puede ser porque hay muchos vídeos de juegos en *YouTube* que tienen vehículos en ellos.

Resulta interesante ver que para ambos modelos se repite un comportamiento. Tanto para el mismo número de vídeos por etiqueta como distinto, en el Modelo 5 como en el Modelo 7, hay una etiqueta que no varía su número de vídeos correctamente etiquetados en el caso de mismo número de vídeos o distinto. Se trata de la etiqueta 5, que se corresponde con la clase "Cartoon".

4.6 Resumen de los modelos

Para poder visualizar mejor los resultados obtenidos se deja la Tabla 4-3 donde se encuentran para cada modelo el porcentaje de acierto sobre cada conjunto, el de entrenamiento, validación y de test, y un pequeño resumen. Los resultados se muestran en el caso del conjunto de test para diferente número de vídeos por clase tal y como se encuentra en la base de datos. El valor del acierto de la validación se ha tomado para la mejor época, y el del entrenamiento para esa misma época, ya que normalmente sigue creciendo.

Modelo	Acierto Entrenamiento	Acierto Validación	Acierto Test	Resumen de los modelos
Modelo 1	97.12%	91.33%	91.23%	Modelo de 1 capa oculta
Modelo 2	95.32%	92.23%	92.98%	Modelo de 1 capa oculta con *Dropout*
Modelo 3	93.22%	93.19%	93.01%	Modelo de 1 capa oculta por entrada
Modelo 4	94.65%	94.22%	94.22%	Modelo de 2 capas ocultas por entrada
Modelo 5	94.73%	94.24%	94.26%	Modelo 1 capa oculta por entrada y otra común
Modelo 6	92.45%	91.78%	91.48%	Modelo 1 capa oculta por entrada y 2 comunes
Modelo 7	92.03%	91.12%	93.03%	Modelo 5 entrenado para igual número de vídeos por clase

Tabla 4-3. Resumen final de los modelos

El mayor porcentaje conseguido se da en el Modelo 1, pero solo en entrenamiento, debido al sobreajuste, ya que las neuronas están "memorizando", pero se ve que ante nuevos datos en validación o test decae mucho. Al añadir *Dropout* se reduce el sobreajuste y se consigue un resultado bastante bueno en acierto en el Modelo 2.

Para el Modelo 3 ya se consigue un resultado bastante estable con un acierto muy elevado con los tres conjuntos. Los modelos 4 y 5 aplican la misma idea de añadir una segunda capa oculta, y los resultados son casi iguales. El modelo 6 en el que se añade más complejidad computacional empeora el acierto.

De nuevo es interesante el Modelo 7, donde el resultado en test es bastante mejor que el de entrenamiento y validación, cuando el comportamiento de las redes neuronales suele ser justo al contrario.

4.7 Aumento del número de etiquetas

Para finalizar con la parte experimental, se hace un aumento del número de clases posibles. Durante todos los apartados anteriores se han hecho uso solamente de 10 clases, aunque la base de datos tiene muchas más, por dos motivos. El primero es que conforme más etiquetas, mayor carga computacional y porque la idea de tener etiquetas más genéricas como "Coche" antes que tener varias de por ejemplo distintos modelos de coches es de mayor utilidad de cara a utilizar este tipo de clasificador en plataformas de clasificación de vídeos con muchos temas distintos como *YouTube*. La segunda razón es que a la hora de hacer las matrices de confusión, tener 10 clases es la manera de conseguir que se puedan ver de manera eficiente los resultados, ya que conforme más clases mayor sería la matriz.

Aun así, tras conseguir unos buenos resultados para clasificar 10 clases distintas, se plantea si ese mismo clasificador se puede utilizar para un mayor número de clases. Para ello se propone duplicar y quintuplicar el número de clases. Los resultados obtenidos resultan más que satisfactorios.

Para las pruebas se ha utilizado el Modelo 5, con el mismo número de capas y neuronas, menos en la capa *softmax* de salida, donde el número de neuronas representan el número de clases posibles, que se han cambiado a 20 y 50 para los dos casos a probar.

Para el clasificador de 10 clases, estas no se escogieron de entre aquellas que tenían identificadores del 0 al 9, sino 10 que estaban entre los identificadores 0 y 15, de manera que las clases tuvieran temáticas más diferenciadas. En cambio para las pruebas de clasificador con 20 y 50 clases sí se han escogido del 0 al 19 y del 0 al 49 respectivamente.

Los resultados mostrados en la Figura 4-15 corresponden primero al acierto y pérdida en entrenamiento y validación para clasificar 20 clases.

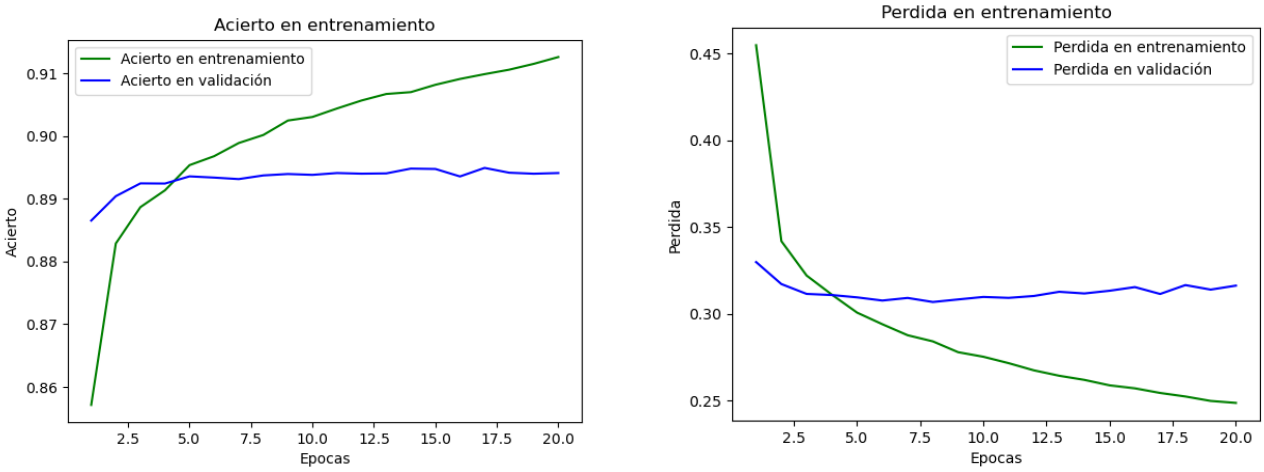


Figura 4-14. Resultados para clasificación de 20 clases

A partir de los resultados en el clasificador de 20 clases se puede observar que, evidentemente, el acierto disminuye mientras que la pérdida aumenta. Aun así en validación se sigue consiguiendo un buen porcentaje de acierto, con un máximo de 89.37%. Teniendo en cuenta que el número de clases posibles es el doble que en el caso anterior (94.24% de acierto), se concluye que el modelo sigue siendo una buena opción para clasificación de vídeos.

El acierto en entrenamiento sigue creciendo aunque el de validación se estabiliza, y el error de validación crece mínimamente a partir de la época 8 aproximadamente, por lo que se debería parar de entrenar en esta.

Ahora en la Figura 4-16, se muestran los resultados de acierto y pérdida para el caso del clasificador de 50.

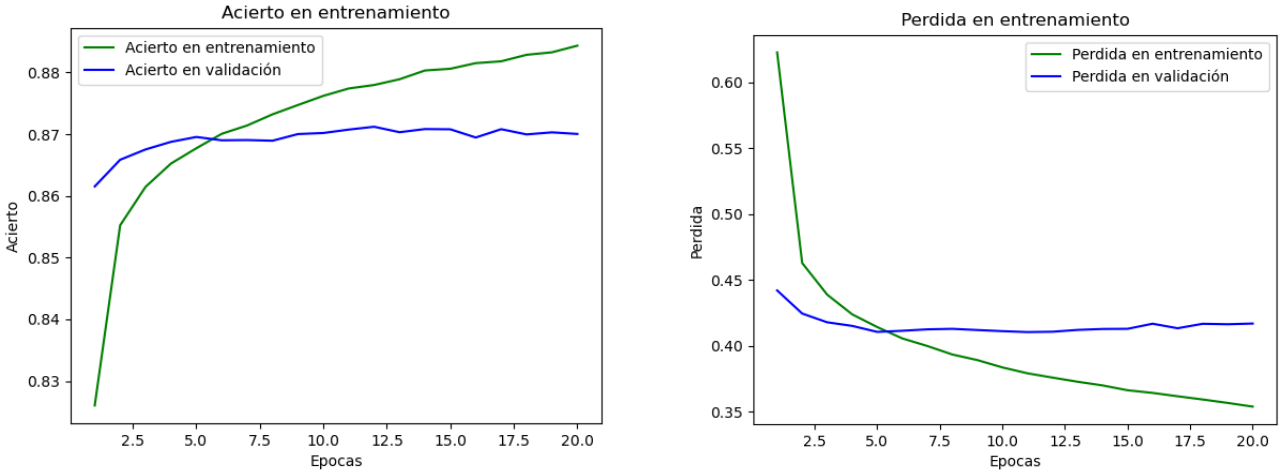


Figura 4-15. Resultados para clasificación de 50 clases

Los resultados para el clasificador de 50 clases dan para acierto un máximo de 87.11%, y un incremento de la pérdida, pero el comportamiento estable tanto del acierto como la pérdida en validación vuelven a conseguirse, aun utilizando un mismo modelo, de solo dos capas ocultas y no muchas neuronas.

Entre el clasificador de 10 y 20 clases, respecto al acierto, hay un decremento de un 4.87%, pero entre en el 10 y 50 clases el decremento es del 7.13%. La diferencia entre duplicar y quintuplicar el número de clases es de 2.26%, es decir, el acierto disminuye bastante al pasar de 10 a 20 clases, y sigue disminuyendo pero en menor medida conforme más clases hay.

La base de datos utilizada es realmente multietiqueta, pero para este trabajo solamente se ha cogido la primera etiqueta dada por cada vídeo. Además, la probabilidad de que la primera etiqueta de cada vídeo tome un cierto índice o identificador va disminuyendo conforme mayor es el índice. Esta característica de la base de datos explica que el acierto disminuya en menor medida al clasificar más clases, ya que las nuevas clases que se van añadiendo tienen muy baja probabilidad de aparición.

Por último en la Figura 4-17 se muestra un esquema con el resumen del modelo completo propuesto para la clasificación de vídeo, donde se identifican 3 partes principales:

1. Decodificación de los vídeos a imágenes a 1 *frame*/segundo.
2. Uso de la red convolucional *Inception* para extracción de características y posterior disminución de dimensiones a su salida. Se consiguen dos tipos de entrada, RGB y audio.
3. Uso de una red neuronal densamente conectada para aprender de las características y clasificar entre una serie de clases posibles.

La base de datos *YouTube-8M* ofrece de manera gratuita los dos primeros puntos, permitiendo descargar libremente los archivos con las características de los vídeos ya procesadas, que son los que con diferencia mayor complejidad y carga computacional requieren, haciendo más sencilla la tarea de clasificar vídeo.

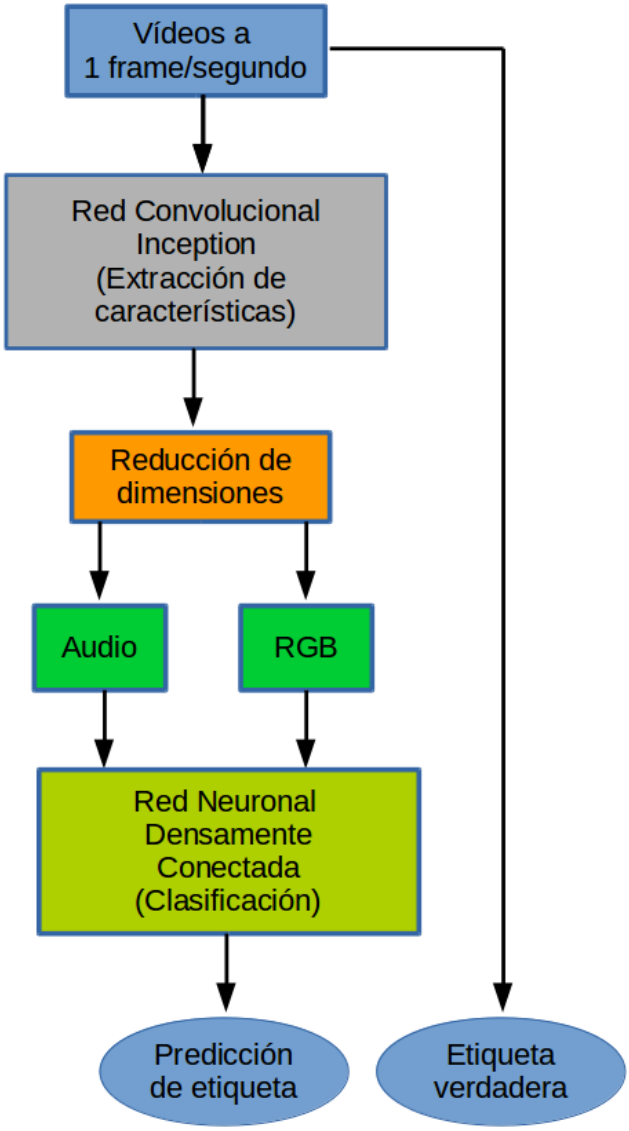


Figura 4-16. Estructura completa del clasificador

5 CONCLUSIONES Y LÍNEAS FUTURAS

Este trabajo ha permitido aprender, incluyendo una base teórica y una implementación programada, como utilizar una red neuronal para implementar un clasificador, en particular de vídeo. De manera sencilla la base de datos de *YouTube-8M* permite el libre acceso para cualquier persona a vídeos procesados de manera que no ocupen tanto espacio y sean manejables para una red neuronal. En este último capítulo se exponen las conclusiones finales extraídas de los resultados experimentales y se exponen unas consideraciones acerca del futuro de esta rama del Aprendizaje Máquina, la clasificación de vídeo.

5.1 Conclusiones

Las tareas de clasificación son de las que mayor tiempo tienen dentro de los campos de IA, Aprendizaje Máquina y Aprendizaje Profundo. La clasificación es esencial en la vida del ser humano para poder entender el mundo que le rodea, aunque ello conlleve a veces hacer representaciones con una pérdida de información de la realidad.

La clasificación de imágenes ha sido el centro de la investigación de una ingente cantidad de ingenieros durante más de 60 años desde la aparición de conceptos como la neurona o perceptrón. Sentó una base en la visualización por ordenador, que aún continúa en investigación, con implicaciones que van desde hacer filtros y fotos graciosas donde puedes ver como serías de anciano en redes sociales a permitir conducir de manera autónoma un vehículo, gracias a, evidentemente, otras muchas tecnologías.

Pero la clasificación de vídeo tiene un menor recorrido, debido a la dificultad computacional que conlleva procesar los vídeos, sea guardado en memoria o en tiempo real. Aun así hay mucha investigación en este ámbito, y competiciones como las de *kaggle* permiten que cada vez haya mayores avances en el tema. Las aportaciones de *Google* al presentar su base de datos *YouTube-8M* y utilizar esta página para presentar hasta tres años seguidos de competiciones haciendo uso de la base de datos, ha incrementado el interés de muchos investigadores en esta rama de la visión por ordenador.

En esta línea, este trabajo pretende ser un primer acercamiento tanto al Aprendizaje Máquina en general, como a la clasificación de vídeo en particular. Se abarca un problema tan complejo como es la clasificación de vídeo de una manera simple pero potente, utilizando una red neuronal densamente conectada para ello.

Los modelos de redes presentados demuestran que con pocas capas y neuronas se puede conseguir un potente clasificador. Se consigue una estabilidad en el error y un porcentaje de acierto más que aceptable, si bien es cierto que perdiendo una característica destacable de la base de datos original como es la multietiqueta. Pero las herramientas, sobretodo de hardware, como estudiante son limitadas. Además el hecho de no tener conocimientos previos de IA ni Aprendizaje Máquina ha hecho que la curva de aprendizaje fuera al principio de crecimiento lento, aunque constante, hasta llegar a poder tratar el problema propuesto, con la complejidad que conlleva.

Se puede concluir que la arquitectura propuesta, los resultados obtenidos y el aprendizaje tanto de *Python* como de Aprendizaje Máquina conseguidos son más que satisfactorios. Este trabajo consigue ser un punto de partida para adentrarse en varios de estos campos que se van imponiendo como tecnología predominante en un futuro digital e inteligente. Las posibilidades que ofrecen son casi infinitas, permitiendo además a ingenieros de diferentes ramas, y personas no tan técnicas, acceder a herramientas que pueden brindar soluciones a problemas que aun ni han sido planteados.

5.2 Líneas futuras de la clasificación de vídeo

Los vídeos pueden verse como una secuencia de imágenes, es decir, de matrices. Pero estas matrices tienen una característica particular que es la relación directa que hay entre una y la siguiente, que han permitido el desarrollo de nuevas tecnologías y arquitecturas que, por cuestión de tiempo y de complejidad computacional, han quedado fuera de este trabajo.

Dentro de la rama de clasificación de vídeo, además habiendo utilizado *YouTube-8M*, hay que señalar las ideas que se usaron en el equipo que quedó en primer posición en la competición de *kaggle* en 2017 [37], donde demostraron que, entre otras, una arquitectura denominada *Long Short Term Memory* (LSTM) resultaba muy eficiente a la hora de clasificar vídeos. Este tipo de arquitectura es denominada con memoria porque toma la entrada actual y datos de entradas ya procesadas, de manera que para datos como vídeos donde hay una relación directa entre *frames* resulta muy efectiva.

Junto a LSTM, existe la LSTM bidireccional (BiLSTM) utilizada también por varios concursantes, ya que es una arquitectura que había probado ser muy eficiente en reconocimiento del lenguaje, y prueban ser arquitecturas con una gran capacidad para capturar dependencias temporales entre *frames* [38]. Esta arquitectura difiere de cómo un ser humano puede llegar a clasificar un vídeo, ya que un ser humano lo ve secuencialmente de principio a fin, pero se este modelo puede incorporar información pasada y futura a la del *frame* que se esté tomando como entrada, logrando conseguir mejores relaciones.

Otra arquitectura que utiliza valores pasados es GRU [39], que es más simple que LSTM y junto a esta forma parte de la familia denomina redes neuronales recurrentes (RNN). GRU se compone de menos parámetros, y busca una mayor eficiencia que LSTM, y ha sido usada junto a esta para las diferentes competiciones relacionadas con *YouTube-8M*.

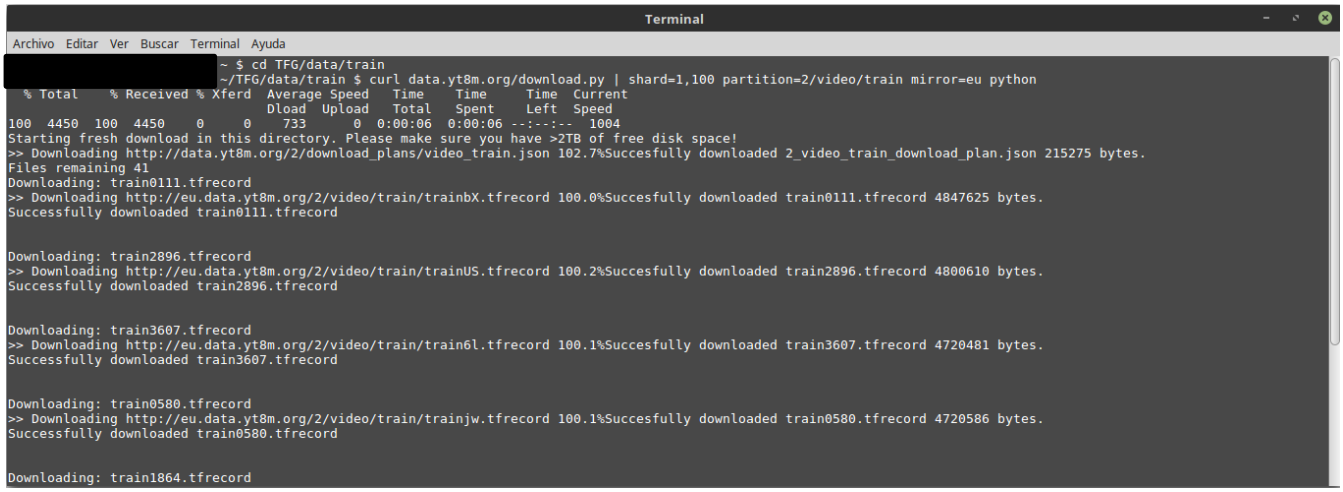
Para este tipo de tareas de clasificación se usa mucho la técnica *a Mixture of Experts* (MoE) [40] que puede resumirse en entrenar diferentes expertos para una misma tarea de clasificación, donde cada experto se especializa en un subconjunto de los casos posibles. Existe una capa *softmax* que da la probabilidad de elegir un experto. El problema es que hace falta mucha potencia computacional para este tipo de arquitecturas, ya que se entrenan diferentes clasificadores para cada vídeo, por lo que tener buenas y varias GPUs puede ser necesario.

En resumen, durante los últimos años se han implementado diversas arquitecturas para la visión por ordenador, y más particularmente para tareas relacionadas con vídeo. La problemática de los vídeos incluye, además de introducir relaciones temporales a las espaciales que ya tienen las imágenes, la cantidad de espacio en disco que requiere almacenar los vídeos y la potencia del *hardware* requerido para su procesamiento. Aun así es una rama de gran importancia y tendrá un impacto cada vez mayor en el mundo, permitiendo hacer uso de herramientas de IA para multitud de tareas distintas.

ANEXO A: DESCARGA BASE DE DATOS

En este anexo se explica cómo proceder para la descarga de la base de datos *YouTube-8M*, donde desde la página oficial [17] se ofrecen los pasos a dar. Para ello se muestran la siguiente captura hecha usando el sistema operativo Linux Mint, pero los pasos son análogos para otros. El procedimiento viene explicado en la web para las diferentes bases de datos que ofrece *YouTube-8M*. Es este trabajo se ha hecho uso de la denominada *Video-level features* dataset, de la versión de 2018.

Desde la terminal, primero se accede a la carpeta donde se vayan a descargar los ficheros y una vez ahí de proceden a utilizar los siguientes comandos, donde se diferencia entre conjunto de entrenamiento, validación y test.



El parámetro *mirror* deberá igualarse dependiendo de la zona geográfica a eu (Europa), us (América) o asia (Asia). Se deberá cambiar *train* por *validate* o *test* para cambiar el conjunto a descargar.

Además hay que tener en cuenta que la base de datos ocupa mucho espacio, por lo que se ofrece un parámetro, *shard* que en este caso se utiliza para descargar una partición de 1/100 del total de la base de datos. Una buena recomendación es primero utilizar una pequeña parte de la misma para realizar todas las pruebas, y una vez se tenga un modelo bueno proceder a la descarga de toda o una mayor parte de ella.

Una vez utilizado uno de los comandos para la descarga aparecerá información del tiempo y velocidad de descarga totales. Además irán apareciendo uno a uno los archivos *.tfrecord* junto al porcentaje de descarga.

ANEXO B: CÓDIGO CLASIFICADOR

En este anexo se deja el código completo utilizado para el modelo que permite la clasificación de 20 clases distintas. Una vez se use este código, si se quieren probar otras arquitecturas solamente hay que modificar la parte de creación de la red, utilizando las funciones de *Keras* para añadir o quitar capas y/o neuronas. Además se pueden variar otros hiperparámetros como la función de activación o incluso el clasificador. En el caso que se quiera modificar el número de clases hay que cambiar los tamaños de los vectores de salida (Y_train, Y_validate, Y_test) al tamaño que se quiera utilizar, y modificar la capa de salida *softmax*, utilizando para ello el número de neuronas igual al número de clases. El resto del código no debe modificarse para un correcto funcionamiento.

Para utilizar el conjunto de test es análogo a los conjuntos de entrenamiento y validación, cambiando donde pone en las variables y rutas *train* o *validate* por *test*. El código para el uso de la matriz de confusión ya se incluye en el texto.

```
import tensorflow as tf
import numpy as np
import os
from tensorflow.python.keras import backend as K
from keras import models
from keras import layers
import keras
from keras import Input
from keras.models import Model

#Cerramos cualquier sesion que pueda haber abierta por si acaso.
K.clear_session()
with tf.device('/gpu:0'):
    #Directorio donde estan los datos .tfrecord
    directorio = './data/train'
    contenido = os.listdir(directorio)
#Creamos una lista con todos los archivos .tfrecord para ir posteriormente recorriendo uno a uno
    ficheros = []
    for fichero in contenido:
        if os.path.isfile(os.path.join(directorio, fichero)) and fichero.endswith('.tfrecord'):
            ficheros.append(fichero)
    #Ceamos un array donde se van a ir incluyendo las entradas y salidas
    X_train_video = np.zeros(shape=(1,1024)) #Las entradas tendran tamaño 1024 y 128
    X_train_audio = np.zeros(shape=(1,128))
    Y_train = np.zeros(shape=(1,20)) #Las salidas tendran dimension igual al numero de clases
    num_n = 0 #Parametro para contabilizar los videos no utilizados

    #En este bucle vamos sacando para cada archivo tfrecord los parametros
    for i in ficheros:
```

```
        print(i+' fichero numero: '+str(i))
        frame_lvl_record = './data/train/'+i

    labels = []
    mean_rgb = []
    mean_audio = []

    #Usamos funciones de tensorflow para abrir los archivos tfrecord
    for example in tf.compat.v1.python_io.tf_record_iterator(frame_lvl_record):
        result = tf.train.Example.FromString(example)
        labels.append(result.features.feature['labels'].int64_list.value)
        mean_rgb.append(result.features.feature['mean_rgb'].float_list.value)
        mean_audio.append(result.features.feature['mean_audio'].float_list.value)

    new_labels = []
    new_mean_rgb = []
    new_mean_audio = []

    #Vamos a ir añadiendo de todos los archivos aquellos que pertenezcan a las clases
    for i in range(len(labels)):
          etiqueta = labels[i][0]
          if(etiqueta<=19): #Solo las primeras 20 etiquetas de la 0 a la 19
              vector_x = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
              vector_x[etiqueta] = 1 #Pasamos a formato one-hot las etiquetas
              new_labels.append(vector_x)
              new_mean_audio.append(list(mean_audio[i]))
              new_mean_rgb.append(list(mean_rgb[i]))

          else: #Contabilizamos los videos que no usamos
              num_n += 1
      #Tenemos como salidas un array (n,20) donde n es el numero de videos en cada tfrecord
      #Añadimos por cada archivo sus videos a las matrices
      new_mean_rgb = tf.convert_to_tensor(new_mean_rgb)
      new_mean_audio = tf.convert_to_tensor(new_mean_audio)
      X_train_video = np.append(X_train_video, new_mean_rgb,axis=0)
      X_train_audio = np.append(X_train_audio, new_mean_audio, axis=0)
      Y_train = np.append(Y_train,new_labels,axis=0)

#Cuando terminen todos los tfrecord se elimina el primero que es un vector de ceros
Y_train = np.delete(Y_train,0,axis=0)
X_train_video = np.delete(X_train_video,0,axis=0)
X_train_audio = np.delete(X_train_audio,0,axis=0)
```

```
    #Directorio donde estan los datos .tfrecord
    directorio = './data/validate'
    contenido = os.listdir(directorio)
#Creamos una lista con todos los archivos .tfrecord para ir posteriormente recorriendo uno a uno
    ficheros = []
    for fichero in contenido:
        if os.path.isfile(os.path.join(directorio, fichero)) and fichero.endswith('.tfrecord'):
            ficheros.append(fichero)

    #Ceamos un array donde se van a ir incluyendo las entradas y salidas
    X_validate_video = np.zeros(shape=(1,1024)) #Las entradas tendran tamaño 1024 y 128
    X_validate_audio = np.zeros(shape=(1,128))
    Y_validate = np.zeros(shape=(1,20)) #Las salidas tendran dimension igual al numero de clases
    num_n = 0 #Parametro para contabilizar los videos no utilizados

    #En este bucle vamos sacando para cada archivo tfrecord los parametros
    for i in ficheros:
        print(i+' fichero numero: '+str(i))
        frame_lvl_record = './data/validate/'+i

      labels = []
      mean_rgb = []
      mean_audio = []

      #Usamos funciones de tensorflow para abrir los archivos tfrecord
      for example in tf.compat.v1.python_io.tf_record_iterator(frame_lvl_record):
          result = tf.train.Example.FromString(example)
          labels.append(result.features.feature['labels'].int64_list.value)
          mean_rgb.append(result.features.feature['mean_rgb'].float_list.value)
          mean_audio.append(result.features.feature['mean_audio'].float_list.value)

      new_labels = []
      new_mean_rgb = []
      new_mean_audio = []

      #Vamos a ir añadiendo de todos los archivos aquellos que pertenezcan a las clases
      for i in range(len(labels)):
            etiqueta = labels[i][0]
            if(etiqueta<=19): #Solo las primeras 20 etiquetas de la 0 a la 19
                vector_x = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
                vector_x[etiqueta] = 1 #Pasamos a formato one-hot las etiquetas
```

```
                new_labels.append(vector_x)
                new_mean_audio.append(list(mean_audio[i]))
                new_mean_rgb.append(list(mean_rgb[i]))

            else: #Contabilizamos los videos que no usamos
                num_n += 1
        #Tenemos como salidas un array (n,20) donde n es el numero de videos en cada tfrecord
        #Añadimos por cada archivo sus videos a las matrices
        new_mean_rgb = tf.convert_to_tensor(new_mean_rgb)
        new_mean_audio = tf.convert_to_tensor(new_mean_audio)
        X_validate_video = np.append(X_validate_video, new_mean_rgb,axis=0)
        X_validate_audio = np.append(X_validate_audio, new_mean_audio, axis=0)
        Y_validate = np.append(Y_validate,new_labels,axis=0)

    #Cuando terminen todos los tfrecord se elimina el primero que es un vector de ceros
    Y_validate = np.delete(Y_validate,0,axis=0)
    X_validate_video = np.delete(X_validate_video,0,axis=0)
    X_validate_audio = np.delete(X_validate_audio,0,axis=0)

    #Creacion de la red neuronal usando funciones de Keras
    video_input = Input(shape=(1024,))
    audio_input = Input(shape=(128,))
    hidden_1_video = layers.Dense(220, activation='tanh')(video_input)
    dropout1 = layers.Dropout(0.5)(hidden_1_video)
    hidden_1_audio = layers.Dense(65, activation='tanh')(audio_input)
    dropout2 = layers.Dropout(0.5)(hidden_1_audio)
    concatenated = layers.concatenate([dropout1, dropout2])
    hidden_2 = layers.Dense(60, activation='tanh')(concatenated)
    salida = layers.Dense(50,activation='softmax')(hidden_2)

    model = Model([video_input,audio_input],salida, name="modelo")
    print(model.summary())
    keras.utils.plot_model(model, modelo.png', show_shapes=True)

    #Se compila y se entrena el modelo
    model.compile(optimizer="rmsprop", loss="categorical_crossentropy", metrics=["acc"])
    historia = model.fit([X_train_video, X_train_audio], Y_train, epochs=20, batch_size=256,
                         validation_data=([X_validate_video,X_validate_audio],Y_validate))
    #Se guardan los parametros del modelo entrenado para poder usarlo
    model.save('tfg_50_clases.h5')

    #Para visualizar las graficas de perdida y acierto durante entrenamiento y validacion
```

```
import matplotlib.pyplot as plt


historia_dict = historia.history
acc = historia_dict['acc']
val_acc = historia_dict['val_acc']
val_loss = historia_dict['val_loss']
loss = historia_dict['loss']
epochs = range(1,len(acc)+1)
plt.plot(epochs, acc, 'g', label='Acierto en entrenamiento')
plt.plot(epochs, val_acc, 'b', label='Acierto en validación')
plt.title('Acierto en entrenamiento')
plt.xlabel('Epocas')
plt.ylabel('Acierto')
plt.legend()
plt.show()
plt.plot(epochs, loss, 'g', label='Perdida en entrenamiento')
plt.plot(epochs, val_loss, 'b', label='Perdida en validación')
plt.title('Perdida en entrenamiento')
plt.xlabel('Epocas')
plt.ylabel('Perdida')
plt.legend()
plt.show()
```

REFERENCIAS

[1] Google Cloud & YouTube-8 Video Understanding Challenge. Kaggle [consulta: 3 marzo 2021] Disponible en: https://www.kaggle.com/c/youtube8m/overview

[2] Benítez, R. *Inteligencia artificial avanzada* **,** Editorial UOC, 2013

[3] Chollet, F. *Deep learning with Python,* Manning Publications, 2018

[4] Rosenblatt, F. *The Perceptron: A Perceiving and Recognizing Automaton.* Cornell Aeronautical Laboratory, 1957

[5] Introducción a las redes neuronales - Parte 1 [consulta: 3 marzo 2021] Disponible en: https://ichi.pro/es/introduccion-a-las-redes-neuronales-parte-1-65637746291018

[6] Pérez Aguila, R. *Una introducción al cómputo neuronal artificial*, El Cid Editor, 2012

[7] Catálogo de componentes de redes neuronales (II): funciones de activación, [consulta: 4 marzo 2021] Disponible en: https://ignaciogavilan.com/catalogo-de-componentes-de-redes-neuronales-ii-funciones-de-activacion/

[8] Raschka, S., & Mirjalili, V. *Python machine learning : aprendizaje automático y aprendizaje profundo con Python, scikit-learn y TensorFlow* (segunda edición revisada y actualizada), Marcombo, 2019

[9] Neuronal Network Implementation in Hardware [consulta: 4 maro 2021] Disponible en: https://diglab.technion.ac.il/projects/neural-network-implementation-in-hardware-fixed-point/

[10] Comprensión de los algoritmos de optimización [consulta: 5 marzo 2021] Disponible en: https://ichi.pro/es/comprension-de-los-algoritmos-de-optimizacion-53453022124445

[11] What is underfitting and overfitting in machine learning and how to deal with it. [consulta: 7 marzo 2021] Disponible en: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

[12] Overfitting and Underfitting Concepts & Interview questions. [consulta: 7 mazo 2021] Disponible en: https://vitalflux.com/overfitting-underfitting-concepts-interview-questions/

[13] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., *Dropout: a simple way to prevent neural networks from overfitting.* JMLR, 2014

[14] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998

[15] Red Neuronal Convolucional CNN [consulta: 10 marzo 2021] Disponible en: https://www.diegocalvo.es/red-neuronal-convolucional/

[16] Max-pooling / pooling [consulta: 10 marzo 2021] Disponible en: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

[17]YouTube-8M: A large and diversed labeled Video Dataset for video understanding research [consulta: 10 marzo 2021] Disponible en: https://research.google.com/youtube8m/

[18] Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B. & Vijayanarasimhan, S. *YouTube-8M: A Large-Scale Video Classification Benchmark*. 2016 Disponible en: https://arxiv.org/abs/1609.08675

[19] C. Szegedy et al., *Going deeper with convolutions*, 2015 IEEE Conference on Computer Vision and

Pattern Recognition (CVPR), 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[20] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, *ImageNet: A large-scale hierarchical image database*, 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

[21] Lin, M., Chen, Q., and Yan, S., *Network In Network*, Disponible en: https://arxiv.org/abs/1312.4400, 2013.

[22] NVIDIA CUDA Zone [consulta: 23 marzo 2021] Disponible en: https://developer.nvidia.com/cuda-zone

[23]TPU de Cloud [consulta: 23 marzo 2021] Disponible en: https://cloud.google.com/tpu

[24] Numpy The fundamental package for scientific computing with Python [consulta: 25 marzo 2021] Disponible en: https://numpy.org/

[25] Matplotlib: Visualization with Python [consulta: 25 marzo 2021] Disponible en: https://matplotlib.org/

[26] Pandas: Python Data Analysis Library [consulta: 25 marzo 2021] Disponible en: https://pandas.pydata.org/

[27] TensorFlow [consulta: 26 marzo 2021] Disponible en: https://www.tensorflow.org/

[28] Keras: the Python deep learning API [consulta: 26 marzo 2021] Disponible en: https://keras.io/

[29] Battle of the deep learning frameworks [consulta: 28 marzo 2021] Disponible en: https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

[30] K. Kiviluoto and E. Oja, *Softmax-network and S-Map-models for density-generating topographic mappings* 1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227), 1998, pp. 2268-2272 vol.3, doi: 10.1109/IJCNN.1998.687214.

[31] Unpingco, J. *Python for Probability, Statistics, and Machine Learning (2nd ed. 2019.)*. Springer International Publishing, 2019 Disponible en: https://doi.org/10.1007/978-3-030-18545-9

[32] D. Petkovic, *The need for accuracy verification of machine vision algorithms and systems*, Proceedings CVPR '89: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1989, pp. 439-440, doi: 10.1109/CVPR.1989.37885.

[33] Ting K.M. *Confusion Matrix*. In: Sammut C., Webb G. (eds) Encyclopedia of Machine Learning and Data Mining. Springer, Boston, MA , 2016. Disponible en: https://doi.org/10.1007/978-1-4899-7502-7_50-1

[34] Reading a confusion matrix [consulta: 24 abril 2021] Disponible en: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781787125933/6/ch06lvl1sec41/looking-at-different-performance-evaluation-metrics

[35] R. Zaheer and H. Shaziya, *A Study of the Optimization Algorithms in Deep Learning,* 2019 Third International Conference on Inventive Systems and Control (ICISC), 2019, pp. 536-539, doi: 10.1109/ICISC44355.2019.9036442.

[36]DotCSV [consulta: 23 marzo 2021] Disponible en: https://www.youtube.com/c/DotCSV/videos

[37] Miech, A., Laptev, I., and Sivic, J., *Learnable pooling with Context Gating for video classification*, 2017.

Disponible en: https://arxiv.org/abs/1706.06905.

[38] Zou, Haosheng et al. *The YouTube-8M Kaggle Competition: Challenges and Methods*, 2017 Disponible en: https://arxiv.org/abs/1706.09274

[39] Cho, K., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, ,2014. Disponible en: https://arxiv.org/abs/1406.1078

[40] M. I. Jordan, *Hierarchical mixtures of experts and the EM algorithm* IEE Colloquium on Advances in Neural Networks for Control and Systems, 1994, pp. 1/1-1/3.