Universidad de Las Palmas de Gran Canaria

MÁSTER UNIVERSITARIO

SISTEMAS INTELIGENTES Y APLICACIONES NUMÉRICAS EN INGENIERÍA

TRABAJO FINAL DE MÁSTER

Título del TFM

Etiquetado semántico de vídeos basado en aprendizaje profundo y minería de texto

---

Autor

Cristian ORTEGA LEÓN

Tutores

José Javier LORENZO NAVARRO

Pedro Antonio MARÍN REYES

Enero - 2018

Lo único imposible es aquello que no intentas.

Agradecimientos

No quería dejar pasar esta oportunidad para recordar y agradecer a todas aquellas personas que han hecho posible que haya llegado hasta aquí, y que me instaron a continuar.

Entre ellas, cabe nombrar a mis padres y hermano, por el sacrificio que realizan diariamente para posibilitarme la valiosa formación que he estado adquiriendo durante estos años, y por ser fuente de apoyo, tanto en las buenas como en las malas.

A mis tutores, Javier y Pedro; primero, por haberme permitido trabajar junto a ellos y, segundo, por la formación, consejos y ayuda que desde el sosiego me han transmitido, siendo de gran valor para mí.

Por último, agradecer a las personas que he conocido durante el camino hasta llegar a este momento, en especial a los profesores y compañeros del Máster SIANI, ya que, en cierta forma, todos han formado parte de este TFM y han aportado su pequeño granito de arena.

A todos, gracias.

Resumen

El vídeo online es responsable del 40 % del tráfico de Internet, con una tendencia al alza debido al crecimiento de las redes sociales y las plataformas destinadas a este fin, tales como YouTube, Vimeo, Viddler, etc. Para estas plataformas, debido al gran volumen de vídeos que manejan, se presenta de forma natural la problemática referente a la clasificación de los mismos en tópicos o clases. Este es, precisamente, el objetivo desarrollado en este TFM: una metodología que permite realizar clasificación de vídeo desde un enfoque basado en minería de texto. Aunque los resultados de clasificación obtenidos han sido dispares, la metodología desarrollada es perfectamente válida y funcional, siempre en función del dataset que se este utilizando.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1 Motivación

El vídeo online es responsable de más del 40 % del tráfico de Internet, con una tendencia al alza. El motivo es el crecimiento y la consolidación de las redes sociales y las plataformas destinadas a este fin, tales como YouTube, Vimeo, Viddler, etc.

Precisamente, YouTube es una de las principales plataformas de contenido online a nivel mundial: cada segundo se suben 60 minutos de vídeo, siendo el segundo motor de búsqueda más importante de Internet; donde recurren cada día millones de usuarios en busca de contenido audiovisual específico sobre sus intereses. Además, siendo una plataforma integrada en redes sociales como Facebook y Twitter, multiplica exponencialmente el alcance de los vídeos, generando un éxito innegable a la hora de viralizar estos contenidos.

El volumen de vídeos contenido en la plataforma mencionada es inmenso, y aunque el etiquetado acerca de la temática es responsabilidad del usuario que sube el vídeo, este, en muchos casos, no etiqueta el vídeo o lo hace incorrectamente. Por lo tanto, se presenta de forma natural la problemática referente a la clasificación de los vídeos en tópicos o clases.

Es por todo lo anterior que en este TFM, se busca el desarrollo de una metodología que permita realizar dicha clasificación de forma automática, lo cual puede ser de gran utilidad para, por ejemplo, buscar vídeos automáticamente según su contenido.

1.2 Hipótesis

Se ha hablado acerca de la necesidad de utilizar clasificadores automáticos de vídeo debido a la explosión de contenido de este tipo en Internet. Metodologías para la clasificación de vídeo hay muchas y variadas, basadas en distintas filosofías y cuyo resultado puede ser o no exitoso dependiendo del problema de clasificación concreto en el que se este trabajando.

Sabiendo lo anterior, se puede plantear la siguiente pregunta:

- **¿Es viable plantear una metodología diferente para clasificar vídeo?**

En base a la pregunta anterior, se puede generar la siguiente hipótesis:

- **Si se pueden convertir las características visuales de un vídeo a características textuales, debiera ser posible clasificar vídeo, de forma efectiva, desde un enfoque basado en clasificación/minería de texto.**

Esta es la hipótesis de partida de este TFM y en el apartado de **Metodología** se desarrolla la aproximación empleada para cumplirla.

1.3 Objetivos

El objetivo principal del TFM es el desarrollo de una metodología que permita realizar clasificación de vídeo desde un enfoque basado en minería de texto. Para ello, se realizarán una serie de tareas que constituyen objetivos parciales, las cuales se resumen a continuación:

- Se convertirán las entidades visuales que podemos visualizar en vídeos, a palabras en formato escrito o etiquetas. Esto se realizará de forma automatizada utilizando un detector de entidades visuales denominado YOLO9000.
- Se construirá una matriz vídeo/palabra a partir de las etiquetas obtenidas empleando YOLO9000, mediante la aplicación de técnicas de minería de texto.
- Se utilizará la matriz construida, la cual representa el contenido de los vídeos, para la aplicación de modelos de clasificación. Dichos modelos, en base a los vídeos analizados, deben ser capaces de clasificar correctamente nuevos vídeos dentro de una serie de clases.

- De forma paralela al punto anterior y con la misma finalidad, se empleará una red neuronal recurrente (RNN), concretamente una Long Short Term Memory (LSTM); la cual permite afrontar el problema de clasificación desde un punto de vista temporal o secuencial, aportando una perspectiva diferente con respecto a los otros modelos empleados.

Mediante la consecución de los cuatro objetivos parciales planteados, estaremos alcanzando el objetivo principal del TFM. Se dispondrá de un procedimiento que, a priori, es capaz de analizar un vídeo y clasificarlo dentro de una serie de clases.

Capítulo 2

Estado del arte

El objetivo último del presente TFM es realizar clasificación de vídeo, con la peculiaridad de que se cambia el enfoque, a la hora de abordar el desarrollo del problema, con respecto a las metodologías utilizadas habitualmente en el estado del arte y que, normalmente, mejores resultados dan.

A continuación, con la finalidad de ver cuales son los principales métodos existentes para clasificación de vídeo, se realizará una pequeña revisión del estado del arte que nos permitirá tener una idea más clara acerca del avance científico en este área, contextualizando así este TFM.

Debajo, se presentan las citas y una breve descripción para cada una:

- **Exploiting Feature and Class Relationships in Video Categorization with Regularized Deep Neural Networks [2]:** En este trabajo, se estudia el problema de categorizar los vídeos según una semántica de alto nivel, como la existencia de una acción humana en particular o un evento complejo. Se propone un novedoso framework que explota, de manera conjunta, las relaciones entre características y las relaciones entre clases para mejorar el desempeño de la clasificación. Específicamente, estos dos tipos de relaciones se estiman y utilizan imponiendo regularizaciones en el proceso de aprendizaje de una red neuronal profunda (DNN). A través de configurar la DNN con una mejor capacidad de aprovechar tanto las relaciones entre características como las relaciones entre clases, la DNN regularizada (rDNN) propuesta es más adecuada para el modelado de la semántica del vídeo.
- **Large-scale video classification with convolutional neural networks [3]:** Las redes neuronales convolutivas (CNNs) se han establecido como modelos de referencia para

problemas de reconocimiento de imágenes. En este artículo, se proporciona una evaluación empírica exhaustiva de las CNNs en la tarea de clasificar vídeo a gran escala, utilizando un nuevo conjunto de datos compuesto por 1 millón de vídeos de YouTube, pertenecientes a 487 clases. Se han estudiado múltiples aproximaciones para extender la conectividad de una CNN en el dominio del tiempo, con el fin de aprovechar la información espacio-temporal local. Además, se ha sugerido una arquitectura capaz de trabajar a resoluciones múltiples, siendo esta una forma prometedora de acelerar el entrenamiento.

- **Beyond short snippets: Deep networks for video classification [4]:** Las redes neuronales convolutivas (CNNs) se han estado aplicando de forma continua en problemas de reconocimiento de imagen, dando resultados avanzados en reconocimiento, detección, segmentación y recuperación. En este artículo, se proponen y evalúan varias arquitecturas de redes neuronales profundas para trabajar con vídeos de larga duración. Concretamente, se proponen dos métodos distintos. El primer método explora varias arquitecturas convolutivas de pooling de características temporales, examinando las diversas opciones de diseño por las que se puede optar a la hora de adaptar una CNN para esta tarea. El segundo método propuesto modela explícitamente el vídeo como una secuencia ordenada de frames. En este caso, se emplea una red neuronal recurrente que utiliza módulos LSTM, los cuales están conectados a la salida de una CNN.

- **Automatic video classification: A survey of the literature [5]:** Hay mucha cantidad de vídeo disponible hoy en día. Para ayudar a los espectadores a encontrar vídeos de interés, se ha comenzado a trabajar en métodos de clasificación automática de vídeo. En este artículo, se hace una revisión de la literatura en materia de clasificación de vídeo. Se encuentra que las características extraídas son de tres tipos (textuales, auditivas y visuales), y que se ha explorado una gran variedad de combinaciones de características y de métodos de clasificación. Se describen las características generales escogidas y se resume que se ha investigado en este área.

- **Large-scale multimodal semantic concept detection for consumer video [6]:** Se presenta un estudio sistemático sobre clasificación automática de vídeos consumer, considerando una amplia variedad de clases, las cuales son entidades visuales reconocibles en los vídeos. Se investigaron distintas aproximaciones estadísticas basadas en características visuales globales y locales, características de audio y combinaciones de ambas, desarrollando tres estrategias para combinar características visuales y de audio (*ensemble*, *context fusion*, y *joint boosting*). Se ha demostrado que los modelos combinados reducen significativamente los errores de detección (en comparación con los

modelos que solo trabajan con un tipo de característica), lo que resulta en una muy buena precisión sobre diversas clases.

- **Video categorization using Object of Interest detection [7]:** La detección de objetos de interés (OOI) ha sido ampliamente utilizada en muchos trabajos de análisis de vídeo. En este artículo, se propone un clasificador de vídeo genérico basado en el algoritmo K-Nearest Neighbors, utilizando la detección de objetos de interés. Se comprobó que, al detectar y describir el objeto de interés, con el fin de categorizar los vídeos en seis clases, la precisión de la clasificación mejora.

- **Exploiting objects with LSTMs for video categorization [8]:** La componente temporal juega un papel importante en la clasificación de vídeo. Se propone utilizar características semánticas de alto nivel en un modelo temporal, concretamente, una red LSTM. Para ello, primero se extraen características de los objetos mediante el uso de una CNN entrenada para reconocer 20000 de ellos. Luego, se aprovechan las características extraídas como entradas para la LSTM, la cual captura la dinámica temporal en los vídeos. En combinación con información espacial y de movimiento, se consiguen mejoras en la clasificación supervisada de vídeo. Además, al enmascarar las entradas, se descubre que la LSTM aprende que objetos son relevantes para reconocer una clase.

- **Video classification algorithm based on improved K-means [9]:** Con el objetivo de solucionar el problema que supone la baja precisión a la hora de clasificar fragmentos de vídeos deportivos, se propone un algoritmo K-Means mejorado como clasificador para dichos fragmentos. En primer lugar, se segmenta el vídeo deportivo y se extraen sus frames clave; a continuación, se forma el conjunto de características a partir de la característica de textura de los frames extraídos y la característica SIFT (Scale Invariant Feature Transform). Por otro lado, el algoritmo de clustering tradicional K-Means se mejora mediante aprendizaje semi-supervisado y optimización del punto de inicio del clustering. Finalmente, se aplica el algoritmo K-Means mejorado utilizando las características mencionadas.

- **Multilayer and multimodal fusion of deep neural networks for video classification [10]:** En este artículo, se afronta la clasificación de vídeo desde la combinación de diversos tipos de capas y modalidades de redes neuronales profundas. En primer lugar, se propone una estrategia multicapa para capturar, de forma simultánea, distintos niveles de abstracción e invariancia en una red, donde las capas convolutivas y completamente conectadas están representadas en el tiempo por unos métodos de agregación de características. Se introduce, además, un esquema multimodal que incluye

cuatro modalidades de redes altamente complementarias para extraer diversas pistas estáticas y dinámicas, a múltiples escalas temporales. En particular, para modelar la información temporal a largo plazo se propone una nueva estructura llamada FC-RNN, para transformar, de forma efectiva, las capas pre-entrenadas completamente conectadas en capas recurrentes. Por último, se optimiza la fusión de las múltiples capas y modalidades de red.

Capítulo 3

Metodología

En este apartado se va a exponer la metodología empleada desde un punto de vista conceptual. De esta forma, podremos ver la base teórica que sustenta el desarrollo del presente TFM y comprender de mejor manera el camino seguido para alcanzar los objetivos planteados.

3.1 Analogía entre vídeos y documentos

El objetivo de este TFM es realizar clasificación de vídeo desde un enfoque basado en minería de texto. Para ello, es necesario establecer y comprender una serie de analogías entre los elementos en los que un vídeo se puede descomponer y los elementos que forman un conjunto de documentos, que sería nuestro equivalente a un vídeo. Estas analogías se muestran en la Figura 3.1. Como podemos ver, se pueden establecer equivalencias entre los elementos utilizados en clasificación de texto y los utilizados en clasificación de vídeo.

Figura 3.1: Equivalencia entre los elementos utilizados en clasificación de texto y los utilizados en clasificación de vídeo.

Teniendo presente las analogías expuestas, y recordando que se quiere llevar la clasificación de vídeo al terreno de la clasificación de texto, se va a profundizar en los conceptos y procedimientos necesarios para entender como llevar a cabo la tarea propuesta.

En clasificación de texto, disponemos de un conjunto de documentos, el cual se puede subdividir en los documentos que lo forman. Cada uno de estos documentos contienen palabras, las cuales son la materia prima que se necesita para el proceso de clasificación. El problema es que las palabras constituyen un texto y, para trabajar con ellas, necesitamos extraerlas del texto y considerarlas de forma individual. Para ello, se utiliza la tokenización.

La tokenización [11] es la forma de separar el texto en palabras, comúnmente llamadas tokens. Este proceso toma en cuenta que las palabras pueden estar interrumpidas por un final de línea, estar pegadas a signos de puntuación y/o no siempre estar separadas por espacios.

Una vez hemos tokenizado los textos contenidos en los documentos, ya tenemos disponibles las palabras para operaciones posteriores. Llegados a este punto, sabiendo como es la forma de proceder cuando se hace clasificación de texto, hay que plantear un procedimiento análogo para realizar clasificación de vídeo. Por tanto, si en este caso disponemos de un vídeo, este lo podemos subdividir en los frames que lo forman. Cada uno de estos frames contienen

entidades visuales reconocibles, como pueden ser personas, animales, objetos, etc.

Teniendo en cuenta lo anterior, podemos abordar la tarea de clasificar vídeo como una clasificación de texto si convertimos las entidades visuales que podemos visualizar en cada frame a palabras en formato escrito. Estas palabras nos permiten realizar un proceso de clasificación de texto, por lo tanto, se consigue clasificar vídeo desde un enfoque de minería de texto, representando cada vídeo como un conjunto de palabras.

Llegados a este punto, es necesario aplicar un concepto importante en clasificación de texto, el concepto de Bag of Words (BoW) [12].

Considérese una serie de documentos, de los cuales hemos extraído las palabras que contienen, como se ha indicado anteriormente. Cada uno de estos documentos contiene un número de palabras distinto, lo cual no es aconsejable a la hora de aplicar técnicas de Machine Learning (ML), ya que la mayoría de algoritmos de este tipo están pensados para trabajar con inputs de longitud fija. Además, los algoritmos de ML que se utilizarán no pueden trabajar con palabras directamente, por lo que las palabras deben ser convertidas a números.

Debido a la problemática comentada, se hace obligatoria la aplicación del BoW para dar a los documentos la misma dimensión independientemente del número de palabras del mismo. Aplicar un modelo BoW consiste en convertir los conjuntos de palabras de cada documento en vectores sparse del tamaño del vocabulario construido a partir de las palabras que forman cada documento. En estos vectores, la mayoría de valores son 0 excepto en las posiciones que correspondan con las palabras presentes en el documento de que se trate. De esta forma, se obtienen tantos vectores como documentos existan, en los cuales los valores distintos de 0 dependen del método escogido para puntuar la presencia de las palabras del vocabulario. Tenemos, por tanto, una serie de vectores que describen la ocurrencia de las palabras dentro de los documentos.

De forma resumida, empleando un modelo BoW se extraen características del conjunto de palabras contenido en un documento para su uso con modelos, tales como algoritmos de ML. La intuición detrás del uso de un modelo BoW es que los documentos son similares cuando sus contenidos también lo son.

Siguiendo con la analogía entre clasificación de texto y clasificación de vídeo, ya se expuso anteriormente el hecho de que cada vídeo es representado como un conjunto de palabras, permitiendo enfocar el problema de clasificación de vídeo como un problema de clasificación de texto. Por lo tanto, es necesaria la aplicación de un modelo BoW por las razones comentadas anteriormente, una de las cuales es el hecho de que cada vídeo tiene un número de palabras distinto debido a que el número de entidades visuales reconocidas en cada uno es distinto.

Como resultado se obtienen tantos vectores numéricos de dimensión fija como vídeos haya.

A continuación, se expone en forma de diagrama el flujo de procesos realizados. La primera flecha indica la transformación necesaria para enmarcar los vídeos dentro de una metodología de clasificación de texto, y la segunda flecha indica la transformación necesaria para poder alimentar modelos utilizando los conjuntos de palabras.

$$\text{Vídeo} \rightarrow \text{Conjunto de palabras} \rightarrow \text{Vector numérico}$$

Para finalizar, se hará hincapié en los métodos disponibles para puntuar la presencia de las palabras del vocabulario en los vectores numéricos, es decir, ¿qué codificaciones podemos utilizar para evaluar las posiciones distintas de 0 dentro de los vectores?

La codificación [13] para los vectores puede ser:

- **Binaria:** cuando se indica con un 1 la aparición de una de las palabras del vocabulario en el vídeo que el vector esta representando.
- **Term Frequency (TF):** donde sustituimos el 1 por la frecuencia de aparición de cada palabra en su vídeo. De esta manera se mide la frecuencia de aparición de una palabra en un vídeo.

$$\text{TF}(p) = \frac{\text{Número de veces que la palabra } p \text{ aparece en un vídeo}}{\text{Número total de palabras en el vídeo}}$$

- **Term Frequency - Inverse Document Frequency (TF-IDF):** donde realizamos un TF y lo modificamos bajando la importancia de las palabras que aparecen frecuentemente en el conjunto de vídeos completo; y subiendo la importancia de las palabras que aparecen frecuentemente pero solo en el vector de que se trate. Así, se mide la importancia de cada palabra con respecto a las demás.

$$\text{IDF}(p) = \log_e\left(\frac{\text{Número total de vídeos}}{\text{Número de vídeos con la palabra } p \text{ en ellos}}\right)$$

$$\text{TF-IDF}(p) = \text{TF}(p) \cdot \text{IDF}(p)$$

Utilizando estas codificaciones a la hora de construir los vectores numéricos que representan a los vídeos, se consigue aportar información extra a los modelos a la hora de realizar la clasificación. Esto se debe a que se obtienen vectores que representan de manera más precisa a los vídeos.

3.2 Conversión de entidades visuales a palabras

Como se ha expuesto anteriormente, podemos abordar la tarea de clasificar vídeo como una clasificación de texto si convertimos las entidades visuales que podemos visualizar en cada frame a palabras en formato escrito. Por lo tanto, se necesita una herramienta capaz de realizar la conversión mencionada de la mejor manera posible, extrayendo la mayor cantidad de entidades visuales de cada frame.

La herramienta utilizada ha sido YOLO9000, un detector de entidades visuales que, básicamente, es una CNN. Sabiendo esto, se va a introducir brevemente que es y como funciona una CNN, para luego tratar acerca de YOLO, y su versión más avanzada YOLOv2 / YOLO9000.

3.2.1. Redes neuronales convolutivas

Las CNNs [14] son una categoría de redes neuronales (NNs) que funcionan de forma efectiva en áreas como el reconocimiento y clasificación de imágenes. Este tipo de redes pueden identificar todo tipo de objetos en imágenes y han mejorado los sistemas de visión en robots y vehículos autónomos.

En la Figura 3.2, una CNN es capaz de reconocer escenas y sugerir subtítulos relevantes, mientras que en la Figura 3.3 se muestra un ejemplo de una CNN usada para reconocer objetos cotidianos, humanos y animales.

Figura 3.2: Reconocimiento de escenas utilizando una CNN [14].

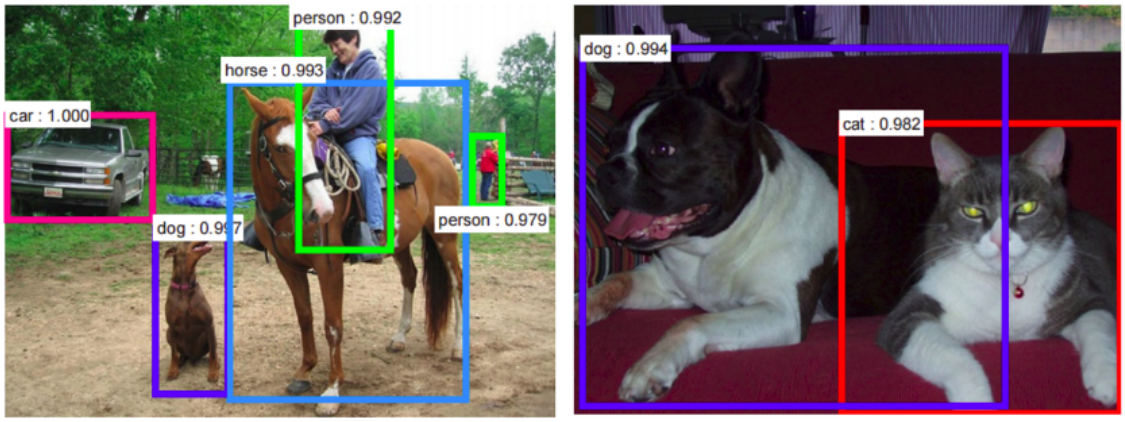

Figura 3.3: Detección de entidades realizada por una CNN [14].

Para poder realizar las tareas de reconocimiento y clasificación que se han comentado, las CNNs utilizan las cuatro operaciones siguientes:

1. **Convolución:** nos permite extraer características de la imagen de entrada, obteniendo mapas de características que aportan información acerca de los distintos elementos que forman la imagen.

2. **No linearidad:** después de cada operación de convolución se realiza una operación adicional llamada Rectified Linear Unit (ReLU), cuyo propósito es introducir una no linearidad en la CNN, lo cual es necesario para que esta pueda ajustarse a las imágenes con las que normalmente va a trabajar, de naturaleza no lineal.

   Es posible utilizar otras funciones no lineales, como la *tanh* o la *sigmoide*, en vez de ReLU. En este caso, se ha elegido la función no lineal ReLU para la explicación por su buen desempeño en muchas situaciones.

3. **Pooling espacial o Subsampling:** esta operación reduce la dimensionalidad de cada mapa de características manteniendo la información más importante y representativa. Al ir reduciendo progresivamente el tamaño de los mapas de características se consiguen varias cosas: hacer que los mapas de características sean más pequeños y manejables; reducir el número de parámetros y cálculos a realizar dentro de la red, controlando así el overfitting; y hacer a la red 'insensible' a pequeñas transformaciones o distorsiones en la imagen de entrada.

4. **Clasificación mediante red neuronal completamente conectada (FCN):** las operaciones anteriores constituyen capas que, trabajando juntas, extraen características

útiles de las imágenes, introducen no linearidades en la red y reducen la dimensión del espacio de características. La salida final del conjunto de estas capas se pasa como entrada a una FCN.

Una FCN es un Perceptron Multicapa [15] que utiliza una función de activación SoftMax en la capa de salida. La salida de las capas convolutivas y de pooling representan características de alto nivel de la imagen de entrada. El propósito de la FCN es usar las características mencionadas para clasificar la misma imagen correctamente, en base a las clases aprendidas durante el entrenamiento. Al utilizar una función de activación SoftMax en la capa de salida, la suma de las probabilidades de salida para el conjunto de clases es igual a uno, por lo que la clase pronosticada es la que ha alcanzado mayor probabilidad.

Las cuatro operaciones comentadas son los bloques básicos para la construcción de cualquier CNN, como podemos ver en la Figura 3.4. La CNN de la Figura 3.4 clasifica una imagen de entrada en cuatro categorías: perro, gato, barco o pájaro. Al recibir la imagen de un barco como entrada, la red asigna correctamente la probabilidad más alta para el barco (0.94) de entre las cuatro categorías.

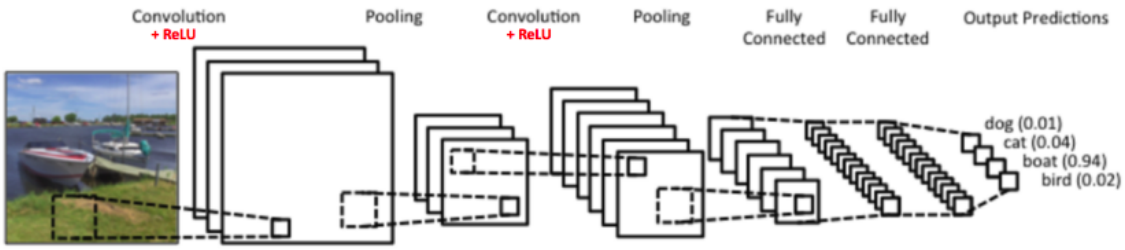

Figura 3.4: Arquitectura básica de una CNN [14].

3.2.2. Detector de entidades visuales YOLO

YOLO [16] es una herramienta compuesta por una CNN que predice simultáneamente, y de forma directa, las coordenadas de múltiples cuadros delimitadores y probabilidades de clase para esos cuadros. Para poder hacer esto, se han unificado los componentes independientes utilizados en la detección de objetos dentro de una sola red neuronal. Dicha red razona de forma global sobre la imagen y los objetos que esta contiene, utilizando características de toda la imagen para predecir cada cuadro delimitador, generando todos los cuadros delimitadores de todas las clases de forma simultánea para una imagen.

Funcionamiento básico: El sistema divide la imagen de entrada usando una rejilla de tamaño $S \times S$. Si el centro de un objeto cae dentro de una celda de la rejilla, esta misma celda es la responsable de la detección de ese objeto.

Cada celda de la rejilla predice B cuadros delimitadores y las puntuaciones de confianza para cada cuadro. Estas puntuaciones de confianza reflejan la confianza que el modelo tiene en que el cuadro contiene un objeto y también la precisión con la que piensa que el cuadro se ajusta a lo que se predice. Formalmente, definimos la confianza como $\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$. Si no existe objeto en una celda, la puntuación de confianza debería ser cero. De lo contrario, queremos que la puntuación de confianza iguale la intersección sobre la unión (IOU) entre el cuadro pronosticado y el verdadero cuadro, el que encuadra perfectamente el objeto.

Cada cuadro delimitador consta de 5 predicciones: x, y, w, h, y puntuación de confianza. Las coordenadas (x, y) representan el centro del cuadro relativo a los límites de la rejilla. La anchura y la altura se pronostican en relación con la imagen entera. Finalmente, la predicción de confianza representa la intersección sobre la unión (IOU) entre el cuadro pronosticado y cualquier cuadro verdadero.

Cada celda de la rejilla también predice C probabilidades condicionales de clase, $\Pr(\text{Class}_i|\text{Object})$. Estas probabilidades están condicionadas a la celda de la rejilla que contiene un objeto. Sólo predecimos un conjunto de probabilidades de clase por celda de la rejilla, independientemente del número de cuadros B.

Para testear, multiplicamos las probabilidades condicionales de clase y las predicciones de confianza de los cuadros individuales,

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

lo que nos da puntuaciones de confianza acerca de la probabilidad que tiene cada una de las clases de aparecer en cada cuadro. Estas puntuaciones codifican tanto la probabilidad de que esa clase aparezca en el cuadro como qué tan bien el cuadro pronosticado se ajusta al objeto.

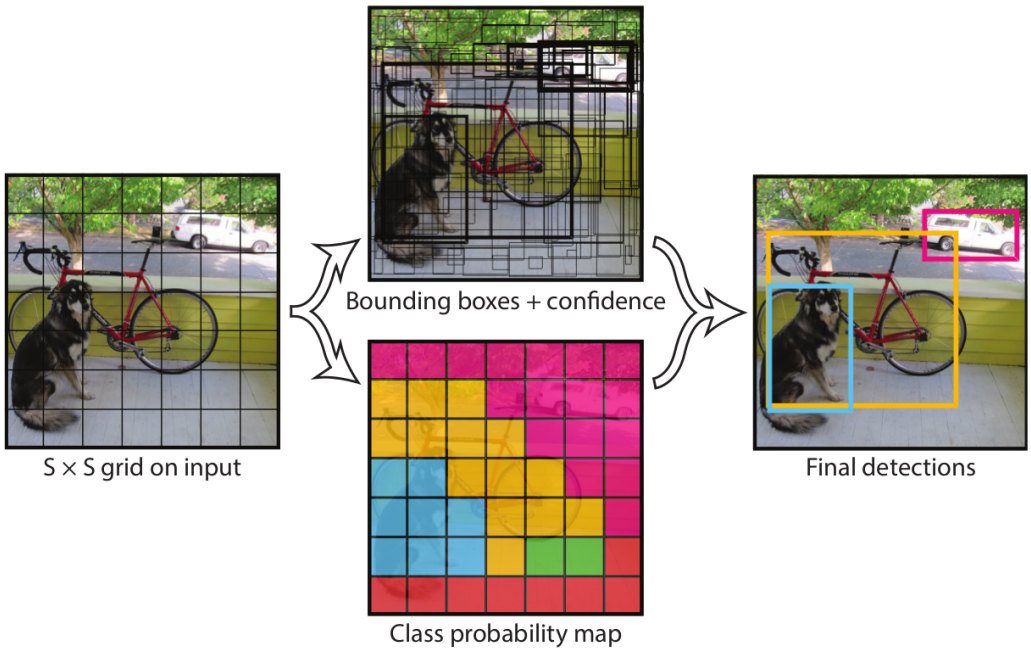

Figura 3.5: El Modelo [16]. El sistema modela la detección como un problema de regresión. Divide la imagen en una rejilla de tamaño $S \times S$ y por cada celda de la rejilla predice B cuadros delimitadores, puntuaciones de confianza para estos cuadros, y C probabilidades condicionales de clase por celda de la rejilla. Estas predicciones son codificadas en un tensor de tamaño $S \times S \times (B \cdot 5 + C)$.

Diseño de la red: El modelo se ha implementado como una CNN y se ha evaluado en el conjunto de datos PASCAL VOC. Las capas convolutivas iniciales de la red extraen características de la imagen mientras que las capas completamente conectadas predicen las probabilidades de salida y las coordenadas.

La arquitectura de la red esta inspirada en el modelo GoogLeNet para clasificación de imágenes. Tiene 24 capas convolutivas seguidas de 2 capas completamente conectadas. En lugar de los módulos Inception utilizados por GoogLeNet, se usan capas de reducción de 1×1 seguido por capas convolutivas de 3×3. La red completa se muestra en la Figura 3.6.

La salida final de la red es un tensor de predicciones de tamaño $7 \times 7 \times 30$ al evaluar en el conjunto de datos PASCAL VOC.

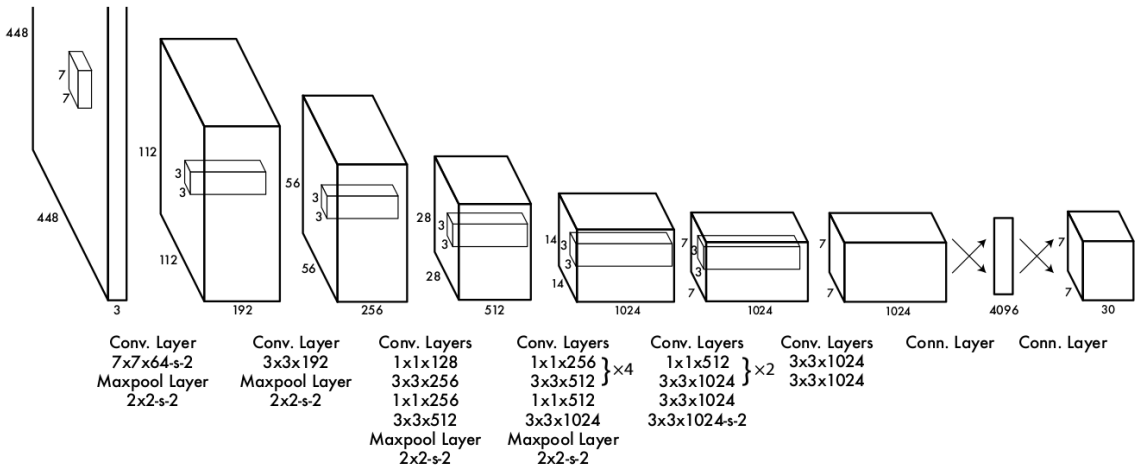

Figura 3.6: La Arquitectura [16]. La red de detección tiene 24 capas convolutivas seguido por 2 capas completamente conectadas. Alternando capas convolutivas de 1×1 reducimos el espacio de características de las capas precedentes.

3.2.3. Detector de entidades visuales YOLOv2 / YOLO9000

Para la creación de YOLO9000 [17] los autores han seguido los siguientes pasos:

- Mejora del sistema básico de detección YOLO para producir YOLOv2, consiguiendo de esta forma un detector en tiempo real con un rendimiento al nivel del estado del arte.
- Utilización de un método de combinación de conjuntos de datos y un algoritmo de entrenamiento conjunto para entrenar un modelo en más de 9000 clases, tanto de ImageNet como del conjunto de datos para detección COCO.
- Obtención de YOLO9000, un detector de objetos en tiempo real que puede detectar más de 9000 categorías de objetos diferentes.

Mejoras sobre YOLO: La visión por computador generalmente tiende hacia redes más grandes y profundas. Un mejor rendimiento a menudo se consigue entrenando redes más grandes o ensamblando varios modelos juntos. Sin embargo, con YOLOv2 se busca un detector más preciso manteniendo un funcionamiento rápido. En vez de ampliar la red, se simplifica y se hace que la representación sea más fácil de aprender.

Son muchas las mejoras implementadas en YOLOv2, como podemos ver en la Figura 3.7.

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figura 3.7: De YOLO a YOLOv2 [17]. La mayoría de las decisiones de diseño enumeradas conducen a incrementos significativos en el mAP.

A continuación, pasamos a comentar las mejoras más importantes introducidas para la creación de YOLOv2:

- **Clasificador de alta resolución:** Todos los métodos de detección de última generación utilizan clasificadores pre-entrenados en ImageNet. Comenzando con AlexNet, la mayoría de los clasificadores operan en imágenes de entrada menores a 256×256. El YOLO original entrena la red de clasificación a 224×224 y aumenta la resolución a 448 para detección, lo cual significa que la red tiene que, simultáneamente, ajustarse a la nueva resolución de entrada y aprender detección de objetos.

  Para YOLOv2 se pre-entrena la red usando ImageNet a resolución completa (448×448) para 10 épocas, dándole tiempo a la red para ajustar sus filtros a trabajar mejor en una entrada de mayor resolución. A continuación, se entrena la red resultante para trabajar en detección. La red de clasificación de alta resolución obtenida da un incremento de casi 4 % mAP.

- **Entrenamiento en múltiples escalas:** Para que el modelo sea robusto trabajando con imágenes de diferentes tamaños, se entrena cambiando aleatoriamente el tamaño de la imagen de entrada cada 10 batches. El tamaño más pequeño es 320×320 y el mayor es 608×608. La red se redimensiona y continua entrenando.

  Este entrenamiento fuerza la red a aprender a predecir correctamente contemplando una variación bastante amplia en las dimensiones de entrada. En la Figura 3.8 se muestra una comparativa entre YOLOv2 y otros frameworks, en términos de resolución, precisión y FPS.

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288×288	2007+2012	69.0	91
YOLOv2 352×352	2007+2012	73.7	81
YOLOv2 416×416	2007+2012	76.8	67
YOLOv2 480×480	2007+2012	77.8	59
YOLOv2 544×544	2007+2012	**78.6**	40

Figura 3.8: Frameworks de detección en PASCAL VOC 2007 [17]. YOLOv2 es más rápido y preciso que otros métodos de detección de uso reconocido. Funciona a distintas resoluciones permitiendo encontrar el balance óptimo entre velocidad y precisión.

- **Darknet-19:** Queremos que la detección sea precisa pero también queremos que sea rápida, lo cual es requisito indispensable en muchas aplicaciones de detección, como la robótica o los vehículos autónomos.

  YOLOv2 se ha diseñado para maximizar el rendimiento, usando una red modificada basada en la arquitectura GoogLeNet. Al igual que en los modelos VGG, se usan principalmente filtros de 3×3 y se duplica el número de canales después de cada fase de pooling. Se usa un average pooling global para hacer las predicciones y filtros de 1×1 para comprimir los mapas de características entre convoluciones de 3×3. Se usa la normalización por lotes para estabilizar el entrenamiento, acelerar la convergencia y regularizar el modelo.

  El modelo final, llamado Darknet-19, tiene 19 capas convolutivas y 5 capas de maxpooling. En la Figura 3.9 podemos ver la estructura completa del modelo.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figura 3.9: Darknet-19 [17].

- **Entrenamiento multi-dataset:** Se propone un mecanismo para entrenar de forma conjunta en datos de clasificación y de detección. El método usa imágenes etiquetadas para detección a la hora de aprender información específica de detección, como la predicción de coordenadas del cuadro delimitador y la probabilidad de contener un objeto, así como la forma de clasificar objetos comunes. Las imágenes que solo llevan asociadas etiquetas de clase se usan para expandir el número de categorías detectables.

  Durante el entrenamiento se mezclan imágenes de los conjuntos de datos de clasificación y de detección. Los conjuntos de datos de detección sólo tienen objetos comunes y etiquetas generales, como 'perro' o 'barco'. Los conjuntos de datos de clasificación tienen una gama de etiquetas mucho más amplia y profunda. Por ejemplo, ImageNet tiene más de cien razas de perros, incluyendo 'Norfolk terrier', 'Yorkshire terrier' y 'Bedlington terrier'. Dado que ambos tipos de conjuntos de datos son diferentes, para entrenar en conjuntos de datos de los dos tipos de manera simultánea se necesita una forma coherente de fusionar sus etiquetas.

  Muchos enfoques en clasificación utilizan una capa de salida SoftMax para calcular la distribución de probabilidad final usando todas las categorías posibles, asumiendo, de esta

forma, que las clases son mutuamente excluyentes. Esto supone un problema a la hora de combinar conjuntos de datos, por ejemplo, no podríamos combinar ImageNet y COCO utilizando este modelo porque las clases 'Norfolk terrier' y 'dog' no son mutuamente excluyentes.

Se puede usar, en su lugar, un modelo multi-etiqueta para combinar los conjuntos de datos sin asumir exclusión mutua entre clases. El modelo es llamado WordTree, un modelo jerárquico de conceptos visuales. En la Figura 3.10 se muestra un ejemplo del uso de WordTree para combinar las etiquetas de ImageNet y COCO.

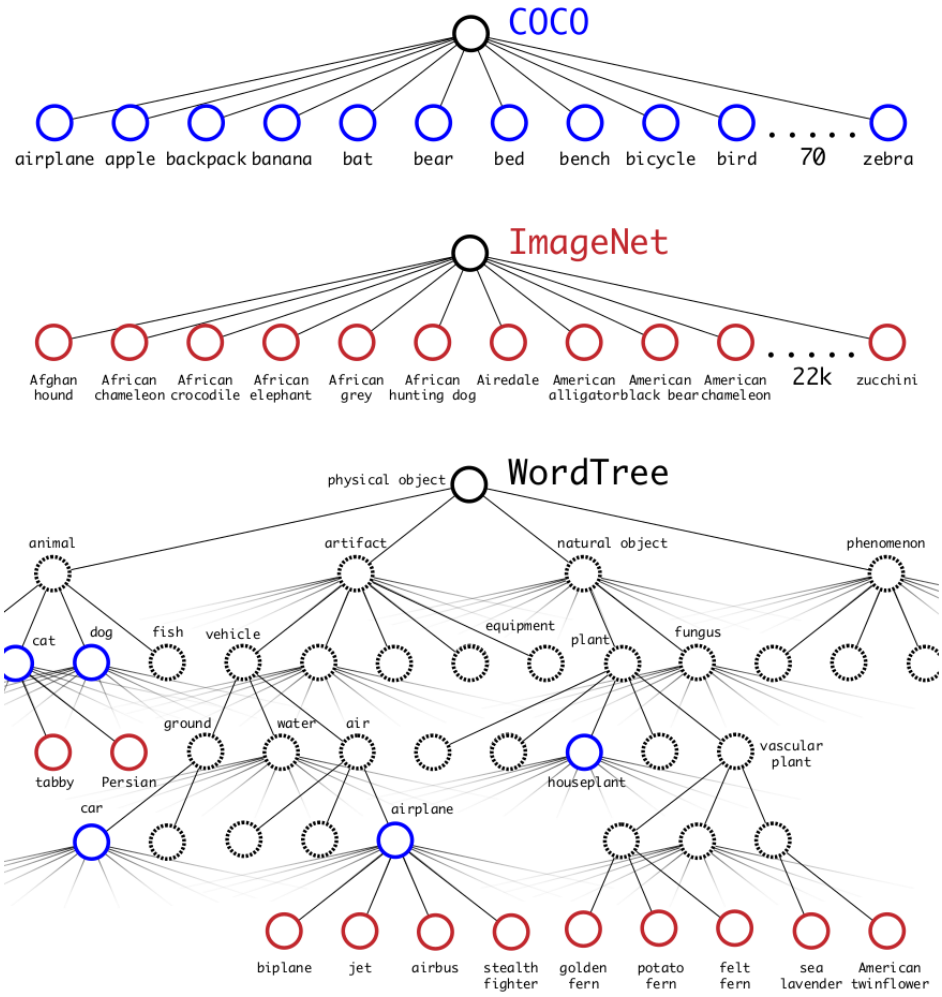

Figura 3.10: Combinación de conjuntos de datos usando la jerarquía WordTree [17]. Usando el concepto gráfico de WordNet se construye un árbol jerárquico de conceptos visuales. Se pueden fusionar conjuntos de datos asignando las clases a los nodos que les correspondan en el árbol. Esta es una vista simplificada de WordTree con fines ilustrativos.

- **YOLO9000:** Para la creación de YOLO9000 se ha entrenado YOLOv2 usando un conjunto de datos que combina el dataset para detección COCO y las 9000 clases principales de ImageNet. YOLO9000 aprende a encontrar objetos en imágenes usando los datos de detección de COCO y aprende a clasificar una amplia variedad de estos objetos usando datos de ImageNet.

3.3 Modelos de clasificación

Como hemos visto anteriormente, es posible abordar un problema de clasificación de vídeo desde la perspectiva de la clasificación de texto, por lo que empleando la metodología vista hasta ahora convertimos los vídeos en vectores numéricos, los cuales nos permiten emplear modelos de clasificación para alcanzar el objetivo perseguido en este TFM.

Un modelo de clasificación [18] se crea mediante la aplicación de un algoritmo a los datos, el cual es un conjunto de heurísticas y cálculos.

Para crear un modelo, el algoritmo comienza analizando los datos proporcionados, en busca de patrones específicos o tendencias. El algoritmo utiliza los resultados de este análisis, realizado a través de un gran número de iteraciones, para determinar los parámetros óptimos y crear el modelo de clasificación. A continuación, estos parámetros se aplican en todo el conjunto de datos para realizar el trabajo de clasificación.

A continuación, se describen los modelos de clasificación utilizados:

- **Naive Bayes [19]:** Se trata del modelo más simple de clasificación con redes bayesianas. En este caso, la estructura de la red es fija y sólo necesitamos aprender los parámetros (probabilidades). El fundamento principal del clasificador Naive Bayes [Duda & Hart 1973; Langley et al. 1992] es la suposición de que todos los atributos son independientes conocido el valor de la variable clase. A pesar de que asumir esta suposición en el clasificador Naive Bayes (NB) es sin duda bastante fuerte y poco realista en la mayoría de los casos, se trata de uno de los clasificadores más utilizados. Además, diversos estudios (p.e. [Michie et al. 1994]) demuestran que sus resultados son competitivos con otras técnicas (redes neuronales y árboles de decisión entre otras) en muchos problemas y que incluso las superan en algunos otros. Como un ejemplo de problema en el que el clasificador NB se está mostrando como una de las técnicas más eficaces, podemos citar la lucha contra el correo basura o spam. Muchos lectores de correo incorporan este clasificador para etiquetar el correo no solicitado.

La hipótesis de independencia asumida por el clasificador NB da lugar a un modelo gráfico probabilístico en el que existe un único nodo raíz (la clase), y en la que todos los atributos son nodos hoja que tienen como único padre a la variable clase. Gráficamente tendríamos la estructura de la Figura 3.11.

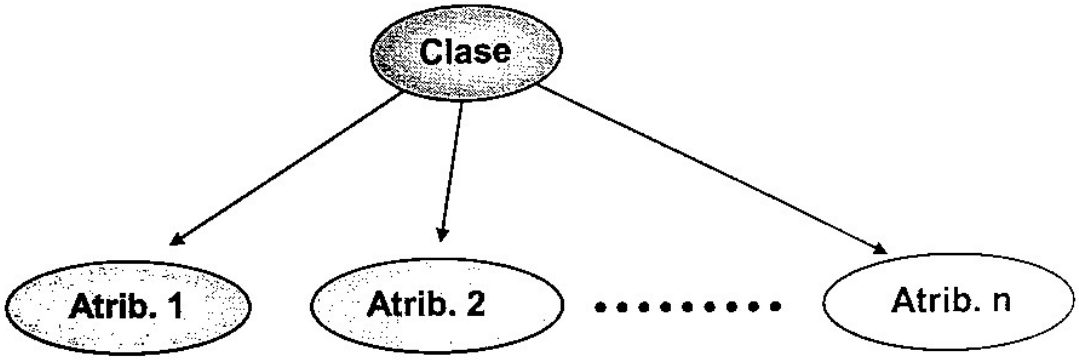

Figura 3.11: Topología de un clasificador Naive Bayes [19].

Estimación de los parámetros: Debido a la hipótesis de independencia usada en el Naive Bayes, la expresión para obtener la hipótesis maximum a posteriori (MAP) queda como sigue:

$$c_{MAP} = \arg\max_{c \in \Omega_c} p(A_1,\ ...,\ A_n \mid c)\, p(c) = \arg\max_{c \in \Omega_c} p(c) \prod_{i=1} p(A_i \mid c)$$

Es decir, la tabla de probabilidad $P(A_1,\ ...,\ A_n \mid c)$ ha sido factorizada como el producto de n tablas que sólo involucran dos variables. Por tanto, los parámetros que tenemos que estimar son $P(A_i \mid c)$ para cada atributo y la probabilidad a priori de la variable clase $P(c)$. Veamos cómo hacerlo dependiendo de que el atributo A_i sea discreto o continuo.

- **Atributos discretos:** En este caso la estimación de la probabilidad condicional se basa en las frecuencias de aparición que obtendremos en la base de datos. Así, si llamamos $n(x_i,\ Pa(x_i))$ al número de registros de la base de datos en que la variable X_i toma el valor x_i y los padres de X_i $(Pa(X_i))$ toman la configuración denotada por $Pa(x_i)$, entonces la forma más simple de estimar $P(x_i \mid Pa(x_i))$ es:

  $$P(x_i \mid Pa(x_i)) = \frac{n(x_i,\ Pa(x_i))}{n(Pa(x_i))}$$

  Es decir, el número de casos favorables dividido por el número de casos totales. Esta técnica se conoce como *estimación por máxima verosimilitud* y tiene como

desventajas que necesita una muestra de gran tamaño y que sobreajusta a los datos. Existen otros estimadores más complejos que palían estos problemas, entre ellos citaremos el *estimador basado en la ley de la sucesión de Laplace*:

$$P(x_i \mid Pa(x_i)) = \frac{n(x_i,\ Pa(x_i)) + 1}{n(Pa(x_i)) + |\Omega_{X_i}|}$$

Es decir, el número de casos favorables más uno dividido por el número de casos totales más el número de valores posibles. Nótese que con pocos ejemplos, la probabilidad se corrige por la probabilidad uniforme a priori, es decir, uno dividido por el número de valores posibles. Con esta estimación lo que se pretende es que todas las configuraciones posibles tengan una mínima probabilidad, ya que con el estimador de máxima verosimilitud cualquier configuración que no esté presente en la base de datos tendría probabilidad cero.

- **Atributos continuos:** En este caso, el clasificador Naive Bayes supone que el atributo en cuestión sigue una distribución normal; por tanto, lo único que tenemos que calcular (a partir de la base de datos) es la media μ y la desviación típica σ condicionadas a cada valor de la variable clase.

$$P(A_i \mid c) \propto \mathcal{N}(\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right)$$

Evidentemente, esta estimación tiene el inconveniente de que los datos no siempre siguen una distribución normal.

- **Regresión Logística Multinomial [20]:** El modelo de regresión logística es un modelo de clasificación supervisado, que utiliza las técnicas del modelo de regresión lineal, en las etapas iniciales del procedimiento, para calcular los logits (puntuaciones). Después, en las etapas posteriores, utiliza los logits estimados anteriormente para entrenar un modelo de clasificación, el cual realiza la tarea de clasificación múltiple. Veamos, a continuación, el funcionamiento detallado del modelo.

  En la Figura 3.12 se ilustra la estructura del clasificador por regresión logística multinomial. Para facilitar la comprensión del mismo se divide en diferentes etapas, desde los inputs hasta los outputs.

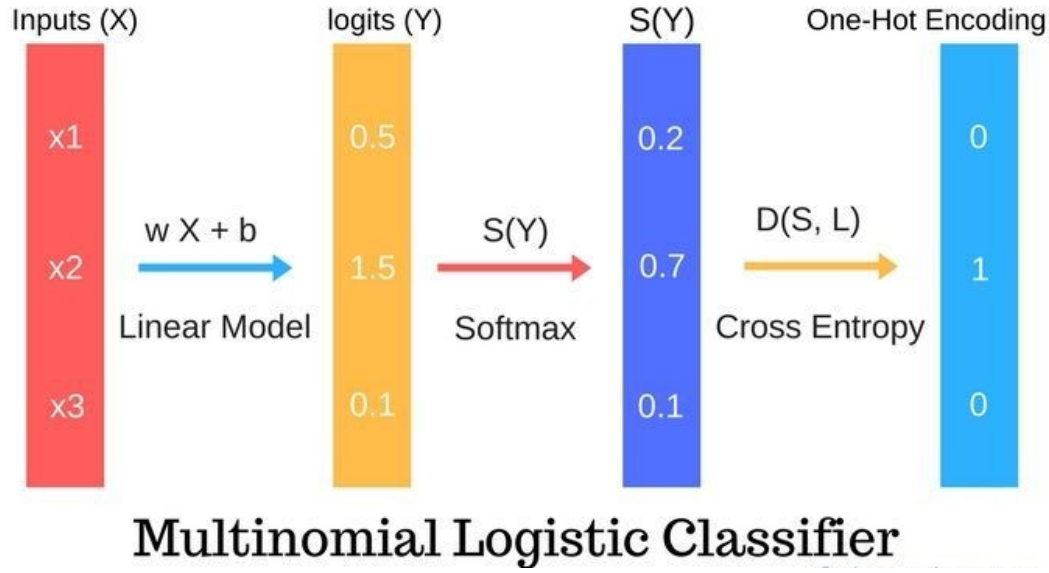

Figura 3.12: Estructura del clasificador por regresión logística multinomial [20].

Cada una de las etapas del clasificador desarrolla la siguiente funcionalidad:

Inputs: Las entradas al clasificador son las características que tenemos en el conjunto de datos. Por ejemplo, si queremos predecir las especies de la flor Iris, las características serían la longitud y anchura del pétalo y el sépalo.

Es importante recordar que los valores de las características deben ser siempre numéricos. En caso de que las características no sean numéricas, es necesario utilizar las técnicas de análisis de datos categóricos apropiadas para convertirlas a valores numéricos.

Modelo lineal: Considérese la siguiente ecuación lineal, **wX + b**; donde **X** es el conjunto de entrada, siendo una matriz que contiene todas las características en forma numérica, $\mathbf{X} = [\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}, ..., \mathbf{X_n}]$; y **w** es la matriz que contiene los pesos, uno para cada característica, $\mathbf{w} = [\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3}, ..., \mathbf{w_n}]$. La salida del modelo lineal resulta: $[\mathbf{w_1} \cdot \mathbf{X_1}, \mathbf{w_2} \cdot \mathbf{X_2}, \mathbf{w_3} \cdot \mathbf{X_3}, ..., \mathbf{w_n} \cdot \mathbf{X_n}]$.

Los pesos de la matriz **w** se actualizan durante el entrenamiento, conforme el clasificador se ajusta al train set.

Logits: Son la salida del modelo lineal, es decir, los valores $[\mathbf{w_1} \cdot \mathbf{X_1}, \mathbf{w_2} \cdot \mathbf{X_2}, \mathbf{w_3} \cdot \mathbf{X_3}, ..., \mathbf{w_n} \cdot \mathbf{X_n}]$. Estas puntuaciones cambian a medida que cambian los pesos y son el resultado en 'bruto' del clasificador.

Función SoftMax: Convertimos la salida del modelo lineal en una distribución de probabilidad, donde las puntuaciones se convierten en probabilidades en el rango [0-1] y la suma de todas las probabilidades es igual a uno. Estas probabilidades constituyen la salida del clasificador, cuya clase pronosticada es la que se corresponde con la probabilidad más alta.

- **Support Vector Machines (SVM) [19]:** Las máquinas de vectores soporte pertenecen a la familia de los *clasificadores lineales* puesto que inducen separadores lineales o hiperplanos en espacios de características de muy alta dimensionalidad (introducidos por funciones núcleo o *kernel*) con un sesgo inductivo muy particular (maximización del margen).

  En primer lugar, recordemos que todo hiperplano en un espacio D-dimensional, $\Re D$, se puede expresar como $h(x) = \langle w, x\rangle + b$, donde $w \in \Re D$ es el vector ortogonal al hiperplano, $b \in \Re$ y $\langle \cdot, \cdot \rangle$ expresa el producto escalar habitual en $\Re D$. Visto como un clasificador binario, la regla de clasificación se puede expresar como: $f(x) = \text{signo}(h(x))$, donde la función signo se define como:

$$\text{signo}(x) = \begin{cases} +1 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases}$$

  En la terminología de clasificación, las $x \in \Re D$ son representaciones vectoriales de los ejemplos, con una componente real por cada atributo, y el vector w se suele denominar *vector de pesos*. Este vector contiene un peso para cada atributo indicando su importancia o contribución en la regla de clasificación. Finalmente, b suele denominarse sesgo (*bias*) y define el umbral de decisión. Dado un conjunto binario (es decir, con dos clases) de datos (ejemplos, vectores o puntos) linealmente separables, existen diversos algoritmos incrementales (*on-line*) para construir hiperplanos (w, b) que los clasifiquen correctamente. Podemos citar, por ejemplo: *Perceptron, Widrow-Hoff, Winnow, Exponentiated-Gradient, Sleeping Experts,* etc. A pesar de que esté garantizada la convergencia de todos ellos hacia un hiperplano solución, las particularidades de cada algoritmo de aprendizaje pueden conducirnos a soluciones ligeramente distintas, puesto que puede haber varios (de hecho infinitos) hiperplanos que separen correctamente el conjunto de ejemplos.

  Suponiendo que el conjunto de ejemplos es linealmente separable, ¿cuál es el 'mejor' hiperplano separador en términos de generalización? La idea que hay detrás de las SVM *de margen máximo* consiste en seleccionar el hiperplano separador que está a la

misma distancia de los ejemplos más cercanos de cada clase. De manera equivalente, es el hiperplano que maximiza la distancia mínima (o *margen geométrico*) entre los ejemplos del conjunto de datos y el hiperplano. Intuitivamente, este hiperplano está situado en la posición más neutra posible con respecto a las clases representadas por el conjunto de datos, sin estar sesgado, por ejemplo, hacia la clase más numerosa. Además, sólo considera los puntos que están en las fronteras de la región de decisión, que es la zona donde puede haber dudas sobre a qué clase pertenece un ejemplo (son los denominados *vectores soporte*). En la Figura 3.13 se presenta geométricamente este hiperplano equidistante (o de margen máximo) para el caso bidimensional.

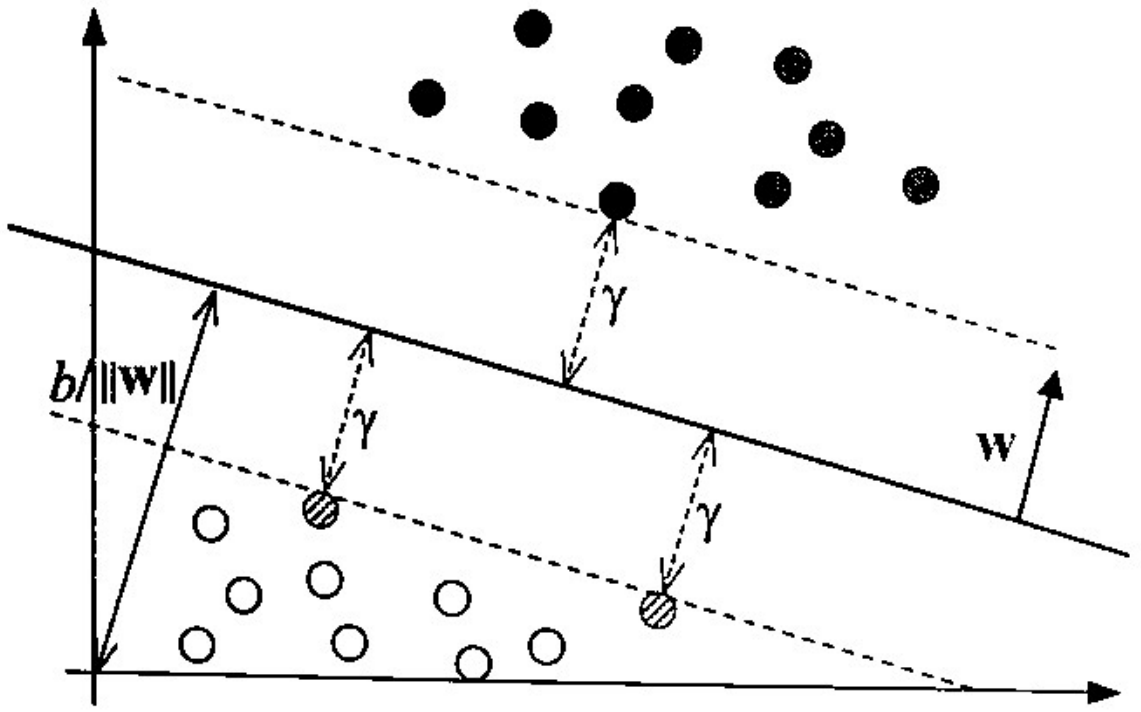

Figura 3.13: Hiperplano $(w,\ b)$ **equidistante a dos clases, margen geométrico** (γ) **y vectores soporte (puntos rayados) [19].**

A nivel práctico, el hiperplano separador de margen máximo ha demostrado una muy buena capacidad de generalización en numerosos problemas reales, así como una robustez notable frente al sobreajuste u overfitting.

A nivel algorítmico, el aprendizaje de las SVM representa un problema de optimización con restricciones que se puede resolver usando técnicas de programación cuadrática (QP). La convexidad garantiza una solución única (esto supone una ventaja con respecto al modelo clásico de redes neuronales) y las implementaciones actuales permiten una eficiencia razonable para problemas reales con miles de ejemplos y atributos.

El aprendizaje de separadores no lineales con SVM se consigue mediante una transformación no lineal del espacio de atributos de entrada (*input space*) en un espacio de características (*feature space*) de dimensionalidad mucho mayor y donde sí es posible separar linealmente los ejemplos. El uso de las denominadas funciones núcleo (*kernel functions*), que calculan el producto escalar de dos vectores en el espacio de características, permite trabajar de manera eficiente en el espacio de características sin necesidad de calcular explícitamente las transformaciones de los ejemplos de aprendizaje.

A veces, los ejemplos de aprendizaje no son linealmente separables ni tan siquiera en el espacio de características. Otras veces no es deseable conseguir un separador perfecto del conjunto de aprendizaje, puesto que los datos de aprendizaje no están libres de errores (ejemplos mal etiquetados, valores de atributos mal calculados, inconsistencias, etc.), comportamientos excepcionales (*outliers*), etc. Focalizarse demasiado en todos los ejemplos de aprendizaje puede comprometer seriamente la generalización del clasificador aprendido por culpa del sobreajuste. En estos casos es preferible ser más conservador y admitir algunos ejemplos de aprendizaje mal clasificados a cambio de tener separadores más generales y prometedores. Este comportamiento se consigue mediante la introducción del modelo de SVM con margen blando (*soft margin*). En este caso, la función objetivo a minimizar está compuesta por la suma de dos términos: el margen geométrico y un término de regularización que tiene en cuenta los ejemplos mal clasificados. La importancia relativa de los dos términos se regula mediante un parámetro, normalmente llamado C. Este modelo, aparecido en 1995, es el que realmente abrió la puerta a un uso real y práctico de las SVM, aportando robustez frente al ruido.

- **K-Nearest Neighbors [19]:** La regla del vecino más próximo simplemente asigna la clase del ejemplo más próximo, utilizando una función de distancia. Esta regla, conocida como *one nearest neighbor* (1-NN), tiene bastantes problemas, ya que ignora la densidad o la región donde se encuentra el ejemplo. Por ejemplo, en la parte izquierda de la Figura 3.14 se muestra un ejemplo de aplicación del vecino más próximo.

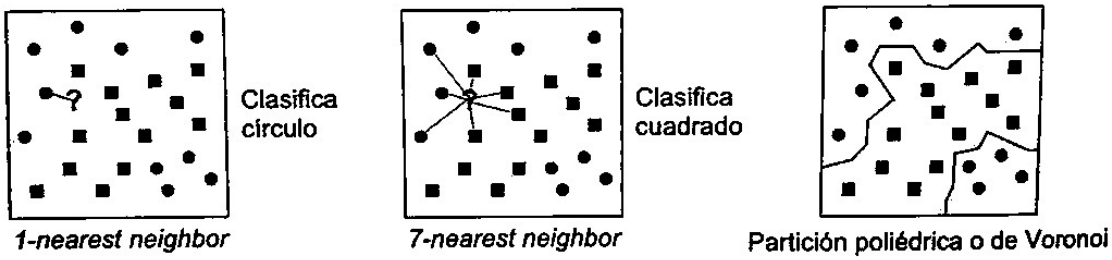

Figura 3.14: Diferencia en el valor de *k* de los vecinos más próximos y partición realizada [19].

Una variante de este método son los *k* vecinos más próximos (kNN, *k-nearest neighbors*) [Cover & Hart 1967] en el que se asigna la clase mayoritaria entre los *k* vecinos más próximos, como se muestra en el panel del medio de la Figura 3.14. Como se puede observar, el valor de *k* es muy importante y es difícil establecer el adecuado. De hecho, como se aprecia en el panel derecho de la Figura 3.14, la expresividad del método es muy alta, y el problema principal de este método es determinar un buen valor de *k*.

- **Árboles de decisión (C4.5) [19]:** La tarea de aprendizaje para la cual los árboles de decisión se adecuan mejor es la clasificación. De hecho, clasificar es determinar de entre varias clases a qué clase pertenece un objeto; la estructura de condición y ramificación de un árbol de decisión es idónea para este problema.

  Debido al hecho de que la clasificación trata con clases o etiquetas disjuntas, un árbol de decisión conducirá un ejemplo hasta una y sólo una hoja, asignando, por tanto, una única clase al ejemplo. Para ello, las particiones existentes en el árbol deben ser también disjuntas. Es decir, cada instancia cumple o no cumple una condición.

  Esta propiedad dio lugar al esquema básico de los primeros algoritmos de aprendizaje de árboles de decisión; el espacio de instancias se iba partiendo de arriba a abajo, utilizando cada vez una partición, es decir, un conjunto de condiciones excluyentes y exhaustivas. Estos algoritmos se llaman algoritmos de partición o algoritmos de 'divide y vencerás'. Otra característica importante de los primeros algoritmos de aprendizaje de árboles de decisión es que una vez elegida la partición dicha partición no se podía cambiar, aunque más tarde se pensara que había sido una mala elección. Por tanto, uno de los aspectos más importantes en los sistemas de aprendizaje de árboles de decisión es el denominado *criterio de partición*, ya que una mala elección de la partición (especialmente en las partes superiores del árbol) generará un peor árbol.

  En la Figura 3.15 se puede observar un algoritmo básico para generar un árbol de decisión

a partir de un conjunto de ejemplos, utilizando la técnica de 'partición'.

```
ALGORITMO Partición(N:nodo, E:conjunto de ejemplos)
  SI todos los ejemplos E son de la misma clase c ENTONCES
     Asignar la clase c al nodo N.
     SALIR;  // Esta rama es pura, ya no hay que seguir partiendo. N es hoja.
  SI NO:
     Particiones := generar posibles particiones.
     MejorPartición:= seleccionar la mejor partición según el criterio de partición.
     PARA CADA condición i de la partición elegida.
        Añadir un nodo hijo i a N y asignar los ejemplos consistentes a cada hijo (Ei).
        Partición(i, Ei).      // Realizar el mismo procedimiento global con cada hijo.
     FIN PARA
  FIN SI
FIN ALGORITMO

Para clasificar un conjunto de ejemplos E, se invoca con la llamada Partición(R,E),
donde R es un nodo raíz de un árbol por empezar.
```

Figura 3.15: Algoritmo de aprendizaje de árboles de decisión por 'Partición' [19].

Simplemente, el algoritmo va construyendo el árbol (desde el árbol que sólo contiene la raíz) añadiendo particiones y los hijos resultantes de cada partición. Lógicamente, en cada partición, los ejemplos se van dividiendo entre los hijos. Finalmente, se llega a la situación en la que todos los ejemplos que caen en los nodos inferiores son de la misma clase y esa rama ya no sigue creciendo. La única condición que hay que exigir es que las particiones al menos separen ejemplos en distintos hijos, con lo que la cardinalidad de los nodos irá disminuyendo a medida que se desciende en el árbol.

Como acabamos de comentar, los dos puntos más importantes para que el algoritmo anterior funcione bien son los siguientes:

- **Particiones a considerar.**
- **Criterio de selección de particiones.**

Esto es lo que diferencia fundamentalmente los distintos algoritmos de 'partición', como CART [Breiman et al. 1984], ID3 [Quinlan 1983][Quinlan 1986], C4.5 [Quinlan 1993], ASSISTANT [Cestnik et al. 1987], etc.

Hasta ahora no hemos tratado cómo aprender sistemas de reglas que se comporten bien con conjuntos de datos con ruido o que no cometan sobreajuste (overfitting). La manera

más frecuente de limitar este problema es modificar los algoritmos de aprendizaje de tal manera que obtengan modelos más generales. En el contexto de los árboles de decisión y conjuntos de reglas, generalizar significa eliminar condiciones de las ramas del árbol o de algunas reglas. En el caso de los árboles de decisión dicho procedimiento se puede ver gráficamente como un proceso de 'poda', como se ilustra en la Figura 3.16.

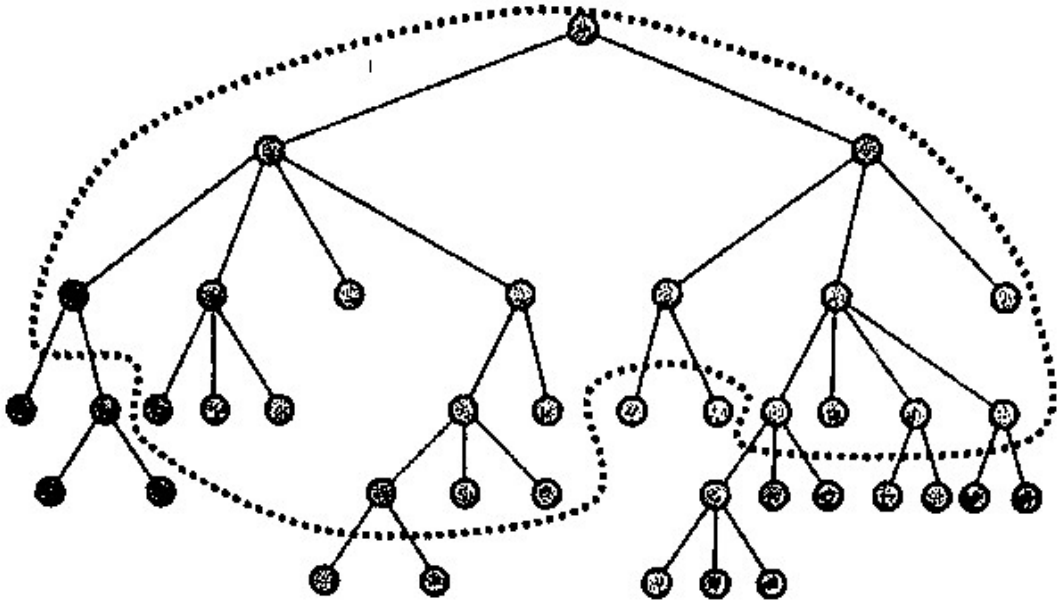

Figura 3.16: Ejemplo de poda [19]. Algunos nodos inferiores son eliminados.

Los nodos que están por debajo del límite de poda se eliminan, ya que se consideran demasiado específicos o, dicho de una manera más informal, se consideran demasiado *ad hoc*.

Podemos distinguir entre métodos de prepoda y pospoda. Evidentemente, se pueden combinar ambas técnicas. De hecho, algunos de los algoritmos más populares, como el C4.5, utilizan una prepoda por cardinalidad (los nodos con una cardinalidad inferior a una cierta constante no se abren) y una pospoda basada en un criterio más sofisticado. Una consecuencia de utilizar prepoda o pospoda (o los dos) es que los nodos hoja ya no van a ser puros, es decir, es posible que tengan ejemplos de varias clases. Normalmente se elegirá la clase con más ejemplos para etiquetar el nodo hoja y hacer, por tanto, la predicción.

- **Random Forest [21]:** Los Random Forest se engloban dentro de los métodos de *ensemble learning*, los cuales generan muchos clasificadores y combinan sus resultados. Dos métodos reconocidos son el boosting (ver, por ejemplo, Shapire et al. 1998) y el bagging

(Breiman 1996) de árboles de clasificación. En el boosting, los árboles sucesivos se construyen otorgando mayor importancia a las muestras clasificadas erróneamente por los árboles anteriores, entendiendo mayor importancia como preferencia a la hora de seleccionarlas para el entrenamiento. Al final, se procede a una votación ponderada para la predicción. En el bagging, los árboles sucesivos no dependen de los árboles anteriores, ya que cada uno de ellos está construido utilizando un subconjunto aleatorio del train set. Finalmente, la predicción se realiza mediante votación por mayoría.

En 2001, Breiman propuso el modelo llamado Random Forest, que añade aleatoriedad adicional al bagging. Además de construir cada árbol utilizando un subconjunto aleatorio distinto de los datos, en el Random Forest cambia la forma en que se construyen los árboles. En los árboles tradicionales, cada nodo se divide utilizando la característica más adecuada, teniendo en cuenta todas las características posibles. En un Random Forest, se utiliza la característica más adecuada para dividir el nodo pero seleccionándola de entre un subconjunto de características aleatorio para ese nodo. Esta estrategia, aunque algo contraria a la lógica, resulta ser muy eficaz en comparación con muchos otros clasificadores, tales como maquinas de vectores soporte o redes neuronales; y es robusta frente al overfitting (Breiman, 2001).

El algoritmo: El algoritmo de un Random Forest, ya sea para clasificación o regresión, se construye de la siguiente manera:

1. Se generan $n_{\text{árbol}}$ subconjuntos aleatorios del train set.
2. Para cada subconjunto aleatorio de los datos de entrenamiento, crece un árbol (al que no se le aplica ningún tipo de poda) con la siguiente modificación: en vez de dividir cada nodo considerando todas las características disponibles, generamos m_{nodo} subconjuntos aleatorios de características y elegimos la característica más adecuada de entre los subconjuntos para dividir cada nodo.
3. Se clasifican nuevos datos combinando las predicciones de los $n_{\text{árbol}}$ árboles.

3.4 Redes LSTM

Como se ha explicado anteriormente, los modelos de clasificación tratados hasta ahora utilizan como entrada vectores numéricos de dimensión fija, los cuales se obtienen al aplicar un modelo BoW a los conjuntos de palabras que representan a los vídeos. Al representar los conjuntos de palabras mediante un modelo BoW, se pierde toda la información acerca del orden

y la estructura original de las palabras. Cada vector numérico obtenido aporta información sobre la aparición de las palabras del vocabulario en el conjunto de palabras de que se trate, pero no informa acerca del orden en el que las palabras aparecen dentro del mismo.

La posición relativa de las palabras dentro de un conjunto de palabras es una información valiosa, teniendo en cuenta el contexto en el que nos encontramos. Cada conjunto de palabras representa la totalidad de los frames de un vídeo ordenados de forma secuencial y, por tanto, el orden en que se presentan las palabras representa el orden temporal de aparición de los frames.

Existen unas redes llamadas Long Short Term Memory (LSTM) capaces de clasificar, no solo en base al contenido de los conjuntos de palabras, al igual que los modelos de clasificación vistos, sino también utilizando la información temporal disponible. De esta forma, son capaces de clasificar un conjunto de palabras (léase 'un vídeo') basándose en el orden de aparición de las palabras que lo forman, es decir, mediante la relación de secuencialidad existente entre los frames del vídeo.

El funcionamiento de una red LSTM es complejo, por lo que la forma más adecuada de exponerlo es mediante una explicación intuitiva [22][23]. La estructura general de este tipo de redes se construye concatenando módulos individuales completamente idénticos, donde cada módulo realiza una serie de operaciones y genera información de salida que, necesariamente, se pasa al módulo siguiente para su correcto funcionamiento. En la Figura 3.17 se ilustra una cadena de tres módulos pero, normalmente, una red LSTM suele tener muchos más.

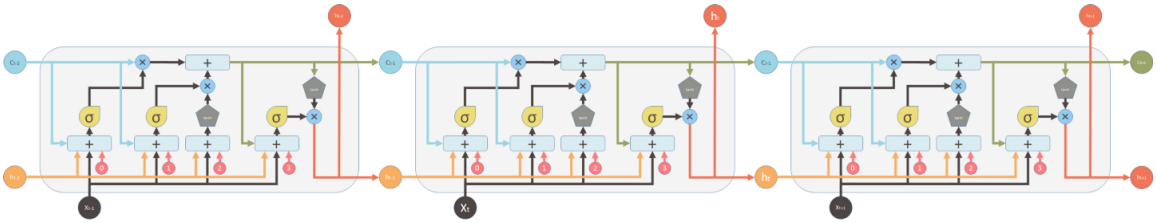

Figura 3.17: Estructura general de una red LSTM [23].

Vista la estructura general, veamos de forma más detallada, en la Figura 3.18, los componentes internos de un módulo para entender el funcionamiento, a grandes rasgos, de la red.

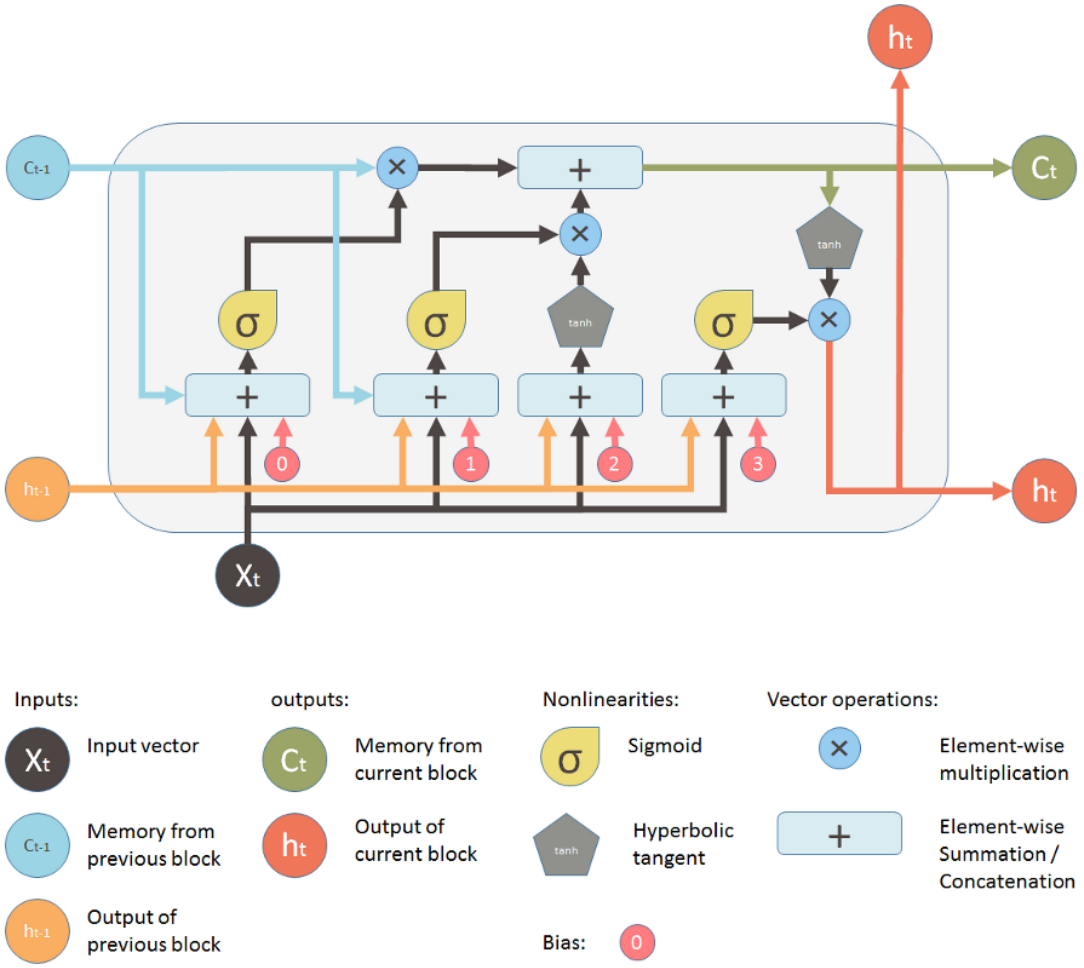

Figura 3.18: Estructura interna de un módulo en una red LSTM [23].

Teniendo en cuenta que cada módulo (*block*) representa un paso de tiempo dentro del marco temporal definido por el número de módulos que haya, es decir, por el tamaño de la red; los inputs y outputs de un módulo quedan indicados en la Figura 3.18. Cada módulo toma decisiones en función de sus inputs y genera unos outputs, utilizando para ello las operaciones combinadas de varias redes neuronales de una capa.

Este tipo de red es capaz de utilizar información vista en el pasado para trabajar de forma más efectiva con la información presente, guardando dicha información pasada en una unidad de memoria llamada *cell state*. Esta unidad de memoria se representa en la Figura 3.18 como la línea superior *C*.

La primera operación que se realiza viene representada por la multiplicación ($\times$) sobre la unidad de memoria y se denomina 'puerta de olvido'. Básicamente, si multiplicamos el vector de memoria que llega del módulo anterior por un vector cercano a cero, se 'olvida' gran parte

de lo guardado en memoria. De forma contraria, si la multiplicación se realiza por un vector cercano a 1 significa que queremos conservar el contenido de la memoria.

La segunda operación se realiza también sobre la unidad de memoria y se representa por la suma (**+**). En este punto, tras haber eliminado anteriormente la información no necesaria de la unidad de memoria, introducimos la nueva información generada por el módulo en la misma. Tras las dos operaciones descritas, el proceso de actualización del contenido de la unidad de memoria finaliza.

Por último, solo queda computar el output del módulo utilizando la información actualizada contenida en la unidad de memoria, controlando que parte de esta información pasa como output y que parte se desecha.

Capítulo 4

Experimentos

Este apartado permitirá exponer los procedimientos utilizados y las tareas concretas realizadas para el desarrollo del presente TFM.

4.1 Datasets

Tanto para el entrenamiento de los clasificadores como para comprobar la validez de la hipótesis propuesta se han necesitado vídeos, necesariamente categorizados o separados por clases. Por lo tanto, con el fin de encontrar gran cantidad de vídeos ya agrupados por categorías y de tener una referencia a la hora de evaluar la bondad de nuestro trabajo de clasificación, se han utilizado datasets como origen de los vídeos empleados en el TFM, los cuales se pasan a describir a continuación.

4.1.1. YouTube-8M

Características generales: YouTube-8M [24] es un dataset multi-etiqueta para clasificación de vídeo. En él se trata la tarea de clasificar vídeo como la de producir etiquetas que son relevantes para un vídeo dado sus fotogramas. En la Figura 4.1 se muestra el explorador del dataset, visualizando un subconjunto de vídeos anotados con la clase 'Guitar'.

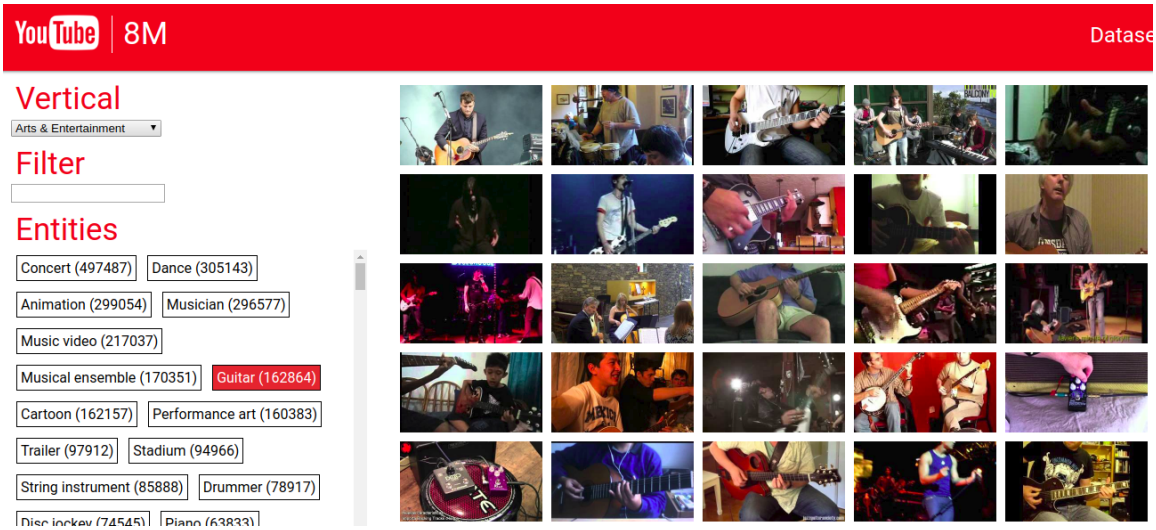

Figura 4.1: YouTube-8M [24]. El explorador del dataset permite navegar entre los vídeos y buscar todos los vídeos etiquetados con cualquiera de las clases. Las clases están agrupadas dentro de 24 categorías.

Los autores construyen un vocabulario de anotaciones visuales a partir de entidades (clases) de Knowledge Graph (banco de datos que recoge millones de datos de las palabras clave que las personas buscan con frecuencia y la intención detrás de dichas palabras) que aparecen como anotaciones acerca de la temática de vídeos de YouTube. Estas anotaciones para cada vídeo las realiza el sistema de anotaciones de vídeos del propio YouTube. Para asegurar que el vocabulario consta de entidades que son visualmente reconocibles, se utilizan varios criterios de filtrado, incluyendo evaluadores humanos. Las entidades en el dataset abarcan actividades, objetos, escenas y eventos, y fueron seleccionadas usando una combinación de su popularidad en YouTube y puntuaciones manuales asignadas por evaluadores humanos sobre su visibilidad.

A continuación, se toma un conjunto de muestra de vídeos para cada entidad, y se usa una red Inception, disponible públicamente, para extraer características de ellos. Específicamente, se decodifican los vídeos a un fotograma por segundo y se extrae la última representación oculta antes de la capa de clasificación para cada fotograma. Se comprimen las características a nivel de fotograma y se ponen en la web para su descarga.

YouTube-8M contiene más de ocho millones de vídeos, asignándoles clases a cada uno de ellos, contemplando una variedad de 4800 clases.

Utilización del dataset: Mediante el explorador del dataset puesto en la red por Google (*https://research.google.com/youtube8m/explore.html*), se han seleccionado 200 vídeos

repartidos en 10 clases dentro de la categoría 'Sports'. Las 10 clases escogidas son las siguientes: 'Basketball', 'Bowling', 'Cycling', 'Football', 'Jumping', 'Parachuting', 'Rallying', 'Surfing', 'Tennis', 'WinterSport'.

El procedimiento utilizado fue visualizar vídeos ofrecidos por el explorador de forma individual. Si el vídeo visualizado realmente pertenecía a la clase en la que se estaba trabajando y la resolución disponible era de 720p como mínimo, el vídeo era descargado. Si el vídeo no cumplía cualquiera de las dos condiciones anteriores, se descartaba. Cabe mencionar el uso del programa 'JDownloader 2' para facilitar la descarga de los vídeos utilizando el código identificativo de cada uno.

4.1.2. Columbia Consumer Video

Características generales: Columbia Consumer Video (CCV) [1] es un dataset cuyos vídeos están grabados, generalmente, de forma no profesional usando dispositivos móviles como smartphones. En ingles, este tipo de vídeos se denominan *consumer videos*. Los vídeos consumer son interesantes porque tienen un rol dominante y creciente en las plataformas de vídeo online, como YouTube.

Para la construcción de este dataset el primer paso es asegurarse de que las categorías seleccionadas son relevantes en función de las necesidades del usuario y válidas para reconocimiento automático. Por tanto, las categorías han sido cuidadosamente escogidas de acuerdo con estudios de interés del usuario y consideraciones de utilidad, detectabilidad, observabilidad y diversidad de contenidos.

Finalmente, las categorías escogidas fueron 20, cubriendo una amplia gama de temas que incluyen objetos, escenas, eventos deportivos y actividades con un significado de alto nivel. La Figura 4.2 muestra un ejemplo para cada categoría.

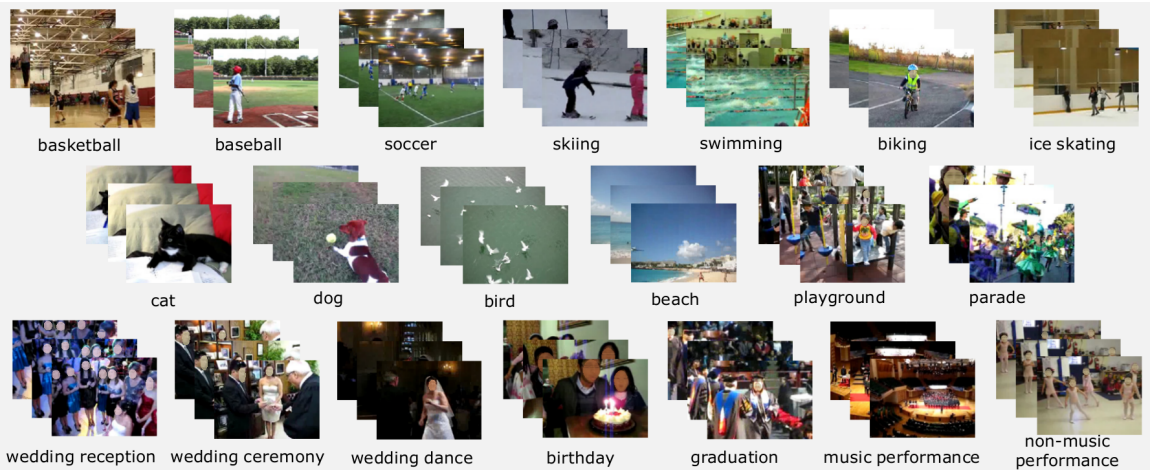

Figura 4.2: Ejemplos en el dataset CCV [1]. Las 5 primeras categorías en la fila de en medio son objetos y escenas, y todas las categorías restantes son eventos.

Los vídeos fueron descargados mediante búsquedas en YouTube. Para asegurarse de que solo se descargaban vídeos consumer, las búsquedas se formularon combinando el string 'MVI' con el nombre de cada una de las categorías. Se utiliza el string 'MVI' porque es el prefijo por defecto en el nombre de los archivos de vídeo grabados por muchas cámaras digitales. Usando esta técnica casi todos los vídeos descargados han resultado ser vídeos consumer.

El dataset final tiene 9317 vídeos, con una duración media de alrededor de 80 segundos. Para el etiquetado de todos estos vídeos los autores utilizan un procedimiento manual, con el que se anotan completamente los vídeos utilizando las 20 categorías. Dicho procedimiento manual se realiza mediante la plataforma Amazon Mechanical Turk (MTurk), que permite emplear usuarios de Internet, normalmente de forma remunerada, para etiquetar los vídeos. La Figura 4.3 muestra la interfaz utilizada en MTurk.

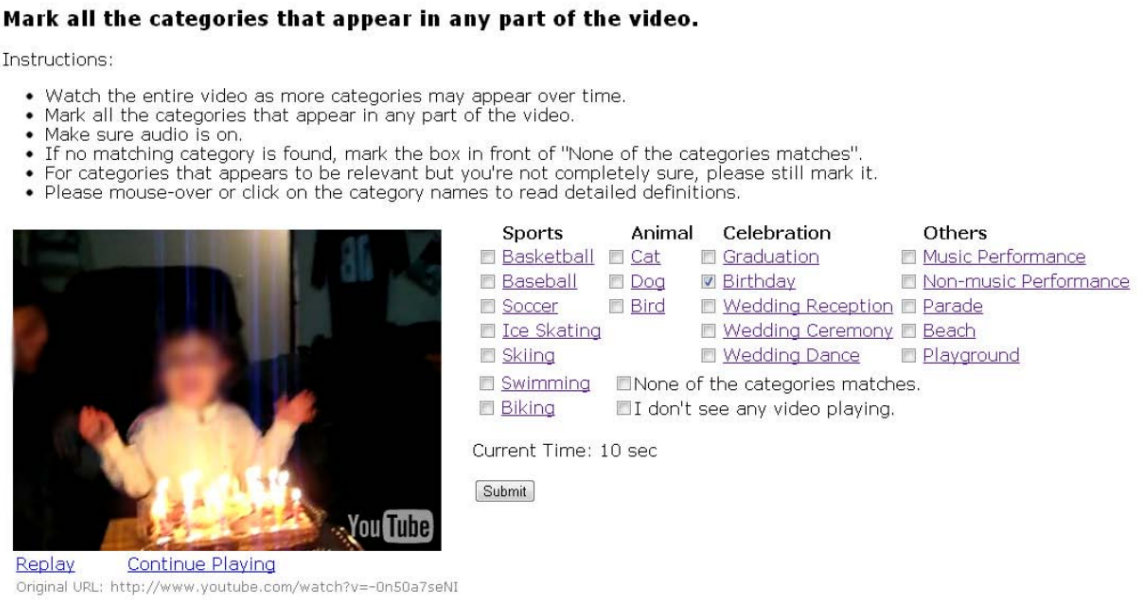

Figura 4.3: Captura de la interfaz en Amazon MTurk [1]. Se muestra un vídeo y el anotador marca todas las categorías que cree que aparecen en el mismo.

Para asegurar que el etiquetado de cada vídeo es correcto los autores usaron dos estrategias. Primero, buscar etiquetas sospechosas y corregirlas en caso de que sean incorrectas. Y segundo, cada vídeo fue asignado a múltiples anotadores independientes, utilizando después una estrategia de votación para filtrar etiquetas erróneas.

Utilización del dataset: En este caso, los autores del dataset suministran una serie de archivos de texto. Tenemos uno con las direcciones de los vídeos que integran el dataset en YouTube. Por otra parte, tenemos un archivo de texto que contiene una matriz vídeo/categoría, cuyas filas se corresponden con el archivo que contiene las direcciones.

Teniendo las direcciones y las categorías para cada dirección, se ha utilizado código Python para construir un archivo de texto único, integrando las direcciones de cada vídeo con las categorías que le corresponden a cada uno. Utilizando este archivo, y también mediante código Python, se han descargado los vídeos. De los 9317 vídeos que integran el dataset se han podido descargar 7578, ya que muchos de ellos ya no se encuentran disponibles en YouTube, ya sea porque el usuario que subió el vídeo lo ha eliminado o bloqueado, o porque YouTube ha eliminado el vídeo por infringir derechos de Copyright.

4.2 Diagramas conceptuales del desarrollo

Antes de exponer las tareas realizadas para el desarrollo del TFM, se ha considerado necesario presentar una serie de diagramas explicativos (Figura 4.4) que, de forma conceptual y no necesariamente rigurosa con respecto a las tareas reales realizadas, permitan visualizar el workflow del TFM. De esta forma, antes de entrar en materia viendo cada uno de los pasos dados, ya se tiene presente el esquema general a desarrollar.

En el 'Diagrama del procesado de los vídeos' tenemos el procesado inicial por el que pasa cada uno de los vídeos. Cada frame del vídeo pasa por un detector que extrae entidades visuales del mismo. Luego, la información extraída de todos los frames se une para conformar la representación textual del vídeo.

Para el segundo y tercer diagrama, 'Diagrama del procesado de la información textual y aplicación de modelos' y 'Diagrama del procesado de la información textual y aplicación de red LSTM', se unifica la información textual de todos los vídeos, se realizan algunos preprocesos necesarios, y se entrenan y aplican distintos modelos, así como una red LSTM.

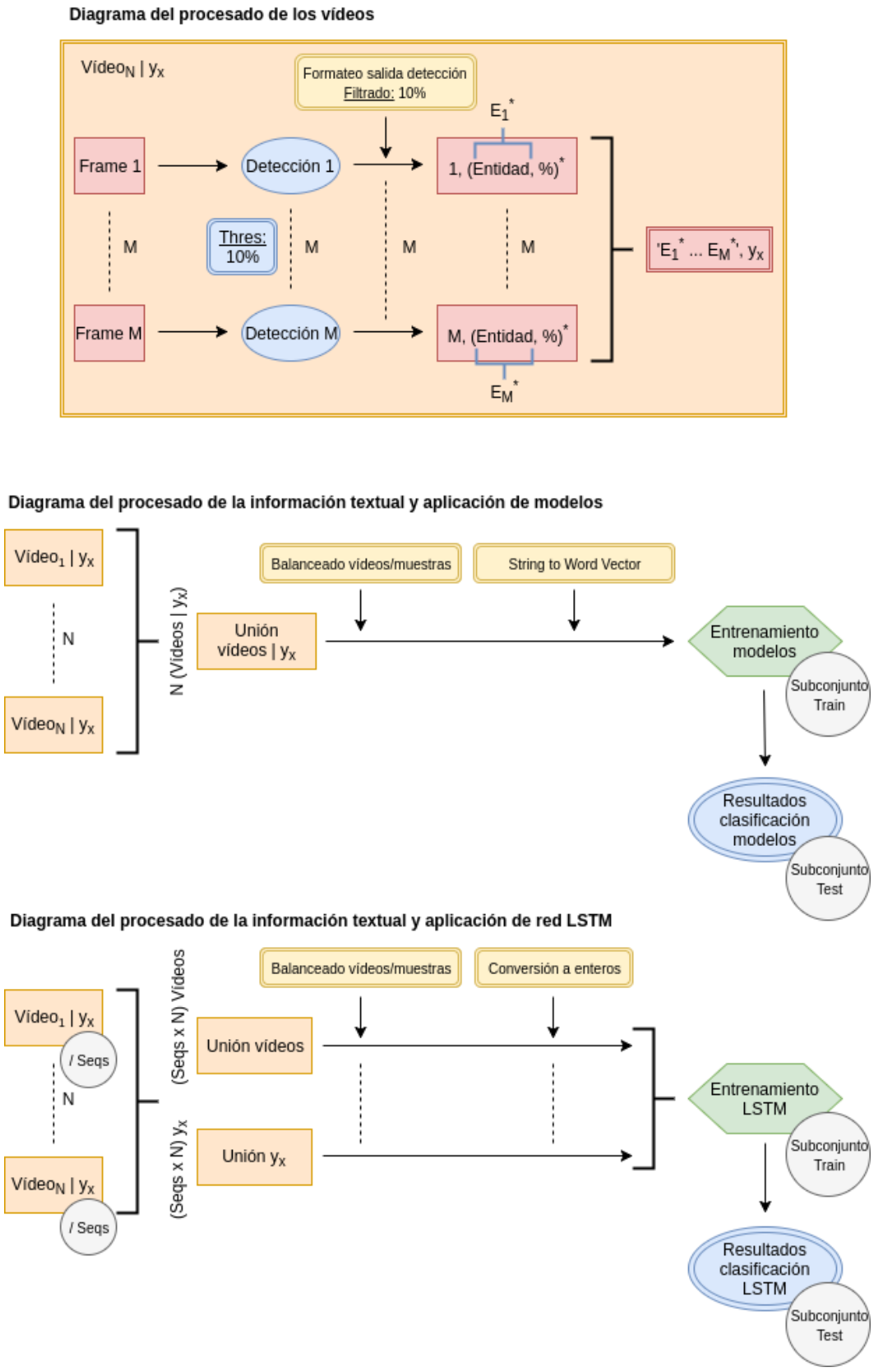

Figura 4.4: Diagramas conceptuales acerca del desarrollo del TFM.

4.3 Utilización de YOLO9000

El procedimiento a seguir para ambos datasets una vez descargados los vídeos es el mismo, se pasan como entrada a la red convolutiva YOLO9000. La salida devuelta por la red para cada vídeo se puede ver en la Figura 4.5.

```
Demo
video file: /home/local/roc/cortega/disco-scratch/0D6PNx2au84.mp4

FPS:19152.1
Frames computados: 1
Objetos:

FPS:3.1
Frames computados: 2
Objetos:
object: 10%

FPS:5.2
Frames computados: 3
Objetos:
device: 12%
instrumentality: 14%

FPS:5.3
Frames computados: 4
Objetos:
person: 11%
device: 13%

```

Figura 4.5: Salida entregada por YOLO9000. Para cada frame del vídeo tenemos los objetos detectados y el porcentaje de confianza acerca de la veracidad de la detección para cada uno de esos objetos.

Se ha configurado la red para que no devuelva objetos con un porcentaje de confianza menor que el 10 %, ya que de lo contrario tendríamos mucho ruido, es decir, muchos objetos con un porcentaje de confianza bajo que realmente no aparecen en el vídeo.

Debido a que YOLO9000, para un funcionamiento rápido, es computacionalmente exigente, se ha ejecutado en un servidor que se encuentra en el Centro de Proceso de Datos del Instituto Universitario SIANI de la ULPGC, cuyas características técnicas son: una CPU Intel Xeon E5-2630 v4, 256 Gb de memoria RAM y dos GPUs Nvidia Tesla K40m con 12 Gb de memoria cada una.

Llegados a este punto, tenemos un archivo de texto con los resultados de la red para cada uno de los vídeos.

4.4 Preprocesado de la salida obtenida de YOLO9000

El preprocesado de la salida devuelta por la red para cada vídeo se ha realizado enteramente usando Python y trabajando con los dos datasets, utilizándolos de forma independiente. Los procesos realizados en este sentido han sido los siguientes:

- **Creación de carpetas:** Se ha automatizado la creación de carpetas cuyos nombres son las categorías con las que se trabaja en cada dataset. A la hora de trabajar con el dataset CCV también se ha creado una carpeta llamada 'Unclassified'. En el siguiente punto veremos para que se usan todas estas carpetas.

- **Copia de resultados a carpetas:** La red YOLO9000 va guardando los archivos de texto de salida en un directorio, independientemente de la categoría a la que pertenezca el archivo. En este punto del preprocesado se ordenan los archivos de resultados copiándolos a las carpetas creadas anteriormente según la categoría a la que pertenezcan. En el caso del dataset CCV también hay una carpeta llamada 'Unclassified', porque en este dataset hay vídeos sin categorizar, por lo que los archivos de resultados correspondientes a vídeos sin categorías se han guardado en esa carpeta.

- **Primera transformación de los resultados:** En la Figura 4.5 se puede ver el formato de los resultados de detección. Se ha realizado una transformación de este formato a uno más limpio y compacto. En la Figura 4.6 podemos ver el nuevo formato generado.

  Cabe destacar que, para cada archivo de resultados correspondiente a cada vídeo, se han generado cinco archivos aplicando este cambio de formato. Se han establecido cinco porcentajes (10 %, 20 %, 30 %, 40 % y 50 %), de tal forma que al generar los archivos con el cambio de formato, el primero conservará los objetos cuyo porcentaje de confianza sea mayor que el 10 %, es decir, todos. El segundo archivo conservará aquellos objetos cuyo porcentaje de confianza sea mayor que el 20 % y así sucesivamente hasta el quinto archivo con el 50 %. Esto se ha hecho con el objetivo de poder trabajar posteriormente jugando con la cantidad de objetos detectados en cada vídeo y con el hecho de que, a mayor porcentaje de confianza para un objeto, más representativo a efectos de una correcta clasificación del vídeo debería ser. Veremos en el apartado de resultados si esto realmente es así.

```
1
2, object, 25, object, 15
3, living thing, 13, instrumentality, 38, living thing, 10, person, 21
4, living thing, 14, instrumentality, 38, organism, 11, person, 22
5, living thing, 16, instrumentality, 38, living thing, 14, person, 20
6, living thing, 16, instrumentality, 37, living thing, 14, person, 17
7, living thing, 15, instrumentality, 36, living thing, 13, person, 14
8, living thing, 14, instrumentality, 36, living thing, 11, person, 14
9, living thing, 13, instrumentality, 37, organism, 12, person, 16
10, living thing, 12, instrumentality, 38, organism, 13, person, 17
11, living thing, 12, instrumentality, 38, organism, 14, person, 18
12, living thing, 12, instrumentality, 39, person, 15, person, 19
13, living thing, 12, instrumentality, 39, person, 16, person, 20
14, living thing, 11, instrumentality, 40, person, 19, person, 20
15, living thing, 11, instrumentality, 42, person, 21, person, 19
16, living thing, 11, instrumentality, 44, organism, 23, person, 18
17, living thing, 11, instrumentality, 44, organism, 23, person, 17
18, living thing, 11, instrumentality, 43, organism, 21, person, 17
19, living thing, 12, instrumentality, 40, person, 19, person, 17
20, nonworker, 10, living thing, 13, instrumentality, 38, person, 17, person, 18
21, living thing, 13, instrumentality, 37, person, 19, person, 18
22, whole, 13, artifact, 36, person, 21, person, 18
```

Figura 4.6: Primera transformación de los resultados. Ahora toda la información para cada frame del vídeo se encuentra en la misma línea. Nos encontramos el número de frame y, seguidamente, los objetos detectados para ese frame con sus porcentajes de confianza.

- **Segunda transformación a 'arff':** Partiendo del procesado de la Figura 4.6 para cada vídeo, se genera el archivo 'arff' final, preparado para su utilización usando las librerías de Weka. En la Figura 4.7 podemos ver el formato de este archivo.

  En este fichero 'arff', cada una de las líneas representa la información de un vídeo distinto. Encontramos strings con palabras, que se forman concatenando las palabras de cada frame usando los archivos procesados en el paso anterior. Para cada línea del 'arff' tenemos un archivo generado en el paso anterior que contiene las palabras de cada frame en cada vídeo. Cada string esta acompañada de una etiqueta que proporciona la clase para cada vídeo.

  Para la generación de este archivo podemos utilizar distintos conjuntos de archivos provenientes del paso anterior, en función del porcentaje de confianza para las palabras que queramos poner como límite.

```
1 @relation test
2 @attribute etiquetas string
3 @attribute clase {Cat, Dog, Set}
4 @data
5 'whole whole living thing living thing instrumentality instru
6 'organism artifact car self-propelled vehicle car car living
  person organism living thing', Dog
7 'clock car self-propelled vehicle', Dog
8 'person person instrumentality person organism living thing o
9 'living thing hunting dog carnivore', Set
10 'contestant person organism living thing', Set
11 'contestant instrumentality person living thing', Dog
12 'instrumentality person organism living thing carnivore', Cat
```

Figura 4.7: Segunda transformación a 'arff'. Las 4 primeras líneas representan la cabecera del archivo 'arff'. Después, cada una de las strings representan un vídeo, acompañadas de su etiqueta de clase.

- **Balanceado de las muestras:** En el caso del dataset construido a partir de YouTube-8M, el número de muestras por clase es de 20, por lo que no es necesario ajustar el número de muestras de ninguna clase. Sin embargo, para el dataset CCV las muestras entre clases están desbalanceadas. En la Figura 4.8 podemos ver la distribución de las muestras para el dataset original.

  Hay tres clases con una cantidad de muestras bastante superior al resto, lo cual no nos interesa a la hora de entrenar modelos de clasificación. Para solventar esto se ha desarrollado un código que permite limitar el número de muestras por clase. Por ejemplo, un límite adecuado para el dataset CCV pudiera ser 350 muestras por clase. De esta forma se elimina el exceso de muestras en tres de las clases y se balancea el dataset.

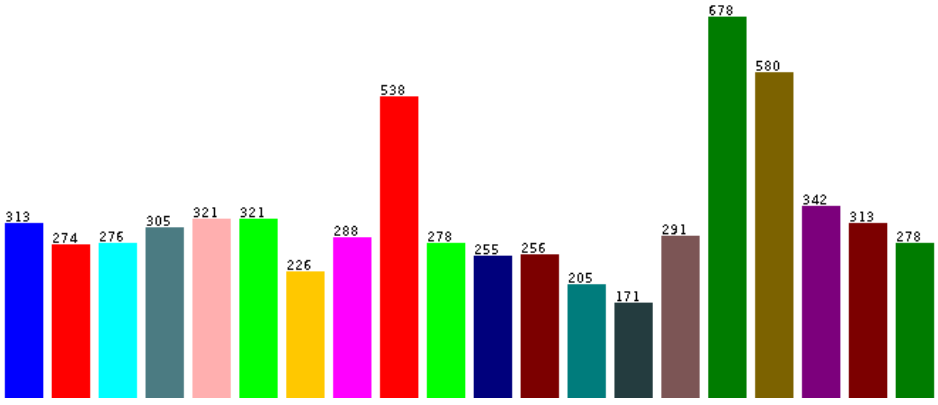

Figura 4.8: Distribución de las muestras en cada clase para el dataset CCV. La distribución de muestras entre las 20 clases es desigual. La clase con más muestras tiene cinco veces más que la clase con menos muestras.

4.4.1. Preprocesado adicional para la aplicación de modelos de clasificación

Tras generar, en la fase de preprocesado general, los archivos 'arff' que contienen cada uno de los datasets, ya se pueden utilizar las librerías para Java de Weka con el fin de aplicar modelos de clasificación, pero realizando antes unos preprocesos adicionales. Para la aplicación de dichos preprocesos, así como para la utilización de los modelos, se ha desarrollado un código en Java utilizando el entorno de desarrollo 'NetBeans IDE 8.2', cuyas funciones son las siguientes:

- **Importación del dataset a utilizar.**
- **Aleatorización de las muestras del dataset.**
- **Aplicación del filtro String To Word Vector (STWV):** Este filtro aplica un modelo BoW, el cual ha sido explicado anteriormente en el apartado de **Metodología**, sobre el dataset.

  En la Figura 4.9 podemos ver el formato obtenido tras aplicar este filtro al archivo 'arff' mostrado en la Figura 4.7. Los vectores no tienen todos el mismo tamaño porque los valores que corresponden a palabras que no aparecen, es decir, los valores que son 0, se han omitido.

```
 3 @attribute clase {Cat,Dog,Set}
 4 @attribute artifact numeric
 5 @attribute canine numeric
 6 @attribute carnivore numeric
 7 @attribute instrumentality numeric
 8 @attribute living numeric
 9 @attribute organism numeric
10 @attribute person numeric
11 @attribute thing numeric
12 @attribute whole numeric
13 @attribute car numeric
14 @attribute clock numeric
15 @attribute contestant numeric
16 @attribute self-propelled numeric
17 @attribute vehicle numeric
18 @attribute dog numeric
19 @attribute hunting numeric
20
21 @data
22 {1 1,4 1,5 1,8 1,9 1}
23 {0 Dog,1 1,4 1,5 1,6 1,7 1,8 1,10 
24 {0 Dog,10 1,11 1,13 1,14 1}
25 {2 1,3 1,4 1,5 1,6 1,7 1,8 1}
26 {0 Set,3 1,5 1,8 1,15 1,16 1}
27 {0 Set,5 1,6 1,7 1,8 1,12 1}
28 {0 Dog,4 1,5 1,7 1,8 1,12 1}
29 {3 1,4 1,5 1,6 1,7 1,8 1}
```

Figura 4.9: 'arff' en formato STWV. El vocabulario, formado por los distintos atributos, se construye a partir de las palabras presentes en las strings originales.

Para finalizar, se ha indicado anteriormente el uso de las librerías para Java de Weka en la aplicación de modelos y algunos preprocesos, como el filtro STWV. Veamos, a continuación, que es Weka exactamente.

Weka [25][26] es un software de código abierto publicado bajo General Public License (GNU). Este contiene una colección de algoritmos de machine learning para tareas de minería de datos, los cuales pueden ser utilizados directamente sobre un dataset o llamados desde un código Java gracias a las librerías. Weka contiene herramientas para preprocesado de datos, clasificación, regresión, clustering, reglas de asociación, y visualización; y también es adecuado para desarrollar nuevos esquemas de machine learning.

Otra de las características de Weka es que posee una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades. Al lanzar Weka se abre el Weka GUI Chooser, ventana intermedia desde la que lanzar las aplicaciones incorporadas. En la Figura 4.10 podemos ver el aspecto de esta ventana.

Figura 4.10: Weka GUI Chooser. Desde esta ventana se pueden lanzar las aplicaciones incluidas en Weka.

Las aplicaciones, de forma resumida, tienen la siguiente utilidad:

- **Explorer:** Es el entorno más utilizado para la realización de operaciones con un único dataset. Dispone de varios paneles que dan acceso a los componentes principales de las diferentes categorías de métodos que incorpora.
- **Experimenter:** Permite la comparación sistemática de una ejecución de los algoritmos predictivos de Weka sobre una colección de conjuntos de datos.

- **Knowledge Flow:** Es una interfaz que, en esencia, implementa las mismas funciones que el Explorer, y además permite 'arrastrar y soltar' en un entorno gráfico. También puede ofrecer aprendizaje incremental.
- **Workbench:** Una aplicación 'todo-en-uno' que combina todas las funcionalidades de las otras dentro de la misma ventana.
- **Simple CLI:** Simple CLI es la abreviatura de 'Simple Command-Line Interface'. Se trata de una consola que permite acceder a todas las opciones de Weka desde la línea de comandos.

4.4.2. Preprocesado adicional para la aplicación de red LSTM

La red LSTM se ha aplicado únicamente sobre el dataset CCV y, para el correcto funcionamiento de la misma, es necesario realizar preprocesos adicionales sobre el archivo 'arff' generado durante la fase de preprocesado general. Recordemos que este archivo 'arff' es el que contiene el dataset CCV al completo, tras aplicar el balanceado de muestras fijando el límite de muestras por clase a 350.

Los preprocesos adicionales llevados a cabo se detallan a continuación:

- **Creación de secuencias de vídeo:** Partimos de un archivo 'arff' con el formato presentado en la Figura 4.7, donde en cada línea tenemos una string representando un vídeo, con su correspondiente etiqueta.

  La naturaleza de las redes LSTM implica el uso del concepto de 'pasos de tiempo'. Una red LSTM es alimentada con vectores que representan secuencias, donde cada posición del vector es un paso de tiempo en una escala temporal. Debido a que, por norma general, las strings que representan a los vídeos son muy largas, es necesario cortarlas para el correcto aprendizaje de la red, ya que esta no es capaz de aprender secuencias extremadamente largas.

  Conociendo lo anterior, se presenta de manera natural la posibilidad de trabajar con secuencias de vídeo más cortas que los vídeos originales. Se ha optado por dividir las strings originales en varias secuencias, consiguiendo así strings de tamaño más adecuado y un mejor aprovechamiento de la información contenida en las mismas, ya que multiplicamos el número de muestras con las que trabajar.

  Mediante código Python se ha dividido cada una de las strings originales en 20 secuencias

y se han creado dos archivos de texto, uno para las nuevas secuencias y otro para las etiquetas de clase correspondientes, línea a línea.

- **Conversión a números enteros:** Keras, para su funcionamiento, requiere que los datos de entrada sean listas de números enteros. Por lo tanto, es necesario convertir las secuencias, que son strings con palabras, a las citadas listas. De la misma forma, es necesario convertir las etiquetas al formato mencionado.

  En el caso de las secuencias, se ha construido un vocabulario aplicando un Term Frequency de forma global sobre el dataset, es decir, la palabra más frecuente del dataset se sustituye por un 1, la segunda más frecuente por un 2, y así sucesivamente para todas las palabras que aparecen en el dataset. En la Figura 4.11 podemos ver el formato obtenido tras la transformación. Una vez se han generado las secuencias en formato de lista de números enteros, las guardamos como *array* de NumPy en un archivo binario.

```
1 [27, 27, 27, 27, 27, 297, 6, 28, 49, 118, 36,
  6, 6, 1, 49, 164, 36, 6, 28, 164, 36, 21, 23,
  21, 1, 6, 49, 15, 6, 1, 11, 6, 49, 6, 36, 6, 1
  6, 16, 1, 11, 11, 49, 5, 140, 131, 16, 28, 11,
  164, 36, 11, 16, 11, 230, 6, 1, 11, 11, 49, 6,
  6, 70, 49, 6, 5, 6, 199, 49, 6, 6, 6, 199, 49,
  6, 49, 118, 36, 6, 70, 55, 49, 118, 36, 15, 6,
  33, 199, 6, 6, 49, 33, 70, 6, 6, 164, 36, 33,
  33, 20, 186, 248, 154, 1, 21, 49, 257, 36, 65,
  6, 199, 49, 269, 154, 257, 36, 65, 6, 199, 49,
  36, 2, 3, 65, 49, 257, 36, 5, 271, 263, 6, 49,
  263, 11, 297, 5, 6, 164, 36, 6, 5, 259, 11, 23
  16, 6, 5, 6, 6, 118, 36, 5, 2, 3, 124, 16, 6,
  5, 6, 230, 427, 34, 230, 33, 6, 199, 248, 154,
  16, 5, 79, 92, 6, 164, 36, 5, 16, 5, 79, 92, 6
  15, 297, 6, 55, 6, 230, 6, 55, 230, 11, 6, 27,
  230, 11, 36, 164, 36, 2, 3, 5, 56, 230, 6, 49,
  257, 36, 6, 257, 36, 36, 257, 36, 124, 36, 257
  11, 230, 65, 269, 154, 6, 49, 257, 36, 6, 5, 5
  61, 56, 124, 49, 257, 36, 6, 61, 56, 49, 257,
  257, 36, 6, 2, 3, 56, 6, 49, 257, 36, 6, 2, 3,
  36, 4, 2, 3, 15, 257, 36, 4, 5, 11, 4, 33, 27,
  11, 5, 4, 16, 6, 6, 296, 5, 27, 6, 6, 70, 15,
2 [6, 6, 70, 40, 5, 11, 6, 6, 5, 70, 40, 15, 6,
  56, 15, 11, 5, 5, 2, 3, 11, 5, 6, 15, 15, 2, 3
  70, 16, 5, 21, 6, 5, 4, 16, 2, 3, 21, 36, 5, 2
  1, 5, 55, 40, 15, 15, 5, 40, 40, 15, 27, 15, 5
  56, 2, 3, 33, 5, 4, 33, 2, 3, 11, 2, 3, 5, 33,
  1, 33, 6, 1, 11, 11, 4, 171, 1, 65, 6, 1, 11,
  6, 2, 3, 40, 15, 5, 2, 3, 33, 6, 11, 4, 40, 6,
  1, 5, 27, 6, 45, 24, 1, 276, 4, 4, 1, 15, 6, 6
  3, 15, 6, 11, 1, 15, 6, 11, 1, 2, 3, 6, 11, 1,
  3, 6, 1, 2, 3, 16, 11, 1, 2, 3, 6, 11, 1, 4, 6
  1, 5, 11, 62, 4, 11, 6, 1, 6, 6, 11, 62, 1, 11
  45, 24, 5, 276, 5, 6, 6, 33, 6, 11, 5, 15, 6,
  6, 6, 11, 1, 6, 15, 6, 5, 6, 11, 1, 11, 15, 6,
  6, 6, 11, 6, 2, 3, 6, 6, 16, 11, 6, 28, 6, 6,
  27, 6, 11, 1, 5, 6, 27, 6, 11, 1, 6, 6, 27, 6,
  56, 56, 21, 5, 27, 6, 56, 1, 11, 21, 5, 15, 6,
  6, 5, 21, 2, 3, 6, 40, 1, 11, 6, 2, 3, 21, 2,
  11, 4, 6, 33, 40, 1, 62, 2, 3, 11, 1, 6, 40, 1
  6, 56, 1, 171, 276, 2, 3, 2, 3, 56, 6, 1, 6, 5
  1, 33, 16, 11, 1, 5, 40, 6, 4, 33, 16, 11, 1,
  6, 4]
```

Figura 4.11: Secuencias en formato de lista de números enteros. Cada lista representa una secuencia de vídeo.

Por otro lado, para las etiquetas se han efectuado dos transformaciones. Primero, se transforma cada etiqueta a número entero. Por ejemplo, si tenemos 20 clases, la numeración para las etiquetas de clase irá del 0 al 19. Segundo, una vez tenemos las etiquetas como números enteros, aplicamos una codificación *one-hot*. Esto es necesario en Keras cuando se quiere clasificar con más de dos clases. Cada número entero pasa a ser una lista con 19 ceros y 1 uno en el índice correspondiente a la clase de la secuencia de que se trate. Una vez realizada la codificación, la guardamos en un archivo de texto.

- **Transformaciones sobre las secuencias:** Tras cargar las secuencias y las etiquetas de clase, aleatorizamos el dataset y lo dividimos en conjuntos de entrenamiento y test. Llegados a este punto, es necesario realizar una serie de transformaciones sobre las secuencias para maximizar el rendimiento de la red LSTM y facilitar su aprendizaje.

  Por ejemplo, considérese la siguiente secuencia:

$$[3,5,2,16,7,7]$$

  Primero, incrementamos en 3 cada uno de los números de la secuencia, con la finalidad de liberar del vocabulario el 1, el 2 y el 3. Nos queda:

$$[6,8,5,19,10,10]$$

  Segundo, añadimos el 1 liberado al inicio de la secuencia. Durante el aprendizaje, la LSTM terminará reconociendo el 1 como un carácter que indica inicio de secuencia. Aunque en este punto esta operación no tenga mucho sentido, lo tendrá en breve. La secuencia queda de la siguiente manera:

$$[1,6,8,5,19,10,10]$$

  Con respecto al 2 y al 3 que también fueron liberados del vocabulario, sus usos son los siguientes. Como ya se ha indicado, el vocabulario palabras-enteros se ha construido aplicando un Term Frequency de forma global sobre el dataset. Esto nos permite establecer un límite, de tal forma que si, por ejemplo, queremos trabajar considerando solo las 8 palabras más frecuentes del vocabulario, todo número mayor que 8 en la secuencia es sustituido por un 2. La secuencia con la que hemos estado trabajando quedaría de la siguiente forma:

$$[1,6,8,5,2,2,2]$$

Finalmente, el número 3 simplemente no tiene ningún uso, no aparece en las secuencias en ningún caso.

A continuación, con el objetivo de que todas las secuencias tengan el mismo tamaño o número de pasos de tiempo, se realiza un truncado o un padding en función de la longitud de la secuencia con respecto a un número de pasos de tiempo fijado. Por ejemplo, si queremos tener secuencias con 10 pasos de tiempo, aquellas secuencias que tengan menos de 10 pasos de tiempo se modificarán mediante padding, y las que tengan más de 10 pasos de tiempo se modificarán mediante truncado.

El padding consiste en añadir ceros delante de la secuencia hasta conseguir el número de pasos de tiempo fijado. Por otra parte, el truncado elimina pasos de tiempo comenzando por el inicio de la secuencia hasta dejar disponible solo el número de pasos de tiempo fijado.

Veamos, a continuación, un ejemplo de padding para conseguir 10 pasos de tiempo:

$$[1,6,8,5,19,10,10] \rightarrow [0\ 0\ 0\ 1\ 6\ 8\ 5\ 19\ 10\ 10]$$

Por otro lado, se muestra un ejemplo de truncado para ajustar la secuencia a 10 pasos de tiempo:

$$[1,8,15,9,13,12,9,9,4,6,4,6,5,7] \rightarrow [13\ 12\ 9\ 9\ 4\ 6\ 4\ 6\ 5\ 7]$$

Tras el proceso de truncado y padding se consigue que todas las secuencias tengan la misma longitud pero de tal forma que la LSTM encuentre coherencia entre los pasos de tiempo. En las secuencias con padding, la red termina interpretando que los ceros son pasos de tiempo vacíos y que el 1 marca el inicio de secuencia. Sin embargo, en las secuencias con truncado no es necesario el 1 puesto que la información relevante está disponible desde el primer paso de tiempo.

Tras todo este proceso de manipulación de las secuencias, ya están preparadas para utilizarse en el modelo construido.

En el apartado de **Resultados** se presenta la configuración empleada y el rendimiento obtenido con este modelo sobre el dataset CCV.

Capítulo 5

Resultados

Se presentan en este apartado los resultados obtenidos al acometer la tarea de clasificar vídeo desde el enfoque de la minería de texto, utilizando para ello distintos modelos de clasificación, así como una red LSTM.

5.1 Modelos de clasificación

Los modelos de clasificación utilizados, haciendo uso de las librerías de Weka mediante código Java, han sido los siguientes: Naive Bayes, Regresión Logística Multinomial, SVM, K-Nearest Neighbors, C4.5 y Random Forest. Al trabajar con el dataset CCV se ha utilizado, además, el MultiClassClassifier. Este clasificador nos permite tener más opciones a la hora de ejecutar la SVM.

La metodología de entrenamiento y test utilizada es el Holdout. Consiste en dividir el dataset en dos subconjuntos, entrenar los modelos de clasificación con el subconjunto de mayor tamaño y calcular la medida de calidad de cada modelo utilizando el subconjunto de menor tamaño, es decir, las muestras no utilizadas en el entrenamiento.

A continuación, en los subapartados siguientes, se presentan los resultados obtenidos con cada modelo sobre cada uno de los dos datasets empleados.

5.1.1. YouTube-8M

Comenzaremos mostrando los resultados obtenidos por los modelos para el dataset construido a partir de YouTube-8M. Cabe destacar que, para la construcción del dataset durante la fase de preprocesado, se han conservado los objetos cuyo porcentaje de confianza es mayor al 10 %, es decir, todos los objetos posibles.

Con el fin de garantizar la reproducibilidad de los resultados, se detallan a continuación las opciones utilizadas tanto en los filtros como en los modelos:

- **String To Word Vector:** Se ha utilizado una codificación TF-IDF y se ha construido el vocabulario utilizando todas las palabras detectadas en el dataset.
- **Naive Bayes:** Se ha especificado el uso de una discretización para convertir los atributos numéricos a nominales.
- **Regresión Logística Multinomial:** Se han utilizado las opciones por defecto.
- **SVM:** El parámetro 'soft-margin *C*' se ha fijado al valor 1 y como kernel se ha utilizado un Normalized Polynomial.
- **K-Nearest Neighbors:** Se ha considerado K = 10.
- **C4.5:** Se han utilizado las opciones por defecto. Estas implican la poda del árbol para evitar el overfitting.
- **Random Forest:** Se han utilizado las opciones por defecto, las cuales no imponen un límite de profundidad a la hora de construir los árboles.

El Holdout se ha hecho de la siguiente manera:

- **Conjunto de entrenamiento:** 70 % del tamaño total del dataset.
- **Conjunto de test:** 30 % restante del tamaño total del dataset.

Los resultados obtenidos con los seis modelos de clasificación utilizados se presentan en el Cuadro 5.1.

	Naive Bayes	RLM	SVM	K-Nearest Neighbors	C4.5	Random Forest
Correctly classified instances	56 / 93.33 %	50 / 83.33 %	59 / 98.33 %	50 / 83.33 %	47 / 78.33 %	56 / 93.33 %
Incorrectly classified instances	4 / 6.66 %	10 / 16.66 %	1 / 1.66 %	10 / 16.66 %	13 / 21.66 %	4 / 6.66 %
Kappa statistic	0.92	0.81	0.98	0.81	0.75	0.92
Mean absolute error	0.01	0.03	0.16	0.08	0.04	0.11
Root mean squared error	0.10	0.17	0.27	0.18	0.19	0.20
Relative absolute error (%)	8.52	19.21	88.98	48.24	25.53	64.10
Root relative squared error (%)	36.18	58.06	90.81	61.27	64.51	67.65
Total number of instances	60	60	60	60	60	60
mean Average Precision (mAP) (%)	94	86.9	**98.5**	90.6	82.4	94

Cuadro 5.1: Resultados para el dataset construido a partir de YouTube-8M.

Como podemos observar en el Cuadro 5.1, los resultados son sumamente buenos, consiguiendo mAP's superiores al 80 % con todos los modelos. En algunos casos, como con la SVM, se consiguen tasas de acierto de casi el 100 %, lo cual verifica la validez de esta metodología para la clasificación de vídeo en datasets de corte parecido al que estamos utilizando. No olvidemos que cada dataset tiene características distintas y, por lo tanto, habrán metodologías de clasificación que den mejores o peores resultados en función de esas características.

5.1.2. Columbia Consumer Video

En este apartado se presentan los resultados obtenidos por los modelos para el dataset CCV. Se han considerado, al igual que en el apartado anterior, los objetos cuyo porcentaje de confianza es mayor al 10 %.

La estructura del dataset es la siguiente: originalmente el dataset consta de 9317 vídeos, distribuidos de manera no uniforme en 20 categorías. Hay que tener en cuenta que, de los 9317 vídeos, se han podido descargar 7578. Los 1739 vídeos restantes no se han podido descargar por razones comentadas en el apartado de **Datasets**. De los 7578 vídeos que se han podido conseguir, al clasificarlos dentro de las 20 categorías obtenemos 8046 vídeos. Esto se debe a que hay algunos vídeos etiquetados con más de una categoría dentro del dataset. A estos 8046 vídeos hay que restar 1537 que no tienen categoría asignada y, por lo tanto, quedarían 6509 vídeos con los que trabajar. Finalmente, dado que hay tres categorías con un número de vídeos bastante superior al resto, se ha limitado el número de vídeos por categoría a 350 para balancear el dataset, por lo que la cantidad final de vídeos disponibles es de 5763.

Las 20 categorías (clases) con las que se trabaja en este dataset son: 'Basketball',

'Baseball', 'Soccer', 'IceSkating', 'Skiing', 'Swimming', 'Biking', 'Cat', 'Dog', 'Bird', 'Graduation', 'Birthday', 'WeddingReception', 'WeddingCeremony', 'WeddingDance', 'MusicPerformance', 'NonMusicPerformance', 'Parade', 'Beach', 'Playground'.

Al igual que en el apartado anterior, se detallan las opciones utilizadas tanto en filtros como en modelos para garantizar que los resultados son reproducibles:

- **String To Word Vector:** Se ha utilizado la misma configuración que en el apartado anterior, codificando con TF-IDF y construyendo el vocabulario utilizando todas las palabras detectadas en el dataset.
- **Naive Bayes:** Se ha especificado el uso de un kernel para estimar la densidad de probabilidad de los atributos numéricos, en vez de una distribución normal.
- **SVM:** El parámetro 'soft-margin *C*' se ha fijado al valor 1 y como kernel se ha utilizado un Normalized Polynomial.
- **K-Nearest Neighbors:** Se ha considerado K = 10.
- **C4.5:** Se han utilizado las opciones por defecto. Estas implican la poda del árbol para evitar el overfitting.
- **Random Forest:** Se han utilizado las opciones por defecto, las cuales no imponen un límite de profundidad a la hora de construir los árboles.
- **MultiClassClassifier:** Este clasificador nos permite tener opciones extra a la hora de ejecutar la SVM. Las opciones de configuración propias de la SVM utilizadas aquí son las mismas que en el caso anterior. Sin embargo, el uso de MultiClassClassifier nos permite añadir a la SVM una componente de corrección aleatoria que mejora levemente los resultados.

El Holdout se ha hecho de la siguiente manera:

- **Conjunto de entrenamiento:** 70 % del tamaño total del dataset.
- **Conjunto de test:** 30 % restante del tamaño total del dataset.

Los resultados obtenidos con los seis modelos de clasificación utilizados se presentan en el Cuadro 5.2.

	Naive Bayes	SVM	K-Nearest Neighbors	C4.5	Random Forest	MultiClassClassifier
Correctly classified instances	1018 / 58.87 %	1107 / 64.02 %	856 / 49.50 %	804 / 46.50 %	995 / 57.54 %	1125 / 65.06 %
Incorrectly classified instances	711 / 41.12 %	622 / 35.97 %	873 / 50.49 %	925 / 53.49 %	734 / 42.45 %	604 / 34.93 %
Kappa statistic	0.56	0.62	0.46	0.43	0.55	0.63
Mean absolute error	0.04	0.09	0.06	0.05	0.07	0.09
Root mean squared error	0.19	0.20	0.18	0.20	0.18	0.21
Relative absolute error (%)	43.34	95.49	66.99	60.16	77.89	97.23
Root relative squared error (%)	88.42	96.30	82.64	95.82	85.93	97.26
Total number of instances	1729	1729	1729	1729	1729	1729
mean Average Precision (mAP) (%)	61.2	64.3	53.1	46.3	58.6	**64.5**

Cuadro 5.2: Resultados para el dataset CCV.

Como vemos en el Cuadro 5.2, los resultados para este dataset no son tan buenos como en el caso anterior, obteniéndose un 65 % de acierto como mejor resultado empleando una SVM. Las razones para estos resultados tan pobres en comparación con los obtenidos con el dataset basado en YouTube-8M son, principalmente, las siguientes:

- **Número de clases:** La complejidad del problema de clasificación aumenta conforme se incrementa el número de clases que el modelo debe distinguir. El dataset CCV tiene 20 clases, 10 más que el basado en YouTube-8M.

- **Complejidad de las clases:** En el dataset basado en YouTube-8M, las clases elegidas han sido distintos deportes, donde, a pesar de que en muchos casos hay bastantes entidades visuales detectadas de forma común en vídeos de distintas clases, siempre hay entidades visuales singulares en los vídeos de cada clase que permiten la diferenciación de estos para su correcta clasificación por parte de los modelos.

  Sin embargo, en el dataset CCV, hay varias clases que, en comparación con otras, no poseen entidades visuales diferenciales. Esto sucede entre las clases: 'wedding reception', 'wedding ceremony' y 'wedding dance'; de la misma forma que también sucede entre 'graduation', 'music performance' y 'non-music performance'. Es muy difícil para los modelos distinguir entre estas clases. De hecho, en algunos casos, es difícil hasta para las personas. Basándonos en [1], la precisión general y el recall de los anotadores de la plataforma Amazon MTurk, los cuales han realizado el trabajo de etiquetar el dataset, es de 77.4 % y 79.9 %, respectivamente. Como vemos, incluso para espectadores humanos, a veces es difícil distinguir la clase correcta.

- **Dataset desbalanceado:** En el dataset CCV, a diferencia del basado en YouTube-8M, las muestras entre clases están desbalanceadas, lo cual afecta a la hora de entrenar los distintos modelos. Por otro lado, dado que la cantidad de vídeos en comparación con

el dataset basado en YouTube-8M es muy superior, también lo es la variabilidad en el tamaño de los vídeos. Esto implica la presencia de vídeos de muy corto tamaño, muestras que apenas aportan información a los modelos, tanto en el entrenamiento como en el test.

No obstante, aunque el mejor resultado obtenido ha sido un 65 % de tasa de acierto, con un mAP del 64.5 %, este es mejor que el obtenido por Jiang et al [1] con la misma base de datos. En la Figura 5.1 vemos como el mejor mAP que alcanzan es de 59.5 %, utilizando todas las características consideradas en el artículo. El modelo que han utilizado es una SVM.

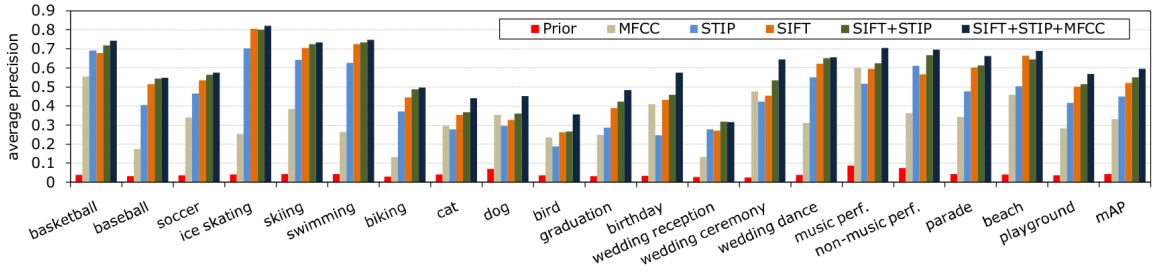

Figura 5.1: Precisiones alcanzadas en [1]. Tenemos las AP's para cada clase y la mAP, en función de las características utilizadas.

En la Figura 5.2, tenemos las AP's para cada clase y la mAP de la SVM que se ha implementado en este TFM. Si comparamos las AP's de nuestro modelo con las del modelo implementado en [1] (Figura 5.1), vemos que, en la mayoría de los casos, se mejora la precisión con respecto a [1]. Esto justifica la mejora del mAP, en casi el 5 %, que conseguimos con nuestro modelo.

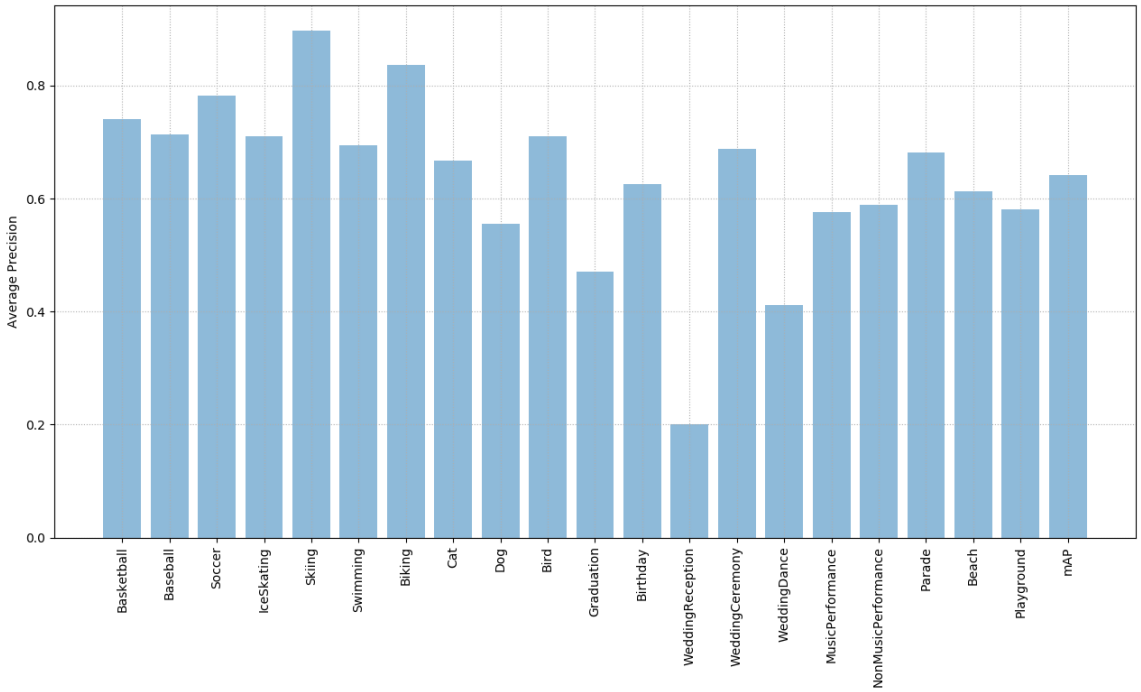

Figura 5.2: Precisiones alcanzadas en nuestro modelo de SVM. Tenemos las AP's para cada clase y la mAP.

Para finalizar, aunque los resultados obtenidos han sido pobres, son aceptables dada la complejidad del dataset empleado y las posibilidades de la metodología utilizada.

5.2 Red LSTM

En este apartado se presenta el resultado obtenido por la red LSTM para el dataset CCV. Para la construcción del modelo se ha utilizado Keras [27], una API de alto nivel para el uso de redes neuronales, escrita en Python y capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollada con el objetivo de permitir la experimentación rápida, pudiendo pasar de una idea a un resultado en poco tiempo mediante la creación de prototipos fácil y rápidamente gracias a su facilidad de uso, modularidad y extensibilidad. Además, funciona a la perfección en CPU y GPU, y soporta tanto redes convolutivas como redes recurrentes, así como combinaciones de ambas.

Antes de continuar, cabe destacar el uso de otras librerías en el desarrollo de esta parte del TFM, como pueden ser: TextBlob [28], NumPy [29] y Pandas [30].

Llegados a este punto, se presenta la arquitectura básica utilizada, la cual consta de tres

tipos de capas distintas y muchos parámetros de configuración ajustables:

1. **Capa de entrada para Word Embeddings [31].** Se ha experimentado con distintas longitudes para los vectores de números reales que la capa genera, los cuales son una representación, en un espacio de alta dimensionalidad, de los números enteros que representan a las palabras.
2. **Capa o capas para la red LSTM.** En este caso, se juega con el número de capas, con el número de módulos por capa y con el dropout.
3. **Capa o capas Fully Connected, la de salida con función de activación SoftMax.** Por último, se juega con el número de capas, con el número de neuronas en cada capa y con el dropout.

Además de los parámetros comentados, tenemos algunos más como la longitud de las secuencias, el número de épocas y el tamaño del batch de entrenamiento.

Los resultados obtenidos con este modelo no han sido satisfactorios para ninguna de las configuraciones de red/combinaciones de parámetros que se han utilizado, lo cual es sorprendente dado que el planteamiento utilizado para resolver el problema de clasificación de vídeo mediante una red LSTM es lógico y correcto, como se ha explicado a lo largo del TFM.

A través de la experimentación con múltiples configuraciones, lo mejor que se ha conseguido son modelos que se ajustan bien a los datos de entrenamiento (90 % de accuracy en el mejor de los casos) pero que obtienen malos resultados a la hora de evaluar con los datos de validación y testeo (52 % de accuracy en el mejor de los casos).

Las causas de estos resultados tan pobres pueden ser muchas y variadas. Algunas de ellas pueden ser las ya citadas anteriormente para el dataset CCV, como puede ser el número de clases dada la complejidad para distinguirlas utilizando únicamente características visuales, o el hecho de trabajar con un dataset desbalanceado. Sin embargo, aunque la idea de utilizar la relación de secuencialidad existente entre los frames de un vídeo para clasificarlo, y el planteamiento para llevar a cabo lo anterior, son correctos; probablemente uno de los problemas es el hecho de que, aunque los conjuntos de palabras que representan a cada frame se presentan ordenados temporalmente, frame tras frame, las palabras que forman esos conjuntos tienen un orden aleatorio entre sí debido a que YOLO9000 las entrega sin orden aparente. Esto hace que la aproximación del problema, que debería ser rigurosamente temporal, se convierta en 'soft-temporal', lo cual no ayuda a la red.

Probablemente, se hubiesen podido conseguir resultados mejores a través de una mayor depuración, tanto del preprocesado realizado a las entidades visuales extraídas de los vídeos, mediante la eliminación de palabras repetidas dentro de cada frame y/o su ordenación alfabética; como de la estructura y configuración de la red. Sin embargo, dado el tiempo limitado del que disponemos para realizar el TFM, ha sido imposible seguir dedicando tiempo a la experimentación.

Capítulo 6

Conclusiones y líneas futuras

El objetivo de este TFM ha sido el desarrollo de una metodología que permita clasificar vídeo basándonos en técnicas propias del ámbito de la clasificación de texto. Para ello, se han extraído entidades visuales de vídeos mediante un detector convolutivo, permitiendo representarlos de forma textual. Esto da pie a manipular dicha información textual, de tal forma que podamos aplicar modelos de clasificación tales como Naive Bayes, SVMs o árboles de decisión; así como aprendizaje profundo mediante una red LSTM.

Los resultados obtenidos por los modelos de clasificación utilizados, en función del dataset, han sido dispares. Por un lado, tenemos los resultados obtenidos para el dataset construido a partir de YouTube-8M, que son sumamente buenos, llegándose a obtener casi el 100 % de tasa de acierto con alguno de los modelos. Por el otro lado, tenemos los resultados obtenidos para el dataset CCV, los cuales son bastante pobres para la mayoría de los modelos.

En base a los resultados comentados, las conclusiones son inmediatas. La metodología para clasificación de vídeo desarrollada en este TFM, aunque lejos de ser la que mejor resultado da con respecto al estado del arte en la materia, es perfectamente válida y funcional, pero con restricciones. Para datasets sencillos, con pocas clases y muy diferenciadas entre ellas, donde el detector convolutivo consiga extraer muchas entidades visuales que sean representativas de cada clase, obtendremos buenos resultados de clasificación. A medida que el dataset aumente su complejidad y dejen de cumplirse las condiciones anteriores, los resultados irán empeorando.

Tras la exposición anterior, podemos concluir el cumplimiento de la hipótesis y objetivos del presente TFM, ya que la metodología para clasificar vídeo implementada es funcional. Sin embargo, dicha metodología es susceptible de ser objeto de mejoras, las cuales pueden enmarcarse dentro de las líneas futuras.

Citemos alguna de estas posibles mejoras:

- Cambiar el detector de entidades visuales utilizado, YOLO9000, por otro más potente, con el fin de extraer de los vídeos mayor cantidad de entidades visuales, y que estas representen cada vídeo de forma más precisa.
- Generar un preprocesado, para las palabras que representan a cada vídeo, con un mayor número de opciones de manipulación de los datos.

Bibliografía

[1] Yu-Gang Jiang, Guangnan Ye, Shih-Fu Chang, Daniel Ellis, and Alexander C. Loui. Consumer video understanding: A benchmark database and an evaluation of human and machine performance. In *Proceedings of ACM International Conference on Multimedia Retrieval (ICMR), oral session*, 2011.

[2] Y. G. Jiang, Z. Wu, J. Wang, X. Xue, and S. F. Chang. Exploiting feature and class relationships in video categorization with regularized deep neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–1, 2017.

[3] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. . Li. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014. Cited By :751.

[4] J. Y. . Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June-2015, pages 4694–4702, 2015. Cited By :164.

[5] D. Brezeale and D. J. Cook. Automatic video classification: A survey of the literature. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 38(3):416–430, 2008. Cited By :155.

[6] S. . Chang, D. Ellis, W. Jiang, K. Lee, A. Yanagawa, A. C. Loui, and J. Luo. Large-scale multimodal semantic concept detection for consumer video. In *Proceedings of the ACM International Multimedia Conference and Exhibition*, pages 255–264, 2007. Cited By :71.

[7] A. Kowdle, K. . Chang, and T. Chen. Video categorization using object of interest detection. In *Proceedings - International Conference on Image Processing, ICIP*, pages 4569–4572, 2010. Cited By :3.

[8] Y. Sun, Z. Wu, X. Wang, H. Arai, T. Kinebuchi, and Y. . Jiang. Exploiting objects with lstms for video categorization. In *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, pages 142–146, 2016. Cited By :1.

[9] Y. Zhou and W. Song. Video classification algorithm based on improved k-means. *Boletin Tecnico/Technical Bulletin*, 55(1):138–144, 2017.

[10] X. Yang, P. Molchanov, and J. Kautz. Multilayer and multimodal fusion of deep neural networks for video classification. In *MM 2016 - Proceedings of the 2016 ACM Multimedia Conference*, pages 978–987, 2016. Cited By :3.

[11] http://www.scientificdatabases.ca/current-projects/english-spanish-text-data-mining/mineria-de-texto-ingles-espanol/mineria-de texto/.

[12] https://machinelearningmastery.com/gentle-introduction-bag-words model/.

[13] http://www.tfidf.com/.

[14] https://ujjwalkarn.me/2016/08/11/intuitive-explanation convnets/.

[15] https://ujjwalkarn.me/2016/08/09/quick-intro-neural networks/.

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2015.

[17] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. 2016.

[18] https://goo.gl/DYzdGG.

[19] José Hernández Orallo, María José Ramírez Quintana, and César Ferri Ramírez. *Introducción a la Minería de Datos*. Pearson Educación, 2004.

[20] http://dataaspirant.com/2017/03/14/multinomial-logistic-regression-model-works-machine learning/.

[21] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. 23, 11 2001.

[22] http://colah.github.io/posts/2015 08-Understanding-LSTMs/.

[23] https://medium.com/mlreview/understanding-lstm-and-its-diagrams 37e2f46f1714.

[24] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. 2016.

[25] https://www.cs.waikato.ac.nz/ml/weka/index.html.

[26] https://es.wikipedia.org/wiki/Weka-(aprendizaje automatico).

[27] https://keras.io/.

[28] https://textblob.readthedocs.io/en/dev/.

[29] http://www.numpy.org/.

[30] https://pypi.python.org/pypi/pandas.

[31] https://www.tensorflow.org/versions/master/tutorials/word2vec.