Identificación de personajes y acciones/emociones mediante inteligencia artificial – Parte A

Titulación: Grado en Ingeniería Informática

Autor: Raúl Mateus Sánchez

Tutorizado por:

Dr. Francisco Alexis Quesada Arencibia

Dr. José Carlos Rodríguez Rodríguez

Julio, 2023

Resumen

Existe un interés creciente de numerosas empresas dedicadas a la producción de cortos animados en 3D por el análisis de vídeos que tienen éxito en redes sociales, atendiendo a criterios como la presencia de ciertos personajes, emociones o acciones. En este trabajo se busca desarrollar una herramienta la cual, a partir de un vídeo, identifique determinados personajes y sus emociones en diferentes escenas, y obtenga las marcas temporales correspondientes. Para ello, se realiza inicialmente una investigación y exploración de diferentes algoritmos y técnicas de Inteligencia Artificial y, mediante el tratamiento de conjuntos de datos específicos, se hace un estudio y desarrollo de las dos fases identificadas para obtener un prototipo de herramienta que persiga la finalidad indicada.

Abstract

There is a growing interest of many companies dedicated to the production of 3D animated short films in the analysis of videos that are successful in social networks, based on criteria such as the presence of certain characters, emotions, or actions. In this work we seek to develop a tool which, from a video, identifies certain characters and their emotions in different scenes, and obtain the corresponding time stamps. For this purpose, an investigation and exploration of different algorithms and Artificial Intelligence techniques is initially performed and, through the treatment of specific data sets, a study and development of the two identified phases is carried out to obtain a prototype tool that pursues the indicated purpose.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

ÍNDICE DE CUADROS

Capítulo 1. Introducción

"El gran motor del cambio es la tecnología". Esta célebre frase de Alvin Toffler resume perfectamente el objetivo de la ingeniería informática, el cual no es otro que impulsar el cambio constante en el mundo y mejorar continuamente todo lo que nos rodea para hacer la vida un poco más fácil. En un mundo en constante evolución, todos los progresos tecnológicos han jugado un papel determinante en la transformación que ha sufrido la sociedad, en como vivimos y nos comunicamos.

En estos tiempos, la moda es hablar de Inteligencia Artificial, pero, ¿qué es la Inteligencia Artificial? Google define la Inteligencia Artificial [1] como un conjunto de tecnologías que permiten que las computadores realicen funciones avanzadas, incluida la capacidad de ver, comprender, analizar datos, etc. Además, la sitúa como la columna vertebral en la computación moderna, aportando valor a las personas y empresas en cualquiera de sus formas.

Con la gran cantidad de herramientas basadas en Inteligencia Artificial que están popularizándose entre la sociedad, queda claro que el futuro pasa por el uso de este tipo de herramientas, no solo desde un punto de vista profesional sino también personal. Dicho esto, una rama que está muy presente en el día a día de las personas es la Visión por Computador, una de las ramas de la Inteligencia Artificial. Esta rama ha permitido nuevos avances en campos como la medicina, robótica o en la creación de vehículos autónomos, pero, ¿qué aporta en nuestra vida diaria? Hoy en día, mediante las cámaras de nuestros teléfonos móviles, se hace uso de técnicas de Inteligencia Artificial para mejorar fotos y vídeos u ofrecer edición inteligente de imágenes sin acudir a programas como Adobe Photoshop, pero esto es solo una mínima parte de lo presente que está en nuestras vidas.

Con esta popularidad en aumento, la comunidad científica y el sector empresarial busca nuevas áreas o ámbitos donde aplicar estas técnicas, y es aquí donde se enmarca este trabajo. En la actualidad existen numerosas empresas que se dedican a la producción de cortos y vídeos publicitarios de diferente índole, con gran interés dentro del sector por el análisis de vídeos que tienen éxito entre su público consumidor atendiendo a criterios como pueden ser el número de visionados y/o el propio *feedback* de los usuarios. El análisis de estos vídeos puede tener una complejidad importante, desde aspectos más sencillos como la identificación de personajes a lo largo de una secuencia hasta la detección de acciones (correr, saltar, bailar,) y/o emociones (reír, llorar, …).

A partir de aquí, se puede relacionar la Inteligencia Artificial con el sector de producción de cortos animados en 3D. En este trabajo se abordará la exploración y validez de diferentes algoritmos y técnicas de Inteligencia Artificial para la identificación de determinados personajes u objetos en diferentes escenas de un vídeo, así como el reconocimiento de acciones y emociones a lo largo de una determinada secuencia, por lo que se requiere una profunda investigación detallada y crítica acerca de la literatura más relevante sobre este campo antes de adentrarse en su implementación. El ámbito de este proyecto es tan grande, que son muchas las vías de estudio y desarrollo posibles para el propósito planteado, por lo que es conveniente dividir esta investigación en dos partes, identificadas como Parte A y Parte B, de forma que se pueda abarcar todo el enfoque global del proyecto.

Para ello, este trabajo se basará en la serie "Pocoyó", conocida por una gran mayoría de personas. El formato de esta serie la hace ideal para el propósito de este trabajo, puesto que sus escenas mayoritariamente son compuestas por personajes animados identificados por un color sobre

un fondo blanco, resaltando al personaje. En la Figura 1-1 se aprecia como todos los personajes tienen una forma distinguida y un color llamativo, que permite que la audiencia reconozca a cada uno de ellos sin esfuerzo.

***Figura 1-1**: Personajes de la serie "Pocoyó". Fuente:* [2]

Por tanto, esta animación en tres dimensiones se considera como la base perfecta de este trabajo ya que, en definitiva, se busca una solución que, tomando como entrada un vídeo, proporcione como salida un registro de marcas temporales en tiempo de vídeo que delimitan dónde aparecen personajes, acciones o emociones de interés.

Además, se busca sentar las bases de la exploración de técnicas basadas en Inteligencia Artificial respecto a la animación 3D, un ámbito donde no existe mucha investigación pública y tiene su relevancia en el día a día de las personas, sobre todo en la audiencia infantil que pasa horas viendo esta clase de dibujos animados, y se puede plantear todo tipo de estudios y desarrollos para aumentar el grado de satisfacción del consumidor y la eficiencia y éxito de los creadores.

En conclusión, se pretende acercar la tecnología más reciente a la industria de la animación digital, mediante Inteligencia Artificial y Visión por Computador, al mismo tiempo que se busca explorar el estado actual de estas ramas y sentar las bases para futuros trabajos orientados en este ámbito.

1.1. Motivaciones

El *boom* que se ha vivido en este último año con la Inteligencia Artificial no ha pasado desapercibido ni por aquellos sectores más alejados de ella. Herramientas como Chat GPT han revolucionado el mundo tal y como lo conocemos, y han cambiado nuestra forma de trabajar y de buscar en Internet. Sin embargo, la Inteligencia Artificial no se reduce solo a Chat GPT. La Visión por Computador es una rama de la IA que está en continuo ascenso y uso, permitiendo nuevas aplicaciones y tecnologías, como la conducción autónoma o semáforos inteligentes.

Por ello, la principal motivación de este trabajo es la investigación y exploración en esta área, de forma que pueda servir como base de futuros proyectos con su objetivo puesto en la animación 3D, o simplemente servir de apoyo para trabajos que empleen la literatura que en este documento se refleja tras un periodo de búsqueda y estudio.

Además, otra de las motivaciones es aplicar la IA, en concreto, la Visión por Computador a la animación 3D. En otras palabras, ayudar a la industria de la creación de animaciones, de forma que puedan obtener *feedback* de sus vídeos para orientar su producción, sus nuevas creaciones o modificar el rumbo de las existentes.

1.2. Objetivos

El desarrollo de este trabajo conlleva ciertos objetivos a cumplir. Debido a la naturaleza investigadora de este, los objetivos irán condicionados a los resultados de la exploración previa.

El objetivo principal que se plantea en este trabajo es la exploración de diferentes algoritmos de Inteligencia Artificial para la identificación de determinados personajes u objetos en las diferentes escenas que se muestren en un vídeo. En otras palabras, se quiere realizar una investigación profunda en la literatura actual acerca de la detección y clasificación mediante técnicas de IA que permitan la identificación de personajes animados sobre un vídeo de entrada. Además, este mismo objetivo se extrapola a la identificación de emociones en determinadas secuencias del vídeo, como el reconocimiento de acciones realizadas por los personajes.

Condicionado por el primer objetivo, se plantea como otro objetivo la búsqueda de patrones estáticos en donde su reconocimiento dependa solo de la dimensión espacial, es decir, lograr detectar y clasificar personajes animados en una escena. Además, se expande al reconocimiento de emociones y acciones, dentro de la búsqueda de patrones dinámicos.

Se considera también como objetivo la creación de una herramienta que permita, tomando como entrada un vídeo, proporcionar como salida un registro de marcas temporales en tiempo de video que delimite cuando aparecen las diferentes entidades, como personajes, emociones y acciones. De nuevo, queda supeditado a los resultados de la investigación previa.

Finalmente, se pretende realizar una comparación dentro de las diferentes aproximaciones empleadas en cada ámbito de interés popular y emplear o proponer un método de medición del rendimiento de cada aproximación.

Capítulo 2. Estado del arte

Dentro del contexto de este trabajo, se debe realizar una revisión acerca del estado actual de la temática a abordar. En su caso, este trabajo se relaciona firmemente con la Inteligencia Artificial, en su rama de Visión por Computador y las técnicas de procesamiento de imágenes y vídeos empleadas en multitud de áreas como la detección y seguimiento de objetos, la identificación de personas o el reconocimiento de patrones, estáticos o dinámicos.

En concreto, el alcance de esta revisión estará definido por las tres subramas definidas en el propósito descrito: Detección de personajes/objetos, reconocimiento de acciones y reconocimiento de emociones. Para ello, se usarán fuentes bibliográficas de artículos de investigación y académicos empleando estos tres conceptos clave.

2.1. Detección y clasificación de personajes animados

Los algoritmos de detección de objetos suelen emplear *Machine Learning* (aprendizaje automático) o *Deep Learning* (aprendizaje profundo) para producir resultados significativos, con el objetivo de que un ordenador pueda localizar y reconocer objetos o personas en un instante, al igual que un humano. Aunque la detección de objetos [3] comienza su historia en la década de 1990, no es hasta 2001 cuando surgen los detectores Viola-Jones, logrando la detección facial en tiempo real por primera vez sin ninguna restricción. Posteriormente, surgieron los detectores HOG (*Histogram of Oriented Gradients)* y DPM (*Deformable Part-Based Model*), englobados en los métodos de detección tradicionales. Después de 2010, la detección de objetos mediante detectores tradicionales se vio condicionada debido a que el rendimiento de las características elaboradas a mano se saturó. Fue entonces cuando, en 2012, el renacimiento de las redes neuronales convolucionales "revolucionó" todo lo conocido, viendo como las redes convolucionales profundas (*deep convolutional networks*) eran capaces de aprender representaciones robustas y de alto nivel de una imagen. En esta etapa de aprendizaje profundo, la detección de objetos se diferenció en dos grupos: detección en una etapa y detección en dos etapas.

Los detectores en una etapa en general son más rápidos y eficientes en la detección de múltiples objetos, ya que se basan en una única red neuronal y para detectar varios objetos, solo necesitan una sola pasada. Entre los detectores de este grupo más utilizados se encuentra:

- **YOLO (*You Only Look Once*)** [4]**:** Este detector fue propuesto por R. Joseph en 2015, siendo el primer detector en una etapa en la era del aprendizaje profundo. Sin duda, su versión YOLOv7 [5], publicada en julio de 2022, ha supuesto una revolución en la detección de objetos debido a su rapidez, potencia y eficiencia, pues supera en velocidad y precisión a todos los detectores conocidos, incluso sus versiones anteriores. YOLO funciona, de manera resumida, dividiendo la imagen de entrada en una cuadrícula de celdas, y realizando predicciones de objetos en cada una de esas celdas en una sola pasada. Sin duda la versión 7 de este detector ha roto cualquier límite y ha demostrado ser capaz de ser aplicado a cualquier caso real. En relación con la temática de este trabajo, si bien no existen artículos relacionados con la detección de personajes animados con la última versión de YOLO, si existen algunos artículos vinculados a la problemática que se aborda. En un estudio reciente, *"Research on Moving Object Detection of Animated Characters"* [6], acerca de la detección de personajes animados

en animaciones 3D, se aplica YOLOv5 con gran precisión y aplicabilidad para personajes animados en movimiento. Estos resultados, aplicando YOLOv7 podrían ser incluso mejores visto los datos aportados por los autores, empleando los hiperparámetros que se aportan en este estudio, lo cual puede ser un punto de partida en este trabajo a la hora de realizar detecciones de personajes animados de forma rápida, eficiente y precisa.

Además, muy recientemente ha surgido la versión 8 de YOLO, YOLOv8 [7], desarrollada por Ultralytics, basada en los últimos avances en *Deep Learning* y Visión por Computador, ofreciendo un rendimiento sin precedentes en términos de velocidad y precisión. Su diseño lo hace apto para diversas aplicaciones y fácilmente adaptable a diferentes plataformas hardware, desde dispositivos periféricos hasta una propuesta de API en la nube. Además, su rendimiento ha resultado ser incluso mejor que todas las versiones anteriores de YOLO, siendo un modelo de última generación [8]

- **RetinaNet** [9]**:** Este detector surge en 2017 tras descubrir ciertos investigadores que la causa principal de que, hasta ese momento, los detectores en una etapa fueran mucho menos precisos que los detectores en dos etapas, respondía a un desequilibrio extremo entre las clases de primer plano y de fondo que se produce durante el entrenamiento de detectores densos. Por ello, se reestructuró la pérdida estándar de entropía cruzada (*cross entropy loss*) para ponderar a la baja la pérdida asignada a los ejemplos bien clasificados. Sus resultados fueron prometedores: con la denominada "Pérdida Focal" (*focal loss*), era capaz de igualar la velocidad de los detectores en una etapa, y superaba la precisión de todos los detectores en dos etapas por aquel entonces. Si bien no existe en la literatura disponible ninguna investigación o publicación relativa sobre la aplicación de este detector en esta temática, existe una publicación muy interesante llamada "*Cartoon Face Recognition: A Benchmark Dataset*" [10], donde además de la creación del mayor *dataset* de personajes de dibujos animados de alta calidad, en la publicación se explica como para reducir la carga de trabajo que supone el etiquetado de imágenes, se elaboró un proceso o algoritmo semiautomático para recopilar y anotar el conjunto de datos. Para ello, se dividió el proceso de filtrado de datos en dos: detección de caras y extracción de características o reconocimiento. Para esta primera etapa, a partir de un conjunto ya etiquetado, se empleó RetinaNet como columna vertebral de la detección de caras de personajes animados, logrando un 89% de mAP (*mean Average Precision)* [11] en 0.5 IoU (*Intersection over Union*) [12] en el conjunto de prueba. Si bien no se ajusta completamente al propósito de este trabajo, sin duda es lo más cercano que existe, y una demostración de que puede ser un punto de partida para la detección de personajes animados.

- **EfficientNet** [13]**:** Se trata de un detector de objetos que emplea una arquitectura de red neuronal eficiente, diseñado para maximizar la precisión y la eficiencia computacional. En otras palabras, se busca lograr una precisión comparable a otros detectores de referencia, pero empleando menos recursos. Esto se logra empleando un método de escalado que escala uniformemente todas las dimensiones de profundidad/anchura/resolución utilizando un coeficiente compuesto, a diferencia de la práctica generalizada que escala arbitrariamente estos factores. Respecto a su vínculo con la temática de este trabajo, se han encontrado algunos artículos interesantes, por ejemplo, la publicación "*A Deep Learning-Based Approach for Inappropriate Content Detection and Classification of YouTube Videos*" [14], donde se busca clasificar en

adecuado o inadecuado ciertos vídeos de dibujos animados mediante su evaluación con EfficientNet, logrando hasta un 95,66% de precisión. Si bien no es exactamente el propósito de este trabajo, es sin duda una demostración de que el empleo de este detector podría ser perfectamente viable para el propósito de este trabajo, incluso reduciendo el costo computacional.

- **SSD (Single Shot MultiBox Detector)** [15]**:** Basado en una única red neuronal, utiliza múltiples cajas (*multibox*) para detectar objetos en diferentes tamaños y escalas. De nuevo, muchos de los trabajos encontrados respecto a esta temática y detector tratan la propiedad intelectual. Destaca la publicación "*Animation Character Detection Algorithm Based on Clustering and Cascaded SSD*" [16], donde se busca el análisis de videos con personajes animados mediante *clustering* y *cascaded-SSD*. Los resultados experimentales demuestran que este método puede detectar eficazmente personajes de animación, y los indicadores de rendimiento son mejores que los de otros algoritmos existentes, por lo que sin duda se coloca como otra técnica o herramienta de Inteligencia Artificial válida para abordar este primer campo.

Por el contrario, los detectores en dos etapas por lo general suelen ser más precisos y adecuados para la detección de objetos pequeños y objetos individuales de alta precisión, ya que en su primera etapa se generan propuestas de objetos, y en su segunda etapa clasifica y ajusta estas propuestas. La historia de estos detectores se remonta hasta Viola-Jones, pero haciendo un ajuste al estado del arte, sin duda se presentan detectores como R-CNN (*Region-based Convolutional Neural Network*), y sus diversas variantes como Fast R-CNN, Faster R-CNN y Mask R-CNN, mejorando significativamente la precisión de la detección de objetos en dos etapas.

Una de las publicaciones más relevantes acerca de esta temática incluye Faster R-CNN como columna vertebral para la detección de personajes de comic. La publicación, *"A Faster R-CNN based Method for Comic Characters Face Detection*" [17], es interesante e ilustrativa a todos los niveles, pues además de rendir mejor que otros métodos, es capaz de funcionar con imágenes de comic con diferentes estilos de dibujo. Los personajes de comic son característicos por su expresión exagerada y en cierta forma, parecidos a los personajes de dibujos animados, por lo que es una aproximación a esta temática relevante y un punto de partida.

Faster R-CNN [18] es probablemente el detector en dos etapas más famoso hoy en día, en el que se emplea una primera etapa donde se utiliza una red convolucional para generar propuestas de regiones candidatas, que se consideran áreas de interés. En la segunda etapa, se utiliza una red convolucional adicional para clasificar y ajustar las regiones propuestas en objetos precisos. Este detector fue clave durante muchos años, entre otros, pues su extensa historia y uso marcan la importancia de este detector y su estado dentro de la literatura.

2.2. Detección y clasificación de emociones en personajes animados

Sin duda este es el campo más complejo de todos los presentados, pues se trata del más subjetivo. Además, surge un debate acerca de que se entiende por reconocimiento de emociones, ya que no solo vienen acompañadas por expresiones o gesticulaciones, sino que también influyen otros factores como el contexto, tono de voz, lenguaje corporal o la propia experiencia personal. Por tanto, muchos de los trabajos orientados en el reconocimiento de emociones están basados en la identificación de expresiones, factor crucial para reconocer la emoción del personaje.

En este campo si existen más avances debido al interés de ciertas empresas. En los últimos años han surgido nuevas técnicas para el reconocimiento de emociones como el uso de redes convolucionales de atención (*Attentional Convolutional Network)* [19]. Si bien los enfoques tradicionales basados en características elaboradas a mano, como SIFT, HOG y LBP, seguidas de un clasificador entrenado en una base de datos de imágenes o vídeos, funcionan razonablemente bien en conjuntos de datos de imágenes capturadas en condiciones controladas, no lo hacen tan bien en conjuntos de datos más difíciles, con más variación de imágenes y rostros parciales. Por ello, se propone un enfoque de aprendizaje profundo basado en una red convolucional atencional que es capaz de centrarse en las partes importantes de la cara y logra una mejora significativa con respecto a los modelos anteriores en múltiples conjuntos de datos. Mediante resultados experimentales, se demuestra que las distintas emociones son sensibles a diferentes partes del rostro, así como un resultado muy aceptable para uno de los *datasets* más desafiantes como es FER-2013, logrando un 70.02% de precisión, solo por detrás de Aff-Wild2. En otros *datasets*, como FERG, logra un 99.3%, siendo el mejor de la comparación realizada, al igual que para CK+.

Por otro lado, surgen nuevas aproximaciones a este campo incluso empleando YOLO [20], [21] o una combinación entre CNN y RNN [22], donde se demuestra una mejora del rendimiento con una RNN más pequeña que el modelo inicial pre-entrenado con un pico de precisión PN (*Positive-Negative*) de 0,76, lo que demuestra que este enfoque es muy capaz de distinguir entre emociones positivas y negativas.

En relación con la temática que aborda este trabajo, existen ya algunos estudios en la línea que se busca. Una aproximación puede ser el estudio "*Deep Learning-Based Classification of the Polar Emotions of "Moe"-Style Cartoon Pictures*" [23], donde se ataca la problemática de los animadores a encontrar contenido relevante durante el proceso de creación de nuevos personajes o material. Por ello, las emociones polares u opuestas (positivas o negativas) en dibujos animados son una referencia importante para los creadores, ya que pueden ayudarles a obtener fácilmente las imágenes que necesitan. Se propone por lo tanto emplear la característica de expresión de los personajes de dibujos animados de este estilo de dibujo, "Moe"[1], para reconocer así las expresiones faciales de los personajes de dibujos animados, extrayendo la escena y las características faciales de las imágenes de dibujos animados. A continuación, se corrige las emociones de las imágenes obtenidas mediante el reconocimiento de expresiones según las características de la escena para obtener las emociones polares de la imagen correspondiente, logrando una precisión experimental del 81,9%.

Otra aproximación más concreta se recoge en el documento "*Facial Expression Recognition of Animated Characters using Deep Learning*" [24]. En esta publicación, se aplican técnicas de visión por computador y *Deep Learning* para detectar personajes en películas o largometrajes para

[1] Estilo de dibujo que enfatiza la ternura, ingenuidad o inocencia, así como la atracción por personajes ficticios, usando en el ámbito del anime o manga [108]

clasificar posteriormente sus emociones en su búsqueda, junto con la empresa *Animated Language Learning*, de ayudar a los niños con trastorno del espectro autista. Una vez preparado el conjunto de datos, se compara dos nuevas CNN (*Convolutional Neural Network*) personalizadas con un análisis de rendimiento frente a las CNN profundas pre-entrenadas más avanzadas. En primer lugar, los modelos se entrenaron y validaron con imágenes de las películas Toy Story 1 a 3, y después se probaron con Toy Story 4 y algunos cortometrajes de Pixar relacionados. Los resultados experimentales muestran que uno de los modelos desarrollados supera al mejor modelo de referencia con una precisión del 77,65%.

2.3. Reconocimiento de acciones

Por último, se abordará la **detección y reconocimiento de acciones.** En primer lugar, se debe definir el reconocimiento de acciones como una tarea de visión por computador que consiste en reconocer acciones humanas en vídeos o imágenes. El objetivo es clasificar y categorizar las acciones que se realizan en el vídeo o la imagen en un conjunto predefinido de clases de acciones [25]. Aún existen cuestiones abiertas, como si el entrenamiento de una red de clasificación de acciones en un conjunto de datos suficientemente grande, dará un aumento similar en el rendimiento cuando se aplique a una tarea temporal diferente o a un conjunto de datos, debido al poco tiempo que se lleva abordando este campo.

Existen ya varias publicaciones haciendo un estudio acerca del estado del arte del reconocimiento de acciones empleando, por ejemplo, técnicas de aprendizaje profundo. En el artículo "*Video-based Human Action Recognition using Deep Learning: A Review*" [26] se identifica el estado del arte de las arquitecturas profundas en el reconocimiento de acciones y luego proporciona las tendencias actuales y los problemas abiertos para futuros trabajos en este campo. En este se concluye que el aprendizaje profundo es actualmente la mejor opción para reconocer y clasificar la acción humana, así como para predecir el comportamiento humano, donde la técnica más precisa hasta el momento era la fusión de redes convolucionales de dos flujos con VGG-16 [27], [28].

Otra publicación más reciente, "*Human Action Recognition: A Taxonomy-Based Survey, Updates, and Opportunities*" [29], se centra en las acciones básicas realizadas por una sola persona, revisando los enfoques de identificación de acciones teniendo en cuenta la extracción de características y el tipo de actividad.

Entre los distintos métodos presentados en la Figura 2-1, sin duda los métodos basados en *Deep Learning* son los más prometedores, incluyendo redes neuronales convolucionales (CNN), recurrentes (RNN) y los modelos híbridos, combinando dos tipos de modelo para aumentar la eficiencia, como CNN y *Transformer* o CNN y LSTM (*Long Short-Term Memory*)

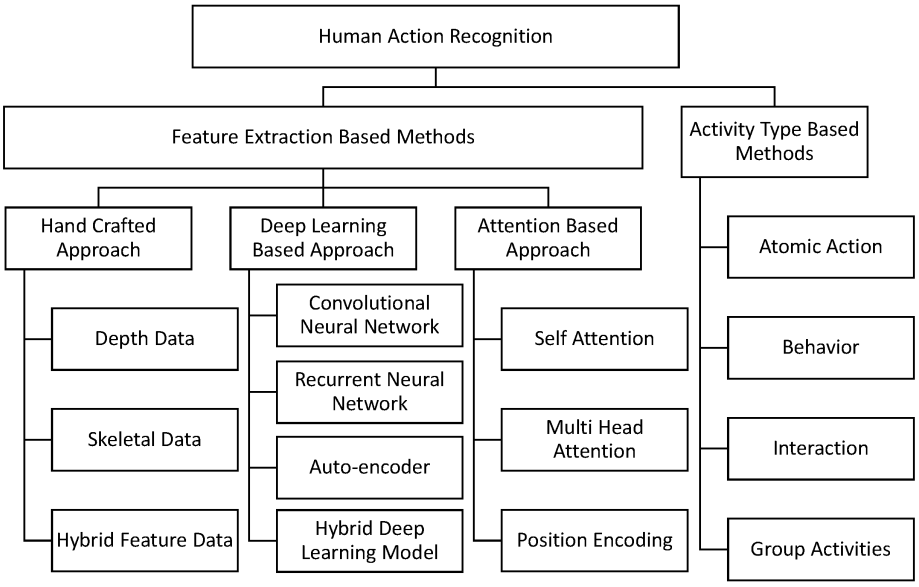

***Figura 2-1**: Taxonomía del reconocimiento de la acción humana. Fuente:* [29]

Por otra parte, se da énfasis a los modelos o enfoques basados en atención, ya que han demostrado resultados prometedores en diversas tareas difíciles de inferencia temporal, como el reconocimiento de subtítulos de vídeo. Una vez realizado un determinado trabajo, sirve para aumentar la interpretabilidad al proporcionar un mapeo diferenciable de todas las ubicaciones de salida a la siguiente entrada.

Transformer es el modelo más reciente que utiliza un mecanismo de atención que atrae a los investigadores en la actualidad. Un *transformer* para el aprendizaje de vídeo puede construirse mediante tres enfoques: autoatención (*Self Attention*), atención multicabezal (*Multi Head Attention*) y codificación de posición (*Position Encoding*). Existen ya varios enfoques que emplean este modelo enfocado al reconocimiento de acciones, recogidos en varias publicaciones, destacando "*Vision Transformers for Action Recognition: A Survey*" [30], donde se hace una revisión acerca del estado del arte respecto a las técnicas desarrolladas, como por ejemplo *Action Transformer* [31].

Sin duda, la carrera para el reconocimiento de acciones está entre las redes neuronales convolucionales y *Vision Transformers*, encontrando varias publicaciones al respecto como *"Convolutional Neural Networks or Vision Transformers: Who Will Win the Race for Action Recognitions in Visual Data?"* [32] .

Respecto a la temática que se aborda existe muy poca investigación, pero si se encuentran modelos que se pueden emplear para abordar la problemática, como ViViT [33], TimeSformer [34], VTN [35], Mformer [36], RViT [37] o MViT [38], además de modelos basados en redes neuronales convolucionales [39] como 3DFCNN [40], o incluso modelos basados en YOLO para el reconocimiento de acciones [41].

Capítulo 3. Metodología del trabajo y planificación del proyecto

3.1. Metodología de trabajo

La metodología de un trabajo puede llegar a resultar un factor clave para su consecución en los plazos fijados y con un estándar de calidad y formalidad. En este caso, este trabajo no se trata de un desarrollo de *software* como tal, sino un trabajo de exploración e investigación, por lo que se ha trabajado de acuerdo a una metodología incremental mediante la sucesión de iteraciones.

Las iteraciones especificadas consisten en las tareas necesarias para completar cada fase representada en la planificación inicial del proyecto, ya que, al tratarse de un trabajo con carácter exploratorio, cada tarea depende de la anterior, por lo que se emplea una continua retroalimentación en este proyecto.

Dentro de las iteraciones, se identifican varias etapas para el desarrollo de este proyecto:

1. Revisión de bibliografía relacionada y especializada en Visión Artificial y *Machine Learning* para la identificación de patrones y para la detección de acciones/emociones.
2. Selección de las distintas tecnologías y herramientas más adecuadas para la gestión y realización del proyecto.
3. Captación de los distintos conjuntos de datos necesarios para la elaboración del proyecto.
4. Desarrollo de software basado en el enfoque de prototipado.
5. Análisis y tratamiento de los distintos datos obtenidos.
6. Elaboración de pruebas de aplicación real del proyecto.
7. Redacción de la memoria del proyecto.

Cabe destacar la primera, cuarta y sexta iteración. La primera consiste en la exploración de la literatura respecto al estado actual de la situación de este trabajo, véase Capítulo 2. Para ello, se realiza una búsqueda de investigaciones, trabajos y todo tipo de publicaciones en páginas de distribución de archivos de acceso público como *Arxiv*, *Google Scholar*, *IEEE Xplore* o *Web of Science*, filtrando por aquellas publicaciones más citadas con la intención de reforzar la autoridad de la fuente de información.

La cuarta se enfoca en la fase de entrenamiento de las redes neuronales pertinentes. Para ello, se debe establecer un entorno de trabajo o emplear aquel que el detector provea, de forma que pueda llevarse a cabo esta etapa.

Por último, la sexta etapa hace referencia a las pruebas realizadas sobre los detectores y la herramienta de generación de marcas temporales, analizando los resultados obtenidos mediante los ensayos realizados tanto durante el entrenamiento como durante la extracción de marcas temporales.

3.2. Planificación del proyecto

En la tabla 3-1 se puede encontrar la planificación inicial del proyecto. Entre las distintas fases representadas, se encuentra en primer lugar una fase de estudio previo, con una estimación de 100 horas. En esta fase se desarrolla la investigación necesaria para este proyecto, recogida en las tareas 1.1, 1.2 y 1.3, la cual se ha ajustado a las 100 horas previstas.

En segundo lugar, se encuentra una fase de desarrollo, en donde se engloba la preparación de los conjuntos de datos necesarios, así como la implementación de los algoritmos seleccionados. La estimación de 120 horas se hace en base a la naturaleza laboriosa de la creación de los conjuntos de datos, ya que se debe recopilar miles de imágenes y deben ser anotadas manualmente para poder proceder con la tarea 2.2, en referencia a la implementación de los algoritmos. Para ello, se lleva a cabo una fase de entrenamiento de las redes neuronales en base a ensayo y error, con el objetivo de maximizar el rendimiento y con el tiempo que conlleva encontrar la configuración óptima.

En tercer lugar, se desarrolla una fase de evaluación de los resultados obtenidos en la fase de desarrollo, así como una prueba general del sistema mediante la extracción de marcas temporales. En base a estas tareas, la estimación es de 40 horas.

Por último, se identifica una etapa de documentación mediante la elaboración de la memoria final y la defensa, estimada en 40 horas.

En general, los plazos establecidos han sido cumplidos en este proyecto, con la excepción de la segunda y tercera fase. Esto se debe a que es un proceso que requiere retroalimentar a su fase anterior, puesto que una evaluación y análisis de resultados conlleva, en muchos casos, el refinamiento de los ajustes de entrenamiento para obtener mejores resultados. Aun así, se compensa con la última fase del proyecto, la cual se puede confirmar que ha conllevado menos de las horas estimadas, llegando a un total final de 300 horas.

Fases	**Duración Estimada (Horas)**	**Tareas (nombre y descripción, obligatorio al menos una por fase)**
Estudio previo / Análisis	100	Tarea 1.1: Búsqueda de información y de artículos académicos sobre el tema y relacionados que permitan la identificación de algoritmos adecuados para los objetivos del proyecto.
		Tarea 1.2: Elección y estudio de las metodologías y técnicas que se emplearán en la fase de implementación.
		Tarea 1.3: Análisis y elección de las herramientas informáticas más acordes para la implementación de la herramienta a desarrollar.
Diseño / Desarrollo / Implementación	120	Tarea 2.1: Preparación de un conjunto de datos para las pruebas del sistema.
		Tarea 2.2: Implementación de los distintos algoritmos seleccionados en la fase 1.2.
Evaluación / Validación / Prueba	40	Tarea 3.1: Evaluación de los algoritmos implementados e introducción de posibles mejoras en los métodos y parámetros empleados.
		Tarea 3.2: Validación final de la herramienta.
Documentación / Presentación	40	Tarea 4.1: Elaboración de la memoria final.
		Tarea 4.2: Preparación de la defensa del TFT.

***Tabla 3-1**: Planificación inicial del proyecto*

Capítulo 4. Competencias específicas

En este capítulo se procede a indicar la justificación de la cobertura de las competencias específicas relacionadas de forma directa con el trabajo desarrollado, reflejadas en el documento TFT01. Las competencias en cuestión son las siguientes:

- **Cl6**: Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos
- **Cl8**: Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados
- **Cl15**: Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica

En el caso de la competencia Cl6, la justificación reside en la propia naturaleza del trabajo, pues se busca realizar una exploración sobre los diferentes algoritmos de Inteligencia Artificial enfocados a nuestro contexto, como es la detección, clasificación y conocimiento. Por tanto, mediante la investigación, se adquiere el conocimiento necesario para aplicar los algoritmos en cuestión en las tareas de interés.

La competencia Cl8 encuentra su justificación en la metodología empleada en el desarrollo. Durante la fase de entrenamiento, los modelos deben ser diseñados y construidos para poder obtener unos resultados que deben ser analizados exhaustivamente debido a los objetivos marcados, como el desarrollo de una herramienta de generación de marcas temporales. Además, todas las modificaciones hechas se han realizado en Python, lenguaje por excelencia de la Inteligencia Artificial.

Por último, la competencia Cl15 queda cubierta, de nuevo, por la propia naturaleza del proyecto. A lo largo de todo el trabajo se indaga en los principios fundamentales de los sistemas empleados, dedicando hasta un marco teórico. Además, su aplicación reside en el entrenamiento de los modelos y su posterior evaluación e inferencia, comprendiendo como se realizan y optimizándolos.

Capítulo 5. Aportaciones

En este capítulo se desarrollan las aportaciones que este trabajo produce sobre nuestro entorno, ya sea técnico, científico o socio-económico.

Este trabajo se centra sobre todo en aportaciones socio-económicas y científicas. Se pretende poder generar unas marcas temporales que describan sucesos en los personajes animados y sus emociones, así como la aparición de estos. Todo ello aportaría, en el marco socio-económico, una gran ayuda a la industria de la creación de animaciones 3D, pues les otorgaría una información muy relevante acerca de sus vídeos, sobre por qué triunfan o por qué fracasan en función del personaje que aparezca, o no, así como sus emociones, después del procesado de los ficheros.

Por otra parte, en el marco científico permite dar a conocer a otros investigadores y desarrolladores la existencia de la animación 3D y el potencial de la Inteligencia Artificial sobre ello. Concretamente, se aporta información relevante acerca de ciertos algoritmos de IA, incluyendo una evaluación de los mismos, lo que permitirá a muchas personas interesadas en este campo conocer el proceso y los resultados de su aplicación e incluso aprender a implementarlos en otros problemas similares.

Capítulo 6. Normativa y legislación

6.1. Licencias software

6.1.1. MIT

La licencia MIT [42] es una licencia de *software* libre permisiva, la cual permite que cualquier persona obtenga gratuitamente una copia del *software* sin restricciones, incluyendo el derecho al uso, copia, modificación, fusión, publicación, distribución, sublicenciar y/o vender copias del software. Además, para que la licencia sea válida, se establece la condición de incluir la pertinente nota de *copyright* y la parte de los derechos en todas las copias o partes sustanciales del *software*.

Bajo esta licencia se encuentra el *pipeline* de entrenamiento de Faster R-CNN [43] y Visual Studio Code.

6.1.2. GNU *Affero General Public License*

La Licencia Pública General de Affero (AGPL) [44] es una licencia libre con *copyleft*, que garantiza la cooperación con la comunidad, específicamente en software de servidores de red. Esta licencia permite el uso comercial, modificación, distribución, patentado y uso privado, pero establece ciertas condiciones, como incluir el aviso de licencia y *copyright* correspondiente o facilitar el código fuente de las versiones modificadas del software cuando se use para prestar servicios en red. Además, entre sus limitaciones destaca la falta de garantía y responsabilidad.

Bajo esta licencia se encuentra el entorno de entrenamiento de YOLOv8 [45].

6.2. Legislación

En base a la normativa española de propiedad intelectual e industrial, regulada en España por el Real Decreto Legislativo 1/1996, de 12 de abril, el contenido fotográfico y los vídeos empleados en este trabajo son propiedad intelectual de la empresa ZINKIA, referido al contenido de la serie Pocoyó, y protegidos por sus condiciones de uso [46].

Capítulo 7. Herramientas y tecnologías empleadas

En la actualidad, existe una gran variedad de tecnologías y herramientas útiles en el ámbito de la Inteligencia Artificial, desde editores de código hasta herramientas de anotación. Por ello, en este capítulo se abordará todas aquellas de relevancia utilizadas en este trabajo.

7.1. Herramientas

Visual Studio Code

Visual Studio Code [47] es un editor de código disponible de forma gratuita, bastante extendido y popular, compatible con prácticamente cualquier lenguaje de programación y los principales sistemas operativos. En este caso, su uso se ha reducido a la edición de ficheros en el lenguaje *Python*.

FFmpeg

FFmpeg [48] es una herramienta multiplataforma que permite grabar, convertir y transmitir audio y vídeo. En el contexto de este trabajo, esta utilidad ha permitido extraer todos los fotogramas (*frames)* de determinados vídeos con la finalidad de crear un conjunto de datos a partir de estas imágenes.

CVAT (*Computer Vision Annotation Tool*)

CVAT [49] es una herramienta interactiva y colaborativa de anotación de imágenes orientada a Visión por Computador. Desarrollada y usada por la empresa Intel, esta herramienta permite, en su modalidad gratuita, realizar el etiquetado de aquellas clases de interés presente en las imágenes de un conjunto de datos para posteriormente obtener el conjunto de datos ya etiquetado.

Roboflow

Roboflow [50] es una plataforma de Visión por Computador que, entre muchas características, destaca por su capacidad de anotación de imágenes de forma individual o colaborativa, así como generar conjuntos de datos con estas anotaciones, permitiendo al usuario redimensionar imágenes o ampliar el conjunto de datos mediante el aumento de datos (*data augmentation*). Debido a las limitaciones de CVAT, esta herramienta se ha vuelto indispensable para crear, mantener, modificar y generar los conjuntos de datos empleados en este trabajo.

GitHub

GitHub [51], [52] es una plataforma de alojamiento de código que permite el control de versiones y la colaboración. Para ello, GitHub se constituye de repositorios en los que los usuarios pueden subir el código pertinente, tanto de forma privada como pública, y gestionarlo de la forma más cómoda. Para este trabajo, los entornos de entrenamiento de las herramientas y modelos que se analizarán se alojan en repositorios de GitHub, lo cual permite, bajo las licencias correspondientes, clonar estos repositorios y usarlos, incluso colaborar en ellos.

Microsoft Word

Word [53] es un software de procesamiento de texto desarrollado por la empresa Microsoft, la cual ha sido empleada para el desarrollo de esta memoria.

7.2. Entornos de desarrollo

Google Colab

Cola*b*, o *Colaboratory* [54], es un servicio alojado en la nube que permite ejecutar cuadernos Jupyter sin configuración previa, permitiendo el acceso a recursos computacionales como GPUs (*Graphics Processing Unit* – Unidad de Procesamiento Gráfico) y TPUs (*Tensor Processing Unit* – Unidad de Procesamiento Tensorial), imprescindibles para el aprendizaje automático, ciencia de datos y proyectos de investigación y educación. En este trabajo, este entorno será muy importante de cara al entrenamiento de los detectores de personajes y emociones de estos, ya que no solo permitirá poder obtener un modelo con el cual se logre uno de los objetivos de este proyecto, sino que el coste computacional de este proyecto no será un factor tan limitante al disponer de la potencia del hardware de Google.

Kaggle

Kaggle [55], [56] es una plataforma en línea orientada a ciencia de datos y aprendizaje automático, cuyas características principales son la colaboración entre usuarios, el almacenamiento de conjuntos de datos y las "competiciones" para resolver retos planteados por la comunidad científica. Además, al igual que Colab, ofrece un servicio de cuadernos con GPUs integradas. De esta forma, esta herramienta permite trabajar de forma paralela a Colab y reducir el tiempo de desarrollo, ya que se puede tener dos modelos a la vez entrenando mientras no se superen las limitaciones de los planes gratuitos.

7.3. Frameworks

PyTorch

PyTorch [57], [58] es un *framework* de código abierto orientado al aprendizaje profundo, popular por su flexibilidad y su facilidad de uso. Este es conocido por su uso a la hora de construir modelos cuyas aplicaciones se engloben en el reconocimiento de imágenes y el procesamiento del lenguaje. Como su propio nombre indica, este está escrito en Python, y tiene un excelente soporte para GPUs, lo cual es otro factor añadido que lo convierte en el marco de trabajo ideal para este tipo de desarrollos.

Capítulo 8. Desarrollo

8.1. Detección y clasificación de personajes animados

En esta primera fase del desarrollo, se tendrá en cuenta el estudio del estado del arte realizado anteriormente para la elección del detector que será entrenado y analizado. Como conclusión de este estudio previo, se presenta en esta fase el detector Faster R-CNN como propuesta.

Uno de los factores de peso para llegar a esta decisión ha sido las investigaciones previas empleando esta red en áreas de gran similitud a la que abarca este trabajo, como se puede ver en [17]. Por otra parte, Faster R-CNN es uno de los detectores en dos etapas más conocido por su precisión e historia previa, por lo que, si bien no se presenta como uno de los modelos más recientes en el campo de la Visión por Computador, se entiende interesante ver como este experimentado detector puede llegar a funcionar a la hora de detectar personajes animados en 3D, aparentemente sencillos, puesto que la serie Pocoyó se caracteriza mayoritariamente por presentar a sus personajes delante de un fondo blanco y bien diferenciados por colores.

8.1.1. Marco teórico: Faster R-CNN

Faster R-CNN se crea como un avance o extensión de Fast R-CNN [59]. Como su propio nombre indica, el principal avance que supone es la reducción del coste computacional que supone este método y mejora en la precisión. Para entender esto de forma más técnica, conviene detallar cómo funciona Faster R-CNN.

Este detector proviene de *R-CNN* [60] (*region-based convolutional neural network),* una red neuronal convolucional basada en regiones cuya principal diferencia o aportación respecto a las otras redes neuronales convolucionales en ese momento resolvía los problemas que suponía que el número de ocurrencias de objetos de interés no fuera constante, ya que esto causaba que se debiese buscar en un gran número de regiones suponiendo una enorme carga computacional.

La solución fue emplear un algoritmo de búsqueda selectiva [61] para extraer solo 2000 regiones de la imagen y alimentar a una red neuronal convolucional que funcione como un extractor de características y alimente a un SVM (*Support Vector Machine)* para clasificar la presencia del objecto dentro de esa región propuesta. Sin embargo, el tiempo de entrenamiento seguía siendo demasiado alto ya que se debe clasificar 2000 regiones propuestas por imagen e impide que se pueda implementar en tiempo real [62].

Para intentar solucionar estos problemas, el mismo autor creó Fast R-CNN. Si bien sigue la misma idea, en vez de alimentar la red neuronal convolucional con regiones propuestas dentro de una imagen, se alimenta con la propia imagen para generar un mapa convolucional de características. Con este mapa se puede predecir la clase de las distintas regiones propuestas mediante varias capas intermedias, por lo que el entrenamiento será mucho más rápido debido a que ya no se debe alimentar a una red neuronal convolucional con 2000 regiones propuestas por cada imagen, sino que la operación de convolución solo se realiza una vez por imagen y genera un mapa de características. Sin embargo, durante el entrenamiento, se producen cuellos de botella debido a las regiones propuestas, afectando al rendimiento del detector [62].

Mientras que estos algoritmos emplean la búsqueda selectiva para encontrar las regiones propuestas, en Faster R-CNN se elimina este algoritmo para permitir a la red que aprenda estas regiones. Por lo tanto, es un detector cuyo funcionamiento es similar a Fast R-CNN*,* donde una imagen se pasa como entrada a una red neuronal convolucional que genera un mapa de características. La diferencia está en que ahora, en vez de usar búsqueda selectiva sobre el mapa para identificar las propuestas de regiones que puede contener objetos en la imagen, se usa una red separada para predecir estas propuestas, conocida como RPN (*región proposal network*). Estas propuestas son redimensionadas usando una capa RoI *pooling* (*Region of Interest Pooling*) para extraer las características de cada región y poder clasificar la imagen dentro de la región propuesta [62].

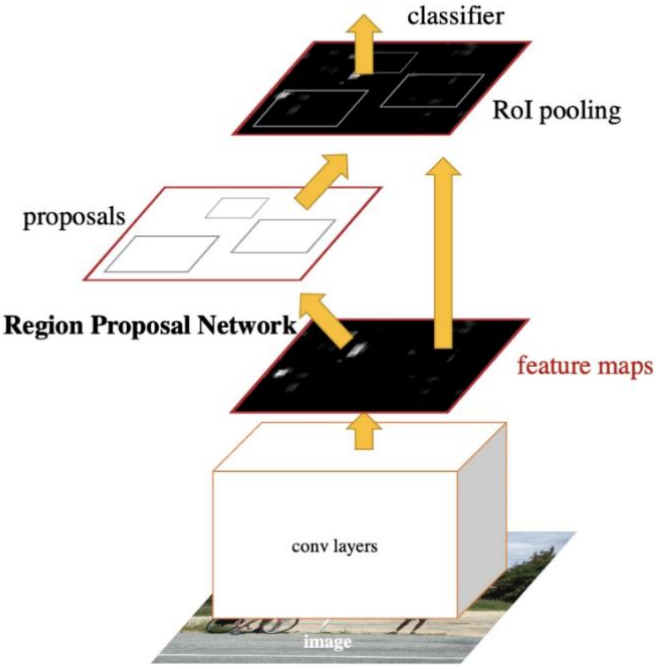

***Figura 7-1**: Funcionamiento de Faster R-CNN. Fuente:* [18]

En resumen, Faster R-CNN como arquitectura de detección usa en primer lugar una red neuronal como columna vertebral (*backbone network*) para extraer algunas características de las imágenes. Una de las más populares es ResNet [63] combinada con FPN (*Feature Pyramide Network)* [64]. Con estas características, la RPN genera regiones propuestas mediante la predicción de objetos en "anclas" (*anchors),* entendidos como cuadros delimitadores. Como la RPN puede llegar a generar muchos cuadros delimitadores para un mismo objeto, se aplica NMS (*Non-Max Suppression)*, de forma que se identifica aquellos cuadros delimitadores con la mayor confianza y se van descartando aquellos que tienen un gran nivel de solapamiento con este. Finalmente, la capa de agrupación de regiones de interés (*RoI pooling layer)* convierte las propuestas generadas de tamaño variable a un tamaño fijo y se procede a su clasificación y refinamiento, prediciendo así la clase y ajustando el cuadro delimitador sobre cada región [65].

8.1.2. Creación y elaboración del conjunto de datos

Para esta fase del desarrollo es necesario contar con un conjunto de datos que recoja los personajes objeto de estudio en esta fase: Pocoyó, Elly, Pato y Nina, véase Figura 7-2. Sin embargo, no existe ningún *dataset* de estas características de dominio público, por lo que este deberá ser creado.

***Figura 7-2**: Personajes de Pocoyó, por orden de aparición: Pocoyó, Nina, Pato, Elly. Fuente: Adaptada de* [66]

El conjunto de datos con el que se realiza el entrenamiento de un detector es, simplemente, imprescindible, además de ser uno de los factores más importantes a la hora de conseguir un detector con gran precisión. Además, no es solo importante la cantidad de imágenes, sino también la calidad de los datos con la que se trabaje [67]. Según el informe del Estado de la Ciencia de Datos 2022 [68], de media, los científicos de datos, investigadores, desarrolladores en Inteligencia Artificial destinan un 37,75% de su tiempo a tener los datos listos antes de poder usarlos para desarrollar modelos. Además, esto se une a que un 12,99% del tiempo se destina a la visualización de datos y un 16,2% a la demostración del valor de esos datos mediante informes y presentaciones, lo que en total supone un 66,94%.

Como primer paso, se debe hacer una búsqueda y captación de imágenes de los personajes de interés. Para ello, la principal fuente de imágenes ha sido *YouTube,* ya que el propósito de este trabajo no es comercial, y sí de exploración e investigación. Esta plataforma alberga gran cantidad de episodios de esta serie, principalmente mediante el canal oficial de Pocoyó en español [69], por lo que una gran cantidad de vídeos han sido descargados para obtener los fotogramas del mismo. Con este objetivo presente, se emplea la herramienta FFmpeg para obtener el número de fotogramas por segundo deseado, permitiendo la opción de comenzar la búsqueda de imágenes desde un momento exacto, o cuanto tiempo de vídeo queremos extraer a imágenes. De esta forma, en total se extraen 1521 imágenes, las cuales deben pasar por el proceso de anotación o etiquetado.

El etiquetado de las imágenes consiste en, una por una, anotar el objeto que se pretende detectar y clasificar mediante un cuadro delimitador presente en la imágenes, si bien esto es variable ya que suelen haber muchas imágenes con más de un objeto de interés, y, por tanto, el proceso de anotación se alarga en el tiempo. Para llevar a cabo este proceso, inicialmente se hace uso de la herramienta *CVAT,* ya que permite el trabajo de etiquetado colaborativo dividiendo el proceso en tareas, donde las personas involucradas pueden acceder a cada una de ellas y comenzar o seguir la tarea. Si bien es una herramienta sencilla y rápida, en su plan gratuito tiene ciertas limitaciones como el número de tareas que se pueden crear, limitando así el propio conjunto de datos, pues si no permite más tareas no se pueden añadir más imágenes. Además, otras limitaciones residen en los formatos de exportación del conjunto de datos ya etiquetado, algo desactualizado. Por todo ello, se realiza una migración a la herramienta Roboflow.

Esta herramienta tiene un comportamiento similar a *CVAT*, pero sus limitaciones no suponen un obstáculo tan grande. De esta forma, se completa en Roboflow el proceso de etiquetado, creando un conjunto de datos lo más equilibrado posible. Posteriormente, esta herramienta permite la generación de versiones del conjunto de datos en donde se puede incluir la técnica de aumento de datos, las cual permite aumentar la diversidad de un conjunto de entrenamiento a través de transformaciones aleatorias, como la conversión de un porcentaje de fotos a escala de grises, o la rotación de imágenes [70].

***Figura 7-3**: Número de instancias de cada personaje en el conjunto de datos*

Para generar una versión en Roboflow, existen 3 pasos fundamentales:

- **División de imágenes**: Selección de porcentaje de imágenes que serán añadidas al conjunto de entrenamiento, validación y test.
- **Preprocesado:** Aplicación de transformaciones a todas las imágenes presentes en el conjunto para decrementar el tiempo de entrenamiento e intentar aumentar el rendimiento.
- **Aumento:** Creación de nuevas imágenes a partir de las ya existentes mediante transformaciones, con la intención de crear modelos de detección más robustos.

Para este conjunto de datos, se ha generado varias versiones a lo largo de la duración de la fase de entrenamiento, incluyendo nuevas imágenes y técnicas de aumento de datos. La versión final contiene aproximadamente 2200 imágenes en el conjunto de entrenamiento y 235 imágenes de validación después de aplicar aumento de datos mediante rotaciones horizontales y verticales [71]. La resolución de estas es de 1920x1080 píxeles, permitiendo así flexibilidad respecto a la resolución luego escogida para realizar el entrenamiento. En otras palabras, si los recursos computacionales lo permiten, es interesante probar diferentes resoluciones y estudiar su impacto, por ello se elige una

resolución alta de imagen para permitir redimensionar a resoluciones más pequeñas con posterioridad.

El conjunto de imágenes de test estará formado por las imágenes de un vídeo totalmente nuevo, sobre el cual se realizará un análisis sobre el funcionamiento del modelo entrenado en imágenes que nunca ha visto, permitiendo así obtener información valiosa sobre el detector entrenado. Este conjunto de test estará formado por 1500 imágenes, correspondiente a un vídeo de 1 minuto y 46 segundos a 25 fotogramas/segundo.

8.1.3. Entorno de entrenamiento

Con el conjunto de datos ya preparado, el siguiente paso es preparar el entorno de entrenamiento. Para ello, después de una pequeña investigación y varios ensayos con algunos entornos (como TensorFlow) los cuales han resultado decepcionantes, se llega a la conclusión de que lo más factible es usar PyTorch como *framework* para el entrenamiento del detector Faster R-CNN.

PyTorch, a través de su librería Torchvision, dispone de varios modelos de este detector tanto pre-entrenado como sin entrenar [72]. Estos modelos se diferencian en su *backbone*, usando ResNet50 FPN, ResNet50 FPN v2, MobileNet v3 Large FPN y MobileNet v3 Large 320 FPN. Si bien se puede disponer de estos modelos, PyTorch también permite crear nuestro *backbone* a partir de los modelos de clasificación de Torchvision, como EfficientNet.

Esto permite cierta flexibilidad a la hora de escoger un modelo a entrenar si el entorno está preparado. Desde Torchvision se ofrece, a través de su repositorio oficial, una serie de *scripts* de referencia para la detección de objetos y, en concreto, para el entreno de las redes que ofrece [73]. Ante la falta de documentación detallada sobre como entrenar este modelo y de un entorno ya preparado e integrado, esta referencia se vuelve válida para nuestro propósito. Tras otra nueva búsqueda, se consigue encontrar un "*pipeline*" o entorno para entrenar modelos de Faster R-CNN en PyTorch, cuyo repositorio se llama fasterrcnn-pytorch-training-pipeline [43], del autor Sovit Ranjan Rath. Este proyecto, de código abierto, toma como referencia los *scripts* mencionados anteriormente y mediante el apoyo de otros desarrolladores, se ha creado un entorno de fácil uso para el entreno del detector en cuestión.

Una de las características principales de este entorno es la cantidad de modelos ya creados, disponibles para ser usados sin configuración previa, visibles en la Figura 7-4. Entre estos, se encuentran los modelos pre-entrenados de PyTorch, y la mayoría de los demás modelos son creados por la comunidad a partir de la flexibilidad que permite en ese sentido Torchvision.

```
'fasterrcnn_convnext_small',
'fasterrcnn_convnext_tiny',
'fasterrcnn_custom_resnet',
'fasterrcnn_darknet',
'fasterrcnn_efficientnet_b0',
'fasterrcnn_efficientnet_b4',
'fasterrcnn_mbv3_small_nano_head',
'fasterrcnn_mbv3_large',
'fasterrcnn_mini_darknet_nano_head',
'fasterrcnn_mini_darknet',
'fasterrcnn_mini_squeezenet1_1_small_head',
'fasterrcnn_mini_squeezenet1_1_tiny_head',
'fasterrcnn_mobilenetv3_large_320_fpn', # Torchvision COCO pretrained
'fasterrcnn_mobilenetv3_large_fpn', # Torchvision COCO pretrained
'fasterrcnn_nano',
'fasterrcnn_resnet18',
'fasterrcnn_resnet50_fpn_v2', # Torchvision COCO pretrained
'fasterrcnn_resnet50_fpn',  # Torchvision COCO pretrained
'fasterrcnn_resnet101',
'fasterrcnn_resnet152',
'fasterrcnn_squeezenet1_0',
'fasterrcnn_squeezenet1_1_small_head',
'fasterrcnn_squeezenet1_1',
'fasterrcnn_vitdet',
'fasterrcnn_vitdet_tiny',
'fasterrcnn_mobilevit_xxs',
'fasterrcnn_regnet_y_400mf'
```

***Figura 7-4**: Lista de modelos de Faster R-CNN. Fuente:* [43]

Para su uso, se emplea la plataforma Kaggle y, en concreto, su servicio de cuadernos o *notebooks*. De esta forma, se realizará el entrenamiento sobre el hardware ofrecido por Kaggle, orientado a este tipo de usos. Con este objetivo, se crea un nuevo cuaderno en donde se clona este repositorio y se instalan los requisitos.

De esta forma, resta descargar el *dataset* en el entorno generado por el cuaderno mediante el código que genera la herramienta Roboflow, en la Figura 7-5, al exportar la versión correspondiente.

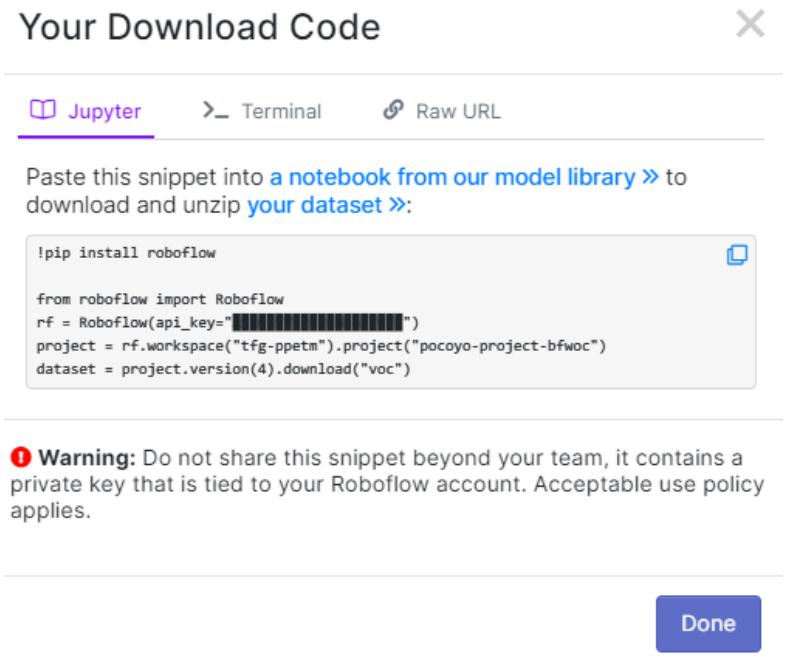

***Figura 7-5**: Código generado por Roboflow para descargar el dataset*

Finalmente, tras varias configuraciones y adaptaciones de archivos, el entorno queda listo para el entrenamiento, mediante el siguiente comando base:

```
python train.py --data {data}.yaml --epochs {NUM_EPOCHS} --model {MODEL_NAME
        } --name {NAME} --batch {BATCH_SIZE} --lr {LEARNING_RATE}
```

Lo esencial para realizar el entreno es especificar el archivo de configuración global del conjunto de datos que se va a usar, establecer el número de épocas, el tamaño del *batch*, la tasa de aprendizaje, el nombre del modelo a usar y el nombre del "proyecto" o carpeta en la que se guardarán los resultados.

El número de épocas se entiende como un hiperparámetro que define el número de veces que el algoritmo de aprendizaje o detector trabajará por todo el conjunto de entrenamiento, mientras que el tamaño de *batch* o tamaño de lote es otro hiperparámetro que define el número de muestras, en este caso imágenes, por el cual se trabajará antes de actualizar los parámetros internos del modelo y el gradiente [74].

Por otra parte, otro de los hiperparámetros más importantes es la tasa de aprendizaje (*learning rate*). Este parámetro, cuyo valor común suele oscilar entre 0.01 y 0.001, se usa para controlar el ritmo con el que un algoritmo actualiza o aprende los valores de la estimación de un parámetro. Esto se debe a que, si bien existen los hiperparámetros, también existen los denominados "Parámetros

susceptibles de ser aprendidos de forma automática", o *Machine learnable parameters,* los cuales son parámetros que el algoritmo aprende o estima por su cuenta [75]. Si la tasa de aprendizaje es muy grande, es posible llegar al mínimo por el descenso del gradiente, pero pasarlo de largo una y otra vez, ya que se dan pasos muy grandes. Por el contrario, si se dan pasos muy pequeños con una tasa bastante pequeña, probablemente encontremos el mínimo buscado, pero se tarde mucho en llegar a él [76]. Por ello, optimizar este valor es fundamental.

Además de las otras funciones que permite este entorno, como aumento de datos o el uso de "mosaicos" para agrupar varias imágenes en una sola, una de las elecciones más importantes de cara al entrenamiento es el optimizador.

```
# Define the optimizer.
# optimizer = torch.optim.SGD(params, lr=args['lr'], momentum=0.9, nesterov=Tr
ue)
# optimizer = torch. optim.AdamW(params, lr=args['lr'], weight_decay=0.0005)
optimizer = torch.optim.Adam(params, lr=args['lr'])
```

El optimizador es quien se encarga de relacionar la función de pérdida con los parámetros del modelo, de forma que, dependiendo de los resultados de la función de pérdida implementada, se actualizarán los parámetros como respuesta, buscando así la mayor precisión posible mediante la actualización de pesos. En otras palabras, la función de pérdida indica al optimizador si se está moviendo en la dirección correcta [77].

En este entorno, si bien se puede usar cualquier optimizador implementado en PyTorch, se realizarán las pruebas con los optimizadores SGD, Adam y AdamW. El descenso del gradiente estocástico (SGD), es probablemente uno de los optimizadores más populares y comunes en el ámbito de los algoritmos de aprendizaje automático y redes neuronales. El objetivo de todo optimizador de cara a obtener un modelo robusto es, mediante el descenso por el gradiente, encontrar el mínimo punto donde el gradiente sea casi 0. En este caso, SGD introduce el término "estocástico" en referencia a aleatorio, puesto que el propio descenso por el gradiente puede ser muy caro computacionalmente hablando en cada iteración, y este algoritmo selecciona una muestra del conjunto de datos para cada iteración, en vez de usar todas las muestras, reduciendo así el tiempo y el coste de cómputo [78].

Otro de los optimizadores más populares es Adam. Este es otra variante del descenso por el gradiente estocástico clásico, donde, en vez de mantener una tasa de aprendizaje fija durante todo el entrenamiento, se mantiene una tasa de aprendizaje para cada peso de la red, y se adapta por separado a medida que el entrenamiento avanza. Otra de sus características es que combina las ventajas de los optimizadores AdaGrad y RMSProp para conseguir una mayor efectividad, convirtiéndolo así en un optimizador relativamente fácil de configurar y con las ventajas de otras variantes incluidas [79].

Por último, se presenta AdamW [80]. Este surge a partir de Adam, debido a un error matemático a la hora de implementar el decaimiento de los pesos (*weight decay*) [81]. Este nuevo optimizador solventa este error desacoplando el decaimiento de los pesos de la actualización del gradiente, mejorando el rendimiento de Adam a la hora de generalizar el modelo.

Como se ha comentado, Faster R-CNN se compone realmente de tres redes neuronales, enfocadas a la extracción de características, las propuestas de regiones y la detección de objetos. La red de propuestas de regiones (RPN) emplea la función de pérdida de entropía cruzada (*cross-*

entropy) para la clasificación de las regiones y una expresión llamada *Smooth L1 Loss* para la regresión de los cuadros delimitadores, es decir, para ajustar el centro y el tamaño de estos cuadros que contendrán las regiones de interés [82].

La función de **pérdida de entropía cruzada** se emplea a la hora de ajustar los pesos del modelo durante el entrenamiento con el objetivo de minimizar la pérdida.

La predicción de un objeto en la imagen supone una probabilidad o confianza asociada que indica cuánto de seguro está el modelo de que el objeto detectado pertenece a la clase predicha. Esta probabilidad es comparada con la salida de la clase real deseada (0-1) y una puntuación o pérdida es calculada, que penalizará la probabilidad en base de si está muy lejos del valor esperado. En otras palabras, si el modelo predice que el objeto pertenece a la clase real con una confianza alta, la diferencia será cercana a 0 y producirá una puntuación pequeña. En cambio, si el modelo produce una predicción de clase errónea o la confianza es muy baja, aumenta la diferencia hasta acercarse a 1, por lo que la puntuación será grande y repercutirá en una penalización mayor. [83].

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

***Figura 7-6**: Función de pérdida de entropía cruzada. Fuente:* [83]

La red de detección de objetos sigue esta misma dinámica, empleando la función de pérdida de entropía cruzada para la clasificación, y *Smooth L1 Loss* para las cajas delimitadores, solo que en este caso se considerarán más clases y no solo el fondo y el primer plano [82].

Entendido el funcionamiento del detector, de las funciones de pérdidas usadas y seleccionado los hiperparámetros, el modelo y el optimizador, puede comenzar el entrenamiento.

8.1.4. Entrenamiento de la red neuronal

Para el entreno de una red neuronal no existe ningún método o forma que permita conocer de forma exacta los hiperparámetros óptimos para cierto modelo, ni tampoco el mejor optimizador que se ajuste a nuestro problema. De hecho, este es el gran condicionante a la hora de entrenar redes neuronales, y es que cada problema es totalmente distinto a cualquier otro, por lo que, si bien la experiencia en otros proyectos puede servir, no se pueden trasladar las mismas configuraciones a este trabajo.

Por tanto, el método para encontrar una configuración óptima es mediante ensayo y error. Probar diferentes combinaciones de hiperparámetros se vuelve fundamental para poder, mediante los resultados de estas pruebas, concluir en un modelo optimizado para el problema que resuelve, sin olvidar que existen más factores de peso como el propio conjunto de datos empleado o las limitaciones de *hardware*.

Se recuerda que el entorno de entrenamiento dispone de un total de 27 modelos para entrenar, evaluar o realizar el proceso de inferencia. Por tanto, la estrategia que se ha seguido es, para una parte de los modelos, realizar las pruebas necesarias hasta llegar a una configuración de parámetros que maximice el rendimiento del detector. Solo se probará una parte de estos modelos debido, principalmente, a las limitaciones de *hardware* y al tiempo que tardan en entrenar. La mayoría de ellos superan las capacidades físicas de entornos como *Colab* y *Kaggle*, en concreto, supera la memoria de las GPUs. Además, otro factor añadido es el tamaño del lote, pues al trabajar con estos modelos, el tamaño del lote suele ser pequeño para evitar errores de agotamiento de memoria.

Por ello, los modelos a entrenar son los siguientes:

Modelo	Parámetros	Resolución de imagen
Faster R-CNN + EfficientNet B0	84.1M	640
Faster R-CNN + EfficientNet B4	137.6M	640
Faster R-CNN + MobileNet v3 Large FPN	19.4M	640
Faster R-CNN + ResNet 50 FPN v2	43.2M	640
Faster R-CNN + ResNet 101	184.2M	640
Faster R-CNN + SqueezeNet 1.1	29.9M	640
Faster R-CNN + ViT Tiny	22.8M	640
Faster R-CNN + RegNetY_400mf	28.8M	640
Faster R-CNN + DarkNet	84.4M	640

***Tabla 7-1**: Modelos entrenados en primera ronda - Faster R-CNN*

Estos modelos funcionan de forma que, el *backbone* proporciona una red para la extracción de características de las imágenes de entrada. En algunos modelos, se proporciona otra red, llamada *Feature Pyramid Network* (FPN), encargada de generar múltiples capas del mapa de características con información de mayor calidad para la detección de objetos [84]. A continuación, es donde entra la red de propuestas de regiones (RPN) y, por último, una red basada en regiones para clasificar (R-CNN).

En un primer momento, para realizar las pruebas de entrenamiento sobre estos modelos se usó una versión anterior del conjunto de datos donde se aplicaron técnicas de aumento de datos como la

rotación horizontal y vertical de imágenes, así como la transformación a escala de grises del 25% de las imágenes, pero los resultados globales se pueden catalogar como decepcionantes. Es por ello por lo que se realizó un paso atrás, hasta la elaboración del conjunto de datos, y se mejoró progresivamente este hasta llegar a la versión final, cuyas características se especificaron anteriormente. Los modelos presentados en la Tabla 7-1, generalmente suelen consumir muchos recursos computacionales y tardan bastante tiempo en entrenar, desde 1 hora en los más ligeros como Faster R-CNN + MobileNet v3 Large FPN hasta más de 8 horas en modelos más pesados como Faster R-CNN + EfficientNet B4. Por ello, se toma la decisión de tomar los 4 mejores modelos para esta segunda ronda de entrenamiento, quedando recogidos en la Tabla 7-2.

Modelo	Parámetros	Resolución de imagen
Faster R-CNN + EfficientNet B4	137.6M	640
Faster R-CNN + MobileNet v3 Large FPN	19.4M	640
Faster R-CNN + ResNet 50 FPN v2	43.2M	640
Faster R-CNN + RegNetY_400mf	28.8M	640

***Tabla 7-2**: Modelos entrenados en segunda ronda - Faster R-CNN*

Por otra parte, la resolución de imágenes usada se ha establecido en 640x640 píxeles. Esta decisión se ha tomado en base al coste computacional que supone entrenar estos modelos con imágenes de mayor resolución, puesto que el tiempo de entrenamiento aumenta exponencialmente.

A continuación, se procede a realizar una visualización de los resultados obtenidos, los hiperparámetros empleados para ello, y una evaluación general.

8.1.5. Evaluación y análisis del rendimiento de los modelos

En primer lugar, antes de proceder con la visualización de resultados, se realizará una explicación sobre los hiperparámetros usados y las métricas de evaluación que se generan. En los entrenamientos realizados, principalmente se ha ido modificando la tasa de aprendizaje o *learning rate*, el tamaño del lote o *batch size,* el número de épocas y el optimizador. Además, en los casos que se presentarán a continuación, todos han sido realizados con la opción de no crear mosaicos, es decir, no crear un *collage* de imágenes como forma de aumento de datos, y se ha empleado la opción *Cosine Annealing*. Esto es un tipo de programación de la tasa de aprendizaje para que comience con un valor de tasa de aprendizaje muy alto, y este se reduce con gran rapidez hasta un valor mínimo antes de volver a incrementar, permitiendo así una exploración más amplia del espacio de búsqueda. Este incremento "rápido" se entiende como "reinicio en caliente", simulando el reinicio del proceso de aprendizaje mientras se reutilizan los buenos pesos como punto de partida del reinicio, en vez de utilizar un nuevo conjunto de pequeños números aleatorios como punto inicial [85], [86].

Además, conviene especificar que los optimizadores suelen estar configurados por una serie de parámetros los cuales no se han modificado, respetando los valores por defecto de PyTorch, siendo los siguientes:

- **Adam**: Beta1=0.9, Beta2=0.999, Epsilon=10^{-8}, Weight Decay=0
- **AdamW**: Beta1=0.9, Beta2=0.999, Epsilon=10^{-8}, Weight Decay=10^{-2}
- **SGD**: Momentum=0, Weight Decay=0

Para cada modelo se han realizado distintas pruebas con distintas configuraciones para intentar obtener el mejor rendimiento posible en la detección de los personajes. En la tabla 7-3 se puede observar cómo, para cada modelo, se especifica el conjunto de parámetros que mejor resultado ha dado.

Modelo	Optimizador	*Learning rate*	Épocas	*Batch size*
Faster R-CNN + EfficientNet B4	AdamW	0.001	30	4
Faster R-CNN + MobileNet v3 Large FPN	AdamW	0.0001	50	16
Faster R-CNN + ResNet 50 FPN v2	AdamW	0.001	50	4
Faster R-CNN + RegNetY_400mf	SGD	0.01	50	16

***Tabla 7-3**: Configuración de hiperparámetros para modelos Faster R-CNN*

Como resultado del entrenamiento, se obtiene cierta información de gran relevancia sobre el rendimiento del modelo. Esta información realmente son métricas de evaluación de la detección, que permiten conocer aspectos como la precisión. En el caso concreto de esta implementación de Faster

R-CNN, se implementa en la fase de validación después de cada época las métricas de evaluación usadas por COCO.

COCO [87] es un conjunto de datos a gran escala, principalmente de detección y segmentación popular por su uso generalizado a la hora de entrenar redes neuronales puesto que permite evaluar nuevos modelos en este *dataset* y así analizar y mostrar los avances respecto a otros modelos. Para ello, ofrece ciertas métricas estándar para poder realizar comparaciones y evaluaciones entre distintos modelos y algoritmos, como AP (*Average Precision*) y AR (*Average Recall*).

En las métricas de COCO, el promedio de la precisión (AP) se realiza entre todas las categorías o clases, por lo que se asume que no existe ninguna distinción entre AP y mAP (*mean Average Precision)*. Esto es importante ya que, a partir de ahora, la métrica de evaluación principal de los modelos será el mAP.

Esta métrica es la métrica de referencia utilizada actualmente por los investigadores en el campo de la Visión por Computador para evaluar la robustez de los modelos en su tarea de detección, por lo que su uso queda justificado [11]. El mAP se sustenta en cuatro pilares, siendo estos la matriz de confusión, Intersección sobre Unión (IoU), precisión y *recall.*

Una matriz de confusión permite conocer los aciertos y fallos del modelo a la hora de detectar y clasificar, pero para el mAP, es relevante conocer que esta matriz se basa en cuatro atributos:

- Positivos verdaderos
- Negativos verdaderos
- Falsos positivos
- Falsos negativos

Por otra parte, la Intersección sobre Unión (IoU) indica el grado de solapamiento del cuadro delimitador predicho con el cuadro delimitador verdadero. Como es obvio, un mayor IoU indicará que las coordenadas de la caja delimitadora son muy similares a la caja delimitadora real.

Otro de los pilares, y una métrica muy importante, es la precisión. Esta mide cómo de bien se puede encontrar los positivos verdaderos, entendiéndose como las predicciones correctas respecto a todas las predicciones positivas, respecto al total de predicciones positivas realizadas. Para ello, se guía de la siguiente fórmula, siendo TP una abreviatura de *True positive* (verdadero positivo) y FP una abreviatura de *False positive* (falso positivo):

$$Precisión = \frac{TP}{TP + FP}$$

Ecuación 7-1*: Fórmula para el cálculo de la precisión*

Finalmente, queda explicar el *recall*. Esta métrica es parecida a la precisión, pero en este caso se mide cómo de bien se puede encontrar positivos verdaderos respecto a todas las predicciones. En otras palabras, mide la capacidad del modelo para encontrar todos los casos positivos.

$$Recall = \frac{TP}{TP + FN}$$

Ecuación 7-2*: Fórmula para el cálculo del recall*

Sustentado en estas métricas, el mAP se calcula encontrando la precisión media para cada clase, y luego se promedia sobre un número de clases. De forma más detallada, ya que el valor de la precisión depende del umbral IoU establecido, para cada clase se calcula la precisión para diferentes umbrales IoU, y luego se toma la media de esos valores para cada clase. Finalmente, el mAP del modelo se calcula haciendo la media de todos los valores mAP por clase.

COCO toma hasta 10 umbrales, desde 0.5 hasta 0.95 en pasos de 0.05 para finalmente determinar el AP, o lo que en este caso es lo mismo, el mAP o mAP@0.5:0.95. En la Ecuación 7-1 y 7-2 se muestra la fórmula de cálculo del mAP para una clase y varias clases.

$$\boldsymbol{mAP} = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

Ecuación 7-3: *Fórmula del mAP. Fuente:* [11]

$$\boldsymbol{mAP} = \frac{1}{N}\sum_{k=1}^{k=n} AP_k$$
$$AP = the\ AP\ of\ class\ K$$
$$n = the\ number\ of\ classes$$

Ecuación 7-4*: Fórmula del mAP multi-clase. Fuente:* [11]

Durante el entrenamiento, se realiza en cada época una actualización de pesos a partir de la pérdida calculada, y se produce una etapa de validación en donde se obtiene, entre otras cosas, el mAP actual sobre el conjunto de validación.

En base al mAP y al *Average Recall*, se observa un modelo que destaca por encima de los otros. Este es Faster R-CNN junto con EfficientNet B4 como *backbone*, obteniendo casi un 83% mAP, y un *recall* promedio del 79,54%, como se puede ver en la Tabla 7-4. Que el rendimiento de este modelo sea mejor comparado con los otros no resulta ninguna sorpresa, ya que es de largo uno de los modelos con más parámetros y su propia naturaleza, popularidad y arquitectura permite estos resultados. EfficientNet, en su versión B4, supera en exactitud a otras redes como ResNet 50, ResNet 152 o Inception-ResNet-v2 bajo el conjunto de datos ImageNet [88], superando en eficiencia a prácticamente todos los modelos con un tamaño de modelo inferior [89].

A continuación, encontramos el modelo que emplea MobileNet v3 [90] como *backbone*, con un 81,89% mAP. Este resultado es, cuanto menos, sorprendente, pues es una red neuronal convolucional diseñada para procesadores de teléfonos móviles. De hecho, sobre el conjunto de datos ImageNet consigue, en la métrica *Top-1 Accuracy,* un 75.2%, mientras que EfficientNet-B4 consigue un 82.6% [91], lo que pone en gran valor los resultados conseguidos por este modelo. Además, el *recall* promedio es el más alto, un 79.54%, lo que se entiende como el porcentaje o proporción de los casos positivos correctamente detectados.

El siguiente modelo en cuanto al mAP logrado es Faster R-CNN + RegNetY_400mf. Este *backbone*, en cuanto a número de parámetros, es muy similar a MobileNet v3, y, de hecho, tiene un porcentaje del 74.1% en el *Top-1 Accuracy* sobre ImageNet, por lo que se entiende como razonable que ambos modelos estén muy cerca en cuanto a rendimiento, y se valora su rendimiento en este entrenamiento cercano a EfficientNet-B4.

El *recall* promedio de estos dos modelos se establece en un 77.47% y 75.29% respectivamente, lo cual es algo peor que EfficientNet-B4, pero no supone una diferencia tan significativa como para catalogar el rendimiento de estos modelos muy lejano a la referencia de EfficientNet.

Por último, se encuentra Faster R-CNN + ResNet 50 FPN v2, un modelo que viene ya implementado en PyTorch y cuya versión base, ResNet-50, obtiene un 79.04% en el *Top-1 Accuracy* sobre ImageNet. En cambio, estos resultados no se trasladan a nuestro caso, pues obtiene un 79.93% mAP, menor que los otros modelos. En cuanto al *recall* promedio, si surge una diferencia notable respecto a los otros modelos, obteniendo un porcentaje del 70,1%, lo cual indica que este modelo no es tan bueno identificando los ejemplos positivos como los otros.

Modelo	**mAP@0.5**	**mAP@0.5:0.95**	***Average Recall***
Faster R-CNN + EfficientNet B4	**0.978**	**0.8295**	**0.795**
Faster R-CNN + MobileNet v3 Large FPN	0.9638	0.8189	0.744
Faster R-CNN + ResNet 50 FPN v2	0.97	0.7993	0.701
Faster R-CNN + RegNetY_400mf	0.967	0.8032	0.752

***Tabla 7-4**: Average Precision y Average Recall en entrenamiento - Faster R-CNN*

Estos resultados, en general, se consideran positivos respecto al objetivo que se busca, pero no terminan de ser convincentes en primera instancia. Esto es debido a que la mayor parte del tiempo los personajes, distinguidos cada uno por un color (verde, rosa, amarillo y azul), se sitúan delante de un fondo blanco, por lo que su reconocimiento para humanos es muy sencillo. Por ello, la expectativa inicial respecto a Faster R-CNN era mayor a los resultados obtenidos.

Si bien la precisión se establece en torno a un 80% respecto al conjunto de validación durante el entrenamiento, normalmente este porcentaje baja cuando se realiza una evaluación sobre el conjunto de test. Este conjunto de test está compuesto por 1500 imágenes, véase apartado 7.1.1, todas correspondientes a un vídeo de prueba sobre el que posteriormente se realiza la inferencia, ya que el objetivo principal de esta fase es poder extraer, sobre vídeos de éxito en redes, ciertas marcas temporales respecto a la aparición de los personajes. Por ello, el conjunto de fotogramas que conforman este vídeo, totalmente distinto a las imágenes de entrenamiento y validación, permite crear un conjunto de test robusto para determinar el rendimiento de los modelos.

Los resultados de la evaluación sobre el conjunto de test arrojan varias conclusiones a la vista. La primera, y más evidente, es que los modelos no rinden tan bien en otros conjuntos de datos, puesto que la diferencia de precisión más baja es la de Faster R-CNN + EfficientNet B4, un 12.72%. Por otra parte, el *recall* promedio se mantiene muy parecido tanto en la evaluación en el conjunto de

test como en el entrenamiento, por lo que se puede deducir que si bien el modelo no varía en base a cómo de bien identifica o detecta los casos positivos, sufre más a la hora de clasificar en umbrales IoU mayores, ya que el mAP@0.5 no varía tanto, como se aprecia en la Tabla 7-5.

Modelo	mAP@0.5	mAP@0.5:0.95	*Average Recall*
Faster R-CNN + EfficientNet B4	**0.9143**	**0.7023**	**0.746**
Faster R-CNN + MobileNet v3 Large FPN	0.8590	0.6455	0.685
Faster R-CNN + ResNet 50 FPN v2	0.9169	0.6122	0.672
Faster R-CNN + RegNetY_400mf	0.888	0.6331	0.679

Tabla 7-5: *Average Precision y Average Recall en conjunto de test - Faster R-CNN*

Como ya se explicó, estos resultados han sido los mejores obtenidos, por lo que cabe un estudio sobre los mismos. Esta diferencia de precisión puede sugerir *overfitting* o sobreajuste, lo cual se corresponde al fenómeno que ocurre cuando un modelo se ajusta a sus datos de entrenamiento. Es decir, se aprende "de memoria" las imágenes que se le pasan durante el entrenamiento, ocasionando que el modelo obtenga peores resultados con datos totalmente nuevos [92].

En una primera ronda de entrenamiento, los resultados habían sido decepcionantes. Ampliando esta sentencia, se daba el caso de que, si bien los resultados del entrenamiento no superaban ninguna expectativa, los resultados de la evaluación sobre el conjunto de test eran aún peores, con diferencias en la precisión mucho más grandes. Por ello, se modificó el conjunto de datos ampliándolo y usando técnicas de aumento de datos que ha permitido mejorar los resultados y reducir este tipo de sucesos.

Además, es necesario incidir en la diferencia entre el conjunto de validación y de test. Mientras el conjunto de validación se compone de entrenamiento de 235 imágenes con un buen número de instancias por cada personaje, el conjunto de test está formado por 1500 imágenes. Esta diferencia se vuelve relevante a la hora de evaluar el modelo, ya que un mayor número de imágenes puede perjudicar los resultados debido al cálculo de la precisión media.

Para demostrarlo, aleatoriamente se reduce el conjunto de test hasta tener el mismo número de imágenes que el conjunto de validación, y se repite la evaluación. Sin duda, en la Tabla 7-6 se aprecia claramente como los resultados respecto a los representados en la Tabla 7-5 han mejorado considerablemente. La diferencia más baja ahora entre precisión en entrenamiento y en test pasa de 12.72% a un 6,1%, prácticamente la mitad. Por ello, queda clara la importancia del conjunto de test, su número de imágenes y la información que genera.

Modelo	mAP@0.5	mAP@0.5:0.95	*Average Recall*
Faster R-CNN + EfficientNet B4	**0.9878**	**0.7685**	**0.797**
Faster R-CNN + MobileNet v3 Large FPN	0.8762	0.6778	0.687
Faster R-CNN + ResNet 50 FPN v2	0.9323	0.6835	0.675

Faster R-CNN + RegNetY_400mf	0.9037	0.692	0.707

***Tabla 7-6**: Average Precision y Average Recall en conjunto de test reducido - Faster R-CNN*

En este contexto, el mejor modelo sobre el conjunto tiene una precisión de un 76.85%, lo cual se considera aceptable respecto al entrenamiento, pero se queda algo corto con respecto a lo esperado, ya que no se considera que el conjunto de datos sea tan complejo. Los otros modelos se quedan aproximadamente un 10% por debajo en cuanto a precisión incluso siendo entrenados por más épocas, lo que transmite la robustez de EfficientNet-B4 como extractor de características para Faster R-CNN.

A continuación, conviene un análisis de la precisión por clases. Esto nos aportará información valiosa sobre cómo se desempeñan los modelos en cada clase de forma individual, para comprobar si existen clases que sean más difíciles de detectar e identificar y tomar las medidas necesarias.

A simple vista en la Tabla 7-7, se observa que la detección y clasificación de Pato y Elly es la más compleja, con una precisión media entre todos los modelos de 63.5% y 60.88% respectivamente. Este hecho coincide con que ambos son los personajes menos "humanos", ya que representan un elefante y un pato, como su propio nombre indica. Por ello, a pesar de contar con colores llamativos y distinguibles, así como una forma única dentro de la serie, es probable que los modelos sufran más a la hora de reconocerlos.

En cuanto a los modelos, de nuevo EfficientNet-B4 destaca frente al resto, siendo el más estable de los 4 y destaca especialmente en la precisión para detectar a Elly, siendo esta una clase más compleja para los otros modelos.

Relación Precisión Media por clase (AP) - Modelo				
Modelo	**Pocoyó**	**Pato**	**Elly**	**Nina**
Faster R-CNN + EfficientNet B4	0.724	0.698	0.758	0.851
Faster R-CNN + MobileNet v3 Large FPN	0.690	0.589	0.505	0.843
Faster R-CNN + ResNet 50 FPN v2	0.726	0.537	0.637	0.634
Faster R-CNN + RegNetY_400mf	0.696	0.716	0.535	0.740

***Tabla 7-7**: Relación AP/clase – Modelo – Faster R-CNN*

Centrados en Faster R-CNN + EfficientNet B4, además de conocer la precisión media por cada clase, es interesante analizar la matriz de confusión, en este caso sobre el conjunto de test.

En la Figura 7-7, se muestra la matriz de confusión generada por la evaluación del modelo Faster R-CNN + EfficientNet B4 sobre el conjunto de test de 235 imágenes. En primer lugar, se encuentra

la clase "*Background*". La fila "*Background*" indica el número de objetos detectados en alguna clase cuando realmente no pertenecen a ninguna, es decir, representa casos de falsos positivos (FP). Por otra parte, la columna "*Background*" representa aquellos objetos que no han sido detectados y que se han considerado como un objeto del fondo. Por tanto, representa casos de falsos negativos (FN).

Para la clase Pocoyó, se extrae de la matriz de confusión que el modelo realiza un buen trabajo, clasificando hasta 199 instancias del objeto correctamente. Por otra parte, 29 instancias se han confundido con Elly, 3 con Pato, y 1 como *background*, es decir, no se ha detectado. La confusión con Elly no deja de ser extraña, ya que ambos personajes tienen muy poco parecido, por lo que sin duda es un caso que se debe estudiar para mejorarlo. Además, hasta 52 objetos de fondo han sido detectados como Pocoyó, afectando a la precisión del modelo en esta clase.

Para Pato, la confusión ha sido mínima. Este es un personaje bastante característico, de un color llamativo y una forma totalmente distinta a cualquier otro personaje, por lo que se entiende complicado que exista confusión. Aun así, 46 objetos del fondo se han detectado como Pato, perjudicando de nuevo la precisión de esta clase.

Con Elly la cosa es aún mejor, solo confundiendo una vez el personaje con Pato, pero de nuevo, 34 objetos del fondo han sido detectados como Elly, además de la mencionada confusión de Pocoyó con este personaje, reduciendo en general la precisión y el rendimiento del modelo.

Finalmente, Nina es detectada al 100% como su clase, a pesar de tener cierto parecido facial con Pocoyó, pero de nuevo se detectan hasta 48 objetos del fondo como Nina.

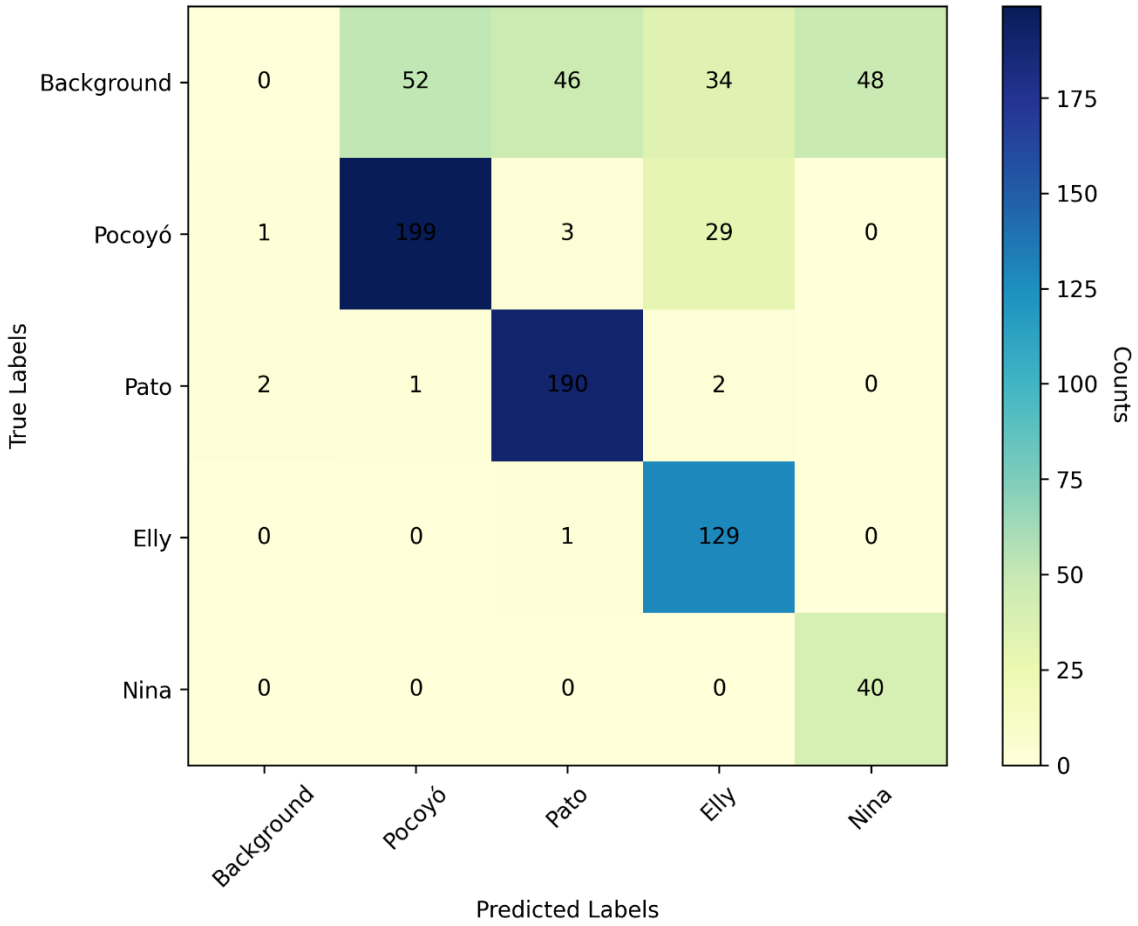

***Figura 7-7**: Matriz de confusión sobre conjunto de test, Faster R-CNN + EfficientNet-B4*

Después de analizar la matriz de confusión, la conclusión es clara: existe un problema de detección de objetos del fondo como clases del modelo. Si bien la confusión entre clases es mínima, este suceso penaliza la precisión de la red a pesar de optimizar los hiperparámetros empleados para su entreno.

En las figuras 7-8 y 7-9 se observa un claro ejemplo de esta problemática. El objeto situado a la izquierda de las imágenes, formado por una especie de mesa de noche con una calabaza típica de *Halloween* es detectada como un objeto de interés, pero en un caso se detecta como Pocoyó con una confianza del 71%, y en otro caso como Nina al 64% de confianza. Realmente es complicado entender esta situación, pues en este objeto predomina el color naranja de la calabaza, que podría ser confundida con la nariz de Pato, pero en su lugar se confunde con Pocoyó, un personaje de mucho mayor tamaño sin ningún tipo de color naranja o blanco en él, o con Nina, cuyo color predominante es el verde y su cuerpo es más voluminoso en pantalla que este objeto.

***Figura 7-8**: Detección de objeto de fondo como Pocoyó*

***Figura 7-9**: Detección de objeto de fondo como Nina*

Otro caso de la problemática en cuestión se da en la Figura 7-10. Esta representación es algo más compleja, ya que se detecta a Elly en el fondo en cuyo cuadro delimitador se encuentra tanto Pocoyó como Elly. En este caso, es difícil saber si se está detectando dos veces el mismo objeto, es decir, dos veces a Elly, si se confunde a Pocoyó con Elly como se apreció en la matriz de confusión o simplemente el color rosa ha permitido hacer creer al modelo que en ese lugar se encuentran dos instancias de Elly cuando solo hay una. Pero si hay algo claro, es que la cuestión de la detección de objetos del fondo como clases de interés no es un caso aislado.

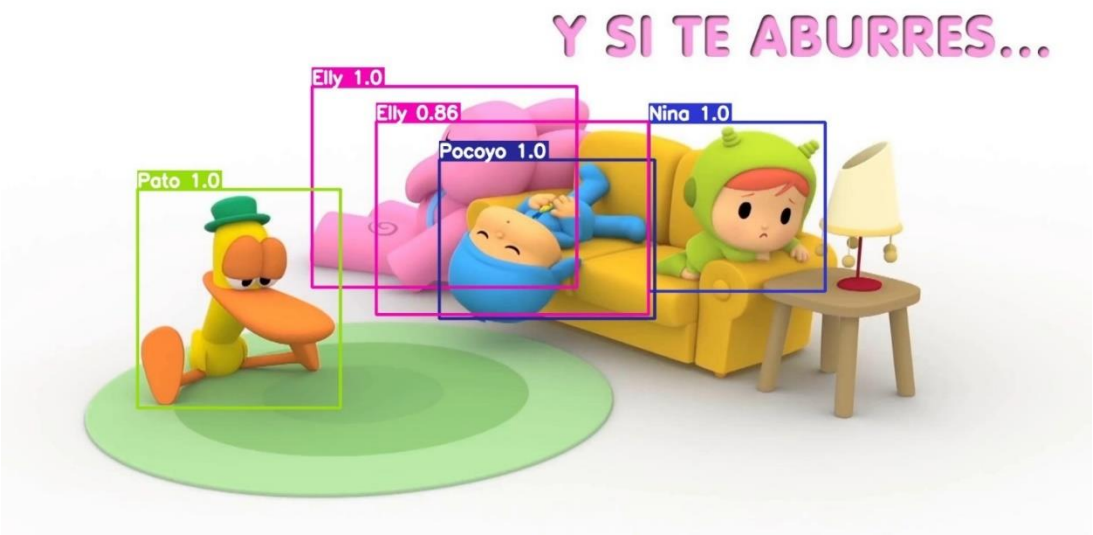

Figura 7-10: *Detección de objeto de fondo como Elly*

A pesar de ello, la detección del modelo es aceptable. En la Figura 7-11 se aprecia un caso de dificultad alta, ya que Pocoyó está prácticamente de espaldas y solo se distingue el cuerpo, mientras que Pato está disfrazado. A pesar de ello, el modelo reconoce a ambos y ajusta el cuadro delimitador casi a la perfección, asignando a ambas predicciones un 100% de confianza.

Figura 7-11*: Detección correcta de Pocoyó y Pato*

Sin duda, este modelo presentado, el cual ha dado los mejores resultados de entre todas las pruebas con otros varios modelos, presenta un comportamiento a la altura de las circunstancias, aunque con ciertos problemas de falsos positivos que lastran a toda la red y la exploración de Faster R-CNN como detector de personajes animados en 3D.

8.1.6. Herramienta de generación de marcas temporales

Otro de los objetivos de este trabajo es, mediante la detección de personajes animados, generar un fichero que contenga las marcas temporales que representen la aparición o desaparición de personajes. Mediante este archivo y su posterior procesado, los creadores de animaciones 3D podrían extraer información relevante sobre los vídeos de éxito.

Para llevar a cabo esta finalidad, es necesario realizar el proceso de inferencia mediante un modelo ya entrenado a un vídeo de entrada. Durante el proceso de inferencia, el modelo descompone el vídeo en fotogramas y realiza una predicción sobre cada uno de los fotogramas, generando los mismos fotogramas con cuadros delimitadores y uniéndolos para generar el vídeo que contenga las predicciones realizadas.

Por ello, para poder extraer a un archivo externo el instante de tiempo en el que sucede la aparición o desaparición de un personaje, es necesario conocer las predicciones del modelo para extraer las clases predichas y poder procesarlo. Para obtener el instante de tiempo al que se corresponde el fotograma actual se puede hacer uso de la librería OpenCV [93], especializada en Visión por Computador. Esta implementación de Faster R-CNN hace uso de esta librería para abrir el vídeo y extraer los fotogramas, por lo que se puede conocer varias propiedades del vídeo gracias a las herramientas que ofrece OpenCV, como la posición actual del vídeo en milisegundos gracias al *flag* CAP_PROP_POS_MSEC [94], por lo que es posible la extracción en milisegundos del momento actual del vídeo que representa el fotograma, y obtener la marca temporal en formato mm:ss.SSS, donde la 'm' representa los minutos, la 's' minúscula los segundos y la 'S' mayúscula, los milisegundos en tres dígitos.

```
current_position_ms = ms_to_time_format(int(cap.get(cv2.CAP_PROP_POS_MSEC)))
```

A continuación, se debe llevar un registro de la presencia de los personajes para poder identificar si el personaje aparece o desaparece mediante la información que el modelo devuelve a la hora de realizar la predicción, y se debe crear el fichero correspondiente donde se volcará las marcas temporales.

```
temp_marks_file = os.path.join(OUT_DIR, 'temporal_marks.txt')
ch_state = {'Pocoyo': False, 'Pato': False, 'Elly': False, 'Nina': False}
```

Si el modelo devuelve predicciones de clases, se debe establecer cierta lógica de forma que, recorriendo las claves del diccionario creado que representa si un personaje está presente en la escena o no, se compara si estas clases se encuentran en la lista generada por el modelo que contiene las clases predichas. En esta lógica se establecen dos casos principales, siendo estos que un nuevo personaje haya sido detectado y no se encontraba en la escena (es decir, en el fotograma anterior), y que el personaje ya estuviese en la escena y en este fotograma ya no aparece, por lo que se entiende que el personaje ha desaparecido de la escena.

Finalmente, se construye el mensaje final el cual será escrito en el fichero que se genera, llamado "*temporal_marks.txt*".

```
base_log = f"{current_position_ms} --> "
```

```
aux_base_log = base_log

for key in ch_state.keys():
    if key in pred_classes:
         if ch_state[key] is False:
            base_log += f"{key} aparece,  "
            ch_state[key] = True

         else:
             if ch_state[key] is True:
                base_log += f"{key} desaparece,  "
                ch_state[key] = False

if base_log != aux_base_log:
   line = base_log.rstrip(" ,")
   f.write(f"{line}\n")
```

Una pequeña prueba se realiza sobre el mismo vídeo del que se obtuvo el conjunto de test de 1500 imágenes, cuya precisión se puede apreciar en la Tabla 7-5 presentada anteriormente. Este vídeo se ha escogido por su diversidad de personajes y escenas, llegando a durar un total de 1 minuto y 46 segundos sobre el que se realiza el proceso de inferencia y se obtiene un fichero de marcas temporales como el que se puede ver en la Figura 7-12, y el vídeo donde se aprecia las predicciones realizadas mediante cuadros delimitadores.

```
00:39.520 --> Pocoyo aparece,  Pato aparece,  Nina aparece
00:39.880 --> Nina desaparece
00:42.360 --> Pocoyo desaparece
00:43.280 --> Pocoyo aparece
00:44.040 --> Pocoyo desaparece
00:44.320 --> Pocoyo aparece
00:45.600 --> Nina aparece
00:45.640 --> Nina desaparece
```

***Figura 7-12**: Ejemplo de marcas temporales en Faster R-CNN*

En fichero cumple el objetivo planteado en esta fase del desarrollo, y puede ser de utilidad para que empresas y profesionales del sector de animación 3D puedan obtener información relevante sobre las razones del éxito de sus vídeos, como qué personajes son los que prefiere la audiencia o con cuales se sienten más intrigados al verlos, al igual que cuánto tiempo en escena pasan, todo ello relevante como *feedback* para encontrar la senda del éxito a medio y largo plazo. De esta forma, se demuestra que sobre el proceso de inferencia de cualquier detector se puede crear una pequeña herramienta que permita extraer información de este estilo, siempre supeditada a la precisión del detector en cuestión.

Por último, se puede unir el proceso de inferencia y evaluación en un solo *script*, el cual permita como entrada argumentos comunes a ambos procesos, así como argumentos particulares para cada uno de ellos, y realizar ambos procesos de una sola vez, lo cual permite no solo generar la inferencia y el fichero de marcas temporales, sino realizar una evaluación sobre el conjunto de fotogramas que conforman el vídeo. Para ello, este conjunto deberá ser previamente etiquetado para poder evaluar el rendimiento del sistema en este vídeo y poder analizar la salida de una manera mucho más estándar y común a otros sistemas, como se realizó en la Tabla 7-5.

Mediante el tratamiento de los argumentos de entrada del *script*, se procede a ejecutar dos subprocesos, llamando a los respectivos *scripts* para simplificar el proceso.

```
def main(inference_args, eval_args):

    # Run the inference script
    print("Running inference\n")
    inference_args = [str(arg) for arg in inference_args]
    print(inference_args)
    subprocess.call(['python', 'inference_video_mod.py'] + inference_args)

    # Run the evaluation script
    print("\nRunning evaluation of video\n")
    eval_args = [str(arg) for arg in eval_args]
    print(eval_args)
    subprocess.call(['python', 'eval.py'] + eval_args)
```

En conclusión, esta herramienta permite, mediante un procedimiento estándar, procesar el vídeo de entrada para generar el fichero de marcas temporales y poder analizar la salida, incluyendo la evaluación pertinente si se tiene el conjunto de imágenes etiquetado para, mediante el mAP, estudiar el rendimiento/calidad de esta aproximación.

8.1.7. Conclusiones

En esta fase del desarrollo, orientada a la detección y clasificación de personajes animados, se ha hecho un estudio previo sobre el origen y funcionamiento del detector empleado, Faster R-CNN. Además, se ha creado un gran conjunto de datos tanto para el entrenamiento como para otras pruebas, como la extracción de marcas temporales sobre la aparición y desaparición de personajes de la escena y su evaluación para medir el rendimiento del sistema. Para ello, se ha llevado a cabo una fase extensa de entrenamiento hasta lograr los mejores resultados posibles, véase Tabla 7-4, y el posterior análisis de los mismos para evaluar el rendimiento del sistema.

La primera conclusión de esta fase se centra en el conjunto de datos y su importancia de cara a proyectos de este tipo. En una fase inicial, el conjunto de datos preliminar era uno de los principales factores por el cual ninguno de los modelos era capaz de superar el 50% de precisión media, un resultado totalmente inaceptable. Una vez este fue refinado, el mejor modelo fue capaz de llegar a un 82.95% de precisión media en el entrenamiento, y un 76.85% sobre el conjunto de test, especificado en las Tablas 7-4 y 7-6.

Esta mejora en el rendimiento ilustra la relevancia de usar un buen conjunto de datos a la hora de entrenar modelos de detección de objetos. Crear un conjunto de datos amplio y variado no es tarea fácil, no por su complejidad, sino por el tiempo que conlleva seleccionar las imágenes, anotar cada una de ellas de forma manual y estudiar su impacto sobre el entrenamiento para realizar ajustes como incluir técnicas de aumento de datos, por lo que es una parte del desarrollo muy importante, laboriosa, y la cual puede, en cierta manera, determinar el éxito del mismo.

Por otro lado, si bien las expectativas en cuanto a los resultados obtenidos eran altas, la realidad ha sido otra distinta, obteniendo como mejor resultado un 82.95% en entrenamiento. Entre los motivos de este resultado, cabe resaltar también la limitación computacional, ya que la mayoría de modelos proporcionados para entrenar requieren de imágenes de muy baja resolución y un tamaño de lote ínfimo, aunque en muchos de estos casos ni siquiera esto es suficiente para poder realizar el entrenamiento, por lo que se reduce el conjunto de modelos para probar y, por tanto, no es posible realizar un estudio completo sobre este detector.

Además, los modelos entrenados han requerido de mucho tiempo para completar el entrenamiento, hasta ocho horas, suponiendo una gran complicación en el desarrollo de esta fase, ya que los entornos usados como Kaggle o Colab tienen limitaciones semanales y diarias en cuanto a horas de uso de las tarjetas gráficas, además de las propias limitaciones de las tarjetas gráficas disponibles.

Además, la implementación de Faster R-CNN en PyTorch, al igual que otras muchas implementaciones, tienen una desventaja común, y es que no recupera el valor de pérdida durante la validación o la exactitud (*accuracy*) en ninguna fase del entrenamiento debido a como está construido, ya que a la hora de realizar el entrenamiento, se pasa las imágenes al modelo para realizar su predicción pero solo devuelve las pérdidas calculadas, mientras que en la fase de validación, entendida como evaluación, cuando se pasa las imágenes al modelo, devuelve las predicciones y no las pérdidas. Esto crea una situación compleja para evaluar si el modelo sufre de sobreajuste (*overfitting*) o subajuste (*underfitting*), limitando su estudio mediante las gráficas de pérdida durante el entrenamiento o mediante la comparación del mAP entre el entrenamiento y la evaluación sobre el conjunto de test.

Entrando de lleno es los resultados, en general se califican como aceptables. La expectativa inicial, sugerida por la alta precisión en objetos individuales de los detectores en dos etapas como Faster R-CNN, se estimaba en torno a un 90%. A esta estimación se suma la complejidad del problema, puesto que, como se ha comentado durante este trabajo, la serie Pocoyó es característica por destacar a cada uno de sus personajes, especialmente los cuatro principales que aquí se procesan, con un color llamativo y diferencial sobre un fondo blanco. Además, estos 4 tienen formas muy distintas (excepto Nina y Pocoyó), y se esperaba un mejor rendimiento de este detector incluso con *backbones* ligeros, como RegNet o MobileNet. En el análisis de los resultados, véase apartado 7.1.4, se estudia cómo, si bien el detector no tiende a confundir personajes, si tiende a detectar objetos de fondo como personajes, en casos donde no se puede aplicar la lógica para llegar a entenderlo.

Esta situación, unido a lo desarrollado en estas conclusiones, refleja que una de las principales acciones que se debería realizar para mejorar los resultados sería refinar y ampliar el conjunto de datos, con mucho más tiempo y recursos. Ya se demostró que, refinándolo una vez, los resultados mejoraron en más de un 20%, por lo que se considera la opción más razonable y segura, a pesar de haber etiquetado más de 3000 imágenes entre el conjunto de entrenamiento, validación y los fotogramas del vídeo usado para la evaluación de la herramienta.

También se concluye que se debería volver al punto inicial de análisis de los resultados del estudio del estado del arte para valorar otros detectores o redes prometedoras para llevar a cabo una tarea en donde no existe tanto interés por parte de los investigadores como es la detección e identificación en animación 3D más allá de detectar su uso comercial fraudulento.

Por último, aunque un 82.95% de precisión no cumpla las expectativas iniciales, si deja un claro camino a seguir para proyectos de este tipo, demostrando que, mediante una fase de estudio previo, de preparación de datos y de entrenamiento, es posible generar todo un *pipeline* que permita, a partir de vídeos animados, extraer mucha información relevante para los investigadores y la industria.

8.2. Detección y clasificación de emociones en personajes animados

Sin duda, una característica de los personajes animados es que suelen mostrar emociones de forma más exagerada y visible que los humanos, enfocados en el público al que se quiere atraer. Incluso existen películas, como *Inside Out*, que son creadas con uno de sus propósitos orientados a enseñar las emociones a la audiencia infantil y como tratarlas, llevando a estudios como el que propone la autora Leticia Porto [95] titulado "Estudio de las emociones en los personajes animados de Inside Out".

Para los creadores de animaciones, no es sorprendente que el éxito de sus vídeos vaya ligado a las emociones que se captan durante el relato de sus historias, puesto que el ser humano por naturaleza conecta por emociones, y aunque estas pueden ser transmitidas por palabras, poder recibirlas también de forma visual incrementa el nivel de excitación de la audiencia.

Por todo ello, este trabajo se enfoca en cómo se puede extraer información sobre las emociones presentes en un vídeo, mediante una exploración y estudio previo de la literatura y seleccionando un detector para llevar a cabo esta fase. Una vez realizado el análisis del estado del arte, véase apartado 2.2, se concluye que la opción más interesante para desarrollar esta fase es el empleo de YOLOv8 [45]. Esta es la última versión disponible de YOLO, creado por *Ultralytics*, basada en las versiones anteriores e introduciendo nuevas mejoras para aumentar el rendimiento y la eficiencia. Además, soporta tareas como la detección, segmentación, estimación de pose, seguimiento y clasificación. Respecto a versiones anteriores como YOLOv5, YOLOv6 y YOLO7, esta nueva versión, con menor número de parámetros es capaz de superarlos a todos en cuanto a mAP sobre el conjunto de datos COCO [87].

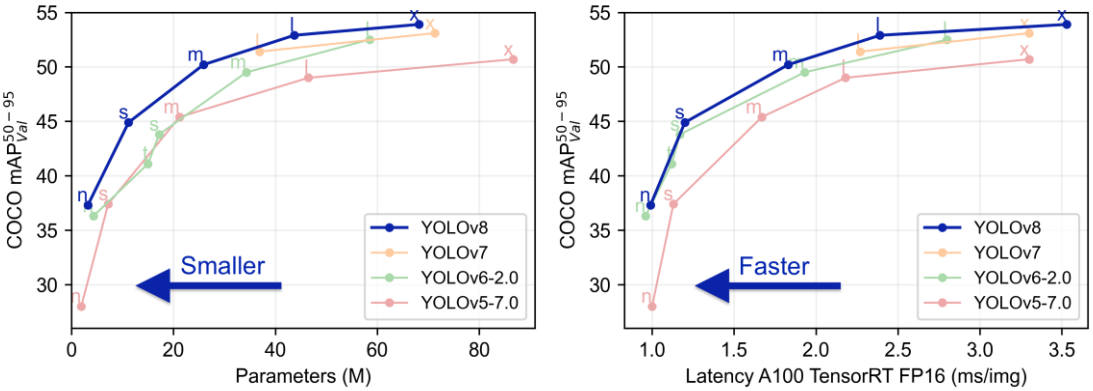

***Figura 7-13**: Comparación entre distintas versiones de YOLO. Fuente: [94]*

En este trabajo, la detección y clasificación de emociones se llevará a cabo de forma general sin consideraciones sobre qué persona la expresa. En otras palabras, la exploración que se realiza en esta parte del proyecto se centra en detectar las emociones correspondientes a felicidad, tristeza, sorpresa y neutral. En vez de reconocer al personaje y su emoción, en esta aproximación se pretende reconocer la emoción de cualquiera de los cuatro personajes de interés, recordando que son Pocoyó, Nina, Elly y Pato. Esta propuesta se basa en el comportamiento humano al reconocer emociones, puesto que este no necesita conocer a la persona o tener conocimiento acerca de sus expresiones faciales para

reconocer emociones básicas como felicidad, relacionada con la aparición de una sonrisa, o tristeza, relacionada con la identificación de unos ojos más pequeños de lo normal, una mueca en la cara o incluso lágrimas. De esta propuesta se infiere que el reconocimiento de emociones en este trabajo, se basará en la identificación de expresiones faciales o gesticulación que, popularmente, se asocian a ciertas emociones.

Nikita Jain et al. [96] publican en 2021 una investigación sobre el uso de redes neuronales profundas para reconocer emociones en personajes animados, basado en la serie 'Tom & Jerry'. En esa investigación se realiza un procedimiento similar al propósito de este desarrollo, extrayendo los rostros faciales de los personajes y clasificando la emoción de cada uno de ellos, obteniendo una precisión de hasta el 90%.

Por ello, se seguirá esta línea de investigación para ver si este detector es capaz ya no solo de clasificar el rostro facial, sino detectar la emoción entre cuatro personajes.

8.2.1. Marco teórico: YOLO

Joseph Redmon et al. [97] presentaron YOLO (*You-Only-Look-Once*) en 2015, cuyo nombre hace referencia a que la detección y clasificación se realiza en una sola pasada, de ahí que sea un detector en una etapa. Realiza una nueva aproximación para la detección de objetos en donde se plantea la detección de objetos como un problema de regresión a cajas delimitadores (*bounding boxes*) separadas especialmente y probabilidades de clases asociadas.

Algunas de las ventajas que presenta YOLO es su alta velocidad en comparación con otros modelos, manteniendo la misma precisión de detección, además de su buena generalización y que es un proyecto de código abierto [98].

La arquitectura de este detector, ilustrada en la Figura 7-14, tiene un total de 24 capas convolucionales, cuatro capas de agrupamiento máximo (*max-pooling layers*) y dos capas totalmente conectadas (*fully-connected layers*). Funciona de forma que redimensiona la imagen a una resolución 448x448 píxeles antes de pasar la primera capa convolucional.

Se aplica una convolución 1x1 para reducir el número de canales, y posteriormente una convolución 3x3 para generar una salida cuboidal, implicando que se extrae características de una región en forma de cubo. Además, durante todo el proceso se usa la función de activación ReLu, excepto en la capa final donde se usa una función de activación lineal, y se aplican técnicas adicionales como la normalización por lotes o el *dropout*, evitando que el modelo sufra de sobreajuste [98].

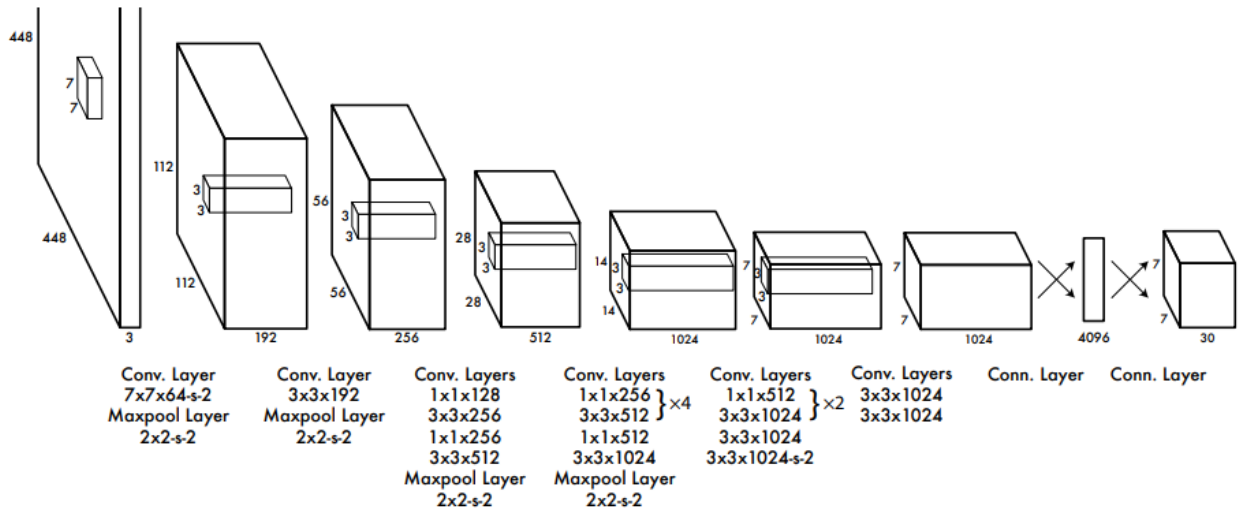

***Figura 7-14**: Arquitectura de YOLO. Fuente: [96]*

Las funciones de activación son esenciales para el diseño de una red neuronal, ya que permite que el modelo sea capaz de aprender y no represente una combinación lineal de capas. En este caso, se usa ReLU, una función no lineal cuyo uso es casi generalizado, la cual define que el valor que devuelva será 0 si el valor de entrada es negativo, mientras que un valor positivo generará una salida igual al valor de entrada [99], definido por la función ilustrada en la Ecuación 7-5

$$f(x) = \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases}$$

***Ecuación 7-5**: Función de activación ReLu. Fuente:* [98]

También se menciona el término *dropout*. Esta técnica hace referencia a la desactivación temporal de un porcentaje de neuronas, permitiendo mejorar el procesamiento y el tiempo de entreno, además de ser una medida que reduce el riesgo de sobreajuste [100].

Una vez queda clara la arquitectura y ciertas características, conviene describir como el algoritmo YOLO realiza la detección de objetos. En primer lugar, se definen los bloques residuales. Se comienza dividiendo una imagen en NxN celdas de igual tamaño, formando así una cuadrícula. Estas celdas son responsables de localizar y predecir la clase del objeto que cubre, si es que hay, además del valor de confianza con la que se realiza la predicción.

El siguiente paso es determinar las cajas delimitadoras que cubrirán todos los objetos de la imagen, empleando un único módulo de regresión para determinar sus atributos, como el valor de la probabilidad de que la celda contenga un objeto, las coordenadas y la altura y el ancho de la caja delimitadora con respecto a la celda que lo envuelve, y las clases detectadas. Muchas veces, un objeto puede tener múltiples celdas candidatas que lo contiene, aunque no todas sean relevantes. Por ello, se aplica el umbral IoU, ya definido en el apartado 6.1.4, para descartar aquellas celdas que no superen el umbral, y se aplica la técnica NMS (*Non-Max Suppression*) para mantener solo aquellas cajas con la mayor probabilidad de detección [98].

A lo largo de los años, distintos miembros u organizaciones de la comunidad han creado distintas versiones de este detector, mediante diferentes mejoras sobre la versión anterior hasta llegar a la versión 8. Este modelo ha sido creado en PyTorch y diseñado para CPUs como GPUs. Esta versión tiene nuevos *backbones* y una nueva función de pérdida, obteniendo el mayor mAP de la historia de YOLO, un 53.9% [101]. Por otra parte, presenta una mejora en la exactitud, mayor velocidad, entrenamiento adaptativo, técnicas avanzadas de aumento de datos o una arquitectura personalizable.

Además, su arquitectura emplea una red neuronal convolucional que unifica los procesos de detección y clasificación, pero se puede dividir en dos partes: el *backbone* y la cabeza o *head*.

El *backbone* se compone de una versión modificada de CSPDarknet53 [102], que consta de 53 capas convolucionales y usa conexiones parciales cruzadas para mejorar el flujo de información entre diferentes capas. Por otro lado, la cabeza o *head* consiste en múltiples capas convolucionales seguidas de una serie de capas totalmente conectadas, responsables de predecir los cuadros delimitadores y las probabilidades de clase [103].

Por último, este detector tiene dos características claves, como es el uso de un mecanismo de autoatención que permite que el modelo se centre en diferentes partes de la imagen y ajuste la importancia de las características encontradas en función de su relevancia para la tarea en cuestión, y la habilidad de detectar objetos a varias escalas empleando una FPN (*Feature Pyramid Network*), compuesta por múltiples capas que detectan objetos de diferentes tamaños a diferentes escalas, mejorando la precisión del modelo a la hora de detectar objetos grandes y pequeños [103].

Sin duda, este detector es muy interesante para explorar la detección de emociones en personajes animados, muy distinto de las emociones en humanos, y comprobar su utilidad y viabilidad para ayudar a los creadores de animaciones 3D y, por otra parte, crear una idea general sobre el rendimiento y potencia de este detector.

8.2.2. Creación y elaboración del conjunto de datos

Al igual que en la fase anterior, para realizar la exploración del detector elegido para identificar y reconocer emociones en los personajes de interés de la serie Pocoyó, es necesario crear previamente un conjunto de datos.

Este conjunto de datos estará compuesto por un total de 4 clases: *Happy, Sad, Surprised* y *Neutral,* representando las emociones de interés. Para ello, en primer lugar, se realiza una recopilación de imágenes filtrando aquellas en las que el rostro facial destaque para poder formar un *dataset* de calidad y balanceado.

La recopilación inicial de imágenes arroja un total de 1874 imágenes, las cuales son procesadas manualmente una a una en la plataforma Roboflow para la anotación de las emociones presentes en cada imagen. De nuevo, la principal fuente de imágenes surge de los vídeos descargados en *YouTube*, concretamente del canal oficial de Pocoyó [69] en el que se encuentran fotos de cada emoción en cada personaje. Además, se vuelve a usar la herramienta FFmpeg para obtener el número de fotogramas por segundo deseado, permitiendo la opción de comenzar la búsqueda de imágenes desde un momento exacto, o cuanto tiempo de vídeo queremos extraer a imágenes.

En total, 1.5K imágenes se han incluido en el conjunto de entrenamiento, mientras que los conjuntos de validación y test contienen 310 imágenes. Por otro lado, se ha desarrollado otra versión en la que se ha empleado técnicas de aumento de datos, generando 4.6K imágenes en el conjunto de entrenamiento [104]. Ambas versiones emplean una resolución de imagen de 640x640 píxeles, ya que es la resolución por defecto de YOLOv8 y la más común en detectores, además del coste computacional que requiere imágenes de mayor calidad, aunque se puede generar otras versiones con distintas resoluciones ya que las imágenes originales son de 1920x1080 píxeles.

Este conjunto de datos es complejo a la hora de procesar, puesto que las emociones son algo tan variable como la percepción de las propias personas. Si bien los personajes Pocoyó y Nina, al tener un rostro más parecido al humano, son más fáciles de identificar sus emociones, con Pato y Elly sucede todo lo contario.

Elly es un personaje en el que cuando su rostro facial expresa felicidad y tristeza, ambas emociones se pueden identificar con facilidad. Sin embargo, su rostro mostrando sorpresa o neutralidad no es tan fácil de identificar. Observando la Figura 7-15, la cual representa el rostro de Elly neutral y sorprendida, a primera vista prácticamente no hay diferencia excepto la posición de las orejas y el hundimiento alrededor de los ojos en la sorpresa. Sin embargo, esto son detalles de los que un ser humano se da cuenta después de pasar un par de segundos analizando ambos rostros, puesto que de una pasada la impresión es que ambos rostros muestran sorpresa.

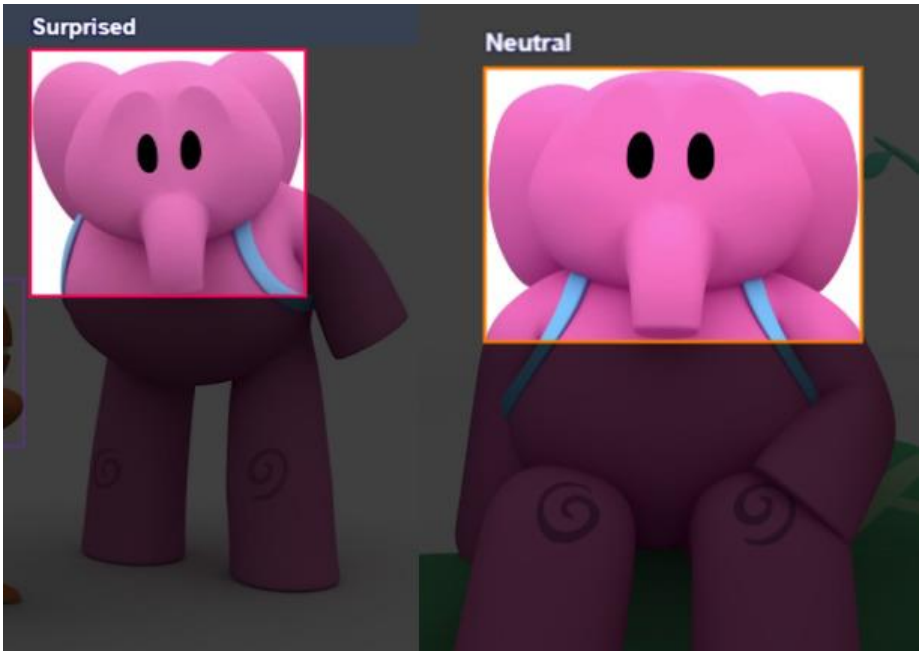

***Figura 7-15**: Comparación entre Elly sorprendida y neutral*

Por otra parte, Pato es otro personaje cuyas emociones son muy complicadas de clasificar, puesto que el personaje siempre expresa cierta "locura" en todas sus acciones. De nuevo, emociones como felicidad y tristeza no son difíciles de identificar en este personaje, ya que la forma de los ojos refleja fácilmente su estado de ánimo, pero la neutralidad y la sorpresa son complicadas de identificar. Como se refleja en la Figura 7-16, ambas caras son prácticamente idénticas, a excepción de la colocación de los ojos y la forma del cuerpo, pues en un estado de sorpresa el personaje tiende a irse hacia delante, mientras que en un estado neutral se mantiene recto, por lo que la emoción ya no solo depende del propio rostro facial, sino de la propia forma del cuerpo, demostrando el dinamismo y complejidad que implica la identificación de las emociones.

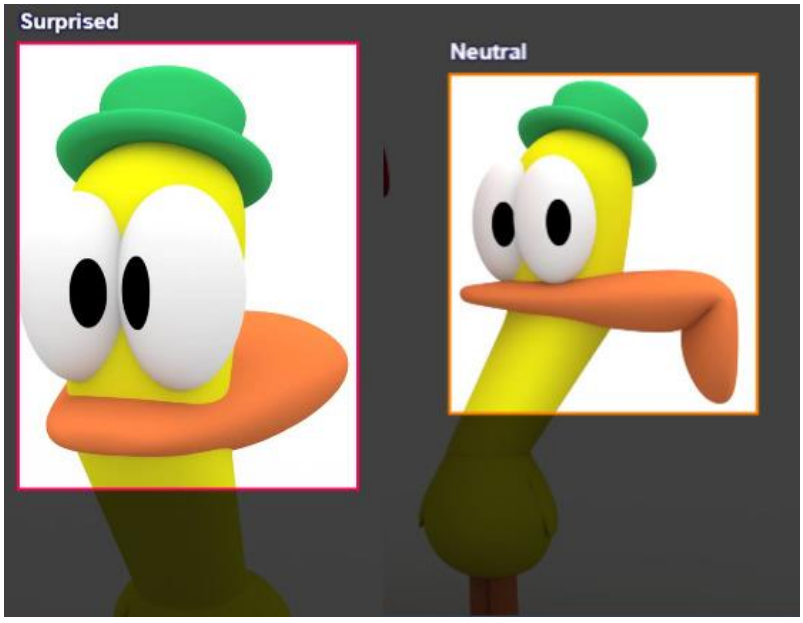

***Figura 7-16**: Comparación entre Pato sorprendido y neutral*

Para clasificar algunas emociones de este tipo, se hace necesario incluso entender el contexto de la escena, ya que en este tipo de personajes animados que no se asemejan tanto a humanos, a simple vista no se puede determinar con total seguridad la emoción que describe, lo cual habla del desafío que supone ya no solo para nosotros, sino para la propia Inteligencia Artificial reconocer emociones basado solo en una imagen de entrada en estos casos concretos, ya que el ser humano a veces necesita entender por qué el personaje está expresando una emoción para poder entenderla y reconocerla. En este sentido, el contexto, la escena e incluso el audio que la acompaña se pueden tornar imprescindibles para acertar en la identificación de emociones.

Mientras se completa el proceso de etiquetado, se comprueba que las clases estén balanceadas para evitar posteriores desajustes en el modelo respecto a la detección y clasificación de distintas emociones entre ellas. La emoción de tristeza y neutral, cuyas clases se representan como *Sad* y *Neutral*, son probablemente las más complicadas de encontrar, ya que, al ser una serie orientada a una audiencia infantil, trata de expresar emociones más alegres y llamativas, por lo que requiere mucho más tiempo que todas las clases estén equilibradas, viendo en la Figura 7-17 el número de instancias por clase.

Figura 7-17: *Distribución de emociones en el conjunto de datos*

Por otra parte, de cara a generar una herramienta mediante la modificación del proceso de inferencia que genere marcas temporales sobre la aparición o desaparición de emociones, se escoge un vídeo totalmente nuevo, de una duración total de 42 segundos, compuesto por 30 fotogramas por segundo, lo que genera un total de 1260 imágenes. Todas estas imágenes han sido también etiquetadas para evaluar el rendimiento del detector sobre este conjunto y poder analizar la salida de marcas temporales de la herramienta y la influencia de la precisión del sistema sobre esta.

8.2.3. Entorno de entrenamiento

Si bien YOLOv8 está construido en PyTorch, la empresa desarrolladora de esta versión, *Ultralytics*, facilita en su repositorio un entorno de fácil instalación y uso para poder entrenar, validar, predecir o desplegar un modelo [45].

Dentro de las opciones de entrenamiento, YOLOv8 ofrece 5 modelos pre-entrenados sobre el conjunto de datos COCO [87] para la detección, y sobre el conjunto de datos ImageNet [88] para clasificación. Estos modelos se clasifican según el mAP logrado sobre el conjunto de datos COCO empleando una resolución de imagen 640x640 píxeles, tal y como se representa en la Tabla 7-8.

Modelo	mAP@0.5:0.95
YOLOv8n	37.3
YOLOv8s	44.9
YOLOv8m	50.2
YOLOv8l	52.9
YOLOv8x	53.9

***Tabla 7-8**: Modelos pre-entrenados de YOLOv8*

Para desarrollar el entrenamiento, se utiliza simultáneamente tanto Kaggle como Colab para reducir el tiempo total que requiere el entrenamiento de todos los modelos, así como para aprovechar tanto las capacidades de uno como de otro. Para ello, el entorno facilita el comando necesario para instalar todo lo necesario:

```
# Install YOLOv8
!pip install ultralytics
```

Una vez instalado, resta descargar en el sistema de almacenamiento de cada *notebook* el conjunto de datos mediante el código que proporciona Roboflow para ello.

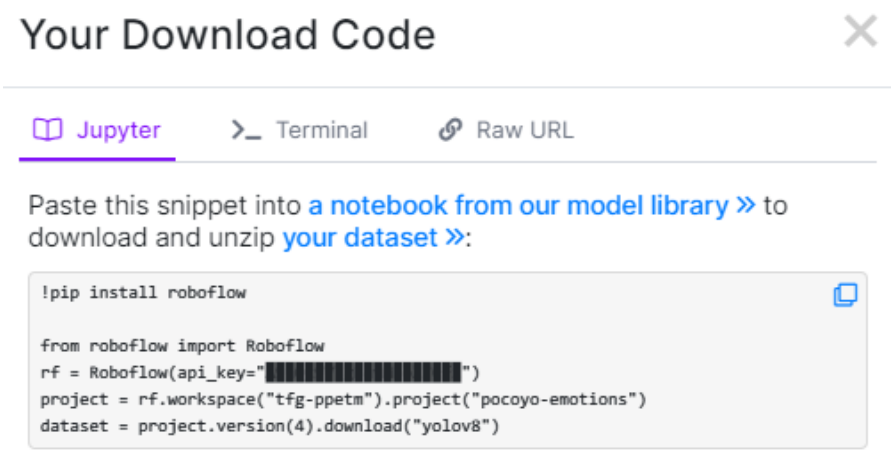

***Figura 7-18:** Código generado por Roboflow para descargar el conjunto de datos de emociones*

Una vez descargado, se edita la ruta del conjunto de datos en el fichero de configuración pertinente y se finaliza la configuración previa al entrenamiento. Para este, YOLOv8 permite tanto utilizar instrucciones en *Python* como un comando con los respectivos argumentos, por lo que se puede elegir indistintamente entre uno u otro.

Respecto a los argumentos, la mayoría de ellos representan la configuración de hiperparámetros, aunque existen otros, como se puede ver en [105]. Entre los argumentos principales, se debe especificar el modelo que se quiere usar, a elegir entre los especificados en la Tabla 7-8 o simplemente ninguno, y permitir que el detector aprenda desde cero. Además, se debe especificar el archivo que define el *dataset*, normalmente con extensión *yaml*, el cual hace saber al detector donde se encuentran las muestras. También en todos los entrenamientos se establecerá un número de épocas y un nuevo parámetro, llamado paciencia. Este parámetro permite introducir un número de épocas tras el cual, si no se detecta ninguna mejora en el entrenamiento, se finaliza se forma anticipada. De esta forma se intenta prevenir la aparición de *overfitting* y optimizar el proceso de aprendizaje en cuando a tiempo y recursos.

A partir de aquí, se establecen varios hiperparámetros que se definen como claves para el entrenamiento. Entre ellos se encuentran la tasa de aprendizaje *(learning rate),* el tamaño del lote *(batch size),* el parámetro *momentum* y el decaimiento de los pesos *(weight decay).*

Por último, se puede elegir el optimizador entre SGD, Adam, Adamax, AdamW, NAdam, RAdam, RMSProp y auto, siendo este último un modo en el cual es el propio detector quien, en base al conjunto de datos y el tamaño del lote, define un optimizador, un *momentum* y el *weight decay*.

8.2.4. Entrenamiento de la red neuronal

La estrategia de entrenamiento planeada se basa, de nuevo, en ensayo y error. Para cada modelo a entrenar, presentes en la Tabla 7-9, se probará todo tipo de combinaciones de hiperparámetros y optimizadores para lograr los mejores resultados posibles. Además, se pondrá en comparación tanto el entrenamiento con un conjunto de datos sin técnicas de aumento de datos como con un conjunto que sí aplique estas técnicas.

Modelo	Parámetros	Resolución de imagen
YOLOv8n	3M	640
YOLOv8s	11.1M	640
YOLOv8m	25.9M	640
YOLOv8l	43.6M	640
YOLOv8x	68.2M	640

***Tabla 7-9**: Modelos entrenados- YOLOv8*

Según el número de parámetros, los modelos consumen más tiempo de entrenamiento y recursos computacionales, por lo que a medida que se entrena un modelo más pesado, las limitaciones de *hardware* requieren que se disminuya el tamaño de lote, entre otros. Por ello, el entrenamiento seguido se ha llevado a cabo de forma progresiva, experimentando y explorando todas las posibilidades con los modelos más ligeros, y utilizando la información obtenida para reducir el campo de exploración en modelos más pesados, especialmente YOLOv8x, en el cual no es posible usar tamaños de lote tan grandes como los demás, y requiere mucho más tiempo de entrenamiento, consumiendo las capacidades de Kaggle y Colab.

Una vez finalizada la fase de entrenamiento, se realiza una evaluación sobre el conjunto de test, que permite analizar el rendimiento del sistema.

8.2.5. Evaluación y análisis del rendimiento de los modelos

En este apartado se realizará el estudio sobre los datos obtenidos una vez se ha realizado el entrenamiento de los modelos.

En primer lugar, se realizará una breve explicación sobre los hiperparámetros usados y las métricas de evaluación que se generan. En los entrenamientos realizados, principalmente se ha ido modificando la tasa de aprendizaje o *learning rate*, el tamaño del lote o *batch size,* el número de épocas y el optimizador.

Entre los optimizadores empleados, se encuentran Adam, AdamW y SGD. Los parámetros se han establecido por defecto, empleando los siguientes:

- **Adam**: Beta1=0.9, Beta2=0.999, Weight Decay=0.0005
- **AdamW**: Beta1=0.9, Beta2=0.999, Weight Decay=0.0005
- **SGD**: Momentum=0.937, Weight Decay=0.0005

Como ya se ha explicado en repetidas ocasiones, no es un proceso automático encontrar los hiperparámetros adecuados, especialmente cuando se trabaja con varios modelos. Por esta razón, cada uno de los modelos especificados en la Tabla 7-9 ha sido entrenado bajo distintas configuraciones para comparar los resultados de cada modelo bajo distintos hiperparámetros y poder finalmente comparar entre las mejores configuraciones de cada modelo.

8.2.5.1. YOLOv8n

En primer lugar, se procede al entreno del modelo YOLOv8n. Para este modelo, se ha probado un total de 28 configuraciones diferentes entre optimizadores, hiperparámetros, y el uso o no uso de aumento de datos. En la Tabla 7-10 se encuentra la relación de hiperparámetros y otras configuraciones empleados durante el entreno del modelo YOLOv8n, el más ligero de todos, limitado a las 10 configuraciones que mejores resultados han dado.

YOLOv8n					
ID	**Optimizador**	***Aumento de datos***	***Learning Rate***	**Épocas**	***Batch size***
1	SGD	No	0.01	50	16
2	SGD	Sí	0.01	50	16
3	AdamW	Sí	0.00125	50	16
4	AdamW	No	0.01	50	16
5	SGD	Sí	0.01	50	32
6	AdamW	No	0.00125	50	16
7	AdamW	No	0.00125	50	32
8	SGD	No	0.01	65	32
9	AdamW	No	0.00125	65	48
10	AdamW	No	0.00125	30	72

***Tabla 7-10**: Configuración de hiperparámetros para modelo YOLOv8n*

Según los resultados del entrenamiento, representados en la Tabla 7-11, a nivel general destaca por encima de todos el entrenamiento con ID 6, cuya configuración consta de una tasa de aprendizaje de 0.00125 y tamaño de lote 16, seleccionando como optimizador AdamW y siendo entrenado en 50 épocas. Comparando sus resultados, en precisión obtiene un 93,4%, siendo el mejor resultado en esta categoría junto con la configuración 1. En *recall,* este se queda en un 90%, no muy lejos del mejor resultado, obtenido por la configuración 3, siendo un 91,2%, una diferencia mínima. Estos resultados siguen siendo buenos en el cálculo del mAP, obteniendo para el umbral IoU de 0.5 un 95.2%, solo superado por la configuración 7, presentando un 96.3%, y consiguiendo un mAP para umbrales IoU desde 0.5 a 0.95 del 85.8%, solo superado por la configuración 1.

Durante esta comparativa, se destaca otras configuraciones, representadas con el ID 1, 3 y 7, las cuales obtienen grandes resultados en precisión, *recall* y mAP comparados con el resto de configuraciones. Estas configuraciones, junto con la configuración 6, tienen en común que en 3 de 4 de ellas no se ha usado técnicas de aumento de datos, lo cual es significativo porque justamente la razón de aplicar aumento de datos es mejorar los resultados. Un ejemplo claro de esto se encuentra en las configuraciones 1 y 2, ya que la única diferencia entre ellas es el aumento de datos, y la configuración 1, sin aumento de datos, es mejor en precisión con una diferencia de un 6.1%, así como en ambos mAP.

El aspecto donde si mejora las configuraciones con aumentos de datos es en el *recall*, véase configuraciones 1-2 y 3-6. Por tanto, si bien las configuraciones con aumentos de datos mejoran el

rendimiento del sistema en cuanto a la detección de todos los objetos de interés en las muestras, sacrifican la precisión a la hora de clasificarlos.

YOLOv8n				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5-:0.95**
1	**0.934**	0.871	0.957	**0.86**
2	0.873	0.881	0.934	0.83
3	0.905	**0.912**	0.952	0.856
4	0.897	**0.906**	0.935	0.831
5	0.891	0.866	0.933	0.83
6	**0.934**	0.9	**0.952**	**0.858**
7	0.918	0.888	**0.963**	0.828
8	0.896	0.89	0.939	0.832
9	0.92	0.861	0.945	0.831
10	0.924	0.882	0.945	0.835

***Tabla 7-11**: Rendimiento del sistema en conjunto de entrenamiento - YOLOv8n*

Los resultados, en general, son bastante positivos, pues a pesar de las situaciones comentadas en el apartado 6.1.2 respecto a las emociones de personajes como Pato y Elly y la dificultad que existe para clasificarlas de una pasada, los modelos son capaces de estar en un mAP@0.5-0.95 de media aproximadamente del 84%, siendo el máximo un 86%, una precisión máxima del 93.4%, y un *recall* máximo del 91.2%. En primera instancia, este modelo, en sus distintas configuraciones, aprovecha la transferencia de pesos para producir grandes resultados, caracterizándose por su alta precisión, lo cual indica que el modelo tiene una tasa alta de clasificación correcta de emociones, y un *recall* excelente para esta tarea, por lo que es capaz de detectar la mayoría de emociones presentes.

A pesar de ello, corresponde una evaluación de las configuraciones sobre el conjunto de test, ya que los resultados pueden variar debido a fenómenos como el sobreajuste. En la Tabla 7-12 se especifican los resultados sobre la evaluación en el conjunto de test. A la vista queda que algunas de las configuraciones destacadas en el entrenamiento, como la 1, 3, 6 y 7, se quedan atrás respecto a otras configuraciones. En especial, destaca la configuración 10, la cual obtiene un mAP@0.5-0.95 del 90.8%, comparado con el 83.5% que obtuvo en el entrenamiento, lo que es un claro indicador de que el modelo, con esta configuración, generaliza bien y es capaz de aplicar su capacidad de detección y clasificación a nuevas muestras.

YOLOv8n				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	0.729	0.767	0.805	0.676
2	0.95	**0.963**	**0.979**	0.886
3	**0.977**	0.94	0.97	0.871
4	0.654	0.718	0.727	0.577

5	0.946	0.949	0.976	**0.888**
6	0.742	0.819	0.829	0.686
7	0.876	0.907	0.942	0.825
8	0.942	0.955	0.973	0.874
9	**0.96**	0.935	0.978	0.885
10	0.956	**0.959**	**0.983**	**0.908**

***Tabla 7-12**: Rendimiento del sistema en conjunto de test - YOLOv8n*

Durante el entrenamiento, la configuración más destacada era la 6, pero en la evaluación en el conjunto de test su rendimiento baja considerablemente, pasando de un mAP@0.5-0.95 del 85.8% al 68.6%. Esta circunstancia podría indicar sobreajuste en el sistema, lo cual se puede analizar mediante las gráficas de pérdida que genera el modelo.

En la Figura 24 se puede ver las gráficas de distintos tipos de pérdida, tales como *box loss, cls loss* y *dfl loss*. El concepto *box loss*, o pérdida de caja, representa como de bien el algoritmo puede localizar el centro de un objeto y en qué medida el cuadro delimitador cubre el objeto [106].

Por otra parte, el concepto *cls loss,* conocido como la pérdida de clasificación, indica lo bien que un algoritmo puede predecir la clase correcta de un objeto [106].

Por último, se encuentra el concepto *dfl loss*. Este concepto, conocido como *Distribution Focal Loss,* ayuda a la red a que se centre en aprender las probabilidades de los valores alrededor de los ubicaciones de las cajas delimitadores reales [107]. En otras palabras, ayuda a combatir conjuntos de datos desequilibrados, donde existan muchas más muestras de una clase que de otra. En esta situación, se les asigna un peso mayor a las muestras de las clases menos representadas, y un peso menor a aquellas clases con muchas muestras para que el modelo centre su atención en clasificar bien las clases minoritarias.

Se puede apreciar en las gráficas, véase Figura 7-19, como las pérdidas en la validación son menores que en el entrenamiento, descendiendo rápidamente para, en las últimas épocas, estabilizarse y equiparar sus valores a las pérdidas del entrenamiento, excepto la pérdida *dfl*. Por ello, puede considerarse que existe un ligero sobreajuste en el modelo con esta configuración y así se explica la pérdida de rendimiento sufrida.

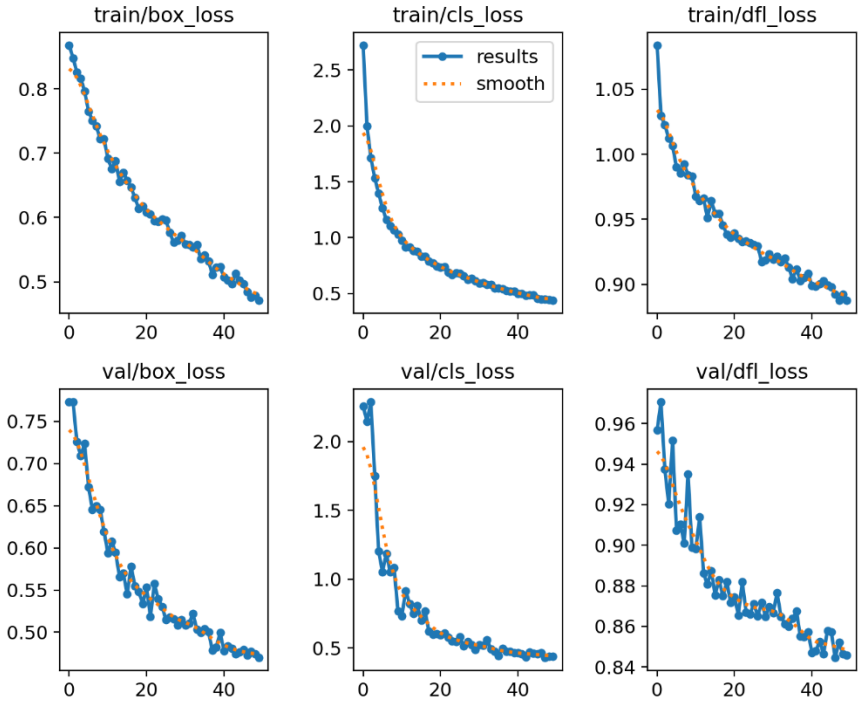

***Figura 7-19**: Gráficas de pérdida de la configuración 6 - YOLOv8n*

Por último, se analiza el mAP@0.5:0.95 por clases de las distintas configuraciones para este modelo. Se puede observar claramente en la Tabla 7-13 como las clases que más cuesta clasificar correctamente son la emoción de tristeza y neutral. En el caso de esta última, no sorprende, ya que como se pudo observar anteriormente, es complicado hasta para un humano diferenciar a algunos personajes cuando están neutros. En cambio, la emoción de tristeza sorprende por su baja precisión, puesto que todos los personajes reflejan de una forma muy identificativa esta emoción. Aun así, se puede ver, por ejemplo, como la configuración 6 no rinde tan bien en esa emoción como los demás, lo cual se puede relacionar con el concepto de *Distribution Focal Loss*. Anteriormente se pudo ver como existía diferencias notorias entre esta pérdida en el entrenamiento y la evaluación, por lo que puede ser un síntoma de que no está considerando las clases con menos muestras con una atención mayor, como sí hacen otras configuraciones.

YOLOv8n				
ID	***Happy***	***Neutral***	***Sad***	***Surprised***
1	0.78	0.632	0.523	0.768
2	0.888	0.883	0.838	0.935
3	0.861	0.863	0.838	0.922
4	0.68	0.484	0.412	0.731
5	0.898	0.869	0.844	0.939
6	0.735	0.663	0.563	0.782
7	0.847	0.814	0.741	0.896
8	0.882	0.872	0.816	0.927

9	0.892	0.876	0.844	0.928
10	0.908	0.914	0.867	0.941
Media	**0.8371**	**0.787**	**0.7286**	**0.8769**

***Tabla 7-13**: mAP@0.5:0.95 por clases en conjunto de test - YOLOv8n*

8.2.5.2. YOLOv8s

El segundo modelo corresponde al nombre de YOLOv8s. Para este modelo, se ha probado un total de 13 configuraciones diferentes entre optimizadores, hiperparámetros, y el uso o no uso de aumento de datos, basado en la información obtenida anteriormente. En la Tabla 7-14 se encuentra la relación de hiperparámetros y otras configuraciones empleados durante el entreno del modelo YOLOv8s, limitado a las 10 configuraciones que mejores resultados han dado.

YOLOv8s					
ID	**Optimizador**	**Aumento de datos**	***Learning Rate***	**Épocas**	***Batch size***
1	SGD	No	0.01	50	16
2	SGD	Sí	0.01	50	16
3	AdamW	Sí	0.00125	50	16
4	AdamW	Sí	0.00125	50	32
5	SGD	No	0.01	50	32
6	SGD	Sí	0.01	50	32
7	SGD	Sí	0.005	50	32
8	SGD	Sí	0.001	50	32
9	SGD	No	0.005	50	32
10	AdamW	No	0.001	65	48

***Tabla 7-14**: Configuración de hiperparámetros para modelo YOLOv8s*

Los resultados del entrenamiento, representados en la Tabla 7-15, se consideran bastante buenos, muy similares entre todas las configuraciones y similares a los obtenidos por el modelo anterior, por lo que, en este momento, un modelo con mayor número de parámetros no representa una mejoría. En base al mAP@0.5:0.95, las configuraciones con mejor precisión global son la 7 y la 9, en donde ambos emplean la misma configuración, presentando diferencia en el conjunto de datos usado, puesto que uno ha empleado aumento de datos y otro no. Estos datos se deben contrastar con la posterior evaluación sobre el conjunto de test.

YOLOv8s				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	0.884	0.892	0.932	0.83
2	0.908	0.879	0.94	0.84
3	**0.921**	0.882	**0.947**	0.842
4	**0.919**	0.861	0.94	0.842
5	0.879	0.905	0.936	0.834
6	0.905	0.874	0.939	0.836
7	0.881	**0.916**	0.946	**0.845**
8	0.858	0.895	0.932	0.842
9	0.899	0.891	0.947	**0.843**
10	0.909	**0.906**	**0.949**	0.84

***Tabla 7-15**: Rendimiento del sistema en conjunto de entrenamiento - YOLOv8s*

Ejecutada la evaluación sobre el conjunto de test, se pueden ver los resultados en la Tabla 7-16. Con este modelo y configuraciones, todos los modelos mantienen el rendimiento e incluso superan las expectativas, destacando la configuración 6 y 7, con un 93,2% y 92,7% respectivamente sobre el mAP@0.5:0.95. En el caso de la configuración 7 ya fue la que mejor resultado dio durante el entrenamiento, por lo que esta configuración consigue un modelo robusto y general. Además, no se detecta en ningún caso alguna sospecha de sobreajuste, aunque la forma de estudiar este fenómeno debe ser mediante las pérdidas y exactitud entre entrenamiento y validación.

YOLOv8s				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	0.955	0.943	0.982	0.894
2	0.981	0.965	0.988	0.925
3	0.966	0.954	0.974	0.908
4	**0.981**	**0.981**	**0.991**	0.924
5	0.978	0.922	0.98	0.892
6	0.967	0.956	0.987	**0.932**
7	0.979	0.964	0.986	**0.927**
8	**0.996**	0.954	**0.991**	0.914
9	0.943	**0.968**	0.976	0.896
10	0.933	0.959	0.98	0.892

Tabla 7-16: *Rendimiento del sistema en conjunto de test - YOLOv8s*

Por último, se analiza el mAP@0.5:0.95 por clases de las distintas configuraciones para este modelo. Se puede observar claramente en la Tabla 7-17 como, de nuevo, la clase que más cuesta reconocer es la emoción de tristeza. Si bien el resultado es mucho mejor que en el modelo anterior, es ligeramente inferior a las otras clases, por lo que es un indicio que el conjunto de datos necesita más imágenes de tristeza.

YOLOv8s				
ID	***Happy***	***Neutral***	***Sad***	***Surprised***
1	0.885	0.889	0.87	0.932
2	0.922	0.923	0.906	0.951
3	0.91	0.911	0.863	0.948
4	0.927	0.915	0.902	0.952
5	0.909	0.889	0.934	0.936
6	0.934	0.918	0.91	0.967
7	0.923	0.917	0.916	0.95
8	0.908	0.906	0.867	0.945
9	0.904	0.901	0.838	0.94
10	0.904	0.872	0.855	0.936
Media	**0,9126**	**0,9041**	**0,8861**	**0,9457**

Tabla 7-17*: mAP@0.5:0.95 por clases en conjunto de test - YOLOv8s*

8.2.5.3. YOLOv8m

El tercer modelo en número de parámetros corresponde al nombre de YOLOv8m. Para este modelo, se ha probado un total de 12 configuraciones diferentes entre optimizadores, hiperparámetros, y el uso o no uso de aumento de datos, basado en la información obtenida anteriormente. En este modelo ya se empieza a acusar las limitaciones computacionales, puesto que, además del incremento en el tiempo de entrenamiento, no permite probar ciertas configuraciones con tamaños de lote como 48. En la Tabla 7-18 se encuentra la relación de ajuste de hiperparámetros y otras configuraciones empleadas durante el entreno del modelo, limitado a las 10 configuraciones que mejores resultados han dado.

YOLOv8m					
ID	**Optimizador**	**Aumento de datos**	***Learning Rate***	**Épocas**	***Batch size***
1	SGD	No	0.01	50	16
2	SGD	Sí	0.01	50	16
3	AdamW	Sí	0.00125	50	16
4	AdamW	Sí	0.00125	50	32
5	SGD	No	0.01	50	32
6	SGD	Sí	0.005	50	32
7	SGD	No	0.005	50	32
8	AdamW	No	0.001	65	32
9	SGD	No	0.001	50	32
10	AdamW	No	0.00125	50	32

***Tabla 7-18**: Configuración de hiperparámetros para modelo YOLOv8m*

Observando la tabla 7-19, en base al mAP@0.5:0.95, se alcanza el mejor rendimiento del modelo empleando las configuraciones 1 y 7. De nuevo, ambas configuraciones comparten el optimizador SGD, el cual se erige como el que mejor rendimiento proporciona para nuestro problema, empleando tasas de aprendizaje 0.005 y 0.01 a medida que el tamaño del lote es más grande o pequeño. Por otra parte, en líneas generales, todas las configuraciones ofrecen un rendimiento similar en la fase de entrenamiento, al igual que el modelo anterior, lo que habla de la flexibilidad que nos ofrece la transferencia de pesos.

YOLOv8m				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	0.882	**0.92**	**0.949**	**0.847**
2	0.9	0.9	0.944	0.846
3	0.887	0.914	0.947	0.844
4	0.914	0.878	0.947	0.847
5	0.919	0.907	0.941	0.839
6	**0.922**	0.862	0.942	0.845
7	0.899	0.89	0.949	**0.857**
8	0.915	**0.915**	**0.947**	0.843
9	0.88	0.889	0.939	0.841

10	**0.926**	0.868	0.947	0.838

***Tabla 7-19**: Rendimiento del sistema en conjunto de entrenamiento - YOLOv8m*

No puede existir fase de entrenamiento sin una evaluación sobre un conjunto de test compuesto por imágenes totalmente nuevas y desconocidas para el modelo. En el caso que se representa, véase Tabla 7-20, se aprecia como en base al mAP@0.5:0.95, la métrica más completa para analizar el rendimiento de un modelo en la tarea de detección de objetos, destaca la configuración 6 y 7, basados en el optimizador SGD donde uno usa técnica de aumento de datos y otro no. De nuevo, prácticamente todos los modelos ofrecen un rendimiento excelente y muy similar entre ellos, con grandes tasas de precisión y de detección de todos los casos positivos, conocido como *recall*.

YOLOv8m				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	0.934	**0.974**	0.982	0.896
2	**0.975**	0.969	0.983	0.915
3	**0.976**	0.939	0.985	0.898
4	0.963	**0.976**	**0.988**	0.913
5	0.965	0.947	0.981	0.897
6	0.969	0.969	**0.991**	**0.93**
7	0.973	0.956	0.986	**0.921**
8	0.959	0.968	0.987	0.904
9	0.945	0.961	0.971	0.893
10	0.929	0.885	0.961	0.853

***Tabla 7-20**: Rendimiento del sistema en conjunto de test - YOLOv8m*

En el análisis por clases, reflejado en la Tabla 7-21, se obtiene una conclusión clara. Aunque el resultado es excepcional, la clase que cuesta más identificar a los modelos es *Sad*, que representa la tristeza. Aun así, en el caso de las configuraciones 6 y 7 se obtiene un mAP del 90.4% y 89.8% respectivamente, una precisión general más que notable y presentan gran robustez en las otras clases, especialmente en la sorpresa.

YOLOv8m				
ID	***Happy***	***Neutral***	***Sad***	***Surprised***
1	0.893	0.898	0.854	0.941
2	0.897	0.929	0.886	0.948
3	0.894	0.892	0.87	0.938
4	0.896	0.916	0.893	0.946
5	0.896	0.898	0.854	0.939
6	0.934	0.924	0.904	0.957
7	0.924	0.918	0.898	0.949

8	0.904	0.904	0.867	0.94
9	0.904	0.893	0.838	0.939
10	0.859	0.834	0.794	0.925
Media	**0,9001**	**0,9006**	**0,8658**	**0,9422**

***Tabla 7-21**: mAP@0.5:0.95 por clases en conjunto de test - YOLOv8m*

8.2.5.4. YOLOv8l

El cuarto modelo es YOLOv8l. Para este modelo, se ha probado un total 8 configuraciones diferentes entre optimizadores, hiperparámetros, y el uso o no uso de aumento de datos, basado en la información obtenida anteriormente. En este modelo las limitaciones computacionales comienzan a ser un obstáculo, debido a la diferencia de parámetros con los otros modelos. Para este caso, los recursos a nivel de *hardware* no son suficientes para poder lanzar una ejecución con un tamaño de lote 32, además del nuevo incremento de tiempo en el entreno, superando de largo las dos horas en cada configuración. Debido a estas limitaciones, en la Tabla 7-22 se encuentra la relación de ajuste de hiperparámetros y otras configuraciones empleadas durante el entreno del modelo, limitado a las 5 configuraciones que mejores resultados han dado.

YOLOv8l					
ID	**Optimizador**	**Aumento de datos**	***Learning Rate***	**Épocas**	***Batch size***
1	SGD	No	0.01	50	16
2	SGD	Sí	0.01	50	16
3	AdamW	No	0.00125	50	16
4	SGD	No	0.005	50	16
5	SGD	No	0.001	50	16

***Tabla 7-22**: Configuración de hiperparámetros para modelo YOLOv8l*

De acuerdo a la Tabla 7-23, el rendimiento del modelo se ha visto maximizado durante el entrenamiento con las configuraciones 4 y 5, de nuevo, correspondientes al optimizador SGD con tasas de aprendizaje 0.005 y 0.001, con un mAP@0.5:0.95 del 85.3% y 85.4% respectivamente. Además, ambas configuraciones destacan en todas las métricas mostradas excepto en precisión, sin ser esto una diferencia muy grande. En general, los resultados vuelven a ser muy similares, ya no solo entre las distintas configuraciones, sino entre este modelo y los anteriores, ya que no se aprecia una mejoría considerable.

YOLOv8l				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	**0.932**	0.888	0.942	0.839
2	**0.92**	0.858	0.944	0.845
3	0.859	0.851	0.911	0.803
4	0.904	**0.892**	**0.954**	**0.853**
5	0.907	**0.915**	**0.95**	**0.854**

***Tabla 7-23:** Rendimiento del sistema en conjunto de entrenamiento - YOLOv8l*

En comparación con los resultados obtenidos mediante la evaluación del modelo con las distintas configuraciones en el conjunto de *test*, reflejados en la Tabla 7-24, la configuración 4, y en especial la 2, son las más destacadas por su mAP@0.5:0.95, reflejando su gran capacidad de detección y clasificación en varios umbrales. Aun así, la configuración 7 y 1 no están tan lejos,

obteniendo resultados similares, pero la configuración 3, la única que empleó el optimizador AdamW, se queda atrás.

YOLOv8l				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	**0.972**	0.963	0.986	0.905
2	**0.975**	**0.965**	**0.989**	**0.937**
3	0.863	0.835	0.906	0.776
4	0.937	0.977	**0.988**	**0.919**
5	0.954	**0.987**	0.979	0.91

***Tabla 7-24**: Rendimiento del sistema en conjunto de test - YOLOv8l*

Debido a los resultados de la configuración 3, se realiza una comparación entre los valores de pérdida durante el entrenamiento y durante la validación. Es notable como, reflejado en la Figura 7-20, la pérdida de clasificación durante la validación comienza en valores muy altos, pero rápidamente baja hasta casi 0, mientras que en el entrenamiento tiene un descenso más progresivo y gradual, lo cual puede indicar sobreajuste en el modelo.

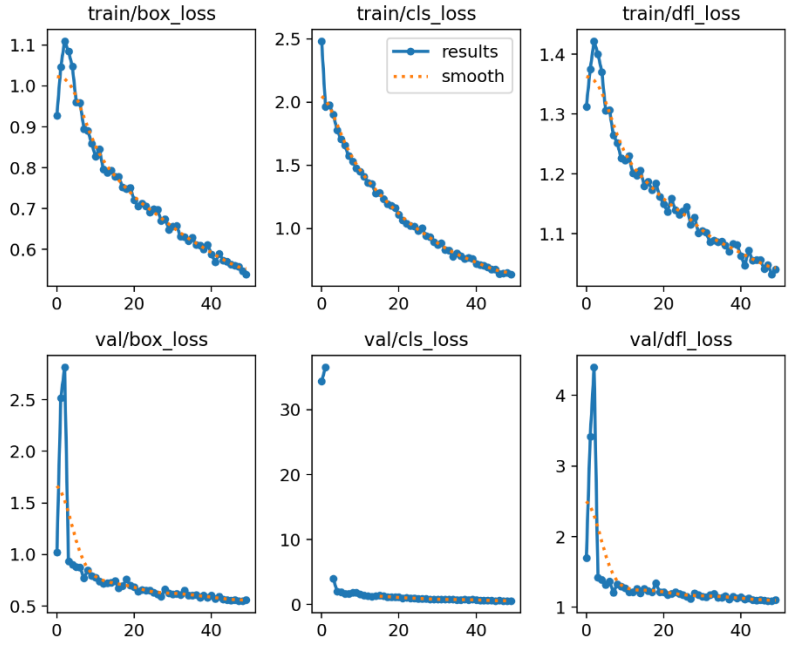

***Figura 7-20**: Gráficas de pérdida de la configuración 3 - YOLOv8l*

Finalmente, se realiza un análisis por clases, especificado en la Tabla 7-25. Sin duda, no es ninguna novedad que, de media, la clase con menor precisión según el mAP sea *Sad,* representando la tristeza. Aunque un 85.22% mAP no es precisamente un resultado malo, no mejora la media de los modelos YOLOv8s y YOLOv8m. De hecho, la media de todas las clases queda por debajo de la media de las clases obtenidas por estos modelos, siendo otro indicio más de que, a pesar de trabajar

con un modelo con mayor número de parámetros, no necesariamente debe tener un mejor rendimiento.

YOLOv8l				
ID	***Happy***	***Neutral***	***Sad***	***Surprised***
1	0.905	0.895	0.881	0.938
2	0.935	0.928	0.923	0.963
3	0.814	0.718	0.71	0.876
4	0.918	0.92	0.885	0.951
5	0.911	0.92	0.862	0.948
Media	**0,8966**	**0,8762**	**0,8522**	**0,9352**

***Tabla 7-25**: mAP@0.5:0.95 por clases en conjunto de test - YOLOv8l*

8.2.5.5. YOLOv8x

Finalmente, se procede al entrenamiento con el modelo YOLOv8x. Para este modelo, se ha probado un total 5 configuraciones diferentes entre optimizadores, hiperparámetros, y el uso o no uso de aumento de datos, basado en la información obtenida anteriormente. En este modelo las limitaciones computacionales son grandes debido al elevado número de parámetros en comparación con los otros modelos. De nuevo, este modelo produce un incremento de tiempo en el entrenamiento, pudiendo superar las 3 horas o más, además de la memoria que ocupa. Debido a estas limitaciones, en la Tabla 7-26 se encuentra la relación de ajuste de hiperparámetros y otras configuraciones empleadas durante el entreno del modelo, limitado a las 5 configuraciones que mejores resultados han dado.

YOLOv8x					
ID	**Optimizador**	**Aumento de datos**	***Learning Rate***	**Épocas**	***Batch size***
1	SGD	No	0.01	50	16
2	AdamW	No	0.00125	50	16
3	SGD	No	0.005	50	16
4	SGD	No	0.001	50	16
5	AdamW	No	0.01	50	16

***Tabla 7-26**: Configuración de hiperparámetros para modelo YOLOv8x*

Con la fase de entreno concluida, se obtienen los resultados presentes en la Tabla 7-27. Comparado con los modelos anteriores, este modelo se queda algo atrás en cuando a rendimiento a pesar de ser el modelo con mayor número de parámetros. Entre las configuraciones, destaca en especial la número 1, empleando el optimizador SGD y una tasa de aprendizaje del 0.01. Las demás configuraciones no se califican como malas, pero están un paso por detrás no solo de la configuración 1, sino en general de todos los otros modelos.

YOLOv8x				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	**0.911**	**0.915**	**0.934**	**0.837**
2	0.837	0.86	0.904	0.796
3	**0.852**	**0.895**	**0.907**	**0.809**
4	0.85	0.893	0.898	0.801
5	0.842	0.872	0.881	0.774

***Tabla 7-27:** Rendimiento del sistema en conjunto de entrenamiento - YOLOv8x*

Durante la evaluación de las configuraciones sobre el conjunto de *test*, se observa en primera instancia, viendo la Tabla 7-28, que los resultados obtenidos en el entrenamiento se mantienen, sin atisbar ningún indicio de sobreajuste o subajuste. De nuevo, destaca la configuración 1, con una clara distancia en cuanto a mAP@0.5:0.95 respecto a los demás.

YOLOv8x				
ID	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
1	**0.938**	**0.958**	**0.964**	**0.874**
2	0.85	0.856	0.91	0.778
3	0.868	0.871	0.914	0.832
4	0.851	0.86	0.892	**0.836**
5	0.863	0.869	0.895	0.826

Tabla 7-28: *Rendimiento del sistema en conjunto de test - YOLOv8x*

En el análisis por clases de las distintas configuraciones, se ve reflejado en la Tabla 7-29 como la configuración 1 es la que mayor mAP obtiene en todas las clases, lo que refleja la complejidad de configuración que conlleva este modelo. Además, sigue la tendencia de los otros modelos, puesto que al sistema le cuesta reconocer la emoción de tristeza más que las demás.

YOLOv8x				
ID	***Happy***	***Neutral***	***Sad***	***Surprised***
1	0.881	0.879	0.797	0.938
2	0.812	0.698	0.718	0.883
3	0.871	0.859	0.721	0.88
4	0.855	0.854	0.752	0.883
5	0.802	0.867	0.759	0.879
Media	**0,8442**	**0,8314**	**0,7494**	**0,8926**

Tabla 7-29*: mAP@0.5:0.95 por clases en conjunto de test - YOLOv8x*

Además, la media del mAP por clase es menor que cualquier otro modelo, lo que refleja que la complejidad de este modelo y su número de parámetros está perjudicando el rendimiento de la red en este problema específico como es la detección y clasificación de emociones.

8.2.6. Herramienta de generación de marcas temporales

Para generar un fichero de marcas temporales que representen la aparición o desaparición de emociones, se sigue un procedimiento similar al desarrollado en el apartado 7.1.6. De esta forma, se modifica el proceso de inferencia del detector YOLOv8 para obtener las predicciones que debe hacer el detector para poder sacar las imágenes de salida con los cuadros delimitadores sobre los objetos. Obtenidas las predicciones, se extrae las clases predichas, y mediante cierta lógica se exportan al fichero de marcas temporales.

La marca temporal tendrá el formato 'mm:ss.SSS → aparición o desaparición de mensaje', donde la 'm' representa los minutos, la 's' minúscula los segundos y la 'S' mayúscula, los milisegundos en tres dígitos. Para extraer el momento del vídeo que se está procesando, se emplea el flag CAP_PROP_POS_MSEC [94], de forma que devuelve el momento actual del vídeo en milisegundos y se transforma al formato deseado.

```
current_pos_ms = self.format_ms(int(vid_cap.get(cv2.CAP_PROP_POS_MSEC)))
```

Además, se lleva un registro de las emociones presentes en la escena para poder identificar si la emoción aparece o desaparece. Para ello, con las clases predichas por el modelo, se recorre las claves de un diccionario creado que representa si un emoción está presente en la escena o no y cuantas se han detectado y se compara si estas clases se encuentran en el diccionario. Si se encuentran en el diccionario, se compara que el número de veces que aparece esa emoción en el fotograma con su valor en el diccionario, para comprobar si existe una nueva aparición, o desaparición. Además, si el modelo no devuelve nada, se entiende que todas las emociones han desaparecido.

```
if emotion == 'Happy' or emotion == 'Happys':
   if emotions['Happy'] > int(count):
      emotions['Happy'] -= int(count)
      if emotions['Happy'] > 0:
         emotions_detected.append('Happy')
      emotions_output += f'{int(count)} Happy emotions disappears, '
   elif int(count) > emotions['Happy']:
        count = int(count) - emotions['Happy']
        emotions['Happy'] += int(count)
        emotions_output += f'{int(count)} new Happy emotions appears, '
        emotions_detected.append('Happy')
```

Finalmente, se construye el mensaje final que se exportará al fichero correspondiente, con el formato visible en la Figura 7-21.

```
00:36.766 --> 1 new Neutral emotions appears
00:36.800 --> 1 new Sad emotions appears, All Neutral emotions disappears
00:36.833 --> 1 new Neutral emotions appears, 1 new Surprised emotions appears, All Sad emotions disappears
00:36.866 --> All Neutral emotions disappears, All Surprised emotions disappears
```

***Figura 7-21**: Ejemplo de marcas temporales en YOLOv8*

Como es lógico, este fichero de marcas temporales se ve condicionado por la precisión del detector. Entre mayor rendimiento ofrezca el sistema, mejores datos se podrán obtener para su posterior procesado.

Por ello, la idea de esta herramienta es complementarla con una evaluación sobre el vídeo de entrada, de forma que se pueda analizar el resultado de la herramienta con datos sobre la precisión del detector. En este contexto, se emplea el conjunto de datos creado a partir del vídeo, explicado en el apartado 6.2.2, para poder comparar la salida de marcas temporales con la precisión del detector, empleando la configuración 2 del modelo YOLOv8l, véase Tabla 7-24, la cual dio el mejor mAP entre todos los modelos.

Visualizando la tabla 7-30, el mAP@0.5:0.95 logrado por este modelo y su configuración específica es del 79.9%. En otras palabras, se puede considerar que el mAP, que proporciona una medida global sobre la precisión del modelo, es del 80% sobre un conjunto de 1260 imágenes que representan el vídeo en cuestión.

YOLOv8l				
Clase	**Precisión**	***Recall***	***mAP@0.5***	**mAP@0.5:0.95**
All	**0.801**	**0.833**	**0.871**	**0.799**
Happy	0.914	0.901	0.956	0.841
Neutral	0.734	0.843	0.833	0.802
Sad	0.719	0.708	0.794	0.736
Surprised	0.838	0.879	0.902	0.817

***Tabla 7-30**: Evaluación sobre el vídeo - YOLOv8 conf. 2*

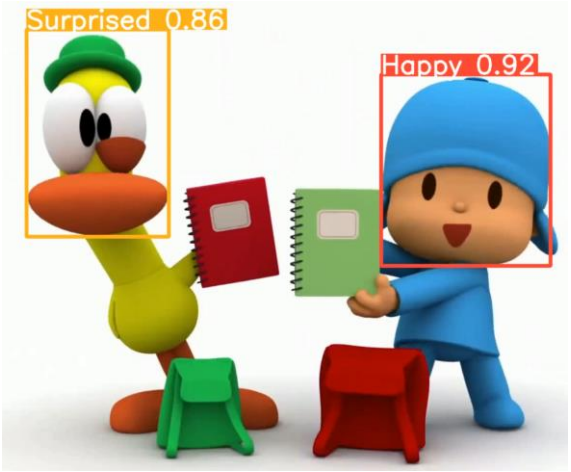

***Figura 7-22**: Ejemplo de un fotograma del vídeo de salida*

Con esta precisión, es posible generar un fichero de marcas temporales cuyos datos representen la realidad, es decir, que sean de calidad. Un modelo con menor precisión podría provocar que existan muchas marcas temporales correspondiente a fotogramas seguidos en donde, sin cambiar la

escena, en unos se detectan ciertas emociones y otros no, añadiendo ruido al archivo. Por ello, aunque el camino a seguir de cara a ayudar a la creación de animación 3D está claro, se debe incidir en refinar los modelos de detección, ya que el éxito reside en ellos.

8.2.7. Conclusiones

En esta segunda fase del desarrollo, orientada a la detección y clasificación de emociones en personajes animados, se comenzó explicando el marco teórico que envuelve el modelo empleado, YOLOv8. Además, se ha creado otro conjunto de datos, dividido en imágenes de entrenamiento, validación y test, tanto para llevar a cabo el entrenamiento como para la realización de otras pruebas, como ha sido el desarrollo de una herramienta para la extracción de marcas temporales sobre la aparición y desaparición de emociones de la escena y su evaluación para medir el rendimiento del sistema, previa anotación de los fotogramas de un vídeo de entrada. Para conseguir esto, se ha llevado a cabo una fase de entrenamiento organizada en modelos hasta lograr los mejores resultados posibles, y el posterior análisis de los mismos para calificar el rendimiento del sistema.

A lo largo de este desarrollo se han ido analizando los resultados de cada modelo con respecto a las configuraciones empleadas. En la Tabla 7-31 se recoge los 5 mejores modelos y configuraciones en base al mAP@0.5:0.95 sobre el conjunto de test. Sin duda, se puede extraer bastante información. En primer lugar, las 5 mejores configuraciones entre modelo e hiperparámetros emplean el optimizador SGD. Es una conclusión interesante, ya que entre las pruebas también se han usado otros optimizadores como Adam y AdamW y, aunque el primero no ha dado los resultados esperados, el segundo ha estado cerca del obtenido por SGD. En definitiva, para propósito concreto parece que el optimizador SGD se adapta mejor.

Por otro lado, otro dato interesante es el hecho de que 4 de los 5 mejores resultados han sido empleando aumento de datos. El denominado *data augmentation* ha sido clave en esta fase llegando a marcar la diferencia en la generalización del sistema, llegando a un mAP del 93.7%. Además, este dato se ha conseguido empleando un tamaño de lote de 16, menor que todos los demás.

Por último, la tasa de aprendizaje ha sido muy importante para prevenir sobreajuste, destacando el valor 0.005, pues, aunque no es tan común durante el entreno de una red neuronal, en la mayoría de modelos junto con el optimizador SGD ha funcionado muy bien.

YOLOv8							
ID	**Modelo**	**Optimizador**	***Aumento de datos***	***Learning Rate***	**Épocas**	***Batch size***	***mAP@0.5:0.95***
2	YOLOv8l	SGD	Sí	0.01	50	16	**0.937**
6	YOLOv8s	SGD	Sí	0.01	50	32	0.932
6	YOLOv8m	SGD	Sí	0.005	50	32	0.93
7	YOLOv8s	SGD	Sí	0.005	50	32	0.927
7	YOLOv8m	SGD	No	0.005	50	32	0.921

Tabla 7-31*: Mejores resultados globales en detección y clasificación de emociones*

Entre las distintas clases del modelo, generalmente se ha obtenido una gran precisión para cada una. En la Figura 7-23 se ilustra la matriz de confusión generada por la configuración con mayor

rendimiento sobre el conjunto de *test*. Se puede ver como todas las clases tienen una precisión excelente, sin casi confusiones entre clases, aún con varias detecciones en el fondo predichas como clases, constituyendo falsos positivos.

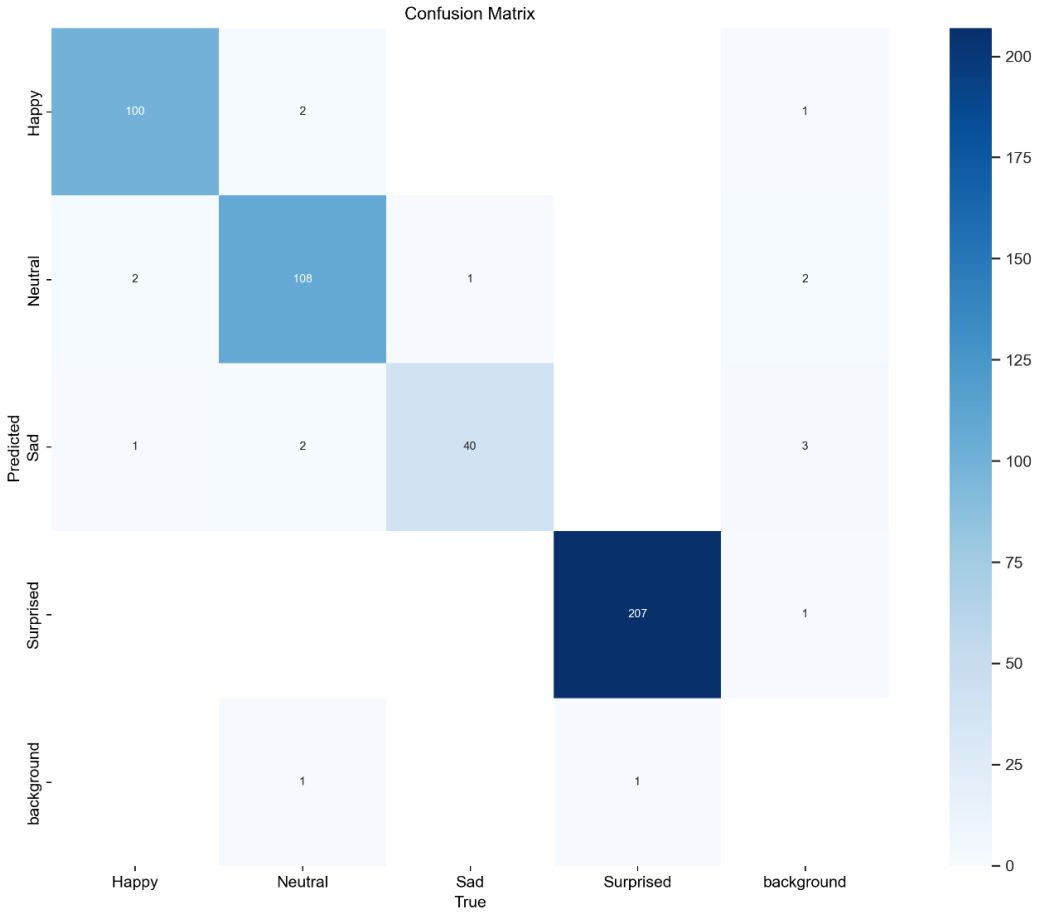

***Figura 7-23**: Matriz de confusión del mejor modelo-configuración sobre el conjunto de test*

En la Figura 7-24 se puede ver un ejemplo real de la predicción realizada. Es destacable la capacidad de detectar en el personaje Pato la emoción neutra, pues este personaje no expresa las emociones de forma tan transparente, y teniendo al lado un personaje triste, podría haberlo clasificado como tal. Por otro lado, es sorprendente la capacidad de detección de emociones en escenas como la segunda, donde Pocoyó está rodeado de personajes sorprendidos, y el detector es capaz de distinguir la forma de la boca, a pesar de ser pequeña, para clasificarlo correctamente como neutral.

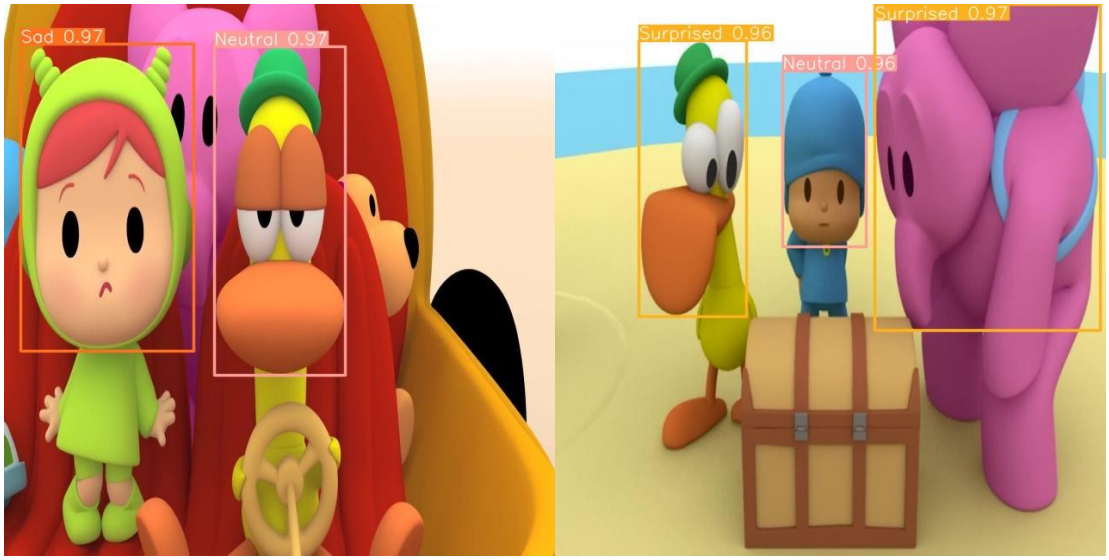

***Figura 7-24**: Predicción de emociones*

Sin duda, este detector configurado correctamente permite hacer grandes cosas con él. En este caso, es capaz de identificar la mayoría de emociones hasta en situaciones complejas, lo cual supera con creces las expectativas iniciales y permite extraer marcas temporales de la aparición y desaparición de emociones de forma robusta, con la futura intención de ayudar a la industria de la creación de vídeos animados en 3D.

Además, se deja patente la necesidad de un buen conjunto de datos e invertir el tiempo necesario en ello y aplicar técnicas de aumento de datos que permitan mejorar la generalización y evitar sobreajuste o subajuste, pues entre otras cosas, es una de las principales razones de que un proyecto de este tipo sea un éxito.

Capítulo 9. Conclusiones y trabajo futuro

9.1. Conclusiones

Desde el inicio, este trabajo se caracteriza por su labor investigadora, exploratoria y aclaradora. La investigación es necesaria en cualquier trabajo de este ámbito, pero su importancia aumenta cuando no existe ninguna contribución con un éxito reseñable y se debe explorar toda la documentación y bibliografía existente, así como estar al día de los avances en la Inteligencia Artificial y en el campo de la Visión por Computador.

Uno de los objetivos principales era la exploración de diferentes algoritmos de Inteligencia Artificial para la identificación de personajes u objetos en diferentes escenas de un vídeo de entrada, así como la posible identificación de acciones y emociones en determinadas secuencias. En otras palabras, se busca realizar una investigación sobre la detección y clasificación de personajes animados y emociones, además de la identificación de acciones.

Para la detección y clasificación de personajes, se ha realizado toda una exploración acerca de este ámbito durante la fase de estudio del estado del arte, analizando la historia de los métodos y algoritmos existentes, su base, y finalizando con el análisis de varios detectores *state-of-the-art* como YOLO, RetinaNet, EfficientNet, SSD o Faster R-CNN. Además, para todos ellos existe alguna publicación o investigación en el ámbito de personajes animados que refuerza su posición en este trabajo. Esta exploración termina siendo un éxito, ya que se determinan varios detectores de interés que pueden ser adaptados al problema en cuestión, el cual es detectar y clasificar personajes animados en 3D, en este caso de la serie Pocoyó. Para ello, se realiza una comparativa entre ellos y basado en sus resultados en ámbitos como este y en su rendimiento general, se decide usar Faster R-CNN.

Para la detección y clasificación de emociones, se recogen publicaciones de interés que guían el camino para desarrollar esta fase mediante una profunda exploración, ya que, si bien la detección de personajes animados está tratada en la bibliografía existente con ciertas investigaciones de interés, para la detección de emociones fuera de humanos no existen numerosos desarrollos, y mucho menos con un éxito notable. Aun así, se consigue desarrollar una amplia retrospectiva del estado actual de esta tarea, y se logra culminar el objetivo de exploración y seleccionar un algoritmo o modelo que permita llevar esta fase a cabo. Para seleccionar el algoritmo, se propone una elección basada en el estudio de los resultados en ámbitos similares, eligiendo YOLO.

Por último, se encuentra el reconocimiento de acciones. La exploración e investigación de esta tarea es compleja por todo lo que conlleva. Durante el estudio de la literatura existente, queda claro que no existe ninguna investigación pública de cara al reconocimiento de acciones en personajes animados. Los pocos enfoques encontrados son aplicados a la acción humana o a publicaciones actualizando el estado de este campo, sin encontrar un entorno o sistema que se pueda aplicar a nuestro caso. Por lo tanto, la exploración e investigación del reconocimiento de acciones mediante Inteligencia Artificial queda en el estudio de la literatura más reciente, pues si bien logra identificar los principales puntos de interés, no se logra encontrar algún método viable, por lo que no se considera implementarlo en este trabajo, ya que excedería tanto el tiempo estimado como los recursos disponibles.

Para poder medir la calidad o el rendimiento de cada aproximación sobre la tarea correspondiente, se debe proceder de forma que cada método elegido sea entrenado sobre un conjunto de datos conformado por muestras de las clases de interés a reconocer, para obtener una métrica como el mAP (*mean Average Precision*), ampliamente usada para medir, de forma global, el rendimiento y la precisión de un sistema sobre una tarea concreta.

Dentro de la detección y clasificación de personajes animados, la aproximación hecha mediante Faster R-CNN no ha resultado tan exitosa como se deseaba, puesto que el mayor mAP conseguido es del 79.7% con el modelo Faster R-CNN + EfficientNet-B4. Los otros modelos entrenados, véase Tabla 6-6, se quedan alrededor del 70%, lo cual supone un contratiempo en el objetivo de esta fase, la cual es conseguir generar una herramienta que, a partir de un vídeo, proporcione como salida un registro de marcas temporales en tiempo de vídeo que delimite cuando aparecen personajes. Si el detector no es lo suficientemente preciso, este archivo quedará lleno de ruido y de detecciones intermitentes, incluso falsas detecciones.

Entre las razones de esta baja precisión, se pudo comprobar que el detector sufre un problema de falsos positivos, tendiendo a detectar muchos objetos de fondo como personajes, así como las limitaciones computacionales encontradas, ya que los modelos entrenados ocupan mucha memoria y abarcan mucho tiempo de entrenamiento, dificultando el proceso de configuración y optimización del sistema.

Entre las causas que expliquen esta situación, puede considerarse el conjunto de datos. Durante el desarrollo se explicó que el conjunto de datos tuvo que ser mejorado para obtener estos resultados, aumentando aproximadamente un 20% la precisión. Por ello, si bien no se considera que el conjunto de datos sea muy complejo, el refinamiento del *dataset* se encuentra como necesario y como primer paso para comprobar si los modelos aumentan su precisión. Además, la implementación de Faster R-CNN carece de ciertas métricas interesantes para estudiar el comportamiento del modelo, por lo que se puede concluir que, si bien el procedimiento para generar la herramienta ha quedado resuelto en este trabajo, no se puede garantizar un fichero de marcas temporales robusto con este método, lo cual también se considera un resultado de este trabajo dentro de los objetivos de investigación marcados.

Para el desarrollo de la herramienta se ha optado por emplear las utilidades que aporta la implementación del detector para predecir, es decir, realizar el proceso de inferencia sobre imágenes o vídeos, así como para evaluar el rendimiento de un modelo sobre un conjunto de imágenes. Por ello, se modifica el proceso de inferencia para, además de producir las predicciones, generar el fichero de marcas temporales.

Todo ello se engloba todo en un pequeño script que llama a ambos procesos, para poder analizar la salida del archivo de marcas temporales junto con la precisión del sistema, siendo necesario etiquetar previamente los fotogramas del vídeo de entrada. De igual forma, no es necesario evaluar el rendimiento del sistema sobre el vídeo en esta fase, ya que la intención era generar un fichero de marcas temporales que posteriormente se analizaría por aquellos interesados, y se ha logrado.

La última fase corresponde a la detección y clasificación de emociones en personajes animados. En este caso, se eligió usar YOLOv8 para desarrollar un detector capaz de realizar la tarea, cuyos resultados superaron las expectativas. Ajustando distintas configuraciones a distintos modelos, cinco en total, se logra el máximo mAP@0.5:0.95 en 93.7%, véase Tabla 6-31. Es sorprendente el

rendimiento de este detector, puesto que más de 5 modelos han logrado superar un mAP del 90%, lo cual habla de lo prometedor que es este método.

De esta fase se extraen ciertas conclusiones, como el mejor rendimiento del optimizador AdamW respecto a Adam, aunque ambos eclipsados por SGD. Además, no se aprecia en ningún momento sobreajuste o subajuste, siendo capaz los mejores modelos de detectar emociones complejas en personajes como Pato aun estando rodeado de otras emociones.

También se hace hincapié en la necesidad de invertir el tiempo necesario en formar un conjunto de datos que permita la generalización del sistema a todas las muestras posibles, porque en ambas fases del desarrollo se ha podido comprobar que este factor es fundamental para el éxito.

En esta fase también se busca el desarrollo de la herramienta que permita realizar la comparación justa entre distintas aproximaciones, al igual que se busca una herramienta de generación de marcas temporales que, en tiempo de vídeo, delimite cuando aparecen o desaparecen las emociones de los personajes. YOLOv8 ofrece en su implementación herramientas de predicción, es decir, para realizar el proceso de inferencia sobre imágenes o vídeos, así como de evaluación del rendimiento de un modelo sobre un conjunto de imágenes. Por ello, en esta fase también se modifica el proceso de inferencia para, además de producir las predicciones, genere el fichero de marcas temporales. De esta forma, es posible analizar la salida del archivo de marcas temporales junto con la precisión del sistema, siendo necesario etiquetar previamente los fotogramas del vídeo de entrada. El objetivo era generar un fichero de marcas temporales que posteriormente se analizaría por el sector dedicado a la producción de vídeos animados en 3D, lo cual se ha podido cumplir.

9.2. Trabajo futuro

Una vez terminada la fase del desarrollo y todo lo que abraca este trabajo, se puede realizar una retrospectiva gracias a esta conclusión y hablar sobre posibles extensiones. Sin duda, el primer trabajo futuro que se puede proponer tomando como base las conclusiones obtenidas debería ser la evaluación de otra aproximación para la detección y clasificación de personajes animados, suponiendo la creación o modificación, si existe, del proceso de inferencia de la red neuronal elegida para poder generar las marcas temporales. Esto se hace necesario debido a la precisión obtenida, ya que el problema no se estima tan complejo como para haber alcanzado el techo en cuanto al rendimiento de un detector para esta tarea.

Otra extensión posible sería el estudio detallado de algún algoritmo para poder llevar a cabo el reconocimiento de acciones de personajes animados, o realizar una investigación profunda para llegar a proponer nuevos algoritmos que cumplan esta tarea. En caso de crear o encontrar un entorno viable, se debe estudiar cómo obtener las muestras que representen la acciones deseadas para poder entrenar el modelo o la herramienta necesaria.

Por último, la ampliación de este trabajo más evidente es la unión de ambos procesos. Esta unión permitiría realizar la detección y clasificación de los personajes en un primer paso, para que las predicciones puedan ser procesadas de forma que se pueda extraer el rostro facial del personaje. Una vez extraído, y según el personaje, se podría realizar la clasificación de la emoción que muestre. Se dice “según el personaje” debido a que lo ideal sería entrenar 4 modelos distintos, basados en las emociones de cada uno de los personajes de interés. De esta forma, la clasificación se volvería particular y no general, tal y como está ahora, y podría aumentar la precisión en cada una de las emociones, centrándose en cada personaje en específico. Así, se generaría un solo fichero de marcas temporales con información mucho más precisa, de la cual la industria podría sacar mucho partido.

La consecución de este trabajo puede beneficiar a muchos creadores de animaciones 3D, para entender a su audiencia y poder ofrecer un contenido a su gusto. Además, abre un nuevo camino en la aplicación de las tecnologías en la animación 3D, y probablemente 2D, puesto que no se hace el mismo hincapié como puede ser en el reconocimiento humano. Este caso no está tan tratado por la Inteligencia Artificial, y abrirá puertas a la investigación y desarrollo en técnicas para animaciones que ayudarán a la industria y, por supuesto, al consumidor final.

Bibliografía

[1] «¿Qué es la inteligencia artificial o IA?», *Google Cloud*. Disponible en: https://cloud.google.com/learn/what-is-artificial-intelligence?hl=es-419. [Accedido: 14 de abril de 2023]

[2] «Pocoyó enseñará español en China con el Instituto Cervantes», *Fundación Consejo España China*, 11 de febrero de 2021. Disponible en: https://spain-china-foundation.org/patronos/pocoyo-ensenara-espanol-en-china-con-el-instituto-cervantes/. [Accedido: 2 de julio de 2023]

[3] Z. Zou, K. Chen, Z. Shi, Y. Guo, y J. Ye, «Object Detection in 20 Years: A Survey», may 2019, Disponible en: https://arxiv.org/abs/1905.05055. [Accedido: 14 de abril de 2023]

[4] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», jun. 2015, Disponible en: https://arxiv.org/abs/1506.02640. [Accedido: 14 de abril de 2023]

[5] C.-Y. Wang, A. Bochkovskiy, y H.-Y. M. Liao, «YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors», jul. 2022, Disponible en: https://arxiv.org/abs/2207.02696. [Accedido: 14 de abril de 2023]

[6] J. Liu y D. Li, «Research on Moving Object Detection of Animated Characters», *Procedia Comput Sci*, vol. 208, pp. 271-276, 2022, doi: 10.1016/j.procs.2022.10.039. Disponible en: https://www.sciencedirect.com/science/article/pii/S1877050922014806. [Accedido: 14 de abril de 2023]

[7] «YOLOv8 Docs», *Ultralytics*. Disponible en: https://docs.ultralytics.com/. [Accedido: 15 de abril de 2023]

[8] «Performance Benchmark of YOLO v5, v7 and v8», *StereoLabs*. Disponible en: https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8/. [Accedido: 15 de abril de 2023]

[9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, y P. Dollár, «Focal Loss for Dense Object Detection», ago. 2017, Disponible en: https://arxiv.org/abs/1708.02002v2. [Accedido: 15 de abril de 2023]

[10] Y. Zheng *et al.*, «Cartoon Face Recognition», en *Proceedings of the 28th ACM International Conference on Multimedia*, New York, NY, USA: ACM, oct. 2020, pp. 2264-2272. doi: 10.1145/3394171.3413726. Disponible en: https://www.webofscience.com/wos/woscc/full-record/WOS:000810735002037. [Accedido: 15 de abril de 2023]

[11] D. Shah, «Mean Average Precision (mAP) Explained: Everything You Need to Know», *V7 Labs*, 2 de marzo de 2023. Disponible en: https://bit.ly/3MKlYGc. [Accedido: 15 de abril de 2023]

[12] V. S. Subramanyam, «IOU (Intersection over Union)», *Medium*, 17 de enero de 2021. Disponible en: https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef. [Accedido: 15 de abril de 2023]

[13] M. Tan y Q. V. Le, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks», may 2019, Disponible en: https://arxiv.org/abs/1905.11946. [Accedido: 15 de abril de 2023]

[14] K. Yousaf y T. Nawaz, «A Deep Learning-Based Approach for Inappropriate Content Detection and Classification of YouTube Videos», *IEEE Access*, vol. 10, pp. 16283-16298, 2022, doi: 10.1109/ACCESS.2022.3147519. Disponible en: https://ieeexplore.ieee.org/document/9696242. [Accedido: 15 de abril de 2023]

[15] W. Liu *et al.*, «SSD: Single Shot MultiBox Detector», dic. 2015, doi: 10.1007/978-3-319-46448-0_2. Disponible en: https://arxiv.org/abs/1512.02325. [Accedido: 15 de abril de 2023]

[16] Y. Wang, «Animation Character Detection Algorithm Based on Clustering and Cascaded SSD», *Sci Program*, vol. 2022, pp. 1-10, ene. 2022, doi: 10.1155/2022/4223295. Disponible en: https://www.hindawi.com/journals/sp/2022/4223295/. [Accedido: 15 de abril de 2023]

[17] X. Qin, Y. Zhou, Z. He, Y. Wang, y Z. Tang, «A Faster R-CNN Based Method for Comic Characters Face Detection», en *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, nov. 2017, pp. 1074-1080. doi: 10.1109/ICDAR.2017.178. Disponible en: https://ieeexplore.ieee.org/abstract/document/8270109. [Accedido: 16 de abril de 2023]

[18] S. Ren, K. He, R. Girshick, y J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks», jun. 2015, Disponible en: https://arxiv.org/abs/1506.01497. [Accedido: 15 de abril de 2023]

[19] S. Minaee, M. Minaei, y A. Abdolrashidi, «Deep-Emotion: Facial Expression Recognition Using Attentional Convolutional Network», *Sensors*, vol. 21, n.º 9, p. 3046, abr. 2021, doi: 10.3390/s21093046. Disponible en: https://www.webofscience.com/wos/woscc/full-record/WOS:000650782000001. [Accedido: 15 de abril de 2023]

[20] G. M, P. A, S. V, S. M, y L. K, «Improved Facial Emotion Recognition using Yolo and DeepFace for Music suggestion», en *2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)*, IEEE, ago. 2022, pp. 1124-1127. doi: 10.1109/ICESC54411.2022.9885456. Disponible en: https://ieeexplore.ieee.org/document/9885456. [Accedido: 16 de abril de 2023]

[21] G. Ogden, «Emotion Detection Using Yolo-V5 and RepVGG». Disponible en: https://github.com/George-Ogden/emotion. [Accedido: 16 de abril de 2023]

[22] Y. Yin, M. Ayoub, A. Abel, y H. Zhang, «A Dynamic Emotion Recognition System Based on Convolutional Feature Extraction and Recurrent Neural Network», 2023, pp. 134-154. doi: 10.1007/978-3-031-16078-3_8. Disponible en: https://www.webofscience.com/wos/woscc/full-record/WOS:000890320100008. [Accedido: 16 de abril de 2023]

[23] Q. Cao, W. Zhang, y Y. Zhu, «Deep learning-based classification of the polar emotions of "moe"-style cartoon pictures», *Tsinghua Sci Technol*, vol. 26, n.º 3, pp. 275-286, jun. 2021, doi: 10.26599/TST.2019.9010035. Disponible en:

https://www.webofscience.com/wos/woscc/full-record/WOS:000595603000004. [Accedido: 16 de abril de 2023]

[24] M. I. Lakhani, J. McDermott, F. G. Glavin, y S. P. Nagarajan, «Facial Expression Recognition of Animated Characters using Deep Learning», en *2022 International Joint Conference on Neural Networks (IJCNN)*, IEEE, jul. 2022, pp. 1-9. doi: 10.1109/IJCNN55064.2022.9892186. Disponible en: https://ieeexplore.ieee.org/document/9892186. [Accedido: 16 de abril de 2023]

[25] «Action Recognition», *Papers with Code*. Disponible en: https://paperswithcode.com/task/action-recognition-in-videos. [Accedido: 15 de abril de 2023]

[26] H. H. Pham, L. Khoudour, A. Crouzil, P. Zegers, y S. A. Velastin, «Video-based Human Action Recognition using Deep Learning: A Review», ago. 2022, Disponible en: https://arxiv.org/abs/2208.03775. [Accedido: 15 de abril de 2023]

[27] C. Feichtenhofer, A. Pinz, y A. Zisserman, «Convolutional Two-Stream Network Fusion for Video Action Recognition», abr. 2016, Disponible en: https://arxiv.org/abs/1604.06573. [Accedido: 15 de abril de 2023]

[28] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition», sep. 2014, Disponible en: https://arxiv.org/abs/1409.1556v6. [Accedido: 16 de abril de 2023]

[29] M. G. Morshed, T. Sultana, A. Alam, y Y.-K. Lee, «Human Action Recognition: A Taxonomy-Based Survey, Updates, and Opportunities», *Sensors*, vol. 23, n.º 4, p. 2182, feb. 2023, doi: 10.3390/s23042182. Disponible en: https://www.mdpi.com/1424-8220/23/4/2182. [Accedido: 15 de abril de 2023]

[30] A. Ulhaq, N. Akhtar, G. Pogrebna, y A. Mian, «Vision Transformers for Action Recognition: A Survey», sep. 2022, Disponible en: https://arxiv.org/abs/2209.05700. [Accedido: 15 de abril de 2023]

[31] V. Mazzia, S. Angarano, F. Salvetti, F. Angelini, y M. Chiaberge, «Action Transformer: A Self-Attention Model for Short-Time Pose-Based Human Action Recognition», jul. 2021, doi: 10.1016/j.patcog.2021.108487. Disponible en: https://arxiv.org/abs/2107.00606. [Accedido: 15 de abril de 2023]

[32] O. Moutik *et al.*, «Convolutional Neural Networks or Vision Transformers: Who Will Win the Race for Action Recognitions in Visual Data?», *Sensors*, vol. 23, n.º 2, p. 734, ene. 2023, doi: 10.3390/s23020734. Disponible en: https://pubmed.ncbi.nlm.nih.gov/36679530/. [Accedido: 16 de abril de 2023]

[33] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, y C. Schmid, «ViViT: A Video Vision Transformer», mar. 2021, Disponible en: https://arxiv.org/abs/2103.15691. [Accedido: 15 de abril de 2023]

[34] G. Bertasius, H. Wang, y L. Torresani, «Is Space-Time Attention All You Need for Video Understanding?», feb. 2021, Disponible en: https://arxiv.org/abs/2102.05095v4. [Accedido: 15 de abril de 2023]

[35] D. Neimark, O. Bar, M. Zohar, y D. Asselmann, «Video Transformer Network», feb. 2021, Disponible en: https://arxiv.org/abs/2102.00719. [Accedido: 15 de abril de 2023]

[36] M. Patrick *et al.*, «Keeping Your Eye on the Ball: Trajectory Attention in Video Transformers», jun. 2021, Disponible en: https://arxiv.org/abs/2106.05392. [Accedido: 15 de abril de 2023]

[37] J. Yang, X. Dong, L. Liu, C. Zhang, J. Shen, y D. Yu, «Recurring the Transformer for Video Action Recognition», en *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, jun. 2022, pp. 14043-14053. doi: 10.1109/CVPR52688.2022.01367. Disponible en: https://ieeexplore.ieee.org/document/9878733. [Accedido: 15 de abril de 2023]

[38] H. Fan *et al.*, «Multiscale Vision Transformers», abr. 2021, Disponible en: https://arxiv.org/abs/2104.11227. [Accedido: 15 de abril de 2023]

[39] Y. Jiaxin, W. Fang, y Y. Jieru, «A review of action recognition based on Convolutional Neural Network», *J Phys Conf Ser*, vol. 1827, n.º 1, p. 012138, mar. 2021, doi: 10.1088/1742-6596/1827/1/012138. Disponible en: https://iopscience.iop.org/article/10.1088/1742-6596/1827/1/012138. [Accedido: 15 de abril de 2023]

[40] A. Sánchez-Caballero *et al.*, «3DFCNN: real-time action recognition using 3D deep neural networks with raw depth information», *Multimed Tools Appl*, vol. 81, n.º 17, pp. 24119-24143, jul. 2022, doi: 10.1007/s11042-022-12091-z. Disponible en: https://link.springer.com/article/10.1007/s11042-022-12091-z. [Accedido: 15 de abril de 2023]

[41] S. Shinde, A. Kothari, y V. Gupta, «YOLO based Human Action Recognition and Localization», *Procedia Comput Sci*, vol. 133, pp. 831-838, 2018, doi: 10.1016/j.procs.2018.07.112. Disponible en: https://www.sciencedirect.com/science/article/pii/S1877050918310652. [Accedido: 15 de abril de 2023]

[42] «The MIT License», *Open Source Initiative (OSI)*. Disponible en: https://opensource.org/license/mit/. [Accedido: 11 de julio de 2023]

[43] S. Ranjan, «sovit-123/fasterrcnn-pytorch-training-pipeline», *GitHub*. Disponible en: https://github.com/sovit-123/fasterrcnn-pytorch-training-pipeline. [Accedido: 27 de junio de 2023]

[44] «GNU Affero General Public License», *Free Software Foundation*. Disponible en: https://www.gnu.org/licenses/agpl-3.0.en.html. [Accedido: 11 de julio de 2023]

[45] G. Jocher, A. Chaurasia, y J. Qiu, «Ultralytics YOLOv8». Ultralytics, 2023. Disponible en: https://github.com/ultralytics/ultralytics. [Accedido: 29 de junio de 2023]

[46] «Condiciones de uso y normas de conducta en la página», *Pocoyó*. Disponible en: https://www.pocoyo.com/condiciones-uso. [Accedido: 11 de julio de 2023]

[47] «Visual Studio: IDE y Editor de código para desarrolladores de software y Teams». Disponible en: https://visualstudio.microsoft.com/es/. [Accedido: 23 de junio de 2023]

[48] «FFmpeg». Disponible en: https://ffmpeg.org/. [Accedido: 24 de junio de 2023]

[49] «CVAT». Disponible en: https://www.cvat.ai/. [Accedido: 23 de junio de 2023]

[50] «Roboflow: Give your software the power to see objects in images and video». Disponible en: https://roboflow.com/. [Accedido: 24 de junio de 2023]

[51] «GitHub». Disponible en: https://github.com/. [Accedido: 24 de junio de 2023]

[52] «Hello World - GitHub Docs». Disponible en: https://docs.github.com/en/get-started/quickstart/hello-world. [Accedido: 24 de junio de 2023]

[53] «Microsoft Word: software de procesamiento de texto | Microsoft 365». Disponible en: https://www.microsoft.com/es-es/microsoft-365/word?activetab=tabs%3afaqheaderregion3. [Accedido: 24 de junio de 2023]

[54] Google, «Google Colab». Disponible en: https://colab.google/. [Accedido: 24 de junio de 2023]

[55] A. Goldbloom y B. Hammer, «Kaggle: Your Home for Data Science». Disponible en: https://www.kaggle.com/. [Accedido: 24 de junio de 2023]

[56] Ç. Uslu, «What is Kaggle?», *DataCamp*, marzo de 2022. Disponible en: https://www.datacamp.com/blog/what-is-kaggle. [Accedido: 24 de junio de 2023]

[57] «PyTorch». Disponible en: https://pytorch.org/. [Accedido: 24 de junio de 2023]

[58] Nvidia, «What is PyTorch?» Disponible en: https://www.nvidia.com/en-us/glossary/data-science/pytorch/. [Accedido: 24 de junio de 2023]

[59] R. Girshick, «Fast R-CNN», abr. 2015, Disponible en: https://arxiv.org/abs/1504.08083. [Accedido: 25 de junio de 2023]

[60] R. Girshick, J. Donahue, T. Darrell, y J. Malik, «Rich feature hierarchies for accurate object detection and semantic segmentation», *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 580-587, nov. 2013, doi: 10.1109/CVPR.2014.81. Disponible en: https://arxiv.org/abs/1311.2524v5. [Accedido: 25 de junio de 2023]

[61] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, y A. W. M. Smeulders, «Selective Search for Object Recognition», *Int J Comput Vis*, vol. 104, n.º 2, pp. 154-171, sep. 2013, doi: 10.1007/s11263-013-0620-5

[62] R. Gandhi, «R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms | by Rohith Gandhi | Towards Data Science», *Medium*, 9 de julio de 2018. Disponible en: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Accedido: 25 de junio de 2023]

[63] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition», dic. 2015.

[64] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, y S. Belongie, «Feature Pyramid Networks for Object Detection», dic. 2016.

[65] «Faster R-CNN», *Hasty*, 17 de mayo de 2023. Disponible en: https://hasty.ai/docs/mp-wiki/model-architectures/faster-r-cnn. [Accedido: 26 de junio de 2023]

[66] «Inicio de la producción de una nueva temporada de Pocoyó», *Zinkia*, 4 de junio de 2019. Disponible en: https://www.zinkia.com/es/prensa/notas-prensa/inicio-de-la-produccion-de-una-nueva-temporada-de-pocoyo. [Accedido: 26 de junio de 2023]

[67] R. Khan, «Importance of Datasets in Machine Learning and AI Research», *DataToBiz*, 13 de mayo de 2022. Disponible en: https://www.datatobiz.com/blog/datasets-in-machine-learning/. [Accedido: 26 de junio de 2023]

[68] «2022 State of Data Science», *Anaconda*, 2022. Disponible en: https://www.anaconda.com/resources/whitepapers/state-of-data-science-report-2022. [Accedido: 26 de junio de 2023]

[69] «POCOYÓ en ESPAÑOL - Canal Oficial», *Youtube*. Disponible en: https://www.youtube.com/@pocoyo. [Accedido: 26 de junio de 2023]

[70] «Aumento de datos», *TensorFlow*. Disponible en: https://www.tensorflow.org/tutorials/images/data_augmentation?hl=es-419. [Accedido: 26 de junio de 2023]

[71] N. C. Durgadas y R. Mateus, «Pocoyó Project Image Dataset - v4», *Roboflow*, 7 de junio de 2023. Disponible en: https://universe.roboflow.com/tfg-ppetm/pocoyo-project-bfwoc/dataset/4. [Accedido: 26 de junio de 2023]

[72] «Faster R-CNN», *PyTorch*. Disponible en: https://pytorch.org/vision/main/models/faster_rcnn.html. [Accedido: 27 de junio de 2023]

[73] «Object detection reference training scripts - PyTorch», *GitHub*. Disponible en: https://github.com/pytorch/vision/tree/main/references/detection. [Accedido: 27 de junio de 2023]

[74] J. Brownlee, «Difference Between a Batch and an Epoch in a Neural Network», *Machine Learning Mastery*, 10 de agosto de 2022. Disponible en: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/. [Accedido: 27 de junio de 2023]

[75] «What is Learning Rate in Machine Learning», *Deepchecks*. Disponible en: https://deepchecks.com/glossary/learning-rate-in-machine-learning/. [Accedido: 27 de junio de 2023]

[76] «Tasa de aprendizaje», *Interactive Chaos*. Disponible en: https://interactivechaos.com/es/manual/tutorial-de-deep-learning/tasa-de-aprendizaje. [Accedido: 27 de junio de 2023]

[77] «Optimizers», *ML Glossary*. Disponible en: https://ml-cheatsheet.readthedocs.io/en/latest/optimizers.html. [Accedido: 27 de junio de 2023]

[78] «ML | Stochastic Gradient Descent (SGD)», *GeeksforGeeks*, 29 de mayo de 2023. Disponible en: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/. [Accedido: 27 de junio de 2023]

[79] J. Brownlee, «Gentle Introduction to the Adam Optimization Algorithm for Deep Learning», *Machine Learning Mastery*, 13 de enero de 2021. Disponible en: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accedido: 27 de junio de 2023]

[80] I. Loshchilov y F. Hutter, «Decoupled Weight Decay Regularization», nov. 2017.

[81] «Adamw», *Hasty*. Disponible en: https://hasty.ai/docs/mp-wiki/solvers-optimizers/adamw. [Accedido: 27 de junio de 2023]

[82] S. Goswami, «A deeper look at how Faster-RCNN works», *Medium*, 11 de julio de 2018. Disponible en: https://whatdhack.medium.com/a-deeper-look-at-how-faster-rcnn-works-84081284e1cd. [Accedido: 27 de junio de 2023]

[83] K. Elijah, «Cross-Entropy Loss Function», *Towards Data Science*, 2 de octubre de 2020. Disponible en: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e. [Accedido: 27 de junio de 2023]

[84] J. Hui, «Understanding Feature Pyramid Networks for object detection (FPN)», *Medium*, 27 de marzo de 2018. Disponible en: https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c. [Accedido: 27 de junio de 2023]

[85] I. Loshchilov y F. Hutter, «SGDR: Stochastic Gradient Descent with Warm Restarts», ago. 2016, Disponible en: https://arxiv.org/abs/1608.03983v5. [Accedido: 27 de junio de 2023]

[86] «Cosine Annealing Explained», *Papers with Code*. Disponible en: https://paperswithcode.com/method/cosine-annealing. [Accedido: 27 de junio de 2023]

[87] Microsoft, «COCO - Common Objects in Context». Disponible en: https://cocodataset.org/#home. [Accedido: 28 de junio de 2023]

[88] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, y Li Fei-Fei, «ImageNet: A large-scale hierarchical image database», en *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, jun. 2009, pp. 248-255. doi: 10.1109/CVPR.2009.5206848. Disponible en: https://ieeexplore.ieee.org/document/5206848. [Accedido: 28 de junio de 2023]

[89] M. Tan, «EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling», *Google Research*, 29 de mayo de 2019. Disponible en: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. [Accedido: 28 de junio de 2023]

[90] A. Howard *et al.*, «Searching for MobileNetV3», may 2019, Disponible en: https://arxiv.org/abs/1905.02244v5. [Accedido: 28 de junio de 2023]

[91] «Image Classification on ImageNet», *Papers with Code*. Disponible en: https://paperswithcode.com/sota/image-classification-on-imagenet. [Accedido: 28 de junio de 2023]

[92] «¿Qué es el sobreajuste?», *IBM*. Disponible en: https://cutt.ly/SwyFBvEs. [Accedido: 28 de junio de 2023]

[93] «OpenCV - Open Computer Vision Library». Disponible en: https://opencv.org/. [Accedido: 29 de junio de 2023]

[94] «OpenCV: Flags for video I/O». Disponible en: https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html. [Accedido: 29 de junio de 2023]

[95] L. Porto Pedrosa, «Estudio de las emociones en los personajes animados de Inside Out», *Revista Mediterránea de Comunicación*, vol. 7, n.º 1, p. 31, jun. 2016, doi: 10.14198/MEDCOM2016.7.1.2

[96] N. Jain, V. Gupta, S. Shubham, A. Madan, A. Chaudhary, y K. C. Santosh, «Understanding cartoon emotion using integrated deep neural network on large dataset», *Neural Comput Appl*, vol. 34, n.º 24, pp. 21481-21501, dic. 2022, doi: 10.1007/s00521-021-06003-9

[97] J. Redmon, S. Divvala, R. Girshick, y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», Disponible en: http://pjreddie.com/yolo/. [Accedido: 29 de junio de 2023]

[98] Z. Keita, «YOLO Object Detection Explained», *DataCamp*, septiembre de 2022. Disponible en: https://www.datacamp.com/blog/yolo-object-detection-explained. [Accedido: 29 de junio de 2023]

[99] «ReLu Definition», *DeepAI*. Disponible en: https://deepai.org/machine-learning-glossary-and-terms/relu. [Accedido: 29 de junio de 2023]

[100] N. Srivastava, G. Hinton, A. Krizhevsky, y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting», *Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, 2014.

[101] M. Moin, «Unleashing the Power of YOLOv8: A Comprehensive Guide to the Real-Time Object Detection Model», *Medium*, 11 de enero de 2023. Disponible en: https://medium.com/augmented-startups/unleashing-the-power-of-yolov8-a-comprehensive-guide-to-the-real-time-object-detection-model-af71014a5e7a. [Accedido: 29 de junio de 2023]

[102] A. Bochkovskiy, C.-Y. Wang, y H.-Y. M. Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection», Disponible en: https://github.com/AlexeyAB/darknet. [Accedido: 29 de junio de 2023]

[103] A. Mehra, «Understanding YOLOv8 Architecture, Applications & Features», *Labellerr*, 16 de abril de 2023. Disponible en: https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/. [Accedido: 29 de junio de 2023]

[104] R. Mateus, «Pocoyo Emotions Dataset», *Roboflow*, 2023. Disponible en: https://universe.roboflow.com/tfg-ppetm/pocoyo-emotions. [Accedido: 30 de junio de 2023]

[105] «Configuration - Ultralytics YOLOv8 Docs», *Ultralytics*, 2023. Disponible en: https://docs.ultralytics.com/usage/cfg/#train. [Accedido: 30 de junio de 2023]

[106] M. Kasper-Eulaers, N. Hahn, P. E. Kummervold, S. Berger, T. Sebulonsen, y Ø. Myrland, «Short Communication: Detecting Heavy Goods Vehicles in Rest Areas in Winter Conditions Using YOLOv5», *Algorithms 2021, Vol. 14, Page 114*, vol. 14, n.º 4, p. 114, mar. 2021, doi: 10.3390/A14040114. Disponible en: https://www.mdpi.com/1999-4893/14/4/114/htm. [Accedido: 30 de junio de 2023]

[107] X. Li *et al.*, «Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection», Disponible en: https://github.com/implus/GFocal. [Accedido: 30 de junio de 2023]

[108] «Moe», *Wikipedia, la enciclopedia libre*. Disponible en: https://es.wikipedia.org/wiki/Moe. [Accedido: 12 de julio de 2023]