Universidad Zaragoza

1542

Trabajo Fin de Grado

Detección de acciones durante el proceso de cocinado mediante técnicas de aprendizaje automático

Action detection during the cooking process using machine learning techniques

Autor

Guillermo Enguita Lahoz

Directores

Ana Serrano Pacheu
Diego Gutiérrez Pérez

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2023

Agradecimientos

Agradecer en primer lugar tanto al *Graphics and Imaging Lab* como a *BSH* por impulsar las distintas líneas de investigación sobre la 'cocina del futuro', que han hecho posible el desarrollo de este trabajo. También agradezco especialmente la labor de mis tutores, Ana Serrano Pacheu y Diego Gutiérrez Pérez, por ofrecerme la oportunidad de unirme al proyecto, además de su asesoría y apoyo durante el mismo.

Así mismo, a los compañeros que he conocido durante el proyecto, por su cercanía y amabilidad, así como a mis amigos, pareja y familia, por su apoyo incondicional y por hacer que estos meses fueran más llevaderos.

Finalmente, quiero agradecer a la comunidad de desarrolladores e investigadores detrás de la Detección de Acciones y otros campos relacionados, principalmente a los equipos encargados de *Epic Kitchens*, *ActionFormer* y *GluonCV*.

Resumen

La reciente evolución de las tecnologías de aprendizaje automático ha supuesto un rápido avance en una gran cantidad de campos, desde el procesado de imágenes que nos permite reconocer los objetos presentes en una fotografía, hasta el procesado del lenguaje natural, utilizado para tareas tan diversas como el análisis de sentimiento o la creación de modelos capaces de mantener una conversación. Por otro lado, las técnicas dedicadas al tratamiento de vídeos suelen seguir a una distancia prudente estos avances, marcada por las limitaciones de *Hardware*, debido a la complejidad añadida por la dimensión temporal, junto al mayor espacio en memoria requerido.

Este trabajo se centrará en la tarea de la Detección de Acciones, que nos permite reconocer las acciones presentes en su vídeo, así como determinar cuando empiezan y cuando acaban, siendo una tarea de alta dificultad que requerirá tanto la segmentación temporal como la clasificación. Esta técnica se utilizará como parte de un proyecto de investigación impulsado por *BSH* y el *Graphics and Imaging Lab*, que pretende evaluar la integración de las nuevas tecnologías en la cocina. Como caso de estudio utilizaremos un conjunto de datos proporcionado por *BSH*, en el cual se recogen vídeos del proceso de cocinado, grabados por distintos participantes en sus hogares, utilizando una cámara sujeta al extractor. El proyecto tendrá como objetivo realizar una investigación exhaustiva del estado del arte, eligiendo un modelo de los analizados para su uso con nuestro *dataset*. Por su similitud a nuestro caso, dicho análisis se centrará en torno al *benchmark* de *Epic Kitchens*, compuesto por una serie de grabaciones de cocina obtenidas con perspectiva egocéntrica.

Tras la investigación, el elegido fue *ActionFormer*, basado en la arquitectura *Transformer* y en el uso de mecanismos de atención. Una vez instalado y verificado su funcionamiento con *Epic Kitchens*, realizamos la adaptación necesaria de nuestros vídeos y anotaciones para su compatibilidad, proceso que incluyó la extracción de *features* utilizando el modelo de Reconocimiento de Acciones *Slowfast*, dicho proceso será detallado más adelante y consiste en la generación de una representación vectorial de los vídeos utilizados como entrada. Después de un análisis en profundidad de los resultados obtenidos, concluimos que nuestro *dataset* presentaba una serie de problemas, principalmente un desbalance entre las distintas clases, los cuales se intentaron mitigar a través de un proceso de reetiquetado del mismo.

Índice

1. Introducción

Como parte de un proyecto de integración de las nuevas tecnologías en la cocina, la empresa *BSH* ha propuesto una serie de líneas de investigación, en colaboración con el *Graphics and Imaging Lab*, que pretenden facilitar al usuario final el proceso de cocinado. Estas se centran principalmente en la explotación de técnicas de Inteligencia Artificial, la creación de nuevas interfaces, y la utilización de la Realidad Virtual. Una de ellas, se dedica a la evaluación de la viabilidad de un sistema de asistencia en la preparación de recetas. A partir de la lista de ingredientes y pasos a seguir, el ayudante automático debería ser capaz de conocer las acciones realizadas por el cocinero, de manera que pueda sugerir los próximos pasos a tomar, así como asistirlo o corregirlo durante el proceso. Tras varios trabajos dentro del grupo relacionados con el Reconocimiento de Acciones y la Detección de Objetos, se ha propuesto continuar la investigación explorando la posible utilidad de la Detección de Acciones para la tarea propuesta. Cabe aclarar que el proyecto no se centrará en el desarrollo de un asistente o su interfaz.

La Detección de Acciones, también conocida como *Action Detection* o *Temporal Action Localization* (TAL), es una disciplina basada en el Aprendizaje Profundo que a partir de una secuencia de vídeo permite conocer las acciones realizadas y sus respectivos instantes de comienzo y finalización. Destacar que en ocasiones el término Detección de Acciones se utiliza para describir otro tipo de técnicas, en las que se busca la clasificación y detección espacial de acciones en un fragmento de la grabación. Dicho campo no será estudiado a lo largo del proyecto.

Esta materia, originada en la Visión por Computador, trata de mejorar las capacidades del entendimiento automático de vídeo, necesidad que ha crecido en relevancia debido al incremento exponencial del contenido visual disponible en la red. El principal interés de la misma reside en el análisis automático de cámaras de videovigilancia, grabaciones deportivas y la división automática de vídeos, además de la ya comentada asistencia en tareas como la cocina. Su aparición surge como una generalización del Reconocimiento de Acciones, que necesita una segmentación previa del metraje para poder realizar la clasificación. El modelo utilizado deberá realizar, por tanto, la detección y la clasificación tras un proceso de entrenamiento completamente supervisado. Dado un vídeo, tratará de encontrar los intervalos temporales con mayor probabilidad de contener una acción. Al mismo tiempo, deberá ser capaz de reconocer su clase.

A lo largo del trabajo, se van a analizar distintas técnicas de Aprendizaje Profundo aplicadas a la tarea de la Detección de Acciones, estudiando el estado del arte y su evolución en los últimos años, buscando aquella que mejor se adapte a un pequeño *dataset* proporcionado por *BSH*.

1.1. Contexto del Proyecto

Para el desarrollo del proyecto, disponemos de un conjunto de vídeos proporcionados por *BSH*, que fueron grabados colocando una cámara en la extractora de la cocina, de manera que se enfocara la placa y parte de la encimera, permitiendo recoger la mayoría del proceso de preparación de un plato. En la Figura 1 podemos ver un fotograma de ejemplo del *dataset*.

De los 200 vídeos disponibles, 91 de ellos ya han sido anotados, indicando para cada acción sucedida los instantes de comienzo y finalización, así como las clases del verbo y el nombre correspondientes. Por ejemplo, si se removiese el agua en una olla, se anotaría utilizando su verbo 'remover' y su nombre 'agua', mientras que si se removiera otro ingrediente, como pasta, la etiqueta del verbo sería la misma y la del nombre pasaría a ser 'pasta'. De esta forma, aunque el conjunto de datos tenga un alto número de acciones e ingredientes distintos, evitamos una explosión combinatoria de clases posibles. Dicho etiquetado se realizó como parte de otro trabajo

Figura 1: Fotograma perteneciente a uno de los vídeos de cocina disponibles

anterior en el laboratorio, centrado en el Reconocimiento de Acciones y la Detección de Objetos, llevado a cabo por Alejandro López[1].

Este sistema de anotaciones usado es el propuesto por el *dataset Epic Kitchens*[2] (extensión de su versión original[3]), que recoge un gran número de vídeos de cocina, y que debido a su similitud con los nuestros será utilizado como principal referencia para el desarrollo del proyecto, centrándose en él el estudio del estado del arte. Pese a ello, no se han ignorado otros *benchmarks*, que aun teniendo características más alejadas al nuestro, también han sido objeto de grandes avances dentro de la Detección de Acciones, siendo su análisis imprescindible para comprender la evolución de dicha disciplina.

A diferencia de la aplicación de la Detección de Acciones a otros ámbitos, su uso para los vídeos de cocina puede presentar un mayor número de dificultades, causadas habitualmente por la variabilidad de los ingredientes e instrumentos utilizados, así como de las recetas realizadas, además de una gran densidad temporal de acciones. Todo esto junto al sistema de etiquetado compuesto, que habitualmente requerirá del entrenado de un modelo específico para la detección de verbos y otro para la detección de nombres, resultará en valores de precisión considerablemente menores que en otros *benchmarks* más sencillos.

1.2. Objetivos del Proyecto

Como hemos comentado, a lo largo del proyecto nuestra meta principal será analizar la viabilidad del uso de la Detección de Acciones, aplicada a vídeos de cocina, para la creación de un asistente virtual a largo plazo. Para ello, el proyecto se ha dividido en los siguientes objetivos:

- **Estudio del Estado del Arte**: contextualización del estado de la disciplina, así como análisis de su evolución y de otras técnicas de Aprendizaje Profundo relacionadas o que tengan influencia sobre ella. El estudio se centrará principalmente en *Epic Kitchens* por su similitud a nuestros vídeos, pero también se tendrán en cuenta otros *datasets* relevantes para el campo. Además, como resultado de esta investigación, se elegirá uno de los modelos analizados para su examen con nuestro conjunto de datos. El modelo elegido fue *ActionFormer*[4], cuya implementación será detallada en la Sección 3.

- **Toma de contacto con *ActionFormer***: una vez elegido el modelo, se procederá al estudio en profundidad de su arquitectura, así como al entrenamiento

y la evaluación con *Epic Kitchens*. Esto nos permitirá instalarlo y configurarlo siguiendo los pasos planteados en la publicación original, pudiendo verificar así el correcto funcionamiento del mismo.

- **Entrenamiento y Evaluación con nuestros datos**: tras la familiarización inicial realizaremos el mismo entrenamiento y evaluación, utilizando en este caso nuestro propio conjunto de vídeos. Para ello será necesario un preprocesado de los mismos, adaptando el formato de nuestras anotaciones para ser compatibles con el modelo. Además, deberemos realizar una extracción de las *features* de nuestro *dataset*, de manera que obtendremos una representación vectorial de cada fragmento de vídeo utilizado, que represente la información relevante del mismo. Dichas *features* o características se utilizarán como datos de entrada.

- **Análisis de los resultados**: una vez entrenado el modelo, se realizará un estudio exhaustivo de los resultados obtenidos, comparándolos con los conseguidos al utilizar *Epic Kitchens* y realizando las modificaciones necesarias a los datos o la configuración con el objetivo de optimizar el rendimiento.

1.3. Herramientas utilizadas y Planificación

1.3.1. Herramientas utilizadas

Al igual que buena parte del material relacionado con el *Deep Learning*, el lenguaje de programación elegido para el desarrollo del trabajo y utilizado por los modelos usados ha sido *Python*. Para su edición se han empleado principalmente los *IDE PyCharm* y *Visual Studio Code*. Cabe destacar el uso de la librería *PyTorch*, que proporciona una interfaz para el desarrollo de modelos de Aprendizaje Automático, facilitando enormemente su desarrollo. Además, fue necesaria la familiarización con la plataforma *CUDA*, desarrollada por *NVidia*, que permite la ejecución de código en *GPU*, acelerando considerablemente el proceso de entrenamiento e inferencia de las redes.

Las tarjetas gráficas usadas fueron proporcionadas por el *Graphics and Imaging Lab* y se trabajó con ellas a través de acceso remoto a dos equipos de altas prestaciones. Los modelos empleados fueron una *NVidia Quadro P5000* de 16 GB para los procesos de entrenamiento e inferencia del modelo, y una *NVidia GeForce GTX TITAN X* de 12 GB para la extracción de *features*. Para dicha extracción, utilizamos *SlowFast*[5] a través de la herramienta *GluonCV*[6]. Finalmente, el trabajo en remoto se realizó utilizando tanto el *IDE Visual Studio Code* como la herramienta *MobaXTerm*.

1.3.2. Planificación

En cuanto a la planificación del proyecto, se ha realizado como continuación de unas prácticas en el *Graphics and Imaging Lab*, dedicadas principalmente al estudio del estado del arte, centrándonos en el *dataset* de *Epic Kitchens*. Durante este período, se analizó la evolución de la Detección de Acciones, así como los modelos que actualmente consiguen un mayor rendimiento. Además, se estudiaron los *benchmarks* más relevantes en el campo, así como otras técnicas relacionadas. Esta parte del proyecto concluyó con la elección de un modelo para su uso con nuestro conjunto de datos, *ActionFormer*[4], así como su instalación y la familiarización con el mismo.

Una vez instalado y configurado, pasamos a la adaptación de nuestro *dataset*, lo cual requirió una modificación del formato de las anotaciones, y realizar la extracción de *features*. Tras obtener un rendimiento inicial muy alejado de lo esperado, fue necesario un proceso de reconfiguración y búsqueda de errores, temporalmente costoso debido a los elevados tiempos requeridos para el entrenamiento del modelo y la extracción. Una vez obtenidos unos resultados con nuestros vídeos equiparables a aquellos conseguidos con *Epic Kitchens*, pasamos a realizar un análisis exhaustivo de

los mismos, buscando las características del conjunto de datos que pudieran afectar negativamente al funcionamiento.

De esta manera, vimos como el *dataset* sufría de un problema de desbalance de clases, un pequeño conjunto de etiquetas englobaba la mayor parte de las instancias, mientras que el resto de clases, más específicas pero menos comunes, aparecían muy puntualmente de manera que el modelo no era capaz de detectarlas correctamente. Este problema, junto a la falta de acciones anotadas o errores en algunas de las existentes, propició la realización de un reetiquetado completo de los vídeos, utilizando en todo momento anotaciones más precisas. Así, conseguimos un considerable aumento del rendimiento del modelo de detección de verbos, mientras que la precisión conseguida para los nombres descendió. Tras un estudio en profundidad concluimos que dicho descenso se debía en gran medida a problemas en la grabación del conjunto de datos, principalmente de iluminación, que dificultaban la diferenciación de los distintos ingredientes utilizados.

Una vez finalizado el estudio de los resultados, comenzamos con la redacción de la memoria, que se extendió hasta la finalización del proyecto. Cabe destacar que debido a los altos tiempos de espera necesarios al utilizar el modelo, el análisis del estado del arte se prolongó a lo largo del trabajo, con el objetivo de ocupar estas esperas. Durante el período de escritura, realizamos también una serie de experimentos (modificando la arquitectura utilizada o las anotaciones de los vídeos) cuyos resultados no fueron tan buenos como los deseados, pero que exponen algunas ideas que podrían ser desarrolladas a futuro (Anexo D). En la Figura 2 podemos ver un cronograma del tiempo empleado para el desarrollo del proyecto, que aproximadamente supuso una inversión de 360 horas.

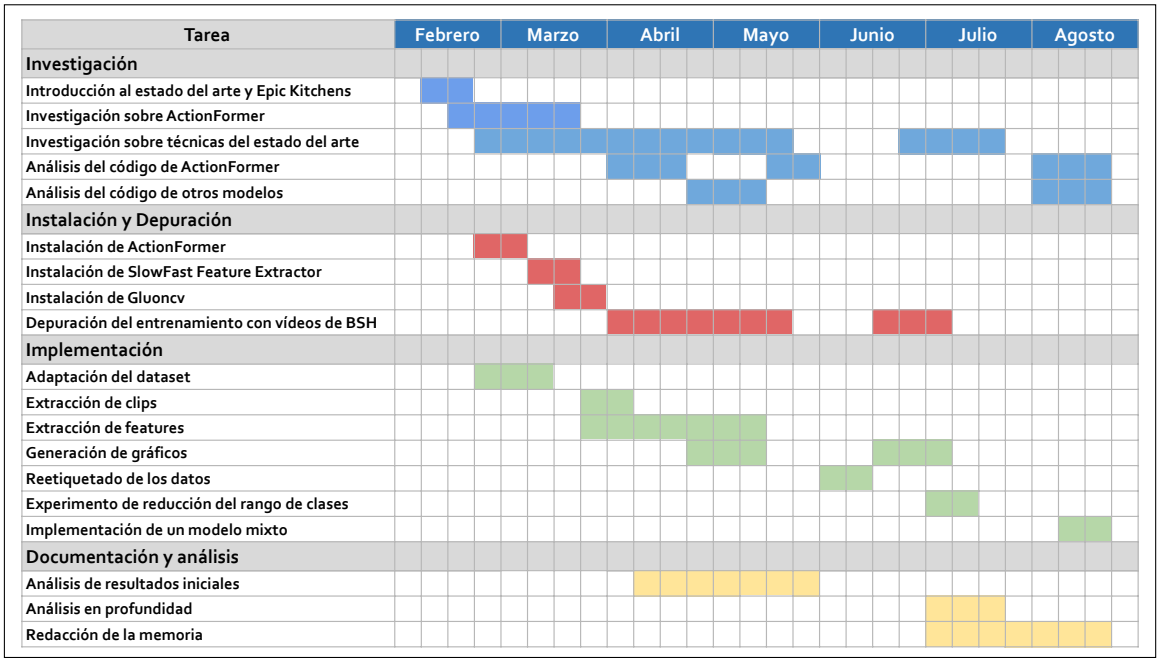

Figura 2: Cronograma con la organización temporal del trabajo realizado para el proyecto.

2. Estudio del Estado del Arte

Con el objetivo de encontrar un modelo para su uso con el *dataset* de *BSH*, realizaremos un análisis en profundidad del estado del arte de la Detección de Acciones, centrándonos en su evolución, los principales conjuntos de datos, y justificando finalmente la elección de una arquitectura concreta. Como hemos comentado anteriormente, este estudio se centrará en torno a *Epic Kitchens*[2] debido a su similitud con nuestros vídeos.

2.1. Evolución del Estado del Arte

En primer lugar, vamos a analizar la evolución reciente del estado del arte de la Detección de Acciones. Para ello, hemos consultado dos publicaciones, [7] y [8], que realizan una investigación exhaustiva de las arquitecturas y *benchmarks* utilizados en 2016 y 2021, respectivamente. Comentaremos los campos relacionados, así como técnicas previas al uso del *Deep Learning*, las actuales y las posibles tendencias futuras.

2.1.1. Trabajo relacionado

La Detección de Acciones está estrechamente relacionada con el Reconocimiento de Acciones, que consiste en la clasificación de segmentos temporales previamente definidos, por lo que la Detección se podría considerar una generalización del mismo. Habitualmente, se utilizarán reconocedores de acciones para la extracción de *features* (proceso comentado en la Sección 3.2), como *SlowFast*[5], *ViVit*[9] o *I3D*[10]. El campo está también influenciado por la Detección de Objetos, que inspiró las técnicas basadas en *anchors*, así como por el Procesado del Lenguaje Natural (*NLP*), del cual se han adaptado mecanismos como la auto-atención. Ambas aproximaciones se detallarán en la Sección 2.1.3. Finalmente, podemos destacar el *Weakly Supervised Temporal Action Localization* o *WTAL*, muy similar a la Detección de Acciones, en el que el modelo conoce las acciones que aparecerán en un vídeo en tiempo de inferencia, detectando únicamente instancias con esas etiquetas e ignorando el resto, por lo que podría ser útil en problemas en los que las clases que se verán en el vídeo se conocen de antemano.

2.1.2. Técnicas anteriores al *Deep Learning*

Como muchos otros campos ahora dominados por el Aprendizaje Profundo, la Detección de Acciones surgió como una línea de investigación de la Visión por Computador tradicional. Al igual que en las técnicas basadas en Inteligencia Artificial, los sistemas debían extraer en primer lugar información semántica de los vídeos a procesar, utilizando *features* o representaciones diseñadas manualmente, como *SIFT*[11] o *HOG*[12], centradas en el cálculo de gradientes, o el *Optical Flow*, que calcula el movimiento producido entre fotogramas consecutivos. Finalmente, se aplicaban módulos de clasificación, ya fueran deterministas, como aquellos basados en *K-Nearest Neighbours* o *Support Vector Machines* (*SVM*)[13], o probabilísticos, que habitualmente utilizaban redes de Bayes.

Tras el desarrollo y la popularización de las redes neuronales convolucionales (CNN) y su extensivo uso para el procesado de imágenes y por extensión de vídeo, las técnicas de detección tradicionales fueron sustituidas rápidamente por el *Deep Learning*, que no requería el diseño de un algoritmo específico para la extracción de *features* ni para la clasificación.

2.1.3. Técnicas basadas en *Deep Learning*

Los modelos de Detección de Acciones actuales se dividen principalmente en dos categorías. La primera de ellas se compone por aquellos centrados en la generación de propuestas (*proposal generation*), es decir, posibles intervalos temporales que contengan acciones, y la clasificación de estos mismos utilizando técnicas propias del Reconocimiento de Acciones (requiriendo en algunos casos el entrenamiento de dos redes independientes). Posteriormente, aparecen arquitecturas basadas en el modelado del contexto temporal a largo plazo, de manera que el modelo sea capaz de aprender una representación de las relaciones existentes entre los distintos instantes del vídeo de entrada. Estas últimas son las que actualmente lideran el estado del arte. Ambas aproximaciones serán estudiadas con más detalle a continuación.

Dentro de los modelos basados en la generación de propuestas, podemos observar las siguientes subcategorías:

- **Anchor-based**. El modelo generará inicialmente una serie de intervalos temporales de referencia (*anchors* o anclas), con longitudes determinadas y distribuidos regularmente a lo largo de todo el vídeo. Posteriormente, se aplicará un refinado de las propuestas iniciales, de manera que se ajusten todo lo posible a las instancias reales. Esta metodología se conoce también como *top-bottom* y deriva directamente de las *Anchor Boxes* utilizadas por algunos algoritmos de Detección de Objetos en imágenes, que generarán una serie de *Bounding Boxes* iniciales que se van ajustando a los objetos presentes. Un ejemplo de arquitectura basada en el uso de *anchors* es *SCNN*[14].

- **Anchor-free**. Al contrario que los anteriores, los intervalos generados no serán obtenidos a partir de *anchors* predefinidos, mejorando en gran manera la flexibilidad. Para ello, emplean una aproximación *bottom-up*, con la que calcularán la probabilidad de la aparición de acciones (*actioness score*) y sus límites temporales (*startness* y *endness score*) en cada instante. Dicha información se utilizará para la creación de las propuestas, que serán posteriormente clasificadas. Entre estos modelos, cabe destacar *BMN*[15], que se suele utilizar para la generación de intervalos en arquitecturas basadas en el *ensemble learning*.

- **Modelos mixtos**. Algunas aproximaciones, como *PRN*[16], utilizan una implementación mixta generando intervalos utilizando dos ramas, una basada en *anchors* y otra sin su uso, para combinar los resultados de ambas. De esta manera, pueden aprovechar la robustez ofrecida por los modelos sin anclas, así como la facilidad de los que las usan para detectar intervalos de longitudes esperadas.

A continuación, comentaremos algunas de las técnicas utilizadas en las arquitecturas centradas en la representación del contexto temporal a largo plazo:

- **Redes Neuronales Recurrentes (RNN)**. Modelos utilizados para el procesado de secuencias, que disponen de una conexión recurrente, permitiendo transmitir la información aprendida al procesar la entrada. Esta unión, conocida como *estado oculto*, nos permitirá tener en cuenta el contexto generado por los valores previos al procesar uno nuevo, permitiéndonos modelar relaciones entre los distintos instantes temporales de un vídeo. Pese a ello, debido a problemas de desvanecimiento del gradiente al procesar un gran número de elementos, así como la dificultad de paralelización, no son las arquitecturas más adecuadas para el procesado de vídeos. Un ejemplo de detector de acciones que utiliza este tipo de métodos es *SS-TAD*[17].

- **Redes basadas en grafos**. También llamadas *Graph Convolutional Networks* (*GCN*), permiten la adaptación de técnicas de aprendizaje a datos estructurados en forma de grafo, sin necesidad de secuencializar la entrada. Para ello, se aplicarán convoluciones sobre la matriz de adyacencia, de manera que podremos modelar automáticamente información sobre su estructura y la relaciones entre nodos. Aplicado a la tarea de Detección de Acciones, cada instante se

representará con un vértice, y sus relaciones a través de las aristas. Algunos modelos que utilizan esta aproximación son *G-TAD*[18] y ACGNet[19].

- **Transformers**. Surgidos del campo del procesado del lenguaje natural (*NLP*), utilizan un mecanismo conocido como auto-atención o *self-attention* (detallado en el Anexo B) para modelar las relaciones entre las distintas partes de una secuencia de entrada. A cambio de una mayor complejidad, y por tanto, mayores tiempos de ejecución y entrenamiento, estas arquitecturas son capaces de modelar relaciones independientemente de la distancia temporal entre entradas. Debido a esto, durante los últimos años han ganado una gran popularidad en las técnicas de tratamiento de vídeo. Algunos ejemplos de arquitecturas basadas en *Transformers* o atención son *ActionFormer*[4] o *BasicTAD*[20].

2.1.4. Tendencias futuras

Recientemente, algunos estudios, entre los que destacan *MetaFormer*[21] y [22], proponen que el uso de la auto-atención en tareas centradas en el procesado de vídeo puede ser perjudicial, ya que pueden provocar pérdidas de información al ser propensas a uniformizar las *features* de entrada. Además, añade un coste computacional elevado. La hipótesis principal sugiere que el buen funcionamiento de estas técnicas no depende del uso de *self-attention*, sino en la estructura general de las arquitecturas basadas en *Transformers*. A raíz de estas publicaciones, surgen modelos como *TriDet*[23] o *TemporalMaxer*[24], que utilizarán *ActionFormer*[4] como base, sustituyendo el uso de auto-atención por operaciones convolucionales más eficientes para el modelado de las relaciones temporales, que permiten preservar las diferencias entre *features*. Como podemos ver en la Tabla 1, ambos obtendrán mejores resultados en el *benchmark* de *Epic Kitchens*[2]. Como medida de rendimiento se utiliza el *mean Average Precision* (mAP), que da una mayor importancia a la precisión de las predicciones realizadas con un alto valor de confianza. Su uso y el proceso de cálculo de la precisión en tareas de Detección de Acciones, se detalla en el Anexo A.

Modelo	mAP medio Verbos	mAP medio Nombres
TriDet	25,4 %	23,8 %
TemporalMaxer	24,5 %	22,8 %
ActionFormer	23,5 %	21,9 %

Tabla 1: Tabla comparativa del rendimiento de *ActionFormer*, *TriDet* y *TemporalMaxer* en el *benchmark* de *Epic Kitchens*. *TriDet* es el modelo que mejores resultados obtiene, seguido de *TemporalMaxer* y finalmente de *ActionFormer*.

Además de estas nuevas aproximaciones, cabe destacar la aparición de distintos modelos fundacionales o *foundation models* para tareas centradas en vídeo, que serán entrenados de manera no supervisada sobre un gran conjunto de datos, pudiendo ser adaptados posteriormente a trabajos más específicos, conocidos como *downstream tasks*. Pese a no estar tan desarrollados como en otros campos (imágenes, lenguaje natural, ...) debido al alto uso de memoria requerido, debemos ser conscientes de su existencia, ya que a medida que avance la tecnología podrían tener una gran influencia en el procesado automático de vídeos. Algunos de ellos, como *VideoMAE V2*[25] e *InternVideo*[26], han conseguido liderar *benchmarks* de Detección de Acciones como *THUMOS'14*[27] o *HACS*[28].

2.2. Conjuntos de Datos

El principal conjunto de datos estudiado durante el análisis fue *Epic Kitchens*[2] compuesto por vídeos de cocina grabados en perspectiva egocéntrica, utilizando una

cámara colocada en la cabeza de los participantes. En total, está formado por aproximadamente 100 horas de grabación, tomadas en 45 cocinas diferentes. Consta de 90000 acciones anotadas de forma compuesta, con 97 clases distintas para los verbos y 300 para los nombres. Se caracteriza por su alta densidad temporal de anotaciones, con una media de 121 por vídeo.

En comparación, nuestro *dataset* recoge un total de 200 vídeos de cocina, de los cuales únicamente 91 han sido anotados, con un total de 2700 instancias. Estos fueron grabados en 3 cocinas distintas. El tiempo medio de cada vídeo es de 18 minutos y tiene una alta densidad de anotaciones (30 por vídeo como valor medio), aunque no tan elevada como *Epic Kitchens*. Además, se ha utilizado su misma metodología para el etiquetado, es decir, la clase de la acción estará compuesta tanto por verbo como por nombre. Pese a disponer únicamente de 11 y 36 clases de verbos y nombres respectivamente, se utilizarán las mismas definidas por *Epic Kitchens* para facilitar el uso de nuestros vídeos. A diferencia de este, las grabaciones se han realizado utilizando una cámara sujetada en la extractora, por lo que en ocasiones las acciones realizadas por los participantes pueden aparecer parcialmente fuera de la imagen, o no ser visibles.

Como hemos visto, nuestro conjunto de datos podría considerarse una simplificación del problema propuesto por *Epic Kitchens*, ya que utiliza su mismo sistema de etiquetado, un número mucho más reducido de clases, una cámara estática en lugar de móvil y tiene una densidad temporal considerablemente menor, respaldando nuestra elección de dicho *dataset* como referencia para el proyecto. Por tanto, un modelo que funcione correctamente con *Epic Kitchens* debería ser capaz de obtener resultados mejores con nuestro conjunto de datos.

Otros *benchmarks* muy relevantes dentro de la Detección de Acciones son *THUMOS'14* [27], *ActivityNet*[29] y *HACS Segments*[28], compuestos por distintos vídeos extraídos de *YouTube*. A diferencia de *Epic Kitchens* y nuestros vídeos, no estarán anotados de manera compuesta, y tendrán una densidad temporal mucho menor, lo cual suele facilitar las tareas de detección, resultando en valores de precisión más elevados. Pese a sus diferencias, limitar el análisis a *Epic Kitchens* habría reducido en gran medida nuestra comprensión general del estado del arte. Como vemos en la Tabla 2, los *datasets* más sencillos, ya sea en cuanto a densidad de acciones o por número de clases, serán aquellos que mejores puntuaciones obtengan. Por otro lado, para un *benchmark* como *Epic Kitchens*, con una gran cantidad de etiquetas y frecuencia de anotaciones, la tarea será más complicada.

Dataset	**Número de Vídeos**	**Número de Clases**	**Duración Media**	**Anotaciones por Vídeo**	**Estado del Arte (mAP %)**
THUMOS'14	413	20	212s	15.5	InternVideo[26] (71.58 %)
ActivityNet-1.3	19994	200	115s	1.54	PRN+BMN[16] (42. %)
HACS Segments	50000	200	156s	2.8	InternVideo[26] (41.55 %)
Epic Kitchens	633	97 + 293	500s	121.46	TriDet[23] (25.4 %)
BSH	91	11 + 36	1092s	29.85	X

Tabla 2: Comparación de los *datasets* presentados, en la que podemos ver el número de vídeos de cada uno, su duración media, el número medio de acciones por vídeo y el número de clases distintas en los conjuntos de datos. Además, se indica el modelo y el *mean Average Precision* (*mAP*) que lidera actualmente el estado del arte. La información sobre *THUMOS*, *ActivityNet* y *HACS* se ha obtenido de [8].

2.3. Elección de un modelo

Tras un exhaustivo estudio del estado del arte, el modelo elegido para el proyecto fue *ActionFormer*[4], el cual consiguió el segundo puesto en la competición de Detección de Acciones organizada por *Epic Kitchens*[2] para el año 2022. En di-

cha competición, *ActionFormer* fue la implementación de código abierto que mejor puntuación obtuvo. Otra razón de peso fue la adaptación existente a *Epic Kitchens*, con características muy similares a nuestro conjunto de vídeos, lo cual facilitaría su uso con respecto a modelos diseñados para otros *datasets*. Además, para realizar el modelado del contexto temporal a largo plazo utilizará una arquitectura inspirada por los *Transformers*, de la que podemos destacar el uso de *self-attention*. Debido a la relevancia actual de estas técnicas, utilizada en arquitecturas como *BERT* [30] o *GPT*[31], otro de los motivos de la elección fue el interés en el estudio de las mismas.

Esta decisión se tomó antes de la publicación de *TemporalMaxer* y *TriDet*, presentados en la Sección 2.1.4, por lo que no se pudieron tener en cuenta. Debido a esto, de manera experimental y con el objetivo de trabajar con uno de los nuevos modelos, se implementó una arquitectura híbrida que utilizaba una rama compuesta por *TemporalMaxer*, para preservar en medida de lo posible las diferencias entre *features* y una rama que empleaba auto-atención, con el objetivo de modelar con mayor precisión las relaciones temporales a largo plazo. Los detalles de dicha implementación se encuentran en el Anexo D.2.

3. Arquitectura de *ActionFormer*

Como hemos comentado en la Sección 2.3, *ActionFormer*[4] es un modelo de detección de acciones basado en la representación del contexto temporal. Es decir, a partir de un vídeo de entrada deberá ser capaz de aprender las relaciones entre los distintos instantes. De esta manera, podrá tener en cuenta vínculos globales, permitiendo una predicción de acciones más precisa. Para ello, utilizará una arquitectura basada en los *Transformer* y el uso de *self-attention*, de forma que se calculará una media ponderada entre las *features* de cada *clip*, utilizando como peso una medida de similitud entre ellos.

ActionFormer seguirá la estructura observada en la Figura 3, compuesta principalmente por:

- **Extractor de *features***: se empleará un reconocedor de acciones preentrenado para obtener una representación vectorial con alta información semántica de cada uno de los fragmentos de vídeo a procesar. Dichos fragmentos o *clips* se extraerán periódicamente, con una separación determinada por un número de *frames* conocido como *temporal stride*, existiendo un cierto solape entre ellos. Así, no será necesario cargar en memoria los vídeos a procesar, lo que supondría un coste muy elevado.

- **Módulo de modelado del contexto temporal**: también conocido como codificador, será el encargado de aprender la relación entre los distintos instantes del vídeo, utilizando el vector de *features* generado para cada uno de ellos. Para ello, se utilizarán bloques de auto-atención, que calcularán la similitud entre los *clips*.

- **Pirámide de *features* temporal**, que permite una representación de la secuencia de entrada en distintas resoluciones temporales, facilitando la Detección de Acciones global. Tras cada nivel de la pirámide, se aplicará el bloque de auto-atención junto a una operación de *down-sampling*.

- **Decodificador**: módulo compuesto por dos redes convolucionales diferentes, que utilizarán como entrada la pirámide de *features* completa. La primera de ellas (clasificación) estará encargada de determinar la clase de acción más probable en cada instante temporal. Finalmente, la segunda red (detección) deberá realizar una regresión para buscar el inicio y el final de la acción en caso de que existiera una en ese momento. Tras este proceso, se realizará un refinado de los intervalos generados para evitar posibles redundancias.

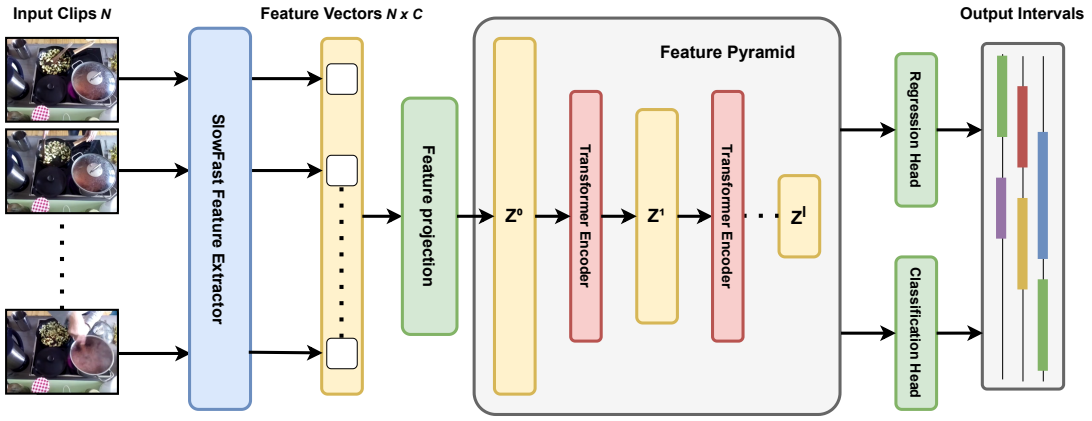

Figura 3: Diagrama de alto nivel de la estructura de *ActionFormer*. El modelo estará compuesto por (1) un reconocedor de acciones preentrenado (*SlowFast*), para la extracción de *features* de los fragmentos del vídeo de entrada, (2) un módulo para el modelado del contexto temporal, compuesto por bloques de *self-attention* y una *pirámide de features temporal* y (3) dos bloques convolucionales para la regresión y clasificación de intervalos. Figura adaptada de [4].

A lo largo de esta sección comentaremos en más detalle la arquitectura de los distintos módulos del modelo, tras presentar la formulación del problema de Detección de Acciones utilizado por *ActionFormer*. Finalmente, veremos brevemente la metodología de uso seguida para entrenarlo con nuestros datos.

3.1. Formulación del problema

Para procesar los vídeos utilizados necesitaremos realizar una fragmentación periódica de los mismos, de manera que exista un cierto solape entre *clips*. Estos serán procesados con nuestro extractor de *features*, obteniendo un vector de *features* para cada uno de ellos utilizados como entrada para el modelo. De esta forma, cada vídeo será una secuencia $X = \{x_1, x_2, ..., x_T\}$, en la que cada x_i es el vector de *features* obtenido del fragmento i, con $i \in \{1..T\}$, siendo T el número total de *clips*. De esta manera, la entrada tendrá una dimensión $X \in \mathbb{R}^{T \times F}$, siendo F la dimensión de las secuencias de *features*, con $F = 2304$ al utilizar *SlowFast*.

El objetivo de *ActionFormer* será predecir los intervalos de las acciones existentes en el vídeo X, así como la clase de cada uno de ellos. De esta manera, la salida será un conjunto $Y = \{y_1, y_2, ..., y_N\}$ de intervalos, con y_i representando la i-esíma instancia predicha, de un total de N. Cada uno de ellos estará compuesto por $y_i = (s_i, e_i, a_i)$, siendo $s_i \in [1, T]$ el instante inicial, $e_i \in [1, T]$ el instante final, y $a_i \in [1, C]$ la clase de la acción (con C representando el número de clases distintas). Este problema se simplifica de manera que el resultado producido por el modelo sea un etiquetado de cada fragmento de vídeo $\hat{Y} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_T\}$, de manera que la etiqueta en el instante t esté compuesta por $\hat{y}_t = (p(a_t), d_t^s, d_t^e)$. $p(a_t)$ será un vector de dimensión $[0, 1]^C$, que indicará para cada una de las clases la probabilidad de que la posible acción en el instante t pertenezca a ella. Por otro lado, d_t^s y d_t^e representarán la distancia desde t, hasta el principio y el final del intervalo.

3.2. Extractor de *features*

Para la extracción de *features*, el modelo elegido fue *SlowFast*[5], un reconocedor de acciones basado en la aplicación de convoluciones tridimensionales que tendrán en cuenta la dimensión temporal. Estará compuesto por dos *pathways* o caminos.

Uno de ellos, el *Slow pathway* procesará el vídeo de entrada utilizando una resolución temporal baja, mientras que el *Fast pathway* empleará una alta resolución. De esta manera, el *Fast pathway* será capaz de detectar movimientos rápidos, aunque tendrá una menor capacidad de procesado de la información espacial. Por otro lado, el *Slow Pathway* se encargará de capturar datos semánticos que se puedan obtener a partir de unas pocas imágenes, y que no cambie rápidamente.

Al juntar lo aprendido por ambos *Pathways*, conseguiremos un vector de *features* que representará tanto los movimientos producidos, como la estructura de la escena. Para cada uno de ellos, se definirá un valor de *temporal stride*, que indicará cuantos fotogramas del vídeo original se utilizan. El *Slow Pathway* habitualmente utilizará 16 *frames* de *stride*, procesando únicamente uno de cada 16 leídos. Por el contrario, el *Fast Pathway* usará normalmente un *stride de 2*, trabajando con la mitad de los fotogramas. En la Figura 4, observamos un diagrama de alto nivel de la arquitectura.

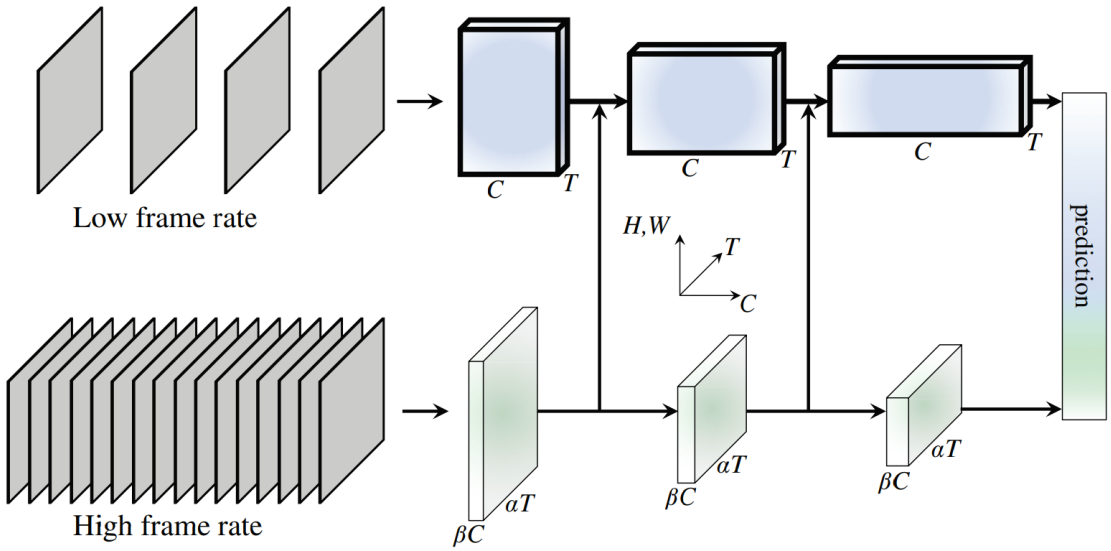

Figura 4: Diagrama en alto nivel de la arquitectura de *SlowFast*, en el que se muestran el *Slow Pathway* (arriba) y el *Fast Pathway* (abajo). Figura extraída de [5].

3.3. Codificador

Antes de introducir los vectores de *features* en el codificador, se realizará una proyección de los mismos, utilizando una capa convolucional seguida de una función de activación $ReLU$[32], bloque conocido como E. De esta manera, las secuencias de *features* se proyectarán sobre un espacio D-dimensional, $Z^0 = \{E(x_1), E(x_2), ..., E(x_T)\}$ con $E(x_i) \in \mathbb{R}^D$, utilizado como entrada del módulo del modelado del contexto temporal. Cabe destacar que las funciones de activación utilizadas se detallarán en el Anexo F. A continuación, la entrada será procesada por un bloque de auto-atención, que calcula una media ponderada entre los distintos vectores de *features*, utilizando como peso una medida de similitud. A $Z^0 \in \mathbb{R}^{T \times D}$, le aplicaremos una proyección utilizando una serie de matrices que el modelo irá aprendiendo, $W_Q \in \mathbb{R}^{D \times D_q}$, $W_K \in \mathbb{R}^{D \times D_q}$ y $W_V \in \mathbb{R}^{D \times D_v}$.

Gracias a ellas, obtendremos distintas representaciones de las *features*, $Q = Z^0 W_Q$, $K = Z^0 W_K$ y $V = Z^0 W_V$, conocidas como *query*, *key* y *value* respectivamente. Finalmente, obtendremos la salida $S \in \mathbb{R}^{T \times D_v}$, aplicando $S = Softmax(QK^T/\sqrt{D_q})\, V$. El funcionamiento de la atención y la auto-atención se detallarán en el Anexo B.

Además, se aplicará una técnica conocida como *Multi-Head Self-Attention* o *MSA*, que permite ejecutar paralelamente distintas operaciones de auto-atención. Con el objetivo de acelerar el entrenamiento, este cálculo no se realizará globalmente, sino que se utilizará una ventana de tamaño W. Procesar la auto-atención en la secuencia completa, tendrá un coste asintótico de $O(T^2D + TD^2)$ tanto en tiempo como en memoria, mientras que limitando su alcance, ese coste se reducirá a $O(W^2TD + TD^2)$. Habitualmente, D será un valor fijo, mientras que T dependerá de la longitud del vídeo a procesar, afectando considerablemente al rendimiento si es muy largo. Por otro lado, el tamaño de la ventana W será mucho menor que T.

Si seguimos esta metodología en conjunto con una pirámide de *features* temporal ($TFPN$) de L niveles, que aplique reducciones de la resolución temporal consecutivas, utilizando la misma ventana de tamaño W en sus distintos niveles, podremos modelar las relaciones contextuales globales sin incurrir en el alto coste computacional y espacial. Las Pirámides de Features (FPN) permiten la representación de la misma información a distintas escalas. Habitualmente, se utilizan en tareas como la Detección de Objetos, permitiendo que el modelo encuentre fácilmente objetos de tamaños variables al disponer de la imagen a procesar en distintas resoluciones, como vemos en la Figura 5. Por otro lado, en la Figura 6 podemos ver un ejemplo de la estructura del codificador, aplicando las técnicas de pirámides de *features* sobre la dimensión temporal del problema.

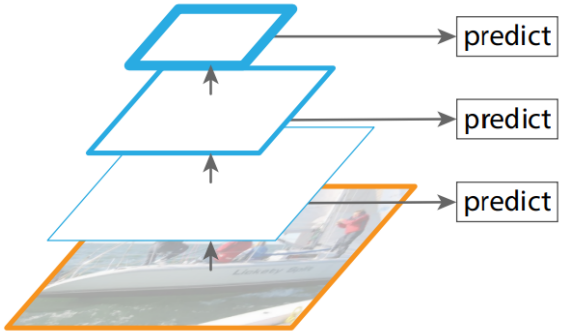

Figura 5: Diagrama de una pirámide de features aplicada a una tarea de reconocimiento de objetos. Figura extraída de [33]

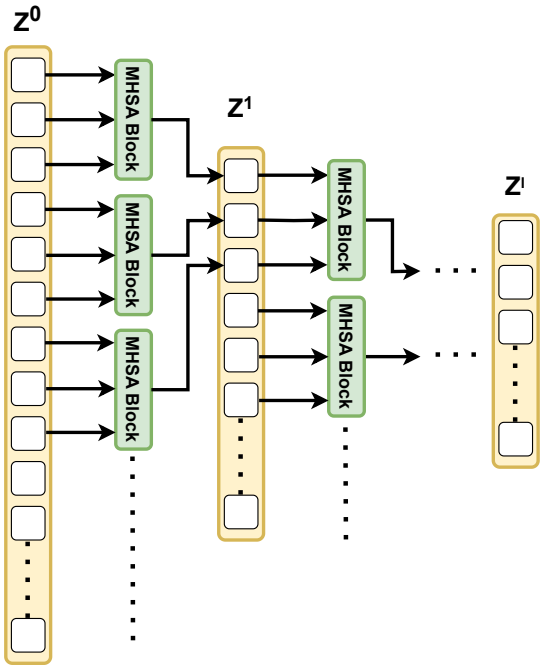

Figura 6: Ejemplo del codificador utilizando una pirámide de features temporal. En todos los niveles de la pirámide se aplicará el modelado del contexto a largo plazo (*Multi-Head Self-Attention* en este caso) con una ventana fija. Aprovechándonos de las distintas resoluciones temporales, utilizando la misma ventana se calcularán tanto relaciones a corto como a largo plazo.

En cada nivel de la pirámide se utilizará un bloque de *Transformer*, que tomará como entrada la salida del nivel anterior. En primer lugar, se aplicará una normalización de dicha entrada, utilizando un bloque de *Layer Norm*[34], que calcula la media y la varianza para normalizar la suma de las entradas a todas las neuronas de una misma capa. A continuación, se procesará la *auto-atención* con el método comentado anteriormente. A esa salida, se le unirá una conexión residual con la entrada inicial del bloque, que tras otra aplicación de *Layer Norm* será procesada por un *Multilayer Perceptron* (*MLP*). Tras otra conexión residual se realizará un *downsampling* temporal. En la Figura 7 podemos ver la estructura de cada bloque del codificador.

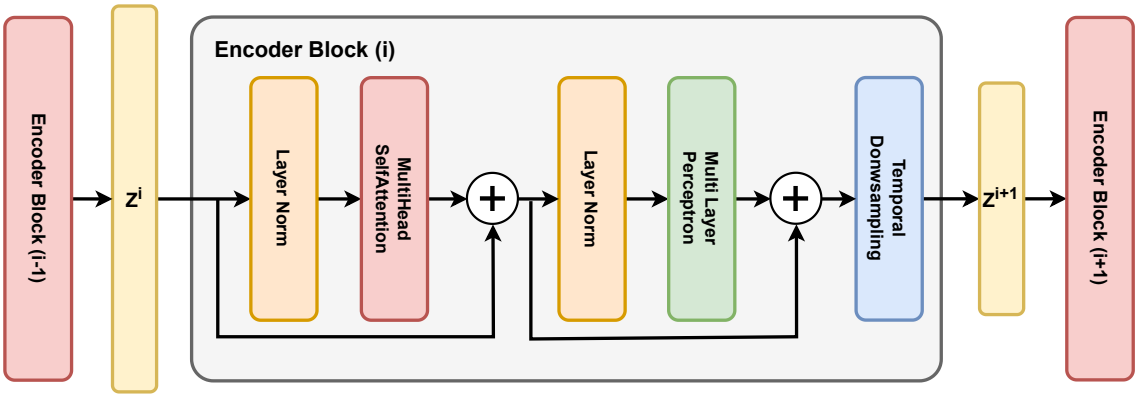

Figura 7: Diagrama de la estructura de un bloque del codificador utilizado por *ActionFormer*. Tomará como entrada la salida del bloque anterior, a la que aplicará una operación de *Layer Norm* y *MultiHead Self-Attention*. Al resultado, combinado con la entrada a través de una conexión residual, se le volverá a aplicar un *Layer Norm* y un perceptrón multicapa. Tras una última conexión residual se aplicará la reducción de resolución temporal. Figura adaptada de [4].

3.4. Decodificador

Finalmente, el modelo utilizará todos los niveles de la *TFPN* como entrada del decodificador, con la doble tarea de predecir para cada instante temporal t, la posibilidad de que pertenezca a una instancia de cualquiera de las C clases, así como la distancia al inicio (d_t^s) y final (d_t^e) de la acción, en caso de que existiera.

El bloque encargado de la clasificación procesará todo momento t en cada uno de los L niveles de la pirámide, compartiendo los mismos parámetros para todos. Aplicará 3 convoluciones unidimensionales, utilizando un *kernel* de tamaño 3, y un *stride* de 2. Además, se utilizarán capas de *Layer Norm* y una activación *ReLU* entre convoluciones. Finalmente, una función *sigmoide* se encargará de obtener una probabilidad para cada una de las C clases. A cada t se asignará la etiqueta que mayor valor obtenga, siendo dicho valor utilizado también como una puntuación de confianza de la acción. Esta se utilizará para ordenar las acciones según su relevancia a la hora de calcular la precisión del modelo.

Por otro lado, el módulo de regresión de intervalos seguirá una estructura similar, añadiendo una activación *ReLU* final, de manera que se pueda realizar una estimación de la distancia a los instantes de inicio y finalización, en caso de que t esté contenido en una acción. En la Figura 8 podemos observar un diagrama de la arquitectura del bloque de detección, aplicado a cada uno de los niveles de la pirámide.

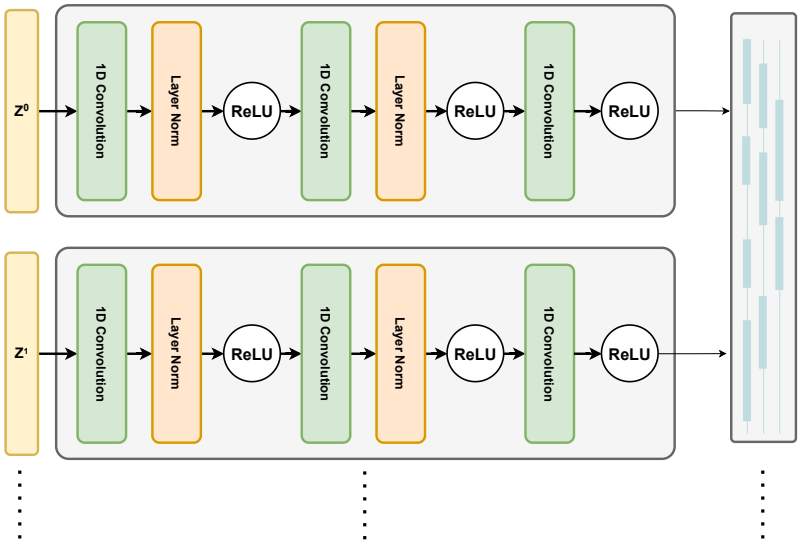

Figura 8: Bloque encargado de la regresión de los intervalos. Sobre cada instante temporal en cada nivel de la pirámide de features, calculará la distancia hasta el principio y final de la posible acción. Para ello utilizará varias capas, compuestas por una convolución unidimensional, *Layer Norm* y una función de activación *ReLU*. La red de clasificación mantiene una estructura similar, utilizando una función sigmoide para generar un valor de probabilidad para cada una de las clases en cada instante temporal.

Cabe destacar, que para cada nivel de la *TFPN* se definirá un *regression range*. Durante el entrenamiento, cada acción perteneciente al *ground-truth* se asignará a un único nivel según su longitud, de manera que esté contenida en el rango de regresión correspondiente. De esta manera, no existirá redundancia entre los distintos niveles de la pirámide. Tras el proceso de detección y clasificación, el modelo realizará un refinado de las instancias generadas, utilizando una técnica conocida como *Non Maximum Supression*[35] ampliamente empleada en tareas de Detección de Acciones. Este algoritmo permite la eliminación de intervalos con un gran solape entre ellos, manteniendo aquellos con una mayor puntuación y será detallado en el Anexo E.

3.5. Entrenamiento y función de pérdida

Durante el entrenamiento, el modelo utiliza *cosine annealing*[36], de manera que el *learning rate* irá variando de forma cíclica. Inicialmente, se definirá un valor elevado que irá disminuyendo rápidamente, de forma que se fuerce la convergencia a un mínimo local tras pocas iteraciones. Una vez los valores se hayan estabilizado, la tasa de aprendizaje volverá a incrementar, provocando la salida del mínimo y su rápida vuelta a otro tras una veloz reducción de la misma. El objetivo de esta metodología es obtener varios modelos, realizando una *snapshot* cada vez que se alcance un óptimo local, que se utilizarán al finalizar el proceso como un modelo conjunto. Este método nos permite realizar un entrenamiento más general, que tiene en cuenta información relevante de los mínimos locales, sin la necesidad de repetirlo completamente varias veces.

Cabe destacar que *ActionFormer* no utilizará el método de *snapshot ensembling*, sino que hará uso del *Exponential Moving Average* o *EMA*, para calcular una media móvil de los parámetros del modelo, que irán perdiendo peso exponencialmente a medida que se vayan realizando nuevas iteraciones.

Finalmente, estudiaremos en mayor detalle la función de pérdida ($\mathcal{L}$) utilizada

por el modelo:

$$\mathcal{L} = \sum_{t}(\mathcal{L}_{cls} + \lambda_{reg}\mathbb{L}_{c_t}\mathcal{L}_{reg})/T_{+} \quad (1)$$

Esta se calculará como la media en cada instante temporal t, de la suma de un error de clasificación ($\mathcal{L}_{cls}$) y otro de regresión ($\mathcal{L}_{reg}$). $\mathbb{L}_{c_t}$ es un valor binario que valdrá 0 si no existe una instancia en el *ground-truth* en t, o 1 en caso contrario, de manera que el error de regresión no se tenga en cuenta en caso de no haber una acción. Además, se añade el parámetro λ_{reg} (1 por defecto), encargado de regular la influencia de $\mathcal{L}_{reg}$ sobre $\mathcal{L}_{cls}$. Finalmente, se aplica un cociente con el término T_{+} en el denominador, que equivale al número de instantes temporales que contienen una acción en el *ground-truth*. Cabe destacar que $\mathcal{L}_{cls}$ utiliza el *Focal Loss*[37], modificación de las funciones de *Cross Entropy* adaptada a conjuntos de datos con un alto desbalance entre clases. Por otro lado, $\mathcal{L}_{reg}$ utilizará una función de *Distance Intersection over Union*[38], que tendrá en cuenta el grado de solape entre el intervalo original y los predichos. En el Anexo G realizamos un análisis en mayor detalle de ambas funciones.

3.6. Uso de ActionFormer

Una vez estudiada en mayor profundidad la arquitectura del modelo, pasamos al proceso de instalación y configuración del mismo. Para verificar su correcto funcionamiento, entrenamos y evaluamos *ActionFormer* utilizando el *dataset* de *Epic Kitchens*, tratando de replicar los resultados obtenidos en la publicación original[4]. Cabe destacar que tanto las *features* utilizadas como las anotaciones fueron proporcionadas por *ActionFormer* y *Epic Kitchens* respectivamente. De esta forma, conseguimos un *average mAP* de 23.26 % para los verbos, y de 22.12 % para los nombres. Tras esta familiarización inicial, pasamos al uso de nuestro propio conjunto de datos, lo cual requirió en primer lugar la transformación de las anotaciones iniciales a un formato compatible, así como la extracción de *features* de los vídeos utilizados.

Para ello, empleamos la herramienta *GluonCV*[6], que entre otras funcionalidades permitía el uso de distintos reconocedores de acciones preentrenados para la extracción, entre los que se encontraba el ya mencionado *SlowFast*. Como salida, se generará una representación vectorial (de dimensión 2304) de cada uno de los fragmentos de vídeo a procesar, que serán extraídos periódicamente con una separación definida por un número de *frames* conocido como *temporal stride*. Para obtener unas *features* todo lo similares posible a las de *Epic Kitchens*, utilizamos un *stride* de 16 *frames*, además se adaptaron los vídeos para que utilizaran el mismo *frame rate* (30*fps*) y se redujo su resolución. Debido a las limitaciones de memoria de la herramienta, fue necesario implementar un *script* que dividiese todos los vídeos en los distintos fragmentos a tratar, y otro que agrupara los vectores de *features* obtenidos. Estos programas, así como los utilizados para la transformación de las anotaciones y cualquier otro código auxiliar, se detallan en el Anexo H.

La adaptación de nuestro conjunto de datos fue un proceso por lo general costoso temporalmente, ya que los fallos en la implementación o en la configuración requerían la modificación y repetición de la extracción o del entrenamiento. Tras unos valores iniciales lejos de los esperados y un proceso de depuración exhaustiva, obtuvimos los resultados mostrados en la Tabla 3, en la que podemos ver como se superó la precisión obtenida con *Epic Kitchens* tanto para verbos como para nombres, tras realizar un entrenamiento con *Fine Tuning*. Dicha técnica nos permite entrenar en primer lugar utilizando un conjunto de datos auxiliar, para retomarlo posteriormente con los datos que realmente se quieren procesar. De esta manera, el modelo tendrá un punto de partida que facilitará su convergencia. Ya que nuestro *dataset* se podría considerar una simplificación de *Epic Kitchens*, al utilizar un número más reducido de clases, una cámara fija en lugar de una móvil y tener una menor densidad temporal, consideramos

que el uso de esta técnica sería adecuado. Esta hipótesis se ve respaldada por el incremento de precisión al realizar el preentrenamiento, 1.5 % para verbos y 8 % para los nombres. Destacar que la adaptación de las anotaciones, la extracción de *features* y la depuración del entrenamiento se detallarán en detalle en el Anexo C.

Dataset	**Average mAP Verbos**	**Average mAP Nombres**
Epic Kitchens	23,26 %	22,12 %
BSH	32 %	16 %
BSH + Fine Tuning	33,5 %	24 %

Tabla 3: *Average mAP* obtenido para verbos y nombres, comparado con los resultados de *Epic Kitchens* y el uso de *Fine Tuning*.

4. Análisis de los resultados

El cálculo de la precisión del modelo puede ser de gran ayuda a la hora de comparar de forma cuantitativa el rendimiento del mismo utilizando distintas configuraciones o conjuntos de datos, pero se quedará corto en caso de necesitar un estudio más detallado, en el que queremos buscar que características de los vídeos facilitan o dificultan su funcionamiento, con el objetivo de encontrar y mejorar los problemas existentes en el *dataset*. Para esto, sería conveniente conocer la precisión individual de cada clase, así como las confusiones entre ellas. Además, como vimos en el Anexo A, cualquier intervalo generado cuya clase no coincida con alguna de las presentes en el *ground-truth* será ignorado, entorpeciendo aún más el análisis de los errores.

Debido a esto, durante este apartado realizaremos un estudio en profundidad de los resultados, valiéndonos de representaciones gráficas de los mismos con el objetivo de evaluar las características de nuestro conjunto de datos y su influencia sobre el rendimiento. Para ello, hemos elegido un conjunto de 13 vídeos, con el objetivo de analizar el efecto sobre el modelo de la variedad de clases, la iluminación, la duración y la densidad temporal de acciones. La Tabla 4 recoge una lista de los vídeos elegidos y sus métricas, así como la precisión obtenida para cada uno de ellos.

Vídeo	**Duración**	**Acciones / minuto**	**Nº Verbos**	**Nº Nombres**	**mAP Verbos**	**mAP Nombres**	**Receta**
P01_3	311,8s	5,196	4	2	44,42 %	49,79 %	Pasta
P01_4	157,44s	7,622	4	8	30,10 %	23,30 %	Sofrito de verduras
P01_9	2106,2s	2,507	5	11	29.43 %	16.56 %	Carne picada
P01_30	231,72s	5,438	2	4	54,06 %	39,39 %	Freír filetes
P01_32	371,96s	5,323	2	1	41,58 %	43,55 %	Freír filetes
P01_33	1993,32s	0,692	2	1	47,66 %	44,09 %	Freír filetes
P01_34	310,68s	3,09	4	4	53,56 %	37,53 %	Freír filetes
P01_67	401,8s	1,792	2	3	76,31 %	55,51 %	Freír carne
P01_68	2849,36s	2,129	3	3	31,72 %	55,51 %	Freír patatas y carne
P01_71	1316,84	0,957	2	5	94,34 %	71,14 %	Sofrito de verdura
P01_78	217,88s	5,959	3	4	25,73 %	28,35 %	Freír filetes
P01_80	604,04s	3,377	4	5	52,21 %	46,17 %	Freír patatas y carne
P01_90	592,04s	0,811	2	2	52,72 %	51,95 %	Pasta

Tabla 4: Tabla con información sobre los vídeos elegidos para el análisis exhaustivo. De cada uno de ellos vemos su duración, el número de acciones anotadas por minuto, el número de clases únicas de verbos y nombres presentes, el valor de *average mAP* obtenido para verbos y nombres, y la receta realizada en el vídeo.

Mencionar especialmente que los elevados valores de *mAP* obtenidos para cada uno de los vídeos de manera individual se deben al método de cálculo comentado en el Anexo A. Ya que las métricas se obtendrán individualmente para cada clase, comparando instancias generadas y *ground-truth* de la correspondiente etiqueta, al evaluar únicamente un vídeo no se tendrán en cuenta las predicciones del resto, aumentando el valor de precisión.

4.1. Variedad de clases

En primer lugar, analizaremos la influencia de la variedad de clases en el rendimiento, para ello hemos escogido los vídeos 3, 4 y 9, todos grabados en la misma cocina. Cada uno de ellos tendrá un rango más amplio de etiquetas para los nombres que el anterior. Podemos ver una comparación entre ellos y la precisión obtenida en la Tabla 5. El 9, pese a tener una densidad de acciones considerablemente menor que la del 3 y el 4, obtiene una puntuación mucho menor, debido a la mayor variedad de etiquetas. Por otro lado, el número 3 al ser mucho más sencillo que el resto, obtiene un mejor rendimiento.

Vídeo	**Duración**	**Acciones / minuto**	**Nº Nombres**	**mAP Verbos**	**mAP Nombres**	**Receta**
P01_3	311,8s	5,196	2	44,42 %	49,79 %	Pasta
P01_4	157,44s	7,622	8	30,10 %	23,30 %	Sofrito de verduras
P01_9	2106,2s	2,507	11	29.43 %	16.56 %	Carne picada

Tabla 5: Tabla de comparación entre vídeos según el número de etiquetas utilizados.

Este análisis se puede realizar más fácilmente comparando las matrices de confusión obtenidas con las predicciones de cada uno de los vídeos, como observamos en la Figura 9. En el número 3, solo hay dos clases distintas en el *ground-truth*, de manera que la tarea de clasificación se facilita, ya que las predicciones de otras etiquetas no presentes no se tendrán en cuenta para el cálculo de la precisión.

Por otro lado, observamos que tanto el vídeo 4 como el 9 tienen una variedad mucho mayor de acciones anotadas, lo cual causa una gran confusión con las etiquetas de mayor frecuencia (como 'food', 'vegetable' o 'pot'), mientras el modelo ignora casi por completo otras clases menos frecuentes (como 'water' o 'spice'). Esto se observa fácilmente en la gráfica, los aciertos no se centran en torno a la diagonal de la matriz, sino que se dispersan, además algunas columnas quedarán casi vacías, indicando que no se ha realizado ninguna predicción con dicha etiqueta. Todo esto, puede apuntar a un problema de desbalance de datos en el *dataset*. Además, como podíamos ver en la Tabla 4 anterior, por lo general los valores de precisión obtenidos para los nombres serán menores que para los verbos, lo cual es de esperar, debido a una mayor dificultad de clasificación propiciada por la diversidad de las etiquetas. Al existir una gran cantidad de clases con pocas apariciones en el *ground-truth*, el modelo tendrá dificultades para detectarlas correctamente durante la inferencia.

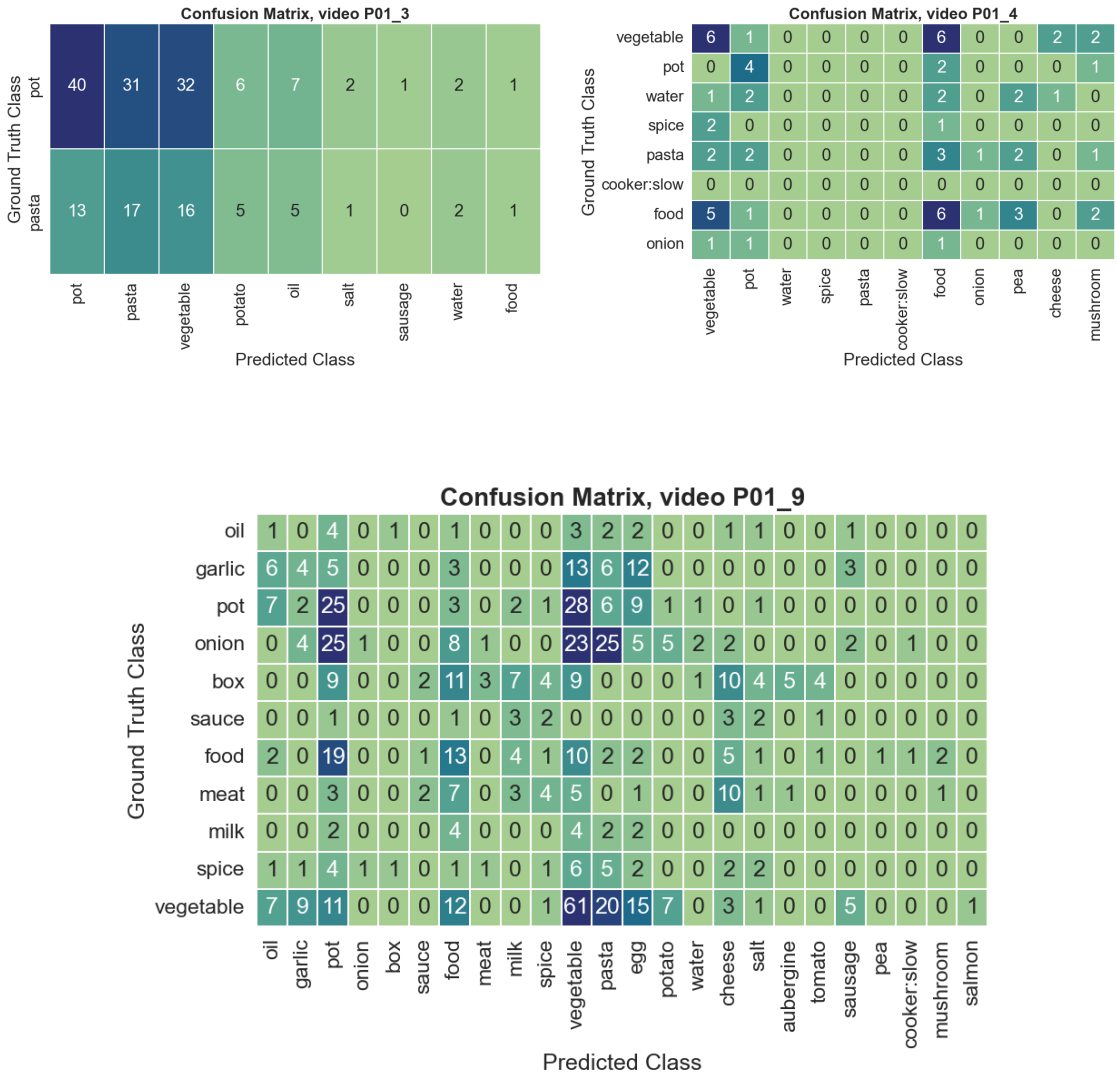

Figura 9: Matrices de confusión para las etiquetas de nombres, de los vídeos 3, 4 y 9, para la comparación del rendimiento del modelo según la variedad de las clases anotadas en el *ground-truth*.

4.2. Iluminación

Al hacer uso del conjunto de datos a lo largo del proyecto, nos dimos cuenta de que algunos de los vídeos presentaban una serie de artefactos o características desfavorables en su grabación, debido a las condiciones de iluminación. Varios de ellos presentaban una alta intensidad lumínica, lo cual causaba la aparición de destellos en la cámara, mientras que otros no disponían de la luz suficiente. En ambos casos, se dificultaba el reconocimiento de los ingredientes utilizados. En la Figura 10 podemos ver ejemplos de estos sucesos. Para estudiar la influencia de estas circunstancias sobre la precisión, hemos elegido 4 vídeos distintos (30, 32, 33 y 34), todos grabados en la misma cocina y con una receta similar, freír filetes de carne en una sartén. Dos de ellos (30 y 33) presentarán destellos, mientras que los restantes (32 y 34) se habrán grabado en un ambiente más oscuro. Vemos en la Tabla 6 una comparación de sus características y el rendimiento del modelo en cada uno de ellos.

P01_30

P01_32

P01_33

P01_34

Figura 10: Ejemplos de artefactos de iluminación en los vídeos, que pueden dificultar la clasificación de los ingredientes.

Vídeo	Duración	Acciones / minuto	Nº Verbos	Nº Nombres	mAP Verbos	mAP Nombres	Artefactos
P01_30	231,72s	5,438	2	4	54,06 %	39,39 %	Destellos
P01_32	371,96s	5,323	2	1	41,58 %	43,55 %	Poca iluminación
P01_33	1993,32s	0,692	2	1	47,66 %	44,09 %	Destellos
P01_34	310,68s	3,09	4	4	53,56 %	37,53 %	Poca iluminación

Tabla 6: Tabla de comparación entre vídeos según su iluminación.

Pese a no observarse una relación clara entre estos artefactos y la precisión, analizando las matrices de confusión obtenidas para estos vídeos (Figura 11), vemos como hay clases de nombres que se confundirán muy frecuentemente. Por ejemplo, en el vídeo 30 el modelo tiende a equivocarse al predecir 'fish' en lugar de 'chicken', lo mismo ocurrirá en el 34 con 'meat' y 'oil'. Por otro lado, tanto en el vídeo 32 como 33, aunque aparezca una única etiqueta en el *ground-truth* ('meat'), hay una gran cantidad de predicciones de otras clases, como 'fish' o 'food'. Por lo tanto, aunque la iluminación no parezca tener un efecto directo en la puntuación, las dificultades de clasificación observadas no

son negligibles y podrían entorpecer el rendimiento del modelo. Al contrario, podemos ver como la precisión de los verbos se mantiene más elevada en la mayoría de los casos, lo cual es esperable, ya que los artefactos en la grabación alterarán principalmente el color y los contornos de los ingredientes y utensilios utilizados, que no afectará especialmente a la capacidad de distinción entre acciones.

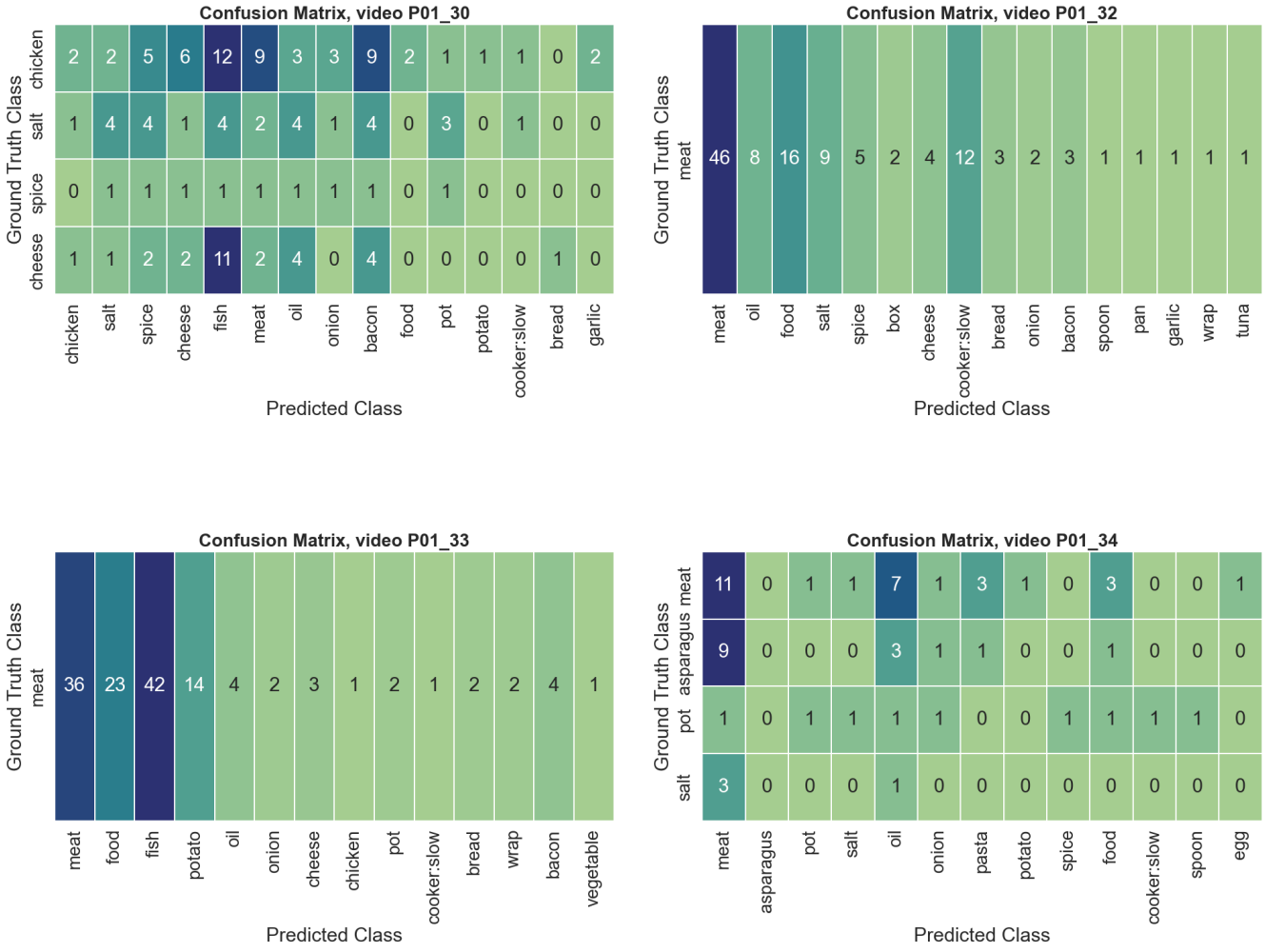

Figura 11: Matrices de confusión de los vídeos 30, 32, 33 y 34, para el análisis de la influencia de artefactos de iluminación en el rendimiento.

4.3. Duración

A continuación, hemos elegido los vídeos 68, 67, y 71, grabados en la misma cocina pero con longitudes diversas, para estudiar como afecta la duración de los mismos sobre el rendimiento del modelo. Podemos ver una comparación de los mismos en la Tabla 7, en la que no se observa ninguna correlación entre su extensión y los resultados de precisión obtenidos tanto para nombres como verbos. La única anomalía encontrada es el considerable descenso en la puntuación para los verbos del vídeo más largo, el 68, lo cual analizando su matriz de confusión (Figura 12) se puede ver que está causado por un gran número de errores al predecir las etiquetas 'flip' y 'add' como 'mix', y no por su longitud. Además, realizando un análisis en más detalle hemos observado que durante el progreso de la receta, en el momento de retirar las patatas no hay ninguna acción anotada, lo cual puede ser un causante del descenso en el rendimiento. Por otro lado, vemos como el

número 71 obtiene una precisión muy elevada para los verbos, la cual no estará relacionada con su duración, sino con su baja densidad temporal y la claridad en la grabación de las acciones (principalmente de mezclado de ingredientes) realizadas.

Vídeo	Duración	Acciones / minuto	Nº Verbos	Nº Nombres	mAP Verbos	mAP Nombres	Receta
P01_67	401,8s	1,792	2	3	76,31 %	55,51 %	Freír carne
P01_68	2849,36s	2,129	3	3	31,72 %	55,51 %	Freír patatas y carne
P01_71	1316,84	0,957	2	5	94,34 %	71,14 %	Sofrito de verdura

Tabla 7: Tabla de comparación entre vídeos según su duración.

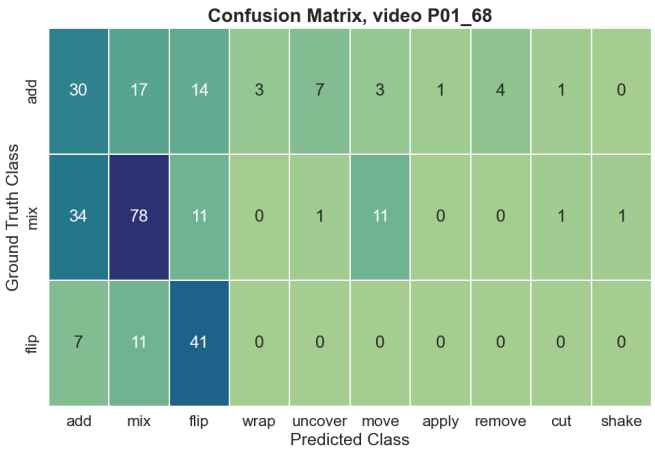

Figura 12: Matriz de confusión de las predicciones de los verbos para el vídeo 68.

4.4. Densidad Temporal de Acciones

Finalmente, hemos elegido tres vídeos (78, 80 y 90) con el objetivo de comparar la densidad temporal de las anotaciones con el rendimiento del modelo. Como podemos ver en la Tabla 8, a medida que va aumentando dicha frecuencia la precisión obtenida se va reduciendo, aunque no de forma lineal, ya que dependerá también de otros factores. Esta relación entre la densidad y los resultados ya se observó en la Sección 2.2, en la que se comparaban distintos *benchmarks* para la Detección de Acciones.

Vídeo	Duración	Acciones / minuto	Nº Verbos	Nº Nombres	mAP Verbos	mAP Nombres	Receta
P01_78	217,88s	5,959	3	4	25,73 %	28,35 %	Freír filetes
P01_80	604,04s	3,377	4	5	52,21 %	46,17 %	Freír patatas y carne
P01_90	592,04s	0,811	2	2	52,72 %	51,95 %	Pasta

Tabla 8: Tabla de comparación entre vídeos según su densidad temporal de acciones.

4.5. Problemas del *dataset*

Como hemos visto a lo largo de los apartados anteriores, es difícil realizar el análisis de la influencia de una característica de los vídeos en concreto sin tener en cuenta el resto de ellas, además, el reducido tamaño del conjunto elegido también entorpece este proceso. Aun así, gracias a este estudio hemos podido concluir que nuestro *dataset* presenta una serie de propiedades que pueden dificultar el trabajo con el mismo, así como el rendimiento del modelo.

En primer lugar, existe un problema evidente de desbalance de datos ya observado en la Sección 4.1, causado por la existencia un pequeño conjunto de clases que acumula la gran mayoría de anotaciones, principalmente 'mix', 'add' y 'flip' para los verbos, 'food' y 'meat' para los nombres. Este desequilibrio puede causar dificultades al modelo, el cual se ajustaría adecuadamente a las etiquetas más frecuentes, pero no podría aprender correctamente las características de aquellas con menos instancias. Dicho obstáculo se agrava además por el uso de clases con un significado muy general, y que pueden englobar a otras más específicas. En el caso de los verbos se puede observar al utilizar 'add', que se emplea también al verter líquidos durante la receta, acción representada por la etiqueta 'pour'. Para los nombres supondrá un inconveniente aún mayor, ya que existen una gran cantidad de intervalos etiquetados con clases genéricas, como 'food', 'meat' o 'vegetable'. Este tipo de desbalance es común en *datasets* de tamaño reducido, para los cuales suele ser costoso la recopilación de nuevas muestras que pudieran ayudar a aliviarlo. Al disponer de vídeos grabados en únicamente 3 cocinas, que recogen recetas habitualmente similares entre sí, y solo tener unos 100 anotados, la aparición de este desequilibrio era previsible, para el cual deberemos buscar una solución que no requiera más grabaciones, ya que están fuera del alcance del proyecto.

Como podemos ver en las Figuras 13 y 14, el problema de desbalance en el conjunto de datos es evidente, tanto para los verbos como los nombres. En ambos histogramas aparece un pequeño grupo de etiquetas que recogen una gran parte de las anotaciones, principalmente 'mix' y 'add' para los verbos, 'food' y 'meat' en el caso de los nombres. Por otro lado, habrá una gran cantidad de clases muy poco comunes, que en algunos casos ven su significado eclipsado por otras más genéricas, como 'chicken' o 'bacon', englobadas por 'meat'.

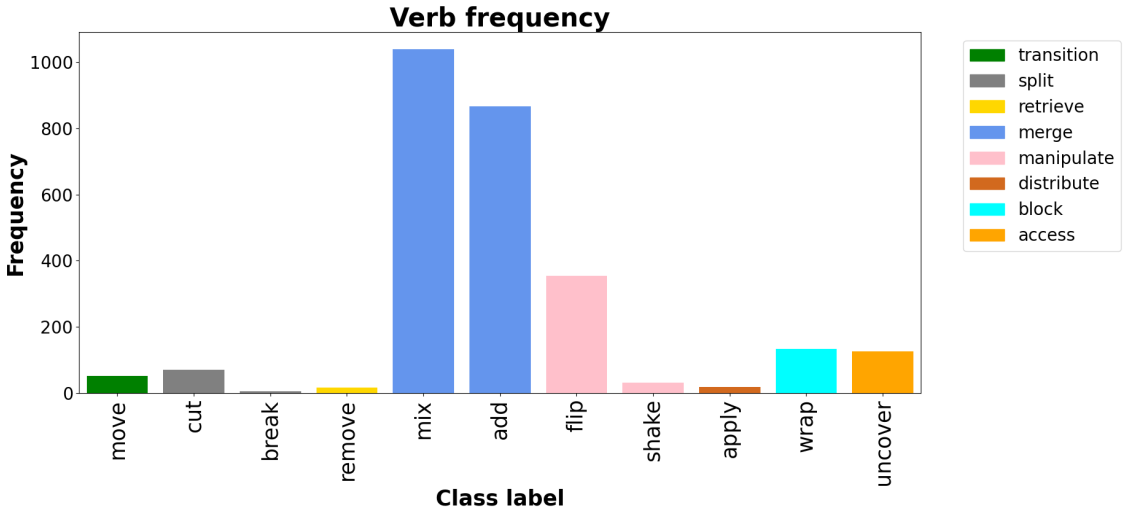

Figura 13: Histograma de la frecuencia de aparición de las distintas clases de verbos en el *ground-truth*. Podemos ver como existe un desbalance en los datos, al concentrar 'mix', 'add' y 'flip' en menor medida la mayoría de las anotaciones.

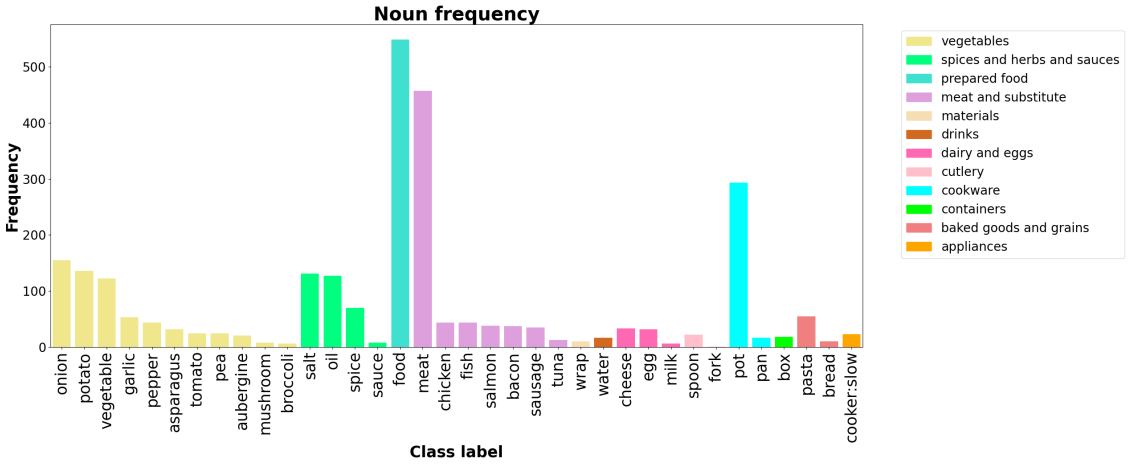

Figura 14: Histograma de la frecuencia de aparición de las distintas clases de nombres en el *ground-truth*. Observamos como un pequeño conjunto de etiquetas, de carácter muy genérico, como 'food' o 'meat' concentran una gran cantidad de anotaciones, eclipsando a otras con significado más específico.

Otro de los principales problemas observados en el conjunto de vídeos es la iluminación (visto en la Sección 4.2), ya que una gran cantidad de ellos fueron grabados en cocinas demasiado oscuras, o excesivamente iluminadas, lo cual provocaba destellos en la cámara. Ambos fenómenos podrían dificultar la capacidad de clasificación del modelo, en especial en el caso de los ingredientes utilizados durante las recetas. Además, debido al uso de una cámara estática, la realización de algunas acciones quedará parcialmente fuera del marco de la imagen, pudiendo entorpecer reconocimiento.

Finalmente, destacar que a lo largo de todo el *dataset* existen acciones que fueron ignoradas durante el etiquetado, para las que se generan predicciones de alta confianza, provocando bajones de precisión como los observados en el vídeo 68 en la Sección 4.3. Todos estos problemas llevaron a la conclusión de que realizar un reetiquetado del conjunto sería la forma más adecuada de aliviar los impedimentos al correcto funcionamiento del modelo.

5. Reetiquetado de los vídeos

Con el objetivo de mejorar el rendimiento e intentando reducir los problemas de desbalance presentes en nuestro *dataset*, realizamos un reetiquetado completo de los vídeos. De esta manera, buscaremos enriquecer el conjunto de datos, evitando el uso de etiquetas que puedan englobar a otras, así como anotando cualquier acción relevante que haya sido ignorada durante el proceso inicial. Como resultado, obtendremos un *dataset* anotado con una mayor variedad de clases, algunas de ellas con un número muy reducido de instancias debido a su aparición puntual o poco común. Aunque esto pudiera agravar los problemas de precisión del modelo, el preentrenamiento con un conjunto tan variado como *Epic Kitchens*[2] debería bastar para la correcta clasificación de las etiquetas poco comunes en nuestros vídeos.

5.1. Proceso de reetiquetado

Dicho proceso consistió en el etiquetado de cualquier acción no anotada previamente, indicando tanto su instante de inicio como de finalización, así como las clases del verbo y nombre correspondientes. También, se corrigieron las anotaciones ya existentes. Tanto para las nuevas instancias como para las actuales, utilizamos

etiquetas lo más específicas posible.

En el caso de los verbos, cabe destacar el desglose de la clase 'add' en la propia acción de añadir ingredientes sólidos y líquidos, que pasaron a ser etiquetados con 'pour'. 'add' se utilizaba también cuando un utensilio de cocina se colocaba en la encimera o placa, por lo que modificamos dichas instancias para utilizar el verbo 'put' en su lugar. También existían confusiones entre algunas clases, como el uso de 'add' al mover un instrumento de sitio en la cocina, en cuyo caso fueron sustituidas por 'move', o al agitar un recipiente, corregida empleando la etiqueta 'shake'. Finalmente, cambiamos apariciones del verbo 'mix' por 'move' cuando un ingrediente se movía dentro de un recipiente sin mezclarse.

Por otro lado, el proceso de reetiquetado para los nombres fue más costoso, ya que las anotaciones originales utilizaban comúnmente clases muy generales, como 'food', 'meat' o 'vegetable'. La clase 'food' se empleaba principalmente para mezclas de varios componentes, que en medida de lo posible fueron sustituidas por la etiqueta del ingrediente principal, siguiendo la metodología observada en *Epic Kitchens*[2]. Además, tuvimos que modificar las anotaciones de las clases 'butter', 'carrot' y 'rice', que utilizaban un identificador incorrecto. Finalmente, las instancias marcadas como 'meat' o 'vegetable' fueron sustituidas por las correspondientes a las etiquetas más específicas, siempre y cuando estas estuvieran presentes en el rango de *Epic Kitchens* y fueran reconocibles en el vídeo.

Los problemas del *dataset* comentados en la Sección 4.5 también han supuesto contratiempos durante el reetiquetado. Principalmente, la gran cantidad de acciones no anotadas provocó una gran inversión temporal al procesar vídeos con pocas instancias previamente etiquetadas. Así mismo, los artefactos de iluminación causaron problemas a la hora de determinar con precisión algunos de los ingredientes utilizados, por lo que en algunos casos tuvimos que mantener las etiquetas generales como 'meat' o 'vegetable'. Cabe destacar, que las recetas realizadas en cada uno de los vídeos no fueron registradas, lo cual habría facilitado la anotación en estos casos. Por otro lado, algunas instancias se han ignorado durante este proceso, principalmente las relacionadas con la manipulación de cubertería (tenedores, cuchillos, cucharas, ...) debido a la alta cantidad de acciones existentes que no influyen sobre el desarrollo de la receta. También se han excluido aquellas que aparecían parcialmente fuera de cámara, anotándose únicamente las que fueran fácilmente reconocibles.

5.2. Resultados con las nuevas etiquetas

Tras el reetiquetado, pasamos de tener un *dataset* con 2716 instancias anotadas a 3433, incrementado también la frecuencia de acciones por vídeo media, pasando de 29.85 a 37.73. Además, al añadir acciones previamente ignoradas, aumentó el número de clases tanto para los nombres (de 36 a 51) como para los verbos (de 11 a 18). En la Figura 15 podemos ver una comparación entre el histograma de frecuencia de verbos y en la Figura 16 la misma equiparación para los nombres. En ambos casos es fácil apreciar como la densidad de aparición se ha repartido, al favorecer el uso de etiquetas lo más específicas posible en cada intervalo anotado y añadir nuevas anotaciones.

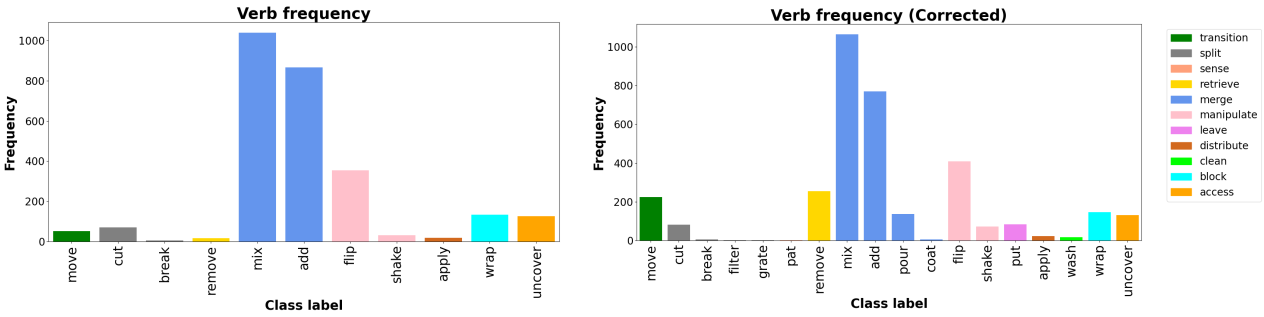

Figura 15: Histograma con la frecuencia de aparición de verbos antes (izquierda) y después (derecha) del proceso de reetiquetado.

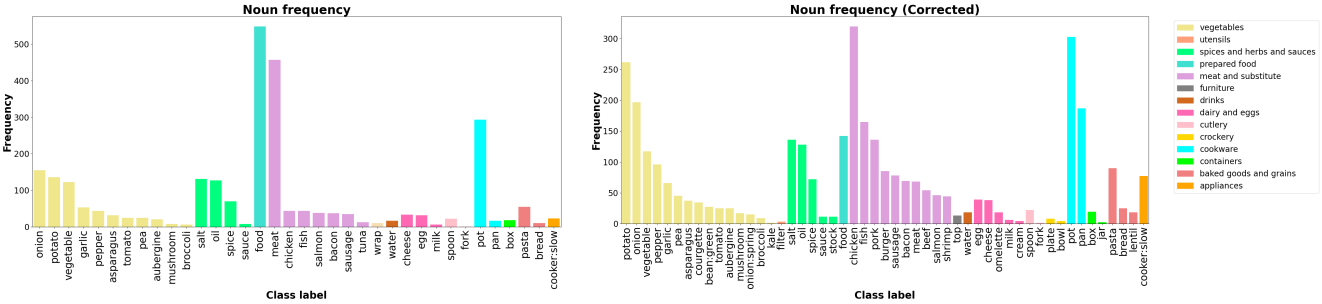

Figura 16: Histograma con la frecuencia de aparición de nombres antes (izquierda) y después (derecha) del proceso de reetiquetado.

En la Tabla 9, observamos los resultados obtenidos al realizar el entrenamiento utilizando las nuevas anotaciones, en comparación con las originales. Como vemos, se ha producido un incremento considerable en la precisión en el caso de los verbos, así como un bajón en el resultado para los nombres.

Etiquetas utilizadas	***Average mAP Verbos***	***Average mAP Nombres***	***Fine Tuning***
Antes del reetiquetado	33,5 %	24 %	Sí
Después del reetiquetado	36.38 %	15.02 %	No
Después del reetiquetado	37.53 %	16.39 %	Sí

Tabla 9: Resultados del entrenamiento del modelo tras el reetiquetado de los datos. Vemos un incremento considerable (un 4 %) en la precisión obtenida para los verbos, por otro lado, se ha producido un desplome (casi un 8 %) del rendimiento para los nombres.

El deterioro en el rendimiento del modelo para la detección de los nombres se puede deber principalmente a problemas en la grabación de los vídeos, así como a los artefactos de iluminación comentados anteriormente. La gran cantidad de clases distintas, que en su mayoría aparecen en un número reducido de ocasiones, y con una calidad de imagen mejorable, provocan que el modelo tenga dificultades a la hora de reconocerlos. Esto causará que el proceso de *Fine Tuning* con *Epic Kitchens* no suponga un incremento significativo de la precisión, ya que las instancias en nuestros vídeos no son lo suficientemente similares a las de dicho *dataset*. Como reflejan los resultados, estos inconvenientes no suponen un compromiso en el rendimiento del modelo en lo que a verbos respecta. La baja calidad de los vídeos o los artefactos de iluminación no impiden el correcto reconocimiento de las acciones, que son fácilmente distinguibles unas de otras. De igual manera, el número reducido

de clases ayudará también a reducir la confusión. Por lo tanto, un etiquetado más preciso de las acciones y de sus clases implicaron una mejora notable en los resultados.

6. Conclusión

A lo largo del proyecto hemos analizado la viabilidad del uso de la Detección de Acciones en vídeo, aplicado a un conjunto de vídeos de cocina. Tras un estudio exhaustivo del estado del arte centrado en el *dataset Epic Kitchens*[2], en el que se han examinado técnicas basadas en la generación de proposiciones temporales y el modelado de las relaciones temporales a largo plazo, elegimos *ActionFormer*[4] como modelo de detección a utilizar. Este se caracteriza por una arquitectura basada en las redes *Transformers*, que utilizan la auto-atención para modelar las relaciones contextuales entre elementos de una misma secuencia. Junto a una pirámide de *features* temporal que permite el modelado del contexto en distintas resoluciones temporales y a una serie de redes convolucionales, el modelo es capaz de predecir para cada instante la clase de la acción más probable, así como el inicio y la finalización de la misma.

El proceso de adaptación de nuestros datos fue costoso, requiriendo la implementación de varios *scripts* para la modificación del formato de las anotaciones y vídeos, además de la extracción de *features* con la herramienta *GluonCV*. Pese a la dificultad de la tarea de la Detección de Acciones, analizando en detalle la configuración del modelo y utilizando técnicas como el *Fine Tuning* hemos obtenido unos resultados mejores a los logrados con *Epic Kitchens*, demostrando que nuestro *dataset* es una simplificación del mismo.

6.1. Limitaciones y trabajo posterior

Realizando un análisis más detallado de los resultados obtenidos, pudimos observar que nuestro *dataset* sufría de un problema de desbalance de clases, el cual intentamos resolver realizando un reetiquetado del mismo. Para ello, anotamos cualquier acción que fuera ignorada, y utilizamos las clases más específicas posibles para evitar que unas pocas agruparan la mayoría de instancias. Dicho proceso resultó en un incremento del rendimiento del modelo para los verbos, pero supuso un bajón considerable de la precisión en el caso de los nombres. Esto se debe principalmente a la dificultad de clasificación de algunos ingredientes durante las recetas, a causa de la reducida resolución de la cámara y los problemas de iluminación durante las grabaciones. Otra de las principales limitaciones del *dataset* es la falta de variedad en los propios datos, causado por la similitud entre las recetas grabadas. Para aliviar estos problemas, debidos al alto coste de grabación y etiquetado de los vídeos, se podría considerar como una línea de investigación a futuro el uso de técnicas de detección de acciones no supervisadas o semi-supervisadas.

Finalmente, destacar el posible trabajo posterior siguiendo las tendencias del estado del arte, intentando reducir el uso de técnicas de *auto-atención*, que pueden provocar la uniformización y pérdida de información de las *features* extraídas de vídeos. En su lugar, se pueden estudiar modelos que siguen la arquitectura *Transformer*, sustituyendo los módulos de *self-attention* por otras técnicas de modelado del contexto temporal más eficientes. Como hemos visto en el Anexo D.2 ofrecen resultados prometedores, pero aún requieren de una investigación en mayor profundidad. Además, debemos ser conscientes del desarrollo de modelos fundacionales para tareas de procesado de vídeo, que podrían suponer una gran evolución del estado del arte en el futuro, como ha ocurrido en campos como el *NLP*.

Bibliografía

[1] Alejandro López Calvo, Diego Gutiérrez Pérez y Ana Belén Serrano Pacheu. "Action recognition and object detection in cooking videos via deep learning". En: (2023).

[2] Dima Damen et al. "Rescaling Egocentric Vision: Collection, Pipeline and Challenges for EPIC-KITCHENS-100". En: *International Journal of Computer Vision (IJCV)* 130 (2022), págs. 33-55. URL: https://doi.org/10.1007/s11263-021-01531-2.

[3] Dima Damen et al. "Scaling Egocentric Vision: The EPIC-KITCHENS Dataset". En: *European Conference on Computer Vision (ECCV)*. 2018.

[4] Chenlin Zhang, Jianxin Wu y Yin Li. *ActionFormer: Localizing Moments of Actions with Transformers*. 2022. arXiv: 2202.07925 [cs.CV].

[5] Christoph Feichtenhofer et al. *SlowFast Networks for Video Recognition*. 2019. arXiv: 1812.03982 [cs.CV].

[6] Jian Guo et al. "GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing". En: *Journal of Machine Learning Research* 21.23 (2020), págs. 1-7. URL: http://jmlr.org/papers/v21/19-429.html.

[7] Soo-Min Kang y Richard P. Wildes. "Review of Action Recognition and Detection Methods". En: *CoRR* abs/1610.06906 (2016). arXiv: 1610.06906. URL: http://arxiv.org/abs/1610.06906.

[8] Elahe Vahdani y Yingli Tian. "Deep Learning-based Action Detection in Untrimmed Videos: A Survey". En: *CoRR* abs/2110.00111 (2021). arXiv: 2110.00111. URL: https://arxiv.org/abs/2110.00111.

[9] Anurag Arnab et al. "ViViT: A Video Vision Transformer". En: *CoRR* abs/2103.15691 (2021). arXiv: 2103.15691. URL: https://arxiv.org/abs/2103.15691.

[10] João Carreira y Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". En: *CoRR* abs/1705.07750 (2017). arXiv: 1705.07750. URL: http://arxiv.org/abs/1705.07750.

[11] David G Lowe. "Object recognition from local scale-invariant features". En: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, págs. 1150-1157.

[12] N. Dalal y B. Triggs. "Histograms of oriented gradients for human detection". En: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886-893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[13] Corinna Cortes y Vladimir Vapnik. "Support-vector networks". En: *Machine learning* 20.3 (1995), págs. 273-297.

[14] Zheng Shou, Dongang Wang y Shih-Fu Chang. *Temporal Action Localization in Untrimmed Videos via Multi-stage CNNs*. 2016. arXiv: 1601.02129 [cs.CV].

[15] Tianwei Lin et al. *BMN: Boundary-Matching Network for Temporal Action Proposal Generation*. 2019. arXiv: 1907.09702 [cs.CV].

[16] Xiang Wang et al. *Proposal Relation Network for Temporal Action Detection*. 2021. arXiv: 2106.11812 [cs.CV].

[17] Sauradip Nag et al. *Semi-Supervised Temporal Action Detection with Proposal-Free Masking*. 2022. arXiv: 2207.07059 [cs.CV].

[18] Mengmeng Xu et al. *G-TAD: Sub-Graph Localization for Temporal Action Detection*. 2020. arXiv: 1911.11462 [cs.CV].

[19] Zichen Yang, Jie Qin y Di Huang. *ACGNet: Action Complement Graph Network for Weakly-supervised Temporal Action Localization*. 2021. arXiv: 2112.10977 [cs.CV].

[20] Min Yang et al. *BasicTAD: an Astounding RGB-Only Baseline for Temporal Action Detection*. 2023. arXiv: 2205.02717 [cs.CV].

[21] Weihao Yu et al. *MetaFormer Is Actually What You Need for Vision*. 2022. arXiv: 2111.11418 [cs.CV].

[22] Yihe Dong, Jean-Baptiste Cordonnier y Andreas Loukas. "Attention is not all you need: pure attention loses rank doubly exponentially with depth". En: *Proceedings of the 38th International Conference on Machine Learning*. Ed. por Marina Meila y Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, págs. 2793-2803. URL: https://proceedings.mlr.press/v139/dong21a.html.

[23] Dingfeng Shi et al. *TriDet: Temporal Action Detection with Relative Boundary Modeling*. 2023. arXiv: 2303.07347 [cs.CV].

[24] Tuan N. Tang, Kwonyoung Kim y Kwanghoon Sohn. *TemporalMaxer: Maximize Temporal Context with only Max Pooling for Temporal Action Localization*. 2023. arXiv: 2303.09055 [cs.CV].

[25] Limin Wang et al. *VideoMAE V2: Scaling Video Masked Autoencoders with Dual Masking*. 2023. arXiv: 2303.16727 [cs.CV].

[26] Yi Wang et al. *InternVideo: General Video Foundation Models via Generative and Discriminative Learning*. 2022. arXiv: 2212.03191 [cs.CV].

[27] Y.-G. Jiang et al. *THUMOS Challenge: Action Recognition with a Large Number of Classes*. http://crcv.ucf.edu/THUMOS14/. 2014.

[28] Hang Zhao et al. "HACS: Human Action Clips and Segments Dataset for Recognition and Temporal Localization". En: *arXiv preprint arXiv:1712.09374* (2019).

[29] Bernard Ghanem Fabian Caba Heilbron Victor Escorcia y Juan Carlos Niebles. "ActivityNet: A Large-Scale Video Benchmark for Human Activity Understanding". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, págs. 961-970.

[30] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[31] Alec Radford et al. "Improving language understanding by generative pre-training". En: (2018).

[32] Vinod Nair y Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". En: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, págs. 807-814. ISBN: 9781605589077.

[33] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

[34] Jimmy Lei Ba, Jamie Ryan Kiros y Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[35] Navaneeth Bodla et al. *Soft-NMS – Improving Object Detection With One Line of Code*. 2017. arXiv: 1704.04503 [cs.CV].

[36] Gao Huang et al. *Snapshot Ensembles: Train 1, get M for free*. 2017. arXiv: 1704.00109 [cs.LG].

[37] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[38] Zhaohui Zheng et al. *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression*. 2019. arXiv: 1911.08287 [cs.CV].

[39] Dzmitry Bahdanau, Kyunghyun Cho y Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[40] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[41] Lucas Smaira et al. *A Short Note on the Kinetics-700-2020 Human Action Dataset*. 2020. arXiv: 2010.10864 [cs.CV].

[42] Olaf Ronneberger, Philipp Fischer y Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[43] Bolin Gao y Lacra Pavel. *On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning*. 2018. arXiv: 1704.00805 [math.OC].

[44] Hamid Rezatofighi et al. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*. 2019. arXiv: 1902.09630 [cs.CV].

Índice de figuras

Índice de tablas

Anexos

Anexo A Cálculo de precisión en problemas de Detección de Acciones

En la mayoría de *benchmarks* observados a lo largo del proyecto, se utilizará la misma metodología para la evaluación del rendimiento de los modelos, basada en el cálculo de la precisión con el uso del *Intersection over Union* o *IoU*, así como del *mean Average Precision* o *mAP*. El *IoU* es una métrica que nos permitirá conocer el solape entre dos intervalos, calculado como el cociente de su intersección y su unión. Su uso deriva principalmente del campo de la Detección de Objetos, en el que es necesario conocer la superposición entre las *Bounding Boxes* generadas y las instancias en la imagen. Siendo A y B dos intervalos, su IoU se calcularía utilizando la siguiente expresión:

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

Habitualmente, se utilizará este valor, equivalente al coeficiente de *Jaccard* para una dimensión, para considerar si un intervalo de los obtenidos por el modelo es una predicción correcta con respecto a uno de los encontrados en el *ground-truth*. Esto requerirá la elección manual de un *threshold*, si el IoU lo supera se considerará como un acierto. Sin embargo, la elección de este umbral no es trivial, si es muy bajo se aceptarán intervalos con un solape muy reducido, mientras que si es muy alto muchas predicciones correctas se ignorarán al calcular la precisión. Por ello, es habitual que la evaluación se realice obteniendo una media de los valores obtenidos para un rango predefinido de *thresholds*, por ejemplo $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ en *Epic Kitchens*.

Normalmente, las instancias generadas por un modelo estarán ordenadas por un valor de confianza, por lo que podremos valernos del *mean Average Precision* para la valoración del rendimiento. Esta métrica, utilizada en campos como la Recuperación de Información, evaluará la precisión dando mayor relevancia a los intervalos con una mayor puntuación. Para ello, se realizará una suma acumulativa de la precisión (*aciertos/intervalos recuperados*) cada vez que se recupere una muestra positiva, es decir, que el intervalo generado tenga un IoU mayor que el umbral determinado con cualquier instancia del *ground-truth* con su misma etiqueta. Finalmente, esa acumulación se dividirá por el número de intervalos relevantes generados.

Este proceso se repetirá para todas las distintas clases presentes en el *ground-truth* (ignorando el resto), así como con todos los valores de *threshold* elegidos, calculando la media para obtener la medida de precisión final del modelo. Cabe destacar que este cálculo se realiza de manera independiente para el modelo encargado de la predicción de los nombres y el de los verbos. En el Algoritmo 1 podemos ver una representación en alto nivel de esta métrica.

Algoritmo 1 Cálculo del *mean Average Precision*

$thresholds = \{0.1, 0.2, 0.3, 0.4, 0.5\}$
$classes = rango\ de\ clases$
$ground\_truth = Intervalos\ anotados$
$predicted = Intervalos\ generados,\ ordenados\ por\ puntuación$
for $threshold \in thresholds$ **do**
  for $class \in classes$ **do**
    $ground\_truth\_class = Intervalos\ de\ ground\_truth\ con\ etiqueta\ class$
    $predicted\_class\_intervals = Intervalos\ de\ predicted\ con\ etiqueta\ class$
    for $interval \in predicted\_class\_intervals$ **do**
      if $\exists\ gti \in ground\_truth\_class : IoU(interval, gti) \geq threshold$ **then**
        $precision\_sum\ +=\ aciertos\ /\ intervalos\ procesados$
      end if
    end for
    $average\_precision\_class = precision\_sum\ /\ |aciertos|$
  end for
  $mean\_average\_precision = \frac{1}{|classes|} \sum_{i=0}^{|classes|} average\_precision\_class[i]$
end for
$mAP\_global = \frac{i}{|thresholds|} \sum_{i=0}^{|thresholds|} mean\_average\_precision[i]$
return mAP_global

Anexo B Atención y auto-atención

La atención[39], es un mecanismo desarrollado en los últimos años en el campo del procesado del lenguaje natural, que permite modelar la relación entre las palabras de dos frases distintas, o de una misma en el caso de la auto-atención. Habitualmente se aplicaban en problemas *Seq2Seq*, en los que a partir de una secuencia de entrada de tamaño variable se busca generar otra de salida, también de longitud indefinida. Dichas técnicas fueron posteriormente adaptadas a distintas tareas, de manera que podrán ser utilizadas para aprender las correspondencias entre los distintos instantes temporales en una secuencia de vídeo, por ejemplo. En la mayoría de los casos, será necesario asignar una representación vectorial a cada elemento de las secuencias a procesar.

La aparición del método original se remonta al uso de redes neuronales recurrentes o *RNN*, y la dificultad que suponía implementar arquitecturas *Encoder-Decoder* debido a los problemas de memoria a largo plazo. Inicialmente, el codificador se encargaría de procesar secuencialmente la entrada, generando un estado oculto que sería utilizado por el decodificador para calcular la secuencia de salida. Al solo disponer del estado final del codificador, en caso de procesar sentencias largas o complejas, la información de sus primeros elementos se perdería. En la Figura 17 vemos una representación en alto nivel de su cálculo, que será más detallada a continuación. Siendo $\bar{X} = \{X_1, X_2, .., X_N\}$ la secuencia de entrada, $\bar{H} = \{h_0, h_1, .., h_N\}$ los estados ocultos en cada uno de los N nodos del codificador, $\bar{Y} = \{y_0, y_1, ..y_t\}$ la secuencia de salida y $S = \{s_0, s_1, .., s_t\}$ los correspondientes al decodificador, calculados hasta el paso t, para calcular el estado de $t+1$ deberemos obtener el *alignment score* (e) de cada $h_i \in H$ con respecto a s_t, que modelará su similitud con una red neuronal (a):

$$e_{t+1,i} = a(s_t, h_i) \tag{3}$$

Posteriormente, calcularemos el peso de cada estado, $\alpha_{t+1,i} = Softmax(e_{t+1,i})$, que será finalmente utilizado para generar un vector de contexto c, con una suma

ponderada, a partir del cual se obtendrá el estado oculto en $t+1$ (siendo f una de las neuronas de la *RNN*):

$$\begin{aligned} c_{t+1} &= \sum_{i=0}^{N} \alpha_{t+1,i} h_i \\ s_{t+1} &= f(s_t, y_t, c_{t+1}) \end{aligned} \tag{4}$$

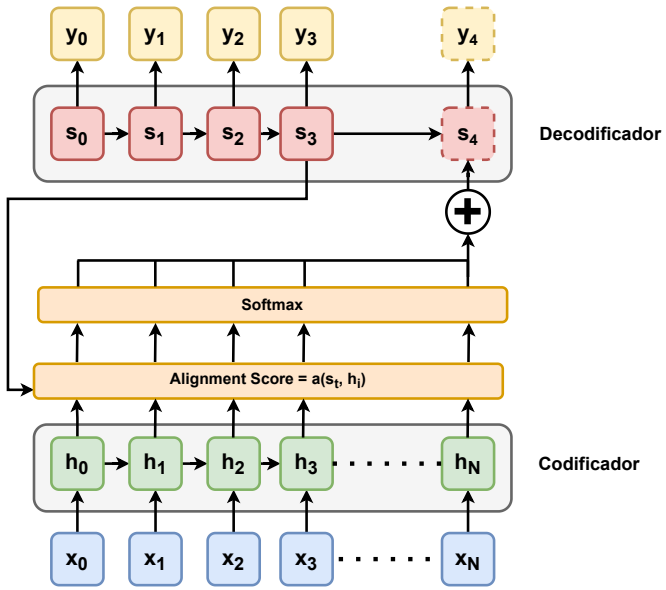

Figura 17: Cálculo de la atención en una arquitectura *Encoder-Decoder* basada en redes recurrentes. Utiliza una suma ponderada de los estados del codificador, teniendo en cuenta el último estado oculto del decodificador, para generar la nueva salida.

Por otro lado, el uso de la auto-atención permite modelar la relación entre elementos de una misma secuencia de forma global, independientemente de la distancia entre ellos, siendo el mecanismo principal en el que se basa la arquitectura *Transformer*[40]. Esta técnica no requerirá del uso de recurrencias, permitiendo aprovechar el paralelismo durante el entrenamiento. Además, evitará la aparición de problemas de desvanecimiento del gradiente. Para su cálculo, el modelo deberá generar 3 representaciones distintas de los datos, de manera que para cada vector original, se calcule una *consulta*, una *clave* y un *valor*. La *query* de cada elemento de la entrada se comparará con la *key* del resto para modelar su similitud, a través del producto escalar o *dot product*, obteniendo un peso para cada relación utilizando la función *Softmax*. Finalmente, esos pesos se utilizarán para calcular una suma ponderada de los valores de todos los elementos de la secuencia, obteniéndose así los valores de atención para el elemento con el que se realizó la consulta. En la Figura 18, podemos observar una representación gráfica de alto nivel de dicho proceso.

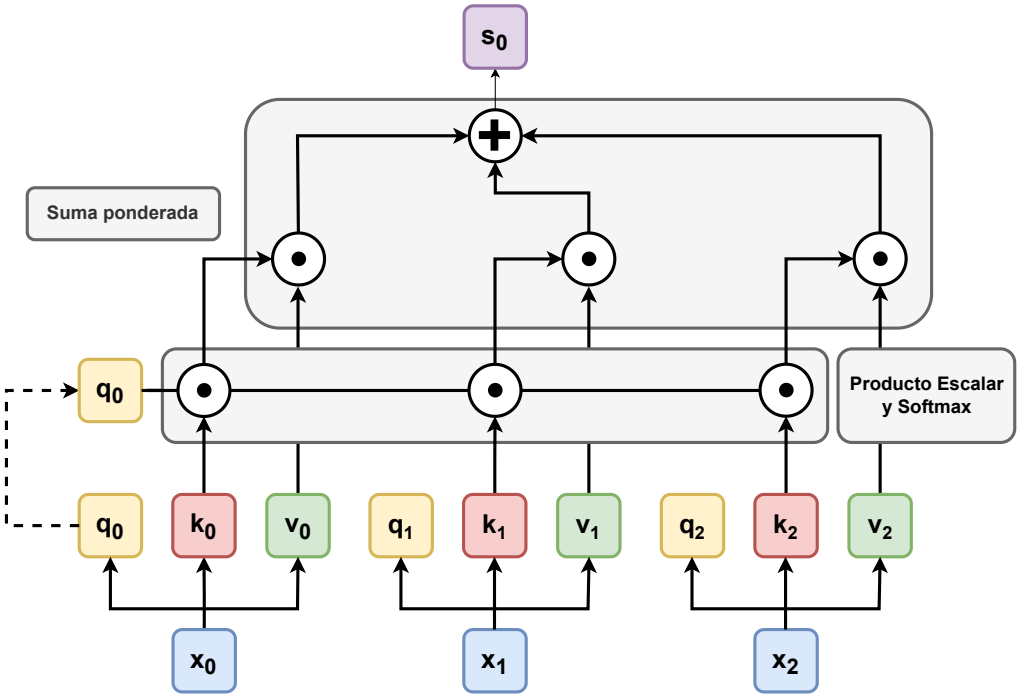

Figura 18: Diagrama de cálculo de la auto-atención, en el que se utiliza la *query* de una de las entradas para obtener un peso que mida su similitud con el resto, a través de una operación de producto escalar con sus *keys*. Tras una operación de *Softmax*, se utilizarán esos pesos para realizar una suma ponderada de los distintos *values*, obteniéndose así los valores de atención para dicho elemento. Figura adaptada de *Illustrated: Self-Attention*.

El cálculo se puede resolver de forma matricial, considerando $X \in \mathbb{R}^{NxD}$ el conjunto de los N vectores (de longitud D) de entrada original, $Q \in \mathbb{R}^{N\times D_q}$, $K \in \mathbb{R}^{N\times D_q}$ y $V \in \mathbb{R}^{N\times D_v}$, las matrices con la *query*, *key* y *value* respectivo a cada uno, calculadas utilizando 3 matrices paramétricas $W_Q \in \mathbb{R}^{D\times D_q}$, $W_K \in \mathbb{R}^{D\times D_q}$ y $W_V \in \mathbb{R}^{N\times D_v}$, que serán aprendidas por el modelo utilizando 3 subredes independientes, tal que $Q = XW_Q,\ K = XW_K,\ V = XW_V$, la matriz de atención $S \in \mathbb{R}^{N\times D_v}$, que contendrá un vector para cada elemento que indique su relación con el resto, se calculará con:

$$S = Softmax(QK^T/\sqrt{D_q})V \tag{5}$$

De manera que la función de *Softmax* se aplicará por filas, y se incluirá una división por la raíz cuadrada de la longitud de las *queries*, como factor de escalado. Habitualmente, con el objetivo de mejorar la precisión, el proceso se repetirá h veces, utilizando para cada una de ellas una proyección (W_Q, W_K, W_V) diferente, concatenando los resultados finales. Dicha práctica se conoce como *Multi-Head Self-Attention* o *MSA*.

Como hemos comentado anteriormente, la arquitectura *Transformer*[40], principal inspiración del modelo estudiado (ActionFormer[4]), se basa en el uso de auto-atención para resolver problemas *Seq2Seq*. En la Figura 19 podemos ver un diagrama de alto nivel de su estructura, que estará compuesta por un codificador (izquierda) y un decodificador (derecha). El primero añadirá información posicional a los elementos de la secuencia de entrada, y calculará su similitud utilizando una capa de *MSA*. La siguiente capa utilizará una red neuronal completamente conectada, y ambas tendrán una conexión residual con sus entradas. Por otro lado, el decodificador trabajará de igual manera, añadiendo un bloque de atención enmascarada, de forma que al predecir un nuevo elemento de la secuencia, el modelo solo conozca los valores de salida anteriores. Además, el segundo bloque de *MSA* incluirá también la información generada por el codificador.

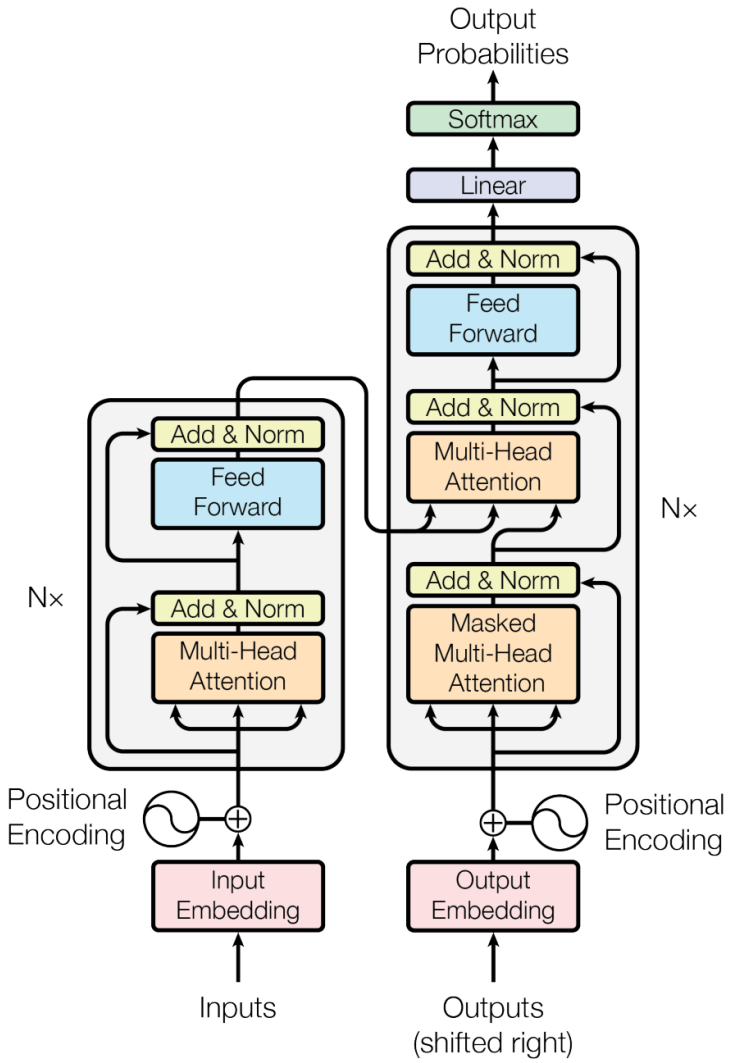

Figura 19: Diagrama de alto nivel de la arquitectura *Transformer*. Figura extraída de [40]

Anexo C Detalles de implementación

A continuación, detallaremos el proceso de entrenamiento y evaluación del modelo con el conjunto de vídeos de *BSH*, profundizando en los procesos de adaptación de las anotaciones y de extracción de *features* necesarios para su uso con *ActionFormer*[4], así como en la depuración realizada con el objetivo de mejorar los resultados de precisión obtenidos.

C.1 Adaptación de los datos de entrada

Nuestro *dataset* está compuesto por 91 vídeos anotados, que fueron grabados por 3 participantes distintos. Dichas anotaciones, realizadas por Alejandro López[1] como parte de un trabajo anterior en el mismo proyecto de *BSH* y el *Graphics and Imaging Lab*, constarán de un instante de inicio y finalización, así como las etiquetas del nombre y verbo correspondientes de la acción realizada en ese intervalo, utilizando las clases definidas por *Epic Kitchens*[2]. Para poder adaptarlas para su uso con el modelo, fue necesaria la implementación de un *script* que tomaba como entrada el fichero *csv* original y lo transformaba a otros dos en formato *JSON*. Uno de ellos recogía las etiquetas correspondientes a los verbos y otro a los

nombres. Cabe destacar que el funcionamiento del programa comentado, así como el de cualquier otro código auxiliar implementado, se detallará en el Anexo H.

C.2 Extracción de *features*

Una vez disponíamos de unas anotaciones aptas para el modelo, pasamos al proceso de extracción de *features*. Para ello, hicimos uso de *GluonCV*[6], un conjunto de herramientas de Aprendizaje Profundo que proporciona distintas implementaciones de modelos aplicados a tareas de Visión por Computador. Entre ellas, utilizaremos el extractor implementado para tareas de Reconocimiento de Acciones, que permite el uso de distintos modelos preentrenados, de los cuales emplearemos *SlowFast*al ser el mismo usado al entrenar con *Epic Kitchens*. El modelo usado fue preentrenado con el *dataset Kinetics*[41], ya que la herramienta no permitía realizar dicho proceso de forma manual y los modelos disponibles en el *Model Zoo* no utilizaban *Epic Kitchens*.

Debido a la implementación de *GluonCV*, el extractor no era capaz de tratar vídeos de una alta duración, por lo que fue necesario implementar un *script* que dividiera las entradas en los distintos fragmentos a procesar. A causa de estas limitaciones, tampoco fue posible utilizar un *temporal stride* de 16 *frames* para el *Slow Pathway* de *SlowFast* (valor empleado para *Epic Kitchens*), utilizando en su lugar uno de *8* frames. Aun así, ya que el uso de un *stride* menor implica la extracción de un número mayor de fotogramas, y por tanto, una mayor precisión de las *features* obtenidas, no se consideró que fuese un factor que repercutiera negativamente sobre el rendimiento. El programa de fragmentación implementado, tomará como entrada la carpeta con el conjunto de vídeos, de manera que cada uno de ellos se dividirá en *clips* de 32 *frames* de longitud, extraídos con una separación de 16 *frames*. Es decir, habrá un solape de 16 fotogramas entre fragmentos contiguos. Además, con el objetivo de obtener unas *features* todo lo similares posibles a *Epic Kitchens*, el programa transformaba también los vídeos de 25 *fps* originales a 30 *fps*. Así mismo, se realizó una reducción de la resolución, pasando de los 640 por 480 píxeles a 340 por 256, más cercana a la usada por *Epic Kitchens* (456 por 256).

Tras la extracción de los fragmentos, haremos uso de la herramienta *GluonCV* que generará un vector de *features* de dimensión 2304 para cada uno. Finalmente, se aplicará un *script* que agrupará las *features* generadas según el vídeo de origen, manteniendo el orden de aparición original. Cabe destacar que, antes de utilizar la herramienta *GluonCV* se intentó utilizar *SlowFast Feature Extractor* pero su uso fue descartado, debido principalmente a la escasa existencia de documentación y la inactividad de la comunidad de desarrollo, que dificultaron en gran medida su uso. Además, la versión de *SlowFast* utilizada en su implementación no coincidía correctamente con la versión actual, por lo que eran necesarias modificaciones manuales al código para su uso.

El proceso de extracción de *features* fue altamente costoso temporalmente, principalmente por problemas durante la instalación, resueltos con la ayuda de los administradores de los equipos utilizados, teniendo en cuenta también el tiempo invertido en el uso de una herramienta que finalmente fue desechada. Además, una vez instalado *GluonCV* y adaptados los vídeos para su uso, la propia extracción de *features* tenía una alta duración, lo cual entorpecía el avance en caso de que fuese necesaria una reconfiguración.

C.3 Entrenamiento y depuración

Al tener disponibles las *features* de cada uno de los fragmentos de los vídeos de nuestro *dataset*, pasamos al proceso de entrenamiento de *ActionFormer*[4]. De esta manera, buscaremos verificar nuestra hipótesis inicial de que un modelo capaz de trabajar con un conjunto complicado como *Epic Kitchens*[2], era capaz

de trabajar con otro con un rango de acciones más reducido, una cámara fija y una densidad temporal de anotaciones mucho menor. Al igual que al utilizar *Epic Kitchens*, fue necesario entrenar dos modelos completamente independientes, uno para las anotaciones de los verbos de las acciones, y otro para los nombres. En la Tabla 10, podemos observar los resultados obtenidos inicialmente, junto a los logrados con *Epic Kitchens* como comparación. Vemos como nuestros resultados se alejan enormemente de los esperados, obteniendo una precisión de alrededor del 4 o 5 %, mientras que el modelo era capaz de superar el 20 % con otros vídeos de entrada.

Dataset	**Average mAP Verbos**	**Average mAP Nombres**
Epic Kitchens	23.26 %	22.12 %
BSH	5.31 %	3.97 %

Tabla 10: Comparación de los resultados del entrenamiento del modelo con *Epic Kitchens* tras 21 épocas, y los vídeos de *BSH* tras 105. Vemos como los resultados son muy alejados a los esperados, ya que al estar compuesto nuestro *dataset* por un conjunto más reducido de acciones, tener una cámara fija y una menor densidad temporal de acciones, confiábamos obtener una precisión superior.

En primera instancia, planteamos que una de las principales razones para el reducido rendimiento obtenido eran las características del *dataset*. Nuestros vídeos son mucho menos variados que los de *Epic Kitchens*, con un rango más reducido de acciones utilizadas y un menor número instancias para cada una de ellas. Es por esto, que nuestra primera aproximación para mejorar los resultados fue la aplicación de técnicas de *Fine Tuning*. El *Fine Tuning* consiste en el entrenamiento de un modelo con un conjunto de datos auxiliar, de manera que los pesos calculados por la red se almacenen, y se utilicen como valores iniciales en un segundo entrenamiento, realizado esta vez sobre los vídeos que realmente se quieren procesar. De esta manera, se facilitará la correcta convergencia del modelo, al disponer de un mejor punto de partida.

Siguiendo esta metodología, realizamos un preentrenamiento con *Epic Kitchens*, *dataset* para el cual *ActionFormer* obtiene resultados considerablemente mejores y que se puede considerar como una generalización de nuestro conjunto de datos. Como podemos ver en la Tabla 11, aunque los resultados obtenidos supusieron una mejora, seguían siendo mucho menores de lo esperado.

Dataset	**Average mAP Verbos**	**Average mAP Nombres**
Epic Kitchens	23.26 %	22.12 %
BSH	5.31 %	3.97 %
BSH + Fine Tuning	7.72 %	4.67 %

Tabla 11: Comparación de los resultados de entrenamiento obtenidos utilizando *Fine Tuning* con los iniciales, así como con *Epic Kitchens*. Vemos un ligero incremento de la precisión, más de un 2 % para los verbos y 0.70 % para los nombres, pero el rendimiento aún se aleja mucho del esperado.

Ya que los valores conseguidos se alejaban mucho de los obtenidos al entrenar con *Epic Kitchens*, decidimos implementar un *script* que creará una gráfica mostrando los intervalos reales (*ground-truth*) del vídeo, junto a los predichos por el modelo. Así, pudimos evaluar de manera cualitativa la precisión de la detección y la clasificación de la red. Gracias a estas gráficas, observamos como las acciones

devueltas como resultado no abarcaban la longitud temporal completa del vídeo y únicamente se concentraban en el primer tercio de su duración. Podemos ver un ejemplo de este comportamiento en la Figura 20.

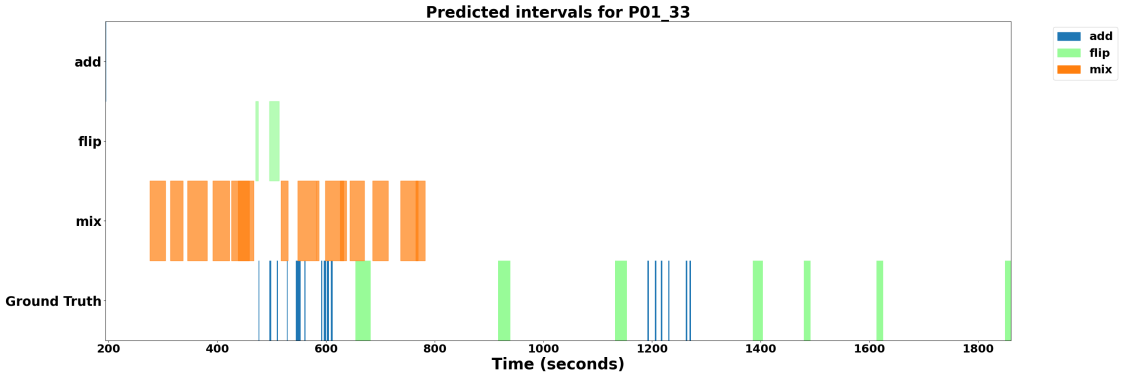

Figura 20: Gráfico con las predicciones originales obtenidas para el vídeo 33. Como podemos ver, todas ellas están presentes en el primer tercio de la duración, lo cual indica un error en la configuración temporal.

Esto, apuntaba a un fallo en la configuración temporal del modelo o del extractor de *features*. Tras un período de depuración, conseguimos encontrar dos errores, el primero de ellos en el *script* de fragmentación, ya que no se había tenido en cuenta el solape entre *clips* contiguos. El segundo se encontraba en los parámetros de *ActionFormer*, donde habíamos especificado que los vídeos de entrada tenían una tasa de refresco de 30 *fps*, pero las anotaciones originales se habían realizado teniendo en cuenta un *frame rate* de 25 *fps*. Como podemos observar en la Tabla 12, los valores de precisión aumentaron considerablemente, superando a los obtenidos con *Epic Kitchens* al utilizar *Fine Tuning*. El incremento en rendimiento obtenido gracias a este preentrenamiento es significativo (1.5 % para verbos y 8 % para los nombres), por lo que la consideración de que nuestro *dataset* es una simplificación de *Epic Kitchens* es correcta. Por otro lado, en la Figura 21 aparecen los intervalos predichos para el mismo vídeo de la Figura 20, observándose fácilmente la mejora en la detección y la clasificación.

Dataset	**Average mAP Verbos**	**Average mAP Nombres**
Epic Kitchens	23.26 %	22.12 %
BSH (sin corregir)	7.72 %	4.67 %
BSH	32 %	16 %
BSH + Fine Tuning	33.5 %	24 %

Tabla 12: Comparación del *average mAP* obtenido con la nueva configuración del modelo junto a los resultados iniciales, ambos obtenidos tras 100 épocas de entrenamiento, y los obtenidos con *Epic Kitchens*. Como podemos ver, se ha producido un gran incremento en la precisión para ambas clases, principalmente en el caso de los verbos. Además, al utilizar *Fine Tuning*, el modelo conseguirá superar tanto para verbos como para nombres los resultados del *benchmark*.

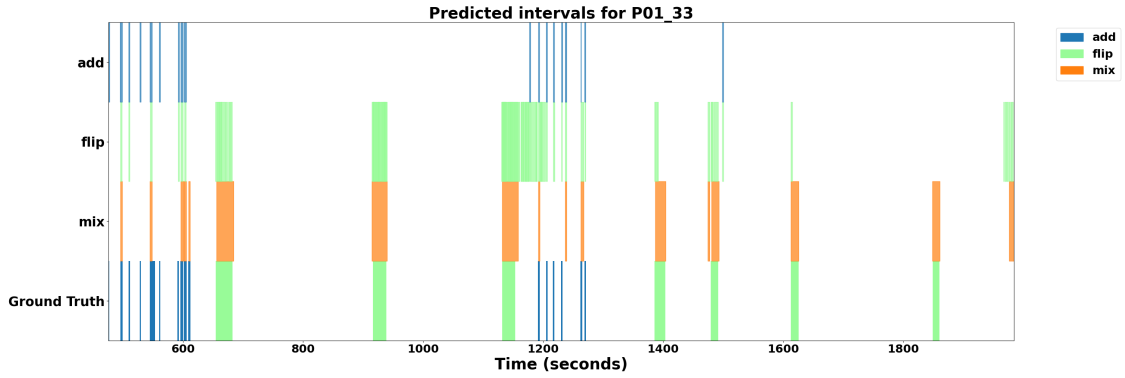

Figura 21: Gráfico con las predicciones obtenidas para el vídeo 33 una vez modificada la corrección del modelo. Como podemos ver, los intervalos se generan a lo largo de toda la duración, y se ajustan razonablemente bien a las instancias del *ground-truth*, a diferencia de los vistos en la Figura 20.

Anexo D Experimentos adicionales

A lo largo de este Anexo comentaremos distintas pruebas llevadas a cabo a lo largo del desarrollo del proyecto, que no se han incluido en el texto principal al ser poco relevantes o al haber producido resultados peores a los esperados.

D.1 Reducción del rango de clases

Como vimos durante el análisis de los conjuntos de datos estudiados (Sección 2.2), nuestro *dataset* de vídeos disponía únicamente de 11 clases distintas para los verbos de las acciones, y otras 36 para los nombres (18 y 51 tras el reetiquetado), pero nuestras anotaciones utilizaban directamente las 97 y 300 del rango de etiquetas completo de *Epic Kitchens*[2], por simplicidad y facilidad de uso de las mismas con modelos adaptados a este.

Por lo tanto, existía un gran número de clases inservibles, que no aparecían ni en el subconjunto de entrenamiento ni de validación, por lo que cualquier predicción que la red realizase de una de ellas supondría añadir una confusión que podría ser fácilmente descartada. De esta manera, decidimos implementar una serie de *scripts*, *reduce_label_range.py* y *remove_unused_actions.py* (cuyo funcionamiento se detalla en el Anexo H). El primero transformará las anotaciones utilizadas de manera que utilicen un nuevo rango de etiquetas, numeradas desde 0 hasta el número de clases menos 1. El segundo eliminará cualquier instancia en las anotaciones de *Epic Kitchens* que no utilicen alguna de las etiquetas presentes en nuestro *dataset* y modificará el resto para que se utilice el nuevo rango, permitiendo así utilizarlas para el preentrenamiento del modelo a través del *Fine Tuning*. En la Tabla 13 podemos observar como el uso de este nuevo rango supuso un ligero aumento de precisión en el caso de los verbos (alrededor de 1 %), y un pequeño descenso para los nombres (cerca de 0.7 %). Por otro lado, el uso de *Fine Tuning* no ayudó a mejorar el rendimiento, siendo muy perjudicial para el rendimiento con los verbos.

Etiquetas	mAP Verbos	mAP Nombres
Rango original	36.38 %	15.02 %
Rango reducido	38.56 %	14.33 %
Rango reducido y Fine Tuning	32.41 %	14.30 %

Tabla 13: Valores de precisión obtenidos al utilizar un rango de clases reducido, frente al original propuesto por *Epic Kitchens*, utilizando las anotaciones tras el proceso de reetiquetado.

Debido a las mínimas diferencias en los resultados, consideramos que esta reducción de rango no constituía una modificación considerable del funcionamiento del modelo, lo cual puede estar relacionado con el método del cálculo de la precisión comentado en el Anexo A, ya que cualquier instancia generada que no pertenezca a una clase presente en el *ground-truth* será ignorada. De esta forma, dichas predicciones no tendrán un impacto significativo en el rendimiento, por lo que el uso de un rango más específico no supondrá apenas mejora.

D.2 Implementación de un modelo mixto

Como vimos durante el estudio del estado del arte, más específicamente en el análisis de las tendencias futuras de la Detección de Acciones (Sección 2.1.4), recientemente han aparecido una serie de publicaciones (*MetaFormer*[21] y [22]) que presentaban los distintos problemas del uso de la auto-atención en tareas dedicadas al tratamiento de secuencias de vídeo, debido a la similitud entre las *features* utilizadas como entrada, causando pérdidas de información. Además, añade una alta complejidad a los modelos. Estos artículos proponen que el buen funcionamiento de las arquitecturas basadas en la atención como *ActionFormer*[4] no se deben a su uso, sino a la estructura general propuesta por los *Transformers*. A partir de estos estudios surgen una serie de modelos, principalmente *TriDet*[23] y *TemporalMaxer*[24], que mantienen la arquitectura principal de *ActionFormer*, sustituyendo los bloques de auto-atención por otros que permiten el cálculo de las relaciones temporales de manera más eficiente. El primero de ellos, usará un bloque compuesto por varias capas convoluciones, mientras que el segundo empleará únicamente una operación de *MaxPooling*. Como vimos en la Tabla 1, ambos obtienen mejores resultados en el *benchmark* de *Epic Kitchens*[2] con implementaciones más sencillas que *ActionFormer*.

Debido a esto, quisimos proponer un modelo híbrido, que hiciese uso de las técnicas convolucionales con el objetivo de preservar las diferencias entre las *features* de los distintos fragmentos de vídeo procesados, pero que aprovechase también las capacidades de la auto-atención para modelar relaciones contextuales a largo plazo. De tal manera se planteó una arquitectura *U-shaped*, originalmente propuesta por *U-Net*[42], compuesta por una rama inicial conocida como codificador, que realizará un *downsampling* de la entrada original, así como un decodificador que tomará como entrada la salida del codificador y aplicará operaciones de *upsampling*, además de una serie de conexiones residuales. Así, el nuevo modelo estará compuesto por dos pirámides de *features*, la primera de ellas utilizará las operaciones de *MaxPooling* ofrecidas por *TemporalMaxer*, mientras que la segunda empleará los bloques de auto-atención proporcionados por *ActionFormer*, realizando además un aumento de la resolución temporal. Finalmente, los resultados obtenidos por la segunda rama serán los utilizados para la detección y la clasificación. Podemos observar un diagrama de alto nivel de esta arquitectura en la Figura 22.

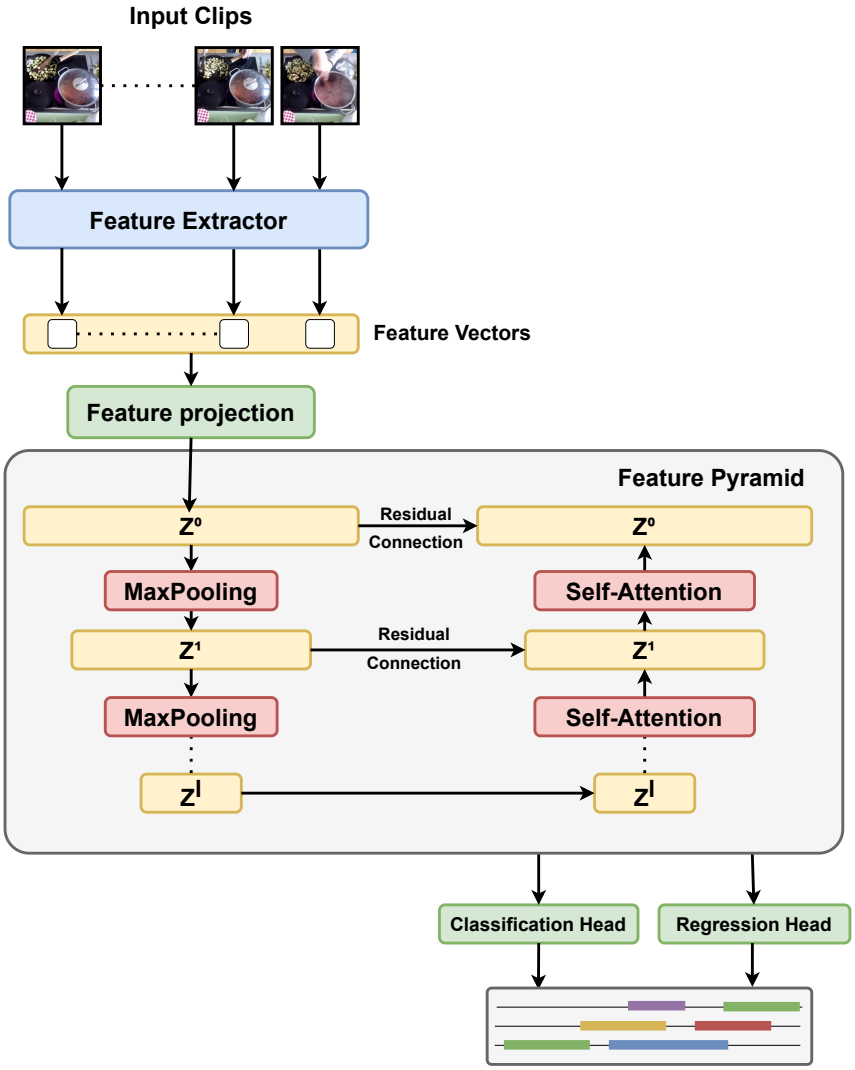

Figura 22: Arquitectura del modelo mixto implementado, compuesto por dos pirámides de *features*, una que realiza un *downsampling* temporal, utilizando bloques de *MaxPooling*, y otra que realizará el *upsampling*, usando auto-atención.

Para controlar la relevancia de la conexión residual, se añadió un parámetro adicional $\alpha \in [0, 1]$, de manera que cuanto mayor sea su valor, más importancia se le dará. En la Tabla 14 podemos observar los resultados obtenidos según distintos valores de α, utilizando las anotaciones originales. La precisión conseguida para los verbos por lo general superará a la original, siendo el mejor resultado el logrado con $\alpha = 0.3$, pero el modelo estará muy lejos del rendimiento deseado en el caso de los nombres, por lo que no será una solución adecuada para el problema. Además, al intentar utilizar un preentrenamiento los resultados no mejoraron, reduciéndose la precisión a 6.30 %. Aun así, la implementación demuestra cierto potencial en la idea de combinar mecanismos de atención con operaciones convolucionales para modelar relaciones temporales a largo plazo y podría ser viable para una investigación futura con una arquitectura más refinada.

Valor de α	*Average mAP Verbos*	*Average mAP Nombres*
0	9.12 %	4.99 %
0.3	36.01 %	13.75 %
0.5	32.11 %	12.77 %
0.7	33.64 %	8.00 %
1	32.29 %	9.98 %
Modelo original	33.5 %	24 %

Tabla 14: Precisión obtenida con el modelo mixto al utilizar distintos valores de α, en comparación a los resultados con *ActionFormer* utilizando *Fine Tuning*

Anexo E *Non Maximum Supression*

El *Non Maximum Supression*[35] es un algoritmo de refinado de intervalos originado en el campo de la Detección de Objetos, que tiene como objetivo eliminar redundancias en las instancias generadas por el modelo, manteniendo aquellas que obtengan una mayor puntuación de confianza. La primera versión, conocida como *Hard Non Maximum Supression* (Algoritmo 2) consistirá en la elección iterativa de los intervalos de una misma clase, ordenados de manera decreciente por su *score*. Cualquier otro intervalo con una puntuación menor, que tenga un solape (calculado con el *Intersection over Union*, Anexo A) mayor a un cierto *threshold* será eliminado. Debido a esto, el algoritmo es altamente sensible al valor de umbral elegido, y podría eliminar instancias con una alta confianza de manera indiscriminada. Podemos ver un ejemplo de aplicación en la Figura 23.

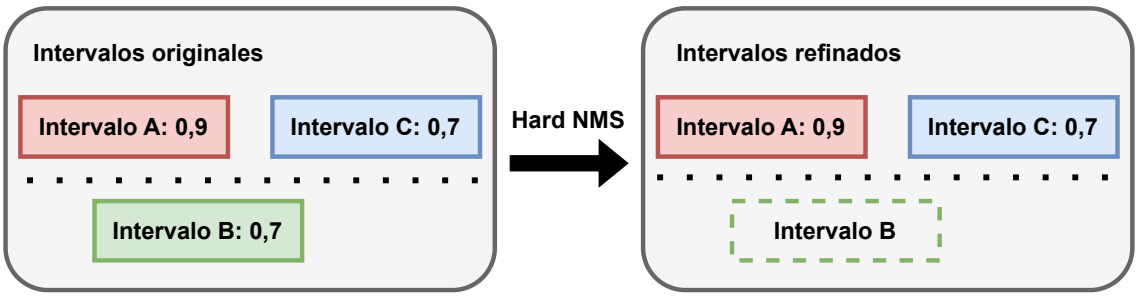

Figura 23: Ejemplo de aplicación del algoritmo básico de *Non Maximum Supression*, podemos ver como el intervalo B, al tener un gran solape con el A y una menor puntuación, se considera redundante, por lo que es eliminado.

Algoritmo 2 *Non Maximum Supression*

NMS(*intervalos, threshold*)
intervalos ← *sort*(*intervalos*, *intervalos.confianza*)
intervalos_fin ← ∅
for *i* ∈ *intervalos* **do**
  intervalos_fin ← *intervalos_fin* ∪ *i*
  posibles_solapes ← *intervalos* − *i*
  for *i_s* ∈ *posibles_solapes* **do**
    if *IoU*(*i_s*, *i*) ≥ *threshold* **then**
      intervalos ← *intervalos* − *i_s*
    end if
  end for
  intevalos ← *intervalos* − *i*
end for
return *intervalos_fin*

Otra aproximación, que recibe el nombre de *Soft NMS (Algoritmo 3)*, no eliminará los intervalos que superen un cierto solape, sino que realizará una reducción de su puntuación proporcional al *IoU* entre ellos. Así, evitaremos la eliminación de posibles instancias relevantes, dando una mayor importancia a los que mejor *score* hayan obtenido. En la Figura 24 observamos un ejemplo de su aplicación. Cabe destacar también la existencia del *Multiclass NMS*, que hará comparaciones independientes a las etiquetas, evitando redundancias temporales entre clases.

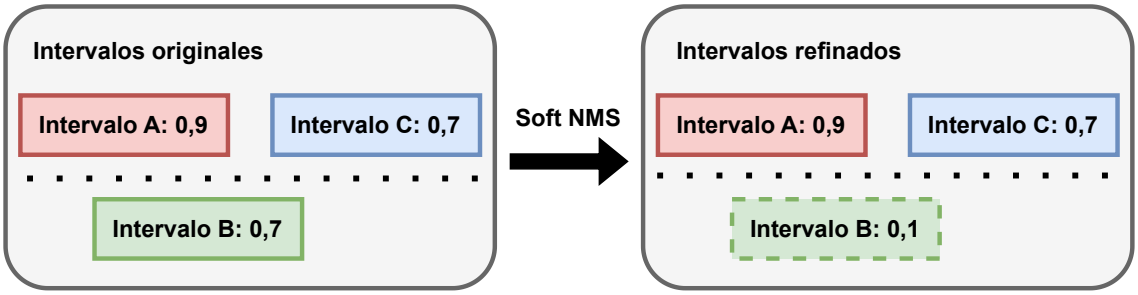

Figura 24: Ejemplo de aplicación del algoritmo de *Soft Non Maximum Supression*, en el que vemos como al existir un intervalo redundante (B), su puntuación se reduce en función al *Intersection over Union* con el intervalo de mayor confianza (A).

Algoritmo 3 *Soft Non Maximum Supression*

SoftNMS(*intervalos, threshold*)
intervalos ← *sort*(*intervalos*, *intervalos.confianza*)
for *i* ∈ *intervalos* **do**
  posibles_solapes ← {*i_s* ∈ *intervalos* : *i_s.confianza* ≤ *i.confianza*}
  for *i_s* ∈ *posibles_solapes* **do**
    if *IoU*(*i_s*, *i*) ≥ *threshold* **then**
      i_s.confianza ← *i_s.confianza*(1 − *IoU*(*i_s*, *i*))
    end if
  end for
end for
return *intervalos*

Anexo F Funciones de activación utilizadas

Durante este Anexo, detallaremos las funciones de activación utilizadas e introducidas brevemente a lo largo de la memoria. Habitualmente, se denominan funciones de activación a aquellas que a partir de los valores de entrada y los pesos correspondientes definirán el valor de salida transmitido, permitiendo al modelo aprender sobre la estructura de los datos procesados.

- Función sigmoide: función no lineal, también conocida como logística, que permite el aprendizaje de estructuras complejas en los datos, transformando la entrada a una salida en el rango $[0.0, 1.0]$, como podemos ver en la Figura 25. Habitualmente, no se utilizará en las capas profundas de las redes neuronales, ya que tienden a saturarse con valores grandes (que pasarán a valer 1.0) o pequeños (que pasarán a valer 0.0), siendo únicamente sensibles a un rango limitado. Además, los errores generados por su derivada serán muy bajos, causando que los gradientes sean cada vez más reducidos a medida que profundicemos en las capas del modelo. Este problema, conocido como *vanishing gradient* puede ralentizar o parar por completo el proceso de entrenamiento. Debido a esto, se suelen utilizar en la última capa, con el objetivo de aprender una distribución probabilística que genere un valor entre 0 y 1, permitiendo clasificar las entradas. Para el cálculo de esta función se utilizará la siguiente expresión:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

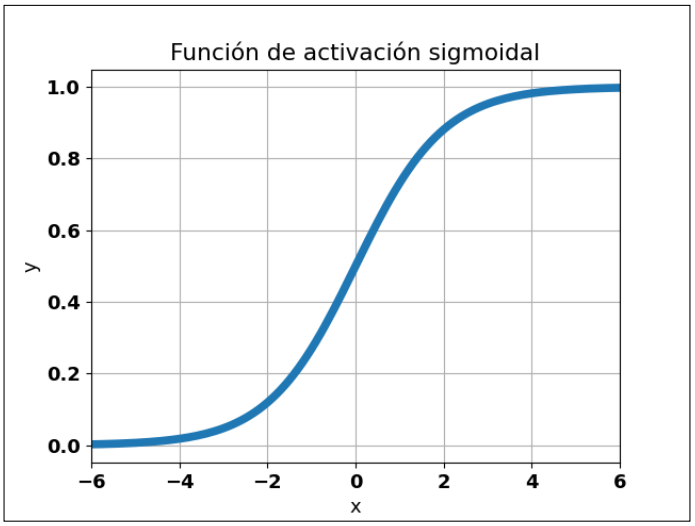

Figura 25: Función sigmoide

- *Rectifier Linear Unit* o *ReLU*[32]: aparece como sustitución de las funciones de activación no lineales para capas profundas, y evita la aparición de problemas de desvanecimiento del gradiente. Esta devolverá el valor de entrada en caso de que sea mayor que 0, o 0 si no lo fuera, como vemos en la Figura 26. Destacan por su rapidez de cálculo, así como su comportamiento casi-lineal, que facilita las tareas de optimización y el uso del algoritmo de *backpropagation*.

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

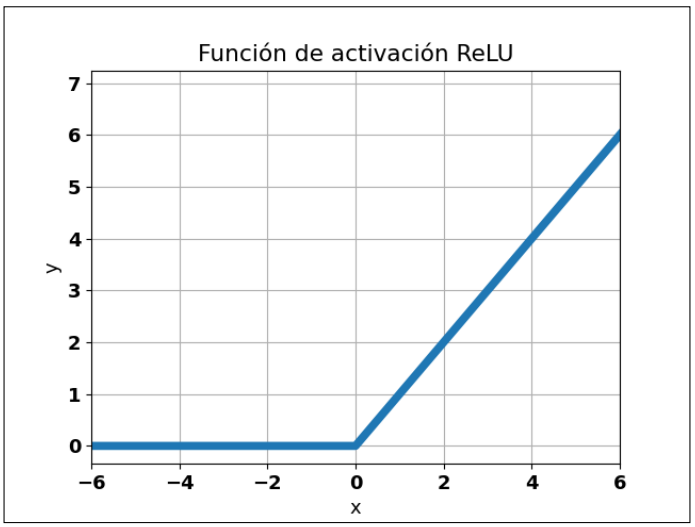

Figura 26: Función ReLU

- Función *Softmax*[43]: función de activación utilizada habitualmente para generar la salida en redes de clasificación multiclase, que generará un vector con la probabilidad de que la muestra pertenezca a cada una de las clases existentes. Se puede considerar una generalización de la función sigmoide, que generaba un valor entre 0 y 1 para una única entrada. En este caso, dado un conjunto de *inputs*, se generarán los valores correspondientes de manera que la suma de todos ellos siempre sea 1, creando así una distribución probabilística que nos indicará el más probable. También se pueden utilizar puntualmente en capas profundas, con el objetivo de normalizar una secuencia de datos. Para obtener el valor de *softmax* para cada elemento de una entrada de tamaño I, se utilizará la siguiente expresión:

$$Softmax(x_i) = \frac{e^{x_i}}{\Sigma_{j=0}^{I} e^{x_j}} \tag{8}$$

Anexo G Funciones de pérdida utilizadas

A lo largo del presente Anexo, presentaremos en mayor profundidad las distintas funciones de pérdida utilizadas por el modelo. Estas se utilizarán para calcular un valor de error sobre las predicciones generadas durante el entrenamiento, de manera que podamos recalcular los pesos con el objetivo de minimizarlas.

- Función de *Cross Entropy*[37]: función de error para problemas de clasificación binaria, también generalizable a problemas multiclase, que genera valores altos de error cuando la probabilidad de pertenecer a la clase es baja y la muestra está etiquetada como positiva, y cuando esta es elevada pero la etiqueta es negativa. Para su cálculo se utilizará la siguiente expresión:

$$CrossEntropy(x, y) = \begin{cases} -\log(x), & \text{if } y = 1 \\ -\log(1-x), & \text{if } y = 0 \end{cases} \tag{9}$$

  De esta manera, el error se calculará teniendo en cuenta el valor de probabilidad de la clase, generando también valores no despreciables para casos fácilmente clasificables, lo cual puede causar que el error originado por una gran cantidad de muestras triviales eclipse al generado por un pequeño grupo de casos menos comunes. Este obstáculo se puede ver más agravado en el caso de una clasificación multiclase, en la que puede existir un desbalance entre las distintas clases. Cabe destacar que este comportamiento podría ser deseado si

lo que se busca es una función robusta ante datos espurios. En la Figura 27 podemos observar un gráfico con la función de *Cross Entropy* ($\gamma = 0$), entre otras.

- Función de *Focal Loss*[37]: modificación de la función de *Cross Entropy*, aplicable a problemas de clasificación binaria y multiclase, que busca equilibrar los valores de pérdida en conjuntos de datos desbalanceados, añadiendo un término que dé una mayor importancia a las muestras poco comunes, regulado por una constante γ. La función seguirá la expresión:

$$FocalLoss(x, y) = \begin{cases} -(1-x)^{\gamma}\log(x), & \text{if } y = 1 \\ -(x)^{\gamma}\log(1-x), & \text{if } y = 0 \end{cases} \tag{10}$$

Cuanto mayor sea el valor de γ mayor importancia (como podemos ver en la Figura 27) se dará a las clases poco comunes o a las muestras difíciles de clasificar, de manera que conseguiremos una función de error capaz de aprender en distribuciones de datos con un desbalance considerable.

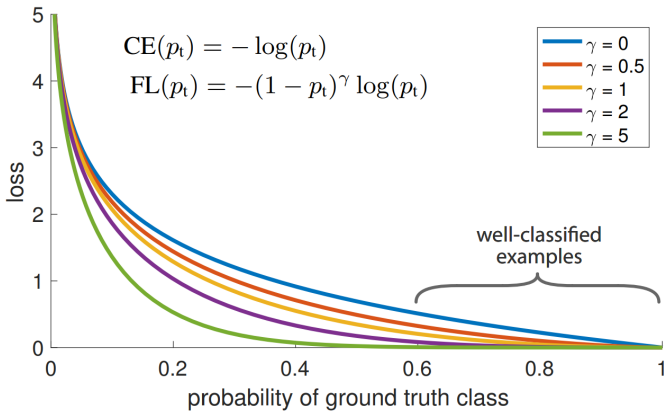

Figura 27: Función de *Focal Loss* según los valores de la constante reguladora γ. Podemos ver como a medida esta aumenta, el error generado para muestras fácilmente clasificables disminuye, de manera que será más difícil que eclipsen a los casos poco comunes. Observamos como al utilizar $\gamma = 0$, la función será equivalente a *Cross Entropy*. Figura extraída de [37].

- Funciones basadas en *Intersection over Union*: en los problemas de Detección de Acciones buscaremos generar intervalos temporales lo más similares posible a los anotados. Para conseguirlo, se utilizarán funciones de pérdida basadas en la métrica de *Intersection over Union* (Anexo A), adaptadas a una dimensión a partir de las ya existentes para Detección de Objetos, que tiene como objetivo generar *Bounding Boxes* ajustadas al área real que ocupan los objetos. El principal problema a la hora de utilizar esta puntuación como función de error ($\mathcal{L}_{IoU} = 1 - IoU$) reside en que si el intervalo generado y el real no tienen superposición alguna, el valor será 0, por lo que el modelo se quedará estancado durante el entrenamiento. Debido a esto, han aparecido diversas funciones alternativas, que evitarán los valores nulos. La primera de ellas, conocida como *Generalized IoU* [44], añade un término de penalización a la métrica original. Dados dos intervalos A y B, el intervalo C será el de mínimo tamaño que englobe tanto a A como a B. De esta manera, se aplicará una penalización según la proporción del área de C que no esté ocupada por A ni por B, como observamos en la siguiente expresión:

$$\mathcal{L}_{GIoU}(A, B) = 1 - IoU(A, B) + \frac{|C - (A \cup B)|}{|C|} \tag{11}$$

Pese a ser una aproximación mucho más robusta, es necesario realizar un alto número de iteraciones para alcanzar la convergencia, ya que cuando el área $C - (A \cup B)$ es pequeña tenderá a funcionar igual que IoU. Otra alternativa, conocida como *Distance IoU*[38] (la utilizada por *ActionFormer*) aplicará una penalización que buscará minimizar la distancia normalizada entre los puntos centrales de los intervalos, consiguiendo converger a mayor velocidad. Si consideramos p_A y p_B los puntos centrales de A y B, c la longitud del intervalo C y d_e la distancia euclídea entre dos puntos, la pérdida se calcularía de la siguiente forma:

$$\mathcal{L}_{DIoU}(A, B) = 1 - IoU(A, B) + \frac{d_e(p_A, p_B)^2}{c^2} \tag{12}$$

En la Figura 28 podemos observar dos ejemplos distintos de cálculo del error con las distintas funciones presentadas. En el primero de ellos vemos como para dos intervalos con cierto solape, tanto IoU como $GIoU$ generan el mismo valor, ya que A y B cubren C por completo. Por otro lado, $DIoU$ consigue un valor más elevado al tener en cuenta la distancia entre ambos intervalos, favoreciendo una convergencia más rápida. En el segundo ejemplo (abajo), $DIoU$ vuelve a generar un error mayor que $GIoU$, y al no existir ningún solape entre A y B, el IoU dará 0 y no se podrá utilizar para la optimización.

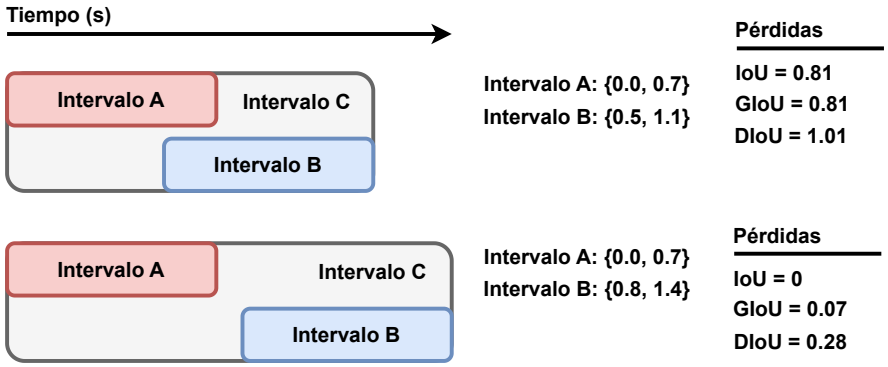

Figura 28: Comparación de las distintas funciones de pérdida basadas en *Intersection over Union*: $\mathcal{L}_{IoU}$, $\mathcal{L}_{GIoU}$ y $\mathcal{L}_{DIoU}$.

Anexo H Código implementado

A lo largo del siguiente Anexo, introduciremos brevemente los distintos programas implementados durante el proyecto, cuyo funcionamiento se centra en la adaptación y el formateo de los datos de entrada para ser procesados por *ActionFormer*[4], la preparación de los vídeos para el proceso de extracción de *features*, y el análisis y representación gráfica de los resultados obtenidos por el modelo.

- *process-videos.py*: programa que toma como entrada todos los vídeos que formen parte del dataset, generando un fichero de salida en formato *JSON* que recoge información de cada uno de ellos. Este incluirá, su nombre, su identificador (P01_XX), su *frame rate*, su duración en segundos y su resolución. Además, asignará aleatoriamente cada vídeo al conjunto de entrenamiento o de validación.

- *convert-annotations.py*: script utilizado para la transformación de las anotaciones originales en formato *csv* a dos ficheros *JSON*, que recojan los intervalos con las etiquetas de los verbos y nombres respectivamente, de manera que sean compatibles con *ActionFormer*. Además, crea el fichero *vid_list.csv* que indica el nombre de los vídeos que contienen alguna anotación.

- *remove-unused-videos.py*: toma como entrada el fichero *vid_list.csv* y elimina todos los vídeos que no se encuentren en esa lista.
- *reduce_label_range.py*: *script* que modificará un fichero *JSON* de anotaciones reduciendo el rango de las clases utilizadas, asignando un nuevo identificador desde 0 hasta el número de etiquetas menos 1. También creará un fichero que indique las clases detectadas, relacionando su anterior identificador con el asignado.
- *remove_unused_actions.py*: tomará como entrada dos ficheros en formato *JSON* con las anotaciones respectivas a los nombres y verbos de las acciones, así como dos *csv* que indican las etiquetas de las clases a conservar. Cualquier instancia cuya etiqueta no esté recogida dentro de ese rango será eliminada.
- *split_videos.py*: *script* utilizado para la división de los vídeos pertenecientes a nuestro *dataset* en fragmentos de pequeña longitud temporal, para adaptarlos a la herramienta *GluonCV*[6], utilizada para la extracción de *features*. Estos *clips* se extraerán periódicamente, con una separación determinada por el *temporal stride*, existiendo un solape de *frames* entre fragmentos contiguos. Además, realizará una transformación de los vídeos, para obtener un *frame rate* de 30*fps* y una resolución de 340 por 256 píxeles.
- *compress_features.py*: tomará como entrada los vectores de *features* generados para cada uno de los *clips* utilizando el extractor proporcionado por *GluonCV*. Estos vectores se agruparán y ordenarán temporalmente, de manera que se cree una única matriz por cada vídeo original.
- *compress_numpy.py*: transforma ficheros de *numpy* en formato *npy* al formato *npz*, con una mayor compresión.
- *top1_accuracy.py*: calcula la precisión del modelo, teniendo en cuenta únicamente el mejor intervalo generado para cada una de las instancias en el *ground-truth*.
- *confusion_matrix.py*: a partir de los intervalos originales y los generados, para un vídeo o conjunto de estos, imprime una matriz de confusión, la cual utilizaremos para analizar las confusiones en las predicciones del modelo. A lo largo de la memoria, podemos ver ejemplos de su uso en las Figuras 9, 12 y 11.
- *show_predictions.py*: *script* que creará un gráfico utilizando los intervalos originales y predichos, mostrando la extensión temporal de cada uno y su etiqueta, permitiendo un análisis a simple vista del desempeño del modelo. Las Figuras 20 y 21 fueron generadas haciendo uso de este programa.
- *plot_frequencies.py*: programa que generará un histograma con el número de apariciones de cada una de las clases existentes en el *dataset*. Vemos ejemplos de su uso en las Figuras 13, 14, 15 y 16.

Anexo I Enlaces externos

A continuación, se listan una serie de enlaces externos que contienen todo el código desarrollado como parte del proyecto, así como los vídeos del *dataset* utilizados para el análisis del modelo:

- Repositorio con código auxiliar y modificaciones a *ActionFormer*, *GitHub*.
- Carpeta con los vídeos utilizados, *Google Drive*.