TRABAJO FIN DE MÁSTER

Máster en Ciencia de Datos e Ingeniería de Datos en la Nube

Clasificación de movimientos de CrossFit: una aplicación con MoViNets

Autor: Agustín Piqueres Lajarín

Tutor: Daniel González Medina

Septiembre, 2022

Aquí va la dedicatoria que cada cual quiera escribir. El ancho se controla manualmente

Declaración de autoría

Yo, … , con DNI … , declaro que soy el único autor del trabajo fin de grado titulado " … ", que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual, y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a … de … de 20 …

Fdo.: Agustín Piqueres Lajarín

Resumen

El CrossFit como deporte como deporte competitivo ha venido ganando adeptos en los últimos años, y el primer paso de clasificación en una competición consiste en realizar una o varias pruebas (*workouts*) que deben ser grabadas y enviadas a la organización para su revisión.

En este trabajo se desarrolla una aplicación para clasificar movimientos de CrossFit como primer paso hacia una aplicación que permita automatizar la revisión de las pruebas enviadas por los atletas. Para llevar a cabo el mismo se crea un conjunto de *clips* de movimientos, basado en videos realizados por competidores de élite de CrossFit perteneciente a la final de los Crossfit Games 2020. El clasificador utilizado se basa en *MoViNets*, una familia de redes neuronales para videos eficiente tanto en términos de computación como de memoria, y se presenta una aplicación desarrollada en AWS. Todo el código está disponible públicamente, y una pequeña librería utilizada para trabajar con *MoViNets* está disponible en PyPI.

Agradecimientos

AGRADECIMIENTOS AQUÍ

Índice general

1 Introducción 1

1.1 Motivación 1

1.2 Objetivos 1

1.3 Estructura del proyecto 2

2 Estado del arte 3

2.1 Deep Learning 3

2.2 Cloud 4

2.3 Trabajos relacionados 4

3 Desarrollo del proyecto 7

3.1 Extracción y recoleccion de datos 7

3.1.1 Introducción 7

3.1.2 Proceso de extracción 7

3.1.3 Datos obtenidos 9

3.2 Experimentación con deep learning 10

3.2.1 Introducción 10

3.2.2 Preprocesado de los datos 11

3.2.3 Experimentos realizados y resultados 12

3.2.4 Evaluación de los resultados 14

3.3 Cloud y despliegue de la aplicación 16

3.3.1 Introducción 16

3.3.2 Arquitectura cloud 16

3.3.3 Resultado y funcionamiento 18

4 Conclusiones y trabajo futuro **21**

A Anexo **23**

A.1 Glosario de movimientos 23

A.2 Probabilidades obtenidas por movimiento en la muestra de test 23

A.3 Pipeline en AWS 24

Referencia bibliográfica **26**

Índice de figuras

3.1 Proceso de extración de datos 9
3.2 Frames de una muestra de *power clean* de la pipeline de entrenamiento . . 12
3.3 Distribución de las etiquetas entre la muestra de entrenamiento y test . . . 13
3.4 Top 1 categorial accuracy por epoch y loss frente por número de iteraciones 13
3.5 Heatmap de la matriz de confusión para la muestra de test. 14
3.6 Heatmap de correlación entre probabilidades. Para los clips clasificados como j, la celda (i, j) representa la correlación entre las probabilidades de predecir el movimiento i y el movimiento j. 15
3.7 Diagrama Cloud 17
3.8 App: Estado inicial 18
3.9 App: Video cargado 19
3.10 App: Extracto de tabla de predicciones 19

A.1 Probabilidad estimadas en la muestra de test. Para cada movimiento observado, cada gráfica representa las probabilidades obtenidas por el modelo. 24

Índice de tablas

3.1 Estadísticos descriptivos de los frames y duración de los clips 10
3.2 Estadísticos principales de las probabilidades predichas para cada movimiento observado 16

1. Introducción

1.1. Motivación

En el presente trabajo se busca desarrollar un modelo que sea capaz de clasificar distintos movimientos de CrossFit. Dentro del este deporte, el proceso para clasificarse para una competición suele ser uno o varios *workouts*[1], que debe ser grabado y enviado a la organización del evento en cuestión, para posibles revisiones del mismo.

Debido a que este deporte viene ganando popularidad con los años, han ido surgiendo más competiciones, y por lo tanto la necesidad de revisar estos videos de los atletas que se presentan. Esta tarea, la de revisar la correcta ejecución y el conteo de repeticiones de un atleta, conlleva la necesidad de un *juez*, una persona encargada de revisar y contar las repeticiones en los eventos.

En el caso de los clasificatorios para las competiciones, esta tarea quizá podría ser automatizada, utilizando para ello una aplicación a la que poder subir videos, y que se encargue de hacer de juez, contando las repeticiones de los distintos movimientos.

1.2. Objetivos

Como punto de partida para solucionar el problema previo, se pretence ofrecer solución a una versión simplificada de este problema, una aplicación que permita clasificar entre distintos movimientos de CrossFit. Para ello, es necesario disponer de un conjunto de videos que representen los distintos ejercicios, un modelo que permita clasificar estos videos, así como una aplicación en la que un usuario pueda subir estos videos y obtener el movimiento del que se trata.

[1] Se entiende por *workout* a un conjunto de ejercicios variados con movimientos funcionales ejecutados a alta intensidad, ver aquí.

1.3. Estructura del proyecto

El trabajo se divide en las siguientes secciones. En la sección 2 se expone el estado del arte actual, mostrando los modelos disponibles en la actualidad que tratan la clasificación de actividades humanas, la explotación de los mismos en el actual entorno *cloud*, así como una breve revisión de trabajos relacionados.

El desarrollo del proyecto contempla las siguientes secciones: La sección 3.1 presenta la metodología para la extracción de datos y el análisis del conjunto de datos obtenido. A continuación, la sección 3.2 trata el tratamiento de los datos necesario para entrenar el modelo, y los resultados obtenidos en el mismo. Por último, la sección 3.3 explica la arquitectura elegida para el despliegue de la aplicación y ejemplos de uso.

Finalmente, en la sección 4 se resumen los resultados obtenidos y se proponen lineas de mejora.

2. Estado del arte

2.1. Deep Learning

El campo del reconocimiento de acciones se ha vuelto un área de investigación activa en los últimos años (ver *Action Recognition*). Si bien las acciones humanas se pueden reconocer de diferentes formas, como el *optical flow* o en representaciones de esqueletos, en este trabajo nos centramos en la clasificación de videos. La clasificación de vídeos consiste en la obtención de una etiqueta representativa del contenido del vídeo, dados los frames que lo componen (ver referencia *Video Classification*).

Con la introducción del dataset *Kinetics 400* (ver [Kay et al., 2017]) se plantea el primer punto de referencia para la clasificación de acciones en humanos. Debido al tamaño del conjunto de datos, se hace posible entrenar redes neuronales desde cero y permite probar distintas arquitecturas sobre el mismo que posteriormente se pueden ajustar a tareas más a la medida por medio de *fine-tuning* (ver [Carreira and Zisserman, 2017]).

En el momento de plantearse este trabajo, aparece *MoViNets*, una familia de modelos eficientes tanto en el uso de memoria como en computación, que permite operar con videos en *streaming*, logrando *accuracies* más elevados en comparación con otros modelos desarrollados en la literatura con sus versiones más complejas (ver [Kondratyuk et al., 2021])[1], que ofrecen buenos resultados sobre *Kinetics 600*, una mejora sobre *Kinetics 600*: [Carreira et al., 2018].

Esta familia de modelos se compone de dos versiones diferentes, que en el paper original llaman *base* y *stream*, con 5 ”niveles” de complejidad (desde a0 hasta a5, donde a0 es una arquitectura simplificada, y a5-base obtiene resultados al nivel de [Feichtenhofer, 2020] (ver Tabla 17 en y Tabla 22 respectivamente en [Kondratyuk et al., 2021]). La gran novedad que aporta este trabajo es la variante *stream*, que permite tratar videos de gran longitud sin tener que procesar todos los frames por completo antes de devolver una predicción, con lo que se hace posible la inferencia en videos de gran longitud y con dispositivos de menor capacidad (ver aquí).

[1] Actualmente el panorama de modelos entrenados sobre Kinetics 600 ha evolucionado mucho, como puede verse en Action Classification on Kinetics-600, incluso añadiendo componentes de audio y texto (subtítulos): [Zellers et al., 2022].

Los autores de *MoViNets* ofrecen el siguiente *colab notebook* en el que muestran ejemplos de como hacer inferencia tanto con los modelos *base* como *stream*, exportar el modelo a *Tensorflow Lite* para hacer predicción en dispositivos móviles, y como hacer *fine-tuning*, utilizando para ello el modelo *MoViNet-A0-Base*. Como se puede ver en el mismo ejemplo, tras solo 3 *epochs*, haciendo fine tuning sobre el dataset UCF101 consiguen un top 1 accuracy de 0.9329 y 0.8514 en training y validación respectivamente, por lo que parece un buen punto de partida para aplicar al conjunto de datos presente.

En cuanto a las arquitecturas de *MoViNets stream*, si bien a la hora de hacer inferencia todo funciona correctamente, y el procedimiento para hacer autores debería ser análogo, no he visto posibilidad de reproducir el entrenamiento. En el siguiente issue 10730 se puede ver los errores. Este comportamiento parece relacionado con los siguientes issues 10463 y 10515, y actualmente se encuentran o bien abiertos esperando respuesta, o se han cerrado sin respuesta. Debido a esto, se ha optado por centrarse en la arquitectura de los modelos *base*.

2.2. Cloud

A la hora de desplegar una aplicación que hace uso de machine learning (ML), los proveedores cloud ofrecen servicios que automatizan una gran parte del ciclo de vida del modelo, tanto en AWS (con *Amazon SageMaker*), Azure: (*Azure ML*) o Google (con *Google Datalab*) por ejemplo.

En este trabajo se ha optado por desplegar el modelo en AWS, partiendo de cero, componiendo los servicios base que ofrece AWS para servir la aplicación por medio de EC2, y obtener las predicciones del modelo haciendo llamadas a una función función lambda respaldada por un contenedor de docker en ECR.

2.3. Trabajos relacionados

El presente trabajo busca un punto de partida de cara a contar repeticiones de distintos movimientos en un evento de CrossFit. Algún trabajo relacionado con esta temática se pueden encontrar por ejemplo en [Chen et al., 2022], donde hacen uso de *Yolo 4* (ver [Bochkovskiy et al., 2020]) y *Mediapipe* para detectar y clasificar distintos movimientos de fitness, o en [Preatoni et al., 2020], donde se hace uso de sensores portátiles para clasificar movimientos a lo largo de un *workout*[2].

Fuera de la literatura, un artículo hace uso de *MoViNets*. En concreto: *tennis-action-recognition* hace uso de *MoViNet stream* para clasificar movimientos en un evento de tenis[3].

En este otro artículo de *towardsdatascience* el autor hace uso de *Optical Flow* y posterior clasificación utilizando una red neuronal propia, para contar repeticiones de movimientos que parece ofrecer buenos resultados.

[2] Se entiende en este contexto por *workout* el conjunto de movimientos funcionales tanto de fuerza como cardiovasculares.

[3] Si bien el trabajo presentado en este blog encaja en gran medida con lo que se busca con este trabajo, no ha sido replicar los resultados, ni ha sido posible encontrar referencias del autor.

Finalmente se ha decidido hacer uso de una versión de *MoViNets base* como modelo para la clasificación de movimientos, ya que ofrece un buen punto de partida para clasificar acciones (en este caso ejercicios de funcionales o de CrossFit), y debido a que el modelo tiene un tamaño adecuado para ser desplegado sin hacer un uso intensivo de recursos (permite hacer inferencia sin necesidad de utilizar GPU para los casos estudiados), lo que permite hacer uso del mismo por medio de una aplicación por un usuario sin conocimiento técnico.

3. Desarrollo del proyecto

3.1. Extracción y recoleccion de datos

3.1.1. Introducción

En el siguiente apartado se expone la metodología de obtención de los datos de entrenamiento. Se presenta el proceso de extracción de los videos, el tratamiento para la generación del conjunto de clips, y el análisis de los datos obtenidos.

No hay constancia de un dataset formado por clips de movimientos de CrossFit etiquetados o modelo que permita clasificar estos movimientos, por lo que el paso inicial en este trabajo consiste en obtener un conjunto de *clips* que con los movimientos etiquetados.

Una particularidad de este deporte, es que gran parte del contenido está disponible en *YouTube*, por lo que es el primer punto al que acceder en busca de estos datos. Debido a que hay una gran diversidad de movimientos en CrossFit, y si en competiciones como los *CrossFit Games* aparecen cada año uno o varios movimientos nuevos, se ha decidido seleccionar un subconjunto de los movimientos para crear este dataset.

Los CrossFit Games son una competición anual creada por *Crossfit, LLC.* que busca descubrir a los atletas más en forma, y consisten en una serie de eventos (*workouts*) basados en ejercicios metabólicos, halterofilia y gimnásticos. Los ganadores de los CrossFit Games obtienen el título de ”Fittest on Earth” (ver la siguiente entrada de wikipedia).

En 2020, debido a la pandemia por COVID-19, esta competición se vió afectada, y 30 hombres y 30 mujeres fueron invitados a competir de forma online durante una primera etapa, pasando solo 5 y 5 a la final.

Los atletas grabaron la realización de los eventos, que eran enviados a *CrossFit* para su corrección, y estos los han publicado en *YouTube*.

3.1.2. Proceso de extracción

Para seleccionar la muestra de videos, se ha partido de 4 *eventos*, que se pueden encontrar en el siguiente Los videos corresponden a los siguientes eventos, que se pueden encontrar

enlace: *Friendly Fran*, *Damn Diane*, *Nasty Nancy* y *Awful Annie*. Estos eventos dan pie a 9 movimientos, las *labels* de nuestro modelo, que se pueden encontrar en el siguiente *gist*.

Dentro de la siguiente lista de videos (listas de CrossFit Games), se pueden encontrar los videos pertenecientes a la primera etapa, donde todos los atletas están grabados de forma individual y con la cámara estática. Todos estos videos tienen en común que la cámara está fija y aunque a veces aparecen personas alrededor (se puede ver a una persona haciendo de juez en todos ellos, o en otros casos también a más gente entrenando al fondo de la imágen), el atleta en cuestión es la parte principal del vídeo.

De cada *evento* se han seleccionado 20 videos (10 de hombres y 10 de mujeres), de los cuales se han etiquetado 15 repeticiones de cada ejercicio. Esto nos deja con una muestra de alrededor de 300 repeticiones por movimiento, 2700 clips en total. Comparemos este dataset con con UCF101 (ver [Soomro et al., 2012]), el utilizado por los autores para hacer fine tuning en el tutorial. UCF101 contiene ejemplos de 101 actividades, con 131 muestras por actividad en media frente a 300.

Los videos se han descargado por medio del siguiente script: `download.py`, que utiliza `yt-dlp`. Todos los vídeos se han descargado con resolución 480 x 854p, en formato mp4.

Una vez tenemos los videos, el siguiente paso para generar el dataset consiste en obtener los clips pertenecientes a cada movimiento con la etiqueta correspondiente. Para ello, la herramienta elegida ha sido *Supervisely*[1].

Una vez se ha etiquetado un video, *Supervisely* genera un `json` con las anotaciones correspondientes al movimiento y los frames en los que se produce. Este fichero hay que transformarlo a un formato que pueda utilizar `ffmpeg-split.py`, para lo cuál se ha utilizado el siguiente script: `manifester.py`.

Por último, para obtener cada uno de los clips se ha utilizado `ffmpeg-split.py`[2]. Como su nombre indica, este script utiliza `ffmpeg` para hacer particiones de los videos. Utilizando la información del *manifest*, la generación del nombre de cada uno de los videos tiene la siguiente estructura: *<movement>_<i>.mp4*, donde *<movement>* es el nombre del movimiento en ese video, e *<i>* es un contador del ejemplo en cuestión.

La figura 3.1 resume el proceso para la obtención de los datos[3].

[1] Entre las distintas herramientas gratuitas, otra alternativa es *LabelStudio*. Sin embargo, *Supervisely* facilita mucho el trabajo al permitir seleccionar rangos de frames por medio de atajos del teclado.

[2] Este script pertenece al siguiente repositorio.

[3] Todos los scripts utilizados para la la extracción de los datos se encuentran aquí.

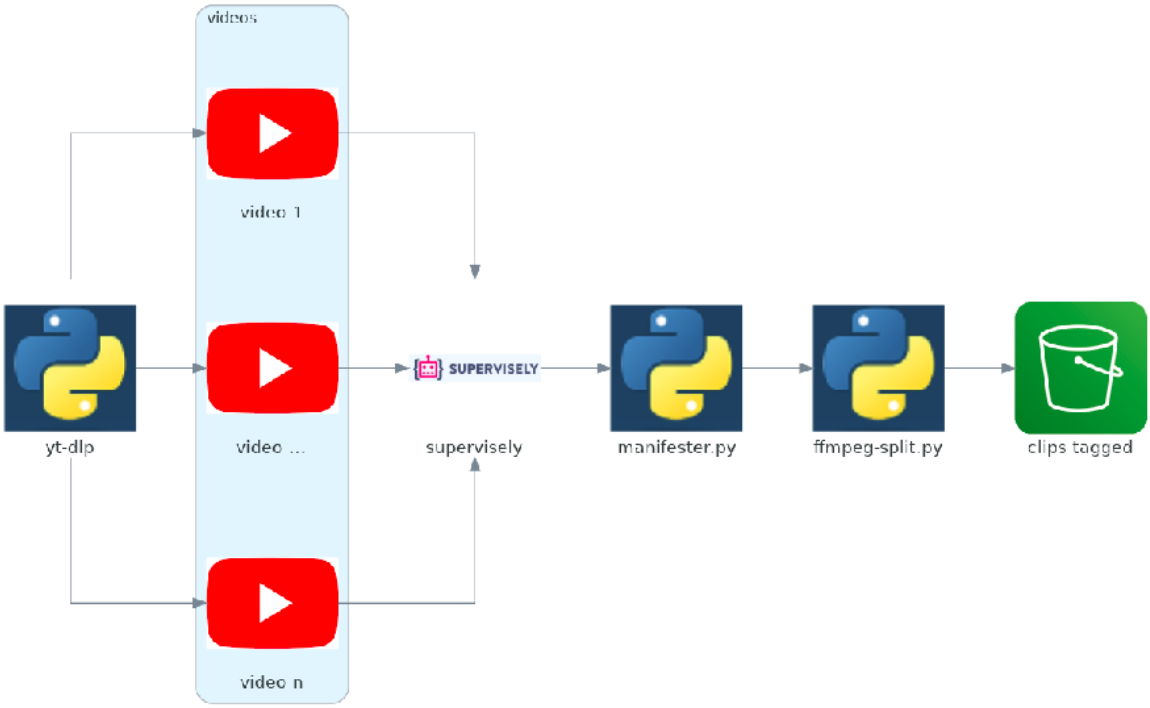

Figura 3.1: Proceso de extración de datos

Se descargan los videos utilizando `download.py`, y se procesan en *Supervisely*. El output de *Supervisely*, se transforma por medio de `manifester.py`, cuyo output se utiliza a su vez por `ffmpeg-split.py`, junto con los videos originales, para obtener los clips etiquetados, que se almacenan en un bucket de S3.

3.1.3. Datos obtenidos

En la tabla 3.1 podemos ver los estadísticos descriptivos (media y desviación típica) del número de frames de todos los videos y duración, desglosado por movimiento.

Tabla 3.1: Estadísticos descriptivos de los frames y duración de los clips

	Frames		Duración (s)	
	Media	Desviación típica	Media	Desviación típica
thruster	27.7396	4.5060	0.9419	0.1274
chest-to-bar	21.0478	7.4233	0.7196	0.2569
double-unders	14.5880	1.2740	0.4841	0.0439
ghd	58.9732	5.5611	1.9654	0.1860
power clean	67.3567	13.0092	2.2448	0.4337
deadlift	24.1229	7.1950	0.8048	0.2395
shspu	43.2100	15.1247	1.4412	0.5055
ohs	31.9767	5.4303	1.0664	0.1799
bar-facing burpee	62.0233	10.0239	2.0688	0.3340
TOTAL	39.0042	7.7275	1.3041	0.2563

Podemos observar que el movimiento más rápido (aquél con menor duración) es el *double-under*, con 14.5580 frames en media, y algo menos de medio segundo de duración (0.4841 segundos, mientras que el movimiento de mayor duración en media es el *power clean*, con 67.36 frames y 2.44 segundos de media. Habría que tener en cuenta, que si bien los *double-unders* son muy similares, el *power clean* podría bajar de tiempo si se tomase una muestra de eventos diferente. Esto es así debido a que en el evento escogido el peso con el que se realiza el movimiento es relativamente alto para este levantamiento.

Si comparamos con UCF101 por ejemplo, la longitud de los videos es menor en este dataset (en UCF101 la duración de los videos está alrededor de 7 segundos, mientras que en Kinetics 600 está alrededor de 10 segundos).

3.2. Experimentación con deep learning

3.2.1. Introducción

En esta sección se explica el modelo seleccionado y el tratamiento aplicado a los datos, el proceso de entrenamiento del modelo y los resultados obtenidos.

Se ha decidido utilizar la implementación de *MoViNets* basada en `tensorflow`. Aunque existe más de una implementación del modelo actualmente (ver *Papers with code*), los autores han decidido utilizar este framework para el desarrollo del modelo, y han publicado el modelo junto con *checkpoints* del mismo en el siguiente repositorio de GitHub, así como en *tensorflow hub*.

3.2.2. Preprocesado de los datos

De cara a construir la pipeline para el entrenamiento del modelo, se ha utilizado la API de `tf.data.Dataset` ya que permite la ingesta de un conjunto potencialmente grande de datos, así como el procesamiento de los mismos de forma eficiente (ver *Better performance with the tf.data API*) de leer los videos como un `Dataset`.

El primer paso para consiste generar un nuevo dataset, reemplazando los videos en formato .mp4 a .tfrecords (ver *TFRecord and tf.train.Example*), un formato para guardar una secuencia de ficheros en binario, basado en protocol-buffers de Google específico para `Tensorflow`.

Para simplificar todo el proceso desde la lectura de los datos hasta el entrenamiento del modelo y posterior consumo del mismo (inferencia), se ha desarrollado una pequeña librería, disponible en PyPI: `movinets_helper`[4]. El paso a paso en el preprocesamiento de los datos se puede ver en el siguiente enlace.

Un punto relevante al transformar el formato de los datos es el tamaño. Los dataset en formato `tf.data.TFRecordDataset` no son eficientes en cuanto al espacio que ocupan, se pierde la ventaja de la compresión que ofrecen los formatos mp4 en video, o jpg en imágenes por ejemplo. El dataset pasa de ocupar alrededor de 400Mb en la resolución original, a ocupar algo menos de 10Gb una vez almacenados como TFRecords. Esto ocurre a pesar de comprimir los ficheros por medio de gzip, almacenar solo 10 frames de cada vídeo y almacenando la resolución necesaria para entrenar la versión a2 base de *MoViNets*.

A la hora de leer el dataset (siguiendo la guía), todos los videos se reescalan de acuerdo al modelo seleccionado (224x224p para el caso de a2 base), y se escalan los valores en RGB para que se encuentren en el rango $[0, 1]$.

Para homogeneizar los datos de entrada se ha decidido fijar al número de frames para todos los videos a 10 ($\bar{f} = 10$)[5]. Debido a que los vídeos en este conjunto de datos son más cortos que los utilizados en el caso de *Kinetics 600*, se ha seleccionado el número de frames como $\bar{f} = \min_v f_v$, donde f_v es el número de frames del vídeo v. El menor número de frames corresponde a uno de los vídeos de *double-unders*. Para aquellos vídeos con un número de frames mayor que $\bar{f}$, se seleccionan $\bar{f}$ frames equiespaciados.

En la siguiente imagen 3.2 podemos ver el ejemplo de uno de los *clips* que entran en la muestra de entrenamiento. Se trata de un *power clean*, donde se pueden ver las distintas imágenes que componen el *clip*.

[4] Aunque aún está pendiente de documentar por completo y añadir los tests correspondientes, pero todo el entrenamiento y tratamiento de datos se ha hecho por medio de este paquete.

[5] Ver en el paper original el tratamiento del *frame rate*. Aunque en el paper original utilizan 50 frames para los videos (Kinetics 600 tiene videos de 10 segundos, a una tasa de 24fps), en este caso no es posible en movimientos como los *double-unders* ya que son mucho más cortos.

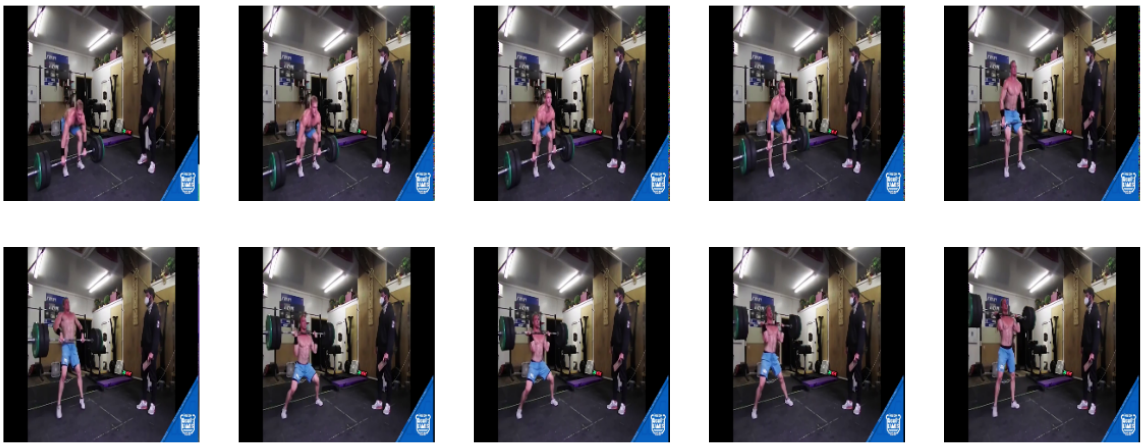

Figura 3.2: Frames de una muestra de *power clean* de la pipeline de entrenamiento

En el momento del entrenamiento, todos los videos están compuestos de 10 frames.

3.2.3. Experimentos realizados y resultados

El modelo se ha entrenado en *Google Colab* con *Tensorflow 2*. Y la versión y arquitectura seleccionada ha sido MoViNets A2 Base[6], ya aunque sea necesario utilizar GPU para el fine-tuning, es el modelo más grande (y con mejor capacidad predictiva) que permite hacer inferencia con CPU en un tiempo razonable[7].

Los hiperparámetros del modelo son los mismos utilizados en el paper original[8], con un tamaño de batch igual a 8[9].

La muestra se divide en un 80% (2164 clips) para training, y un 20% (541 clips) para test, donde los ejemplos se distribuyen de manera balanceada (ver figura 3.3).

[6]No ha sido posible hacer fine-tuning del modelo Stream debido a un error en los modelos que han guardado los autores, ver issue 10730.

[7]En este artículo de *Tensorflow* hacen referencia a aquellos modelos que pueden llegar a correr en tiempo real, lo que consideran como 20fps o superior.

[8]Ver apartado *B.1 More Details of the Architectures and Training* de METER BIBLIOGRAFÍA PARA EL PAPER

[9]El tamaño del batch seleccionado es el mismo utilizado por los autores en el tutorial a modo de ejemplo sobre el dataset UCF101.

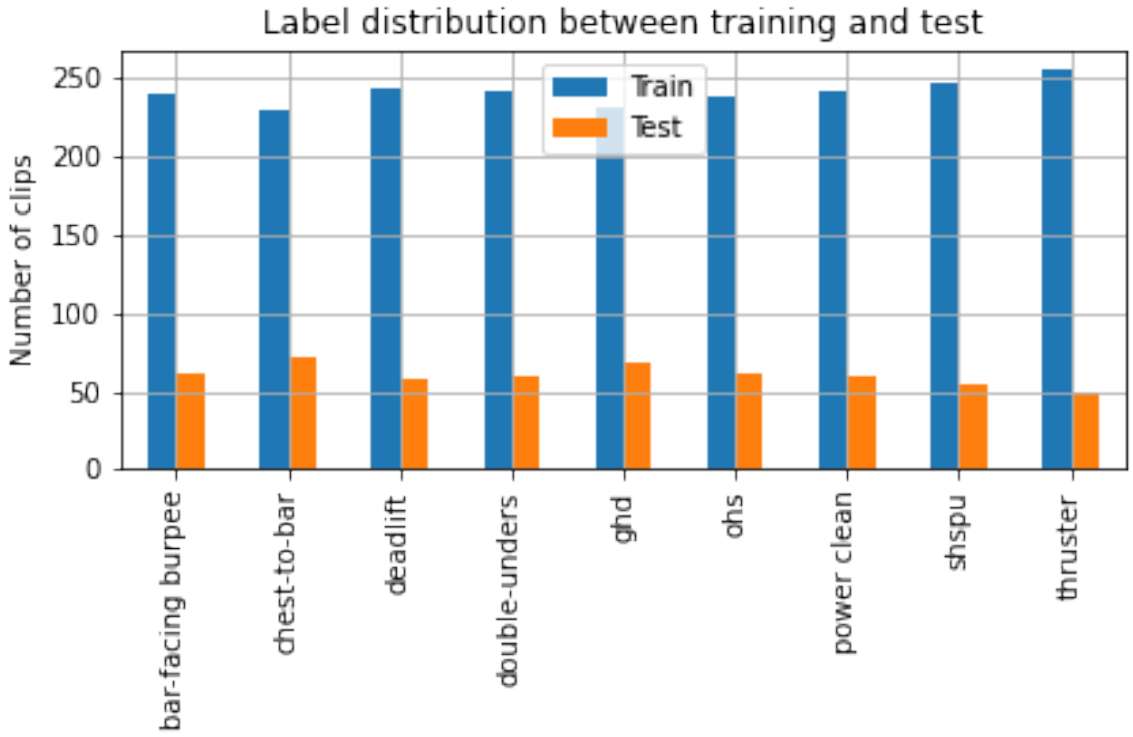

Figura 3.3: Distribución de las etiquetas entre la muestra de entrenamiento y test

El modelo se ha entrenado por 10 *epochs*, evaluando top 1 y top 5 accuracy, obteniendo un resultado de 0.9995 y 1 en entrenamiento y test respectivamente, como podemos observar en la siguiente imagen extraída de *tensorboard.dev*:

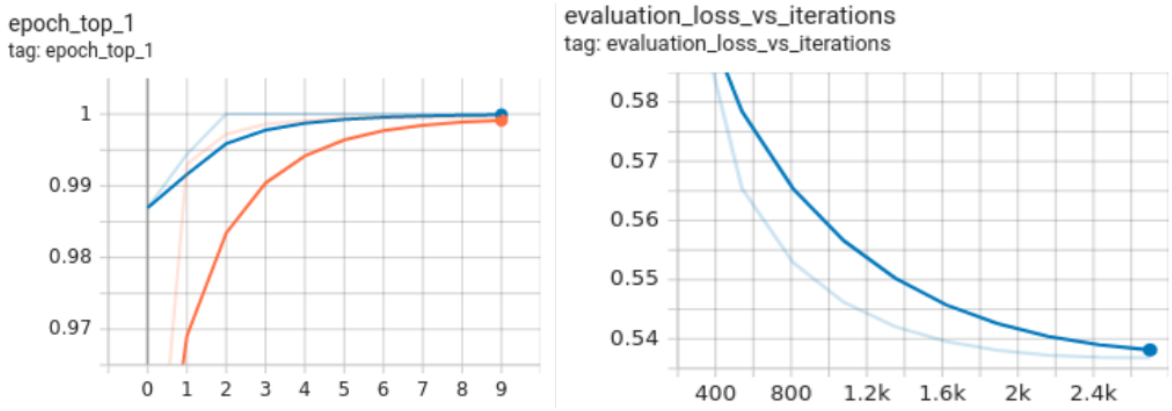

Figura 3.4: Top 1 categorial accuracy por epoch y loss frente por número de iteraciones

El modelo base generaliza a un gran número de movimientos, incluyendo algunos movimientos similares[10] , lo que puede facilitar el aprendizaje de algunos de movimientos similares.

Por otro lado, los videos en este caso, aunque se han seleccionado de una muestra distinta de atletas, todos ellos son atletas de élite en su deporte, por lo que la ejecución de los movimientos son muy similares entre si, lo que facilita aprender a clasificar los mismos.

Los resultados del entrenamiento se pueden ver en de forma interactiva en el siguiente enlace a *Tensorboard.dev*. En vista de estos resultados, no ha sido necesario modificar ninguno de los hiperparámetros propuestos por los autores del modelo.

3.2.4. Evaluación de los resultados

A la hora de evaluar los resultados, empezamos analizando el heatmap de la matriz de confusión. Como se vio al analizar el accuracy en la muestra de test, todos los movimientos se clasifican bien (ver imagen 3.4).

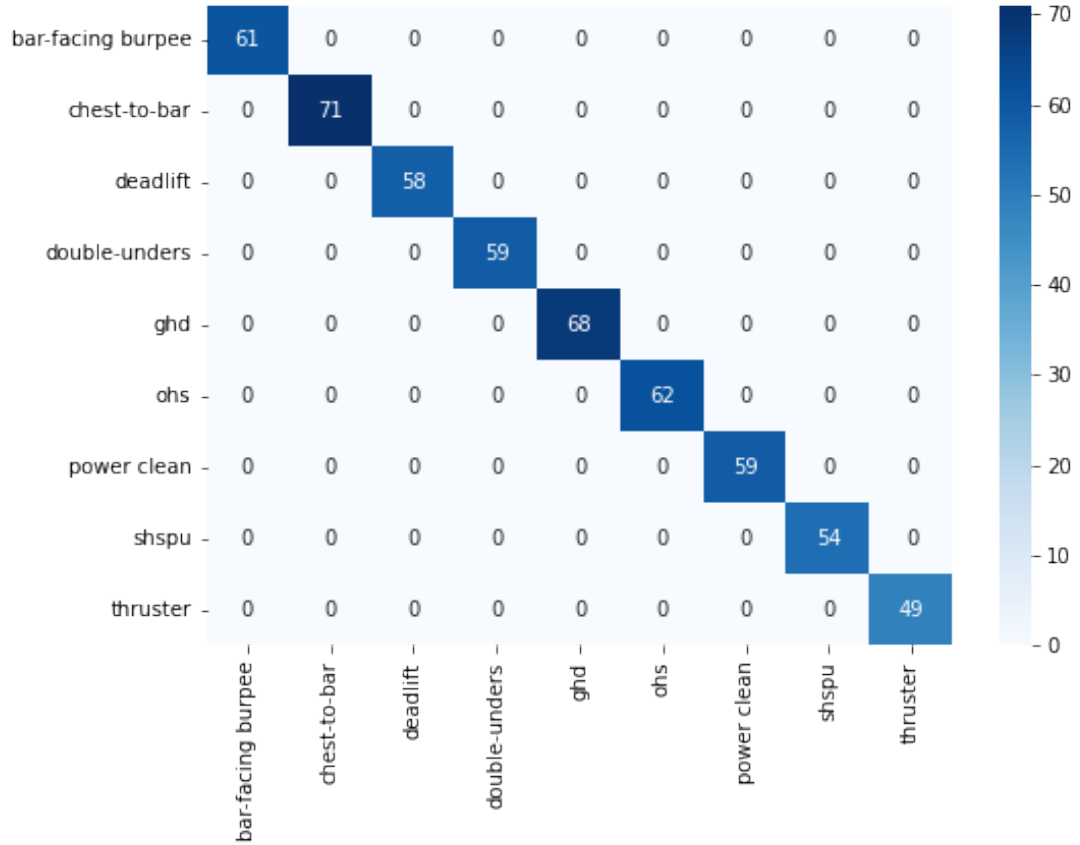

Figura 3.5: Heatmap de la matriz de confusión para la muestra de test.

[10] Por ejemplo, entre las *labels* de Kinetics 600 se encuentran *squat*, *clean and jerk* o *snatch weight lifting*.

Con vistas a proporcionar un análisis más robusto, en caso de que se decidiera extender el modelo a un mayor número de movimientos o muestras de distintos atletas, puede ser útil la información que proporciona el heatmap de la figura 3.6. La construcción de este gráfico se hace de la siguiente forma:

Partiendo de las probabilidades asignadas a cada vídeo en el período de test, se fija para cada movimiento (cada *label* correcta) y se calculan las correlaciones. Se selecciona la correlación (de Pearson) con el movimiento correctamente etiquetado, y se concatenan todos los vectores columna, de forma que obtenemos una matriz cuadrada. Estas correlaciones ayudan a ver cuáles son los movimientos que más se ”confunden” entre si. Dado que las probabilidades deben sumar 1, aquellos movimientos con una correlación negativa más cercana a 1, son aquellos en los que se puede observar una mayor intercambio de probabilidades a la predicción.

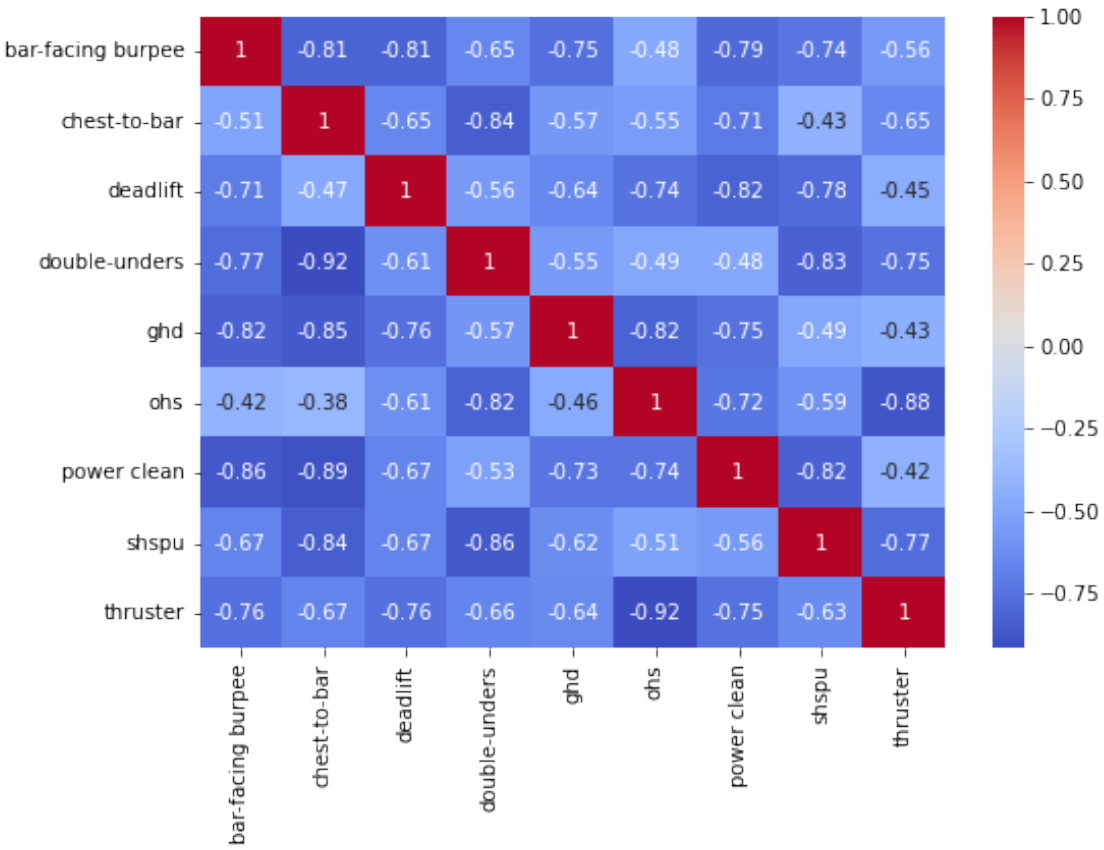

Figura 3.6: Heatmap de correlación entre probabilidades. Para los clips clasificados como j, la celda (i, j) representa la correlación entre las probabilidades de predecir el movimiento i y el movimiento j.

Por ejemplo, si nos fijamos en la columna que corresponde a *ohs*, vemos que el mayor

coeficiente de correlación es de -0.92, con *thruster*. Esto tiene sentido si se tiene en cuenta que en ambos movimientos, la barra parte de una sentadilla y acaba con la barra bloqueada por encima de la cabeza.

Por último, se puede ver en la siguiente tabla las probabilidades medias y desviación estándar obtenidas para cada movimiento correctamente etiquetado. Esto nos permite ver cuáles son los movimientos que se predicen con una mayor confianza, como en este caso los *double-unders*, o los que menos, los *ohs*. Las probabilidades que dan lugar a esta tabla se pueden ver en A.2.

Tabla 3.2: Estadísticos principales de las probabilidades predichas para cada movimiento observado

	Media	Desviación típica
double-unders	0.8921	0.0748
ghd	0.8904	0.0533
bar-facing burpee	0.8876	0.0756
thruster	0.8838	0.0590
deadlift	0.8746	0.0796
shspu	0.8731	0.0843
chest-to-bar	0.8552	0.1282
power clean	0.8443	0.0851
ohs	0.8438	0.1163

3.3. Cloud y despliegue de la aplicación

3.3.1. Introducción

En el siguiente apartado se presenta la aplicación para clasificar movimientos de crossfit, la arquitectura elegida y su funcionamiento para un ejemplo en concreto.

Para la explotación del modelo se ha elegido desplegar una aplicación en AWS desarrollada en Plotly Dash. Un usuario de la misma solo necesita subir un video en formato mp4, y recibir a cambio las predicciones del mismo. En los siguientes apartados se puede ver la arquitectura de la aplicación en la nube y los resultados obtenidos.

3.3.2. Arquitectura cloud

A continuación se puede ver en la figura 3.7 el diagrama de la aplicación desplegada en AWS.

El punto de entrada para un usuario es la aplicación desplegada en *EC2*, cuyo código se puede ver en el repositorio `movinets_dash_app`. La app está desarrollada en *Dash*, y ofrece la funcionalidad para subir un video en formato mp4 (y reproducirlo), obtener las

probabilidades asociadas a cada movimiento, y mostrar ejemplos de los movimientos para los que se ha entrenado el modelo.

Se pueden distinguir dos bloques:

- *App pipeline*. Por un lado tenemos la aplicación con la que interactúa el usuario. Partiendo de un repositorio público en github que contiene el código del front end, por medio de AWS CodePipeline y AWS CodeDeploy se automatiza el despliegue de la misma en una instancia de *EC2*, que viene desencadenado por medio de un `git push` a la rama master del repositorio.
- *Model predictions*. Al subir un video, este se puede enviar a predecir (solo se reproduce en pantalla por defecto). El proceso en este caso contempla la interacción con el modelo por medio de una función *AWS Lambda*. Debido a las restricciones de tamaño de lambda, no es posible desplegar el código de la misma simplemente comprimiendo las dependencias ya que tensorflow por si solo excede el límite. El código de la lambda (ver aquí) se debe empaquetar en Docker y subir la imagen a ECR. La lambda se encarga por tanto de leer el video, cargar el modelo almacenado en S3, transformar el video al formato adecuado, y devolver las probabilidades asociadas al mismo.

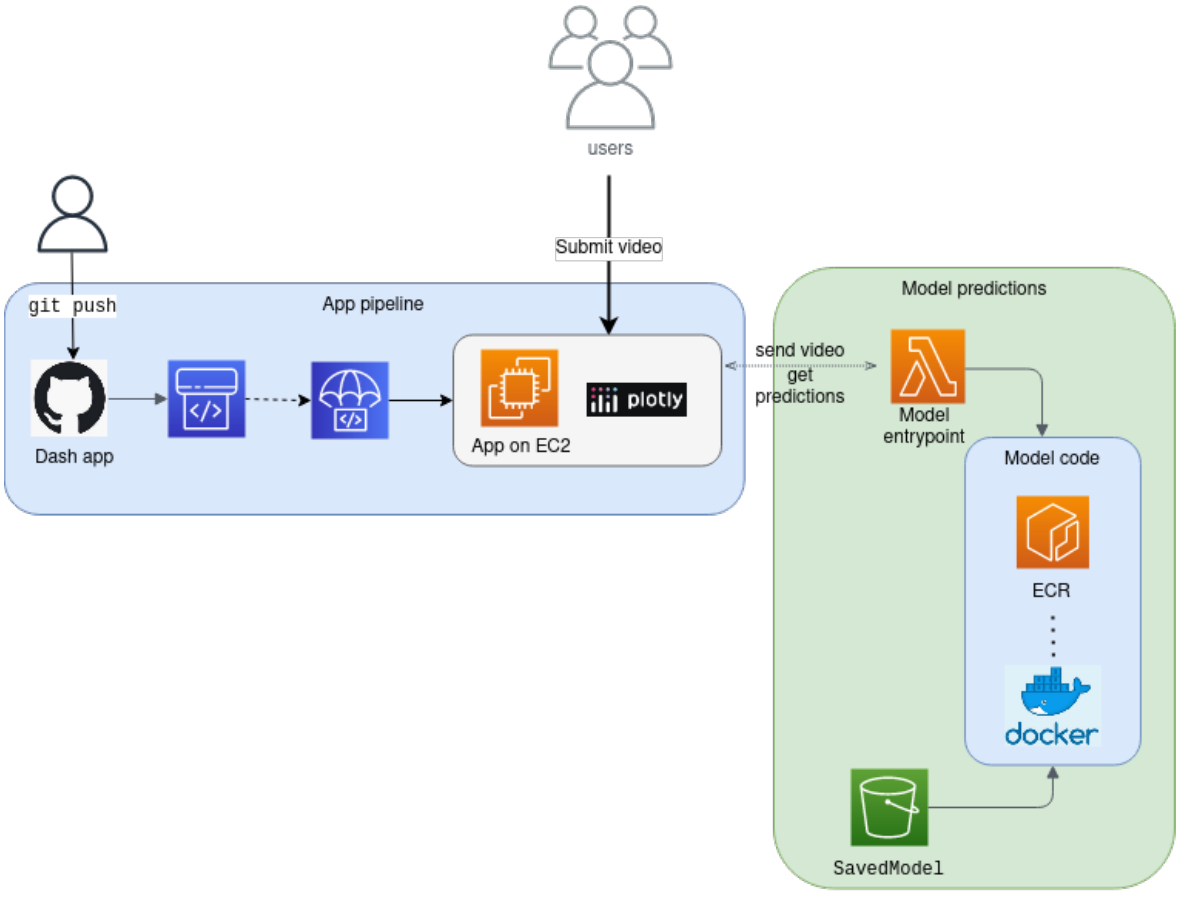

Figura 3.7: Diagrama Cloud

3.3.3. Resultado y funcionamiento

A continuación se puede ver una serie de capturas de la app desplegada. La figura 3.8 muestra la pantalla inicial de la app. La pestaña principal *Clip prediction* permite cargar el vídeo arrastrando el archivo o buscándolo por su nombre. La pestaña secundaria *Examples* permite ver ejemplos de los movimientos.

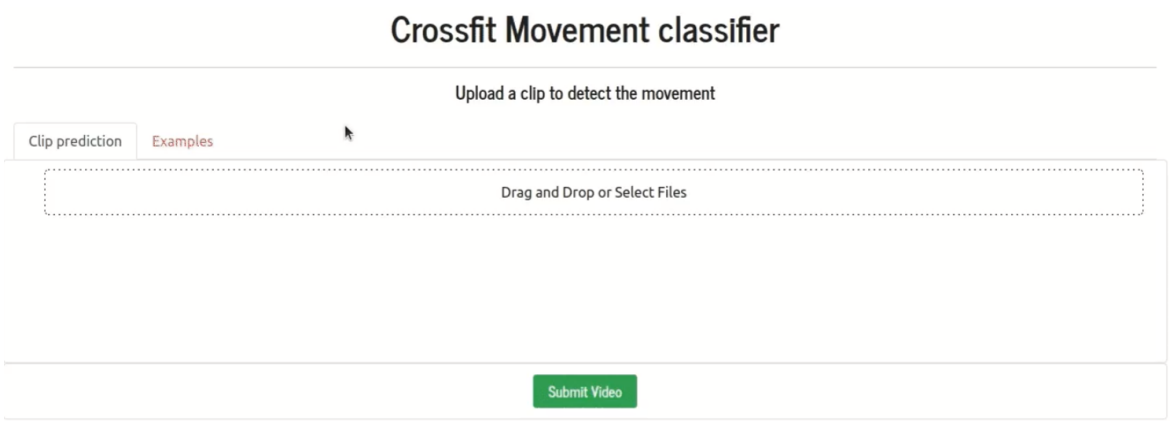

Figura 3.8: App: Estado inicial

Una vez se ha subido un clip, este se reproduce de forma automática (imagen 3.9). Para poder obtener las probabilidades asociadas al movimiento en cuestión habría que pulsar el botón *Submit Video*, que envía el contenido a la función lambda.

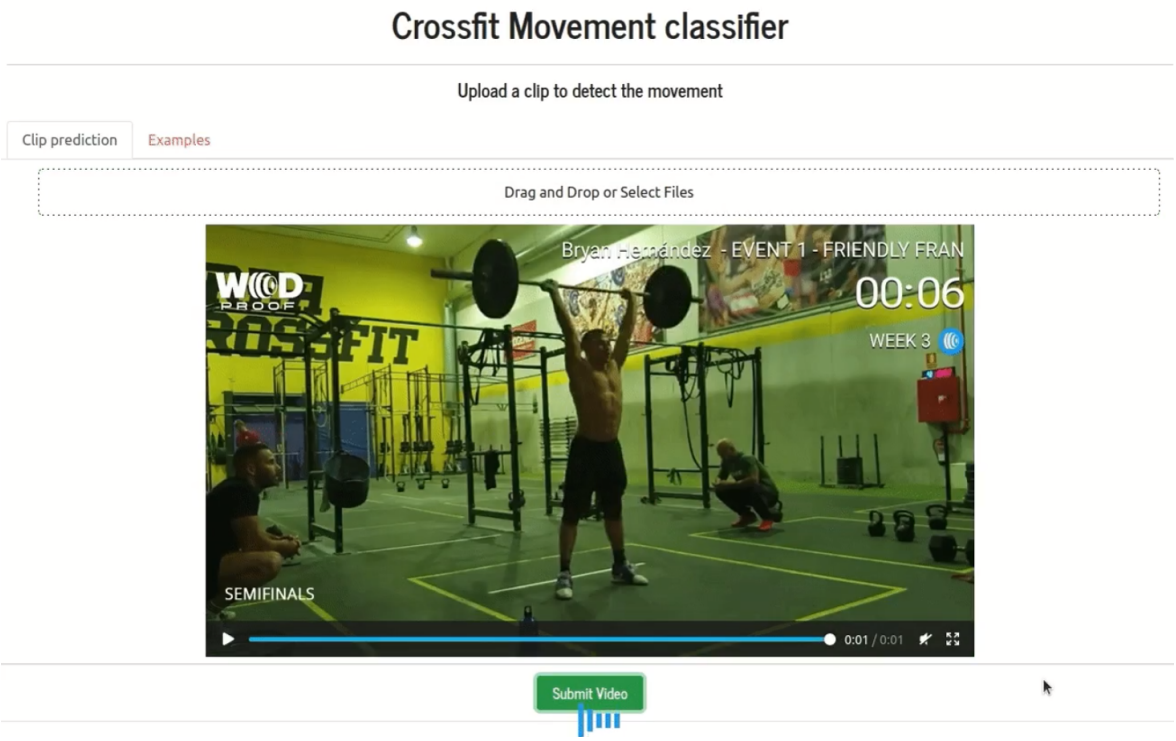

Figura 3.9: App: Video cargado

Tras obtener la predicción del modelo, justo debajo aparece la tabla con los 5 movimientos más probables y la probabilidad asociada a cada uno de ellos, ordenados de mayor a menor, como muestra la figura 3.10.

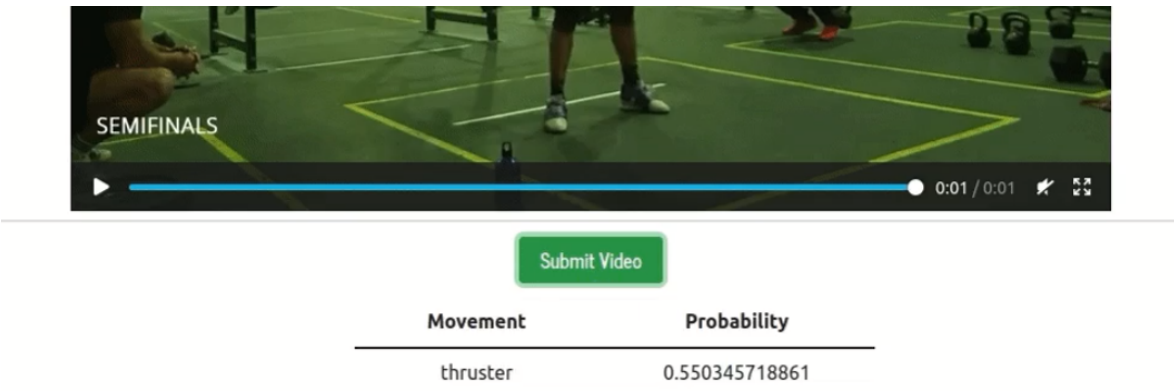

Figura 3.10: App: Extracto de tabla de predicciones

Si bien los vídeos se han entrenado con un máximo de 10 frames, en este caso el vídeo se pasa al modelo sin hacer un muestreo de los mismos. Aunque solo se ha comprobado para

unos pocos ejemplos de los vídeos completos, las probabilidades se ven afectadas, pero la ordenación de los mismos se mantiene.

El tiempo aproximado para obtener las predicciones asociadas a un clip es de alrededor de 14 segundos para un thruster, que es un movimiento relativamente corto (ver tabla 3.1), por lo que clasificar los movimientos es lento.

Como alternativas para ganar velocidad, se podría reducir el tamaño del modelo para mejorar la latencia de la CPU, limitando el número de posiciones decimales por ejemplo, un procedimiento conocido como *post-training quantization*. Los autores originales ofrecen versiones del mismo ya entrenadas para este caso aquí.

Existen otras alternativas. Se podría hacer un muestreo del vídeo de la misma forma que se hace durante la pipeline de entrenamiento (cabe recordar que durante el entrenamiento, el modelo solo utiliza 10 frames de cada vídeo), con lo que la información a procesar disminuiría. Otra opción sería entrenar un modelo de menor complejidad, ya sea la versión a0 o a1, ya que el accuracy actual ya es excepcionalmente alto.

4. Conclusiones y trabajo futuro

Este trabajo presenta una aplicación de *MoViNets* para la clasificación de movimientos de CrossFit, la creación de un nuevo conjunto de datos de basado en clips de videos extraídos de YouTube con una muestra de ejercicios de CrossFit, y el desarrollo de una aplicación en AWS para la clasificación de los clips.

Como siguientes pasos se plantea el entrenamiento de las arquitecturas *stream* para poder tratar videos completos en los que aparezcan distintos movimientos, y comparar este enfoque con otros como metodologías basadas en *optical flow*. En cuanto al despliegue y explotación del modelo en la nube, sería interesante pasar el modelo a *Tensorflow Lite* para disminuir el tamaño del mismo y mejorar los tiempos de ejecución.

A. Anexo

A.1. Glosario de movimientos

	Significado	Traducción
thruster	Thruster	Thruster
chest-to-bar	Chest-to-bar pull-up	Dominada al pecho
double-unders	Double-under rope jump	Salto doble de comba
ghd	Glute-ham developer machine sit-up	Abdominales en máquina GHD
power clean	Power clean	Cargada de fuerza
deadlift	Deadlift	Peso muerto
shspu	Strict handstand push-up	Flexión de pino estricta
ohs	Overhead squat	Sentadilla de arrancada
bar-facing burpee	Bar-facing burpee	Burpee frente a la barra con salto

A.2. Probabilidades obtenidas por movimiento en la muestra de test

En la siguiente imagen se puede observar, para cada movimiento observado, las probabilidades que devuelve el modelo.

Si nos fijamos en la imagen *ohs vs rest* por ejemplo, podemos ver los movimientos entre las probabilidades que dan pie a las figuras de los heatmap de las correlaciones, figura 3.6, donde se puede observar como cuando disminuye probabilidad de *ohs*, aumenta la probabilidad de *thruster*.

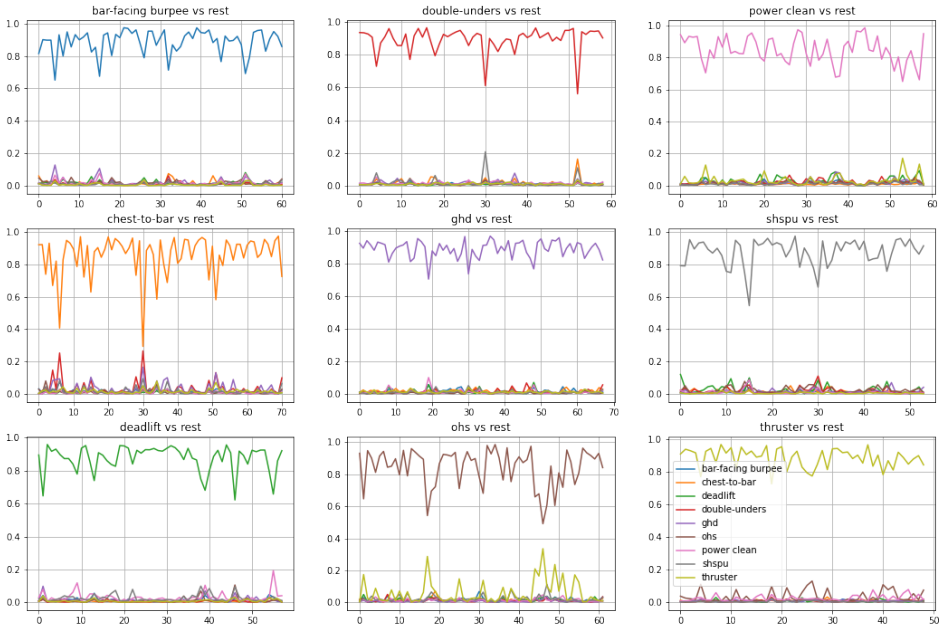

Figura A.1: Probabilidad estimadas en la muestra de test. Para cada movimiento observado, cada gráfica representa las probabilidades obtenidas por el modelo.

A.3. Pipeline en AWS

A la hora de tratar los videos en la función lambda hay que tener en cuenta lo siguiente, si se parte de la imagen de docker que ofrece AWS (como es el caso). Para leer los videos en mp4 y traducirlos a tensores para que los pueda entender tensorflow, se ha utilizado la siguiente función: decode_video. Esta función parte de unos binarios que fallan en la distribución de linux que utiliza la imagen de docker, ver issue. Como alternativa para leer los videos en este caso se ha utilizado OpenCV, si bien el impacto impacto que tiene es mínimo a la hora de leer los clips, en los decimales usados al pasarlo a tensor.

Referencia bibliográfica

[Bochkovskiy et al., 2020] Bochkovskiy, A., Wang, C., and Liao, H. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934.

[Carreira et al., 2018] Carreira, J., Noland, E., Banki-Horvath, A., Hillier, C., and Zisserman, A. (2018). A short note about kinetics-600. *CoRR*, abs/1808.01340.

[Carreira and Zisserman, 2017] Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? A new model and the kinetics dataset. *CoRR*, abs/1705.07750.

[Chen et al., 2022] Chen, K.-Y., Shin, J., Hasan, M. A. M., Liaw, J.-J., Yuichi, O., and Tomioka, Y. (2022). Fitness movement types and completeness detection using a transfer-learning-based deep neural network. *Sensors*, 22(15).

[Feichtenhofer, 2020] Feichtenhofer, C. (2020). X3D: expanding architectures for efficient video recognition. *CoRR*, abs/2004.04730.

[Kay et al., 2017] Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., Suleyman, M., and Zisserman, A. (2017). The kinetics human action video dataset. *CoRR*, abs/1705.06950.

[Kondratyuk et al., 2021] Kondratyuk, D., Yuan, L., Li, Y., Zhang, L., Tan, M., Brown, M., and Gong, B. (2021). Movinets: Mobile video networks for efficient video recognition. *CoRR*, abs/2103.11511.

[Preatoni et al., 2020] Preatoni, E., Nodari, S., and Lopomo, N. F. (2020). Supervised machine learning applied to wearable sensor data can accurately classify functional fitness exercises within a continuous workout. *Frontiers in Bioengineering and Biotechnology*, 8:664.

[Soomro et al., 2012] Soomro, K., Zamir, A. R., and Shah, M. (2012). UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402.

[Zellers et al., 2022] Zellers, R., Lu, J., Lu, X., Yu, Y., Zhao, Y., Salehi, M., Kusupati, A., Hessel, J., Farhadi, A., and Choi, Y. (2022). MERLOT reserve: Neural script knowledge through vision and language and sound. *CoRR*, abs/2201.02639.