Detección de participantes en pruebas deportivas

Proyecto creado por Jonay Antonio Sarmiento Ramírez

Supervisado por:
Javier Lorenzo Navarro
Modesto Castrillón Santana

Fecha:
14 de abril de 2021

Agradecimientos

Agradecer a mi familia y amigos por el increíble apoyo que he tenido estos meses de desarrollo y por su paciencia para escucharme hablar de batallitas de aprendizaje automático sin saber ni a qué me refiero.

Por supuesto agradecer a mis tutores, sin ellos no habría sido capaz de avanzar de esta manera, ni de obtener estos buenos resultados, agradezco su guía y confianza en mí para realizar este proyecto.

Por último, agradecer a la ULPGC por todos estos años de aprendizaje, ha sido un camino especialmente complicado para mí, pero ha terminado en buen puerto y me han otorgado la confianza que necesitaba para el ámbito laboral.

Resumen

En las competiciones de carrera a pie, hoy en día, es muy necesario llevar un control de identificación de los participantes para así poder determinar si están siguiendo el transcurso de la vía marcada y su posición correspondiente. Una solución propuesta a este tipo de problema es la utilización de métodos de detección y re-identificación, para detectar y reconocer a un mismo participante.

Este proyecto nace con el objetivo de validar si las trayectorias de las personas que aparecen en una escena de exterior, correspondiente a una prueba deportiva de carrera, permiten diferenciar entre espectadores y corredores. Comparando sus comportamientos, los espectadores suelen tener un comportamiento más estático o errático, a diferencia de los participantes, que suelen tener un comportamiento más homogéneo, ya que deben seguir el itinerario de la carrera definido por los organizadores.

Para ello partimos de una serie de vídeos de carreras deportivas celebradas en las islas y realizamos una detección de las personas que hay en cada fotograma, conformando al final cada trayectoria. Posteriormente, calculamos una serie de descriptores a partir de dichas trayectorias y alimentamos con ellos a una red neuronal recurrente artificial. Todo esto con el objetivo de comprobar si podemos catalogar a las personas de dichos vídeos mediante la red y con que fiabilidad.

Abstract

In sport competitions such as running races, nowadays, is very important to keep track of the participants in order to determine if they are following the course of the marked track and their corresponding position.A proposed solution to this type of problem is the use of detection and re-identification methods to detect and recognise the same user.

This project was born with the aim of validating whether the trajectories of people appearing in an outdoor scene, corresponding to a sporting race event, make it possible to differentiate between spectators and runners.Comparing their behaviour, spectators tend to behave in a more static or erratic way, unlike participants, who tend to behave in a more homogeneous way, as they have to follow the itinerary of the race defined by the organisers.

To do this, we start from a series of videos of sport races held on the islands and we detect the people in each frame, making up each trajectory. Subsequently, we calculate a series of descriptors from these trajectories and feed them to an artificial recurrent neural network. All this with the aim of testing whether we can catalogue the people in these videos using the network and how reliably.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

Las carreras a pie son eventos deportivos realizados durante todo el año al que acuden participantes de prácticamente todas las edades. Con una popularidad en aumento, son cada vez más los ingresos generados para los organizadores y empresas colaborativas, los cuales deben asegurarse de ofrecer una competición justa y reglada.

Por ejemplo, en la isla destacan pruebas deportivas como el maratón de Gran Canaria, cuya décima edición fue celebrada en 2019 y con sus 4400 corredores y corredoras entró en el listado de los 10 mejores maratones de España [1]. En dicho maratón se celebraban diferentes pruebas, desde la más famosa y dura de 42km hasta variaciones de 21 y 10 kilómetros. Por desgracia no se realizó ninguna otra prueba en 2020 o 2021 debido a la terrible situación dispuesta por el Covid-19.

Ilustración 1.1: Maratón Gran Canaria 2019 [1]

Sin embargo, carreras como la Transgrancanaria HG consiguieron celebrarse tanto en 2020 como en 2021, con muchas restricciones de por medio y con un número reducido de corredores. Aun así, en 2021 la edición número 22, participaron 1700 corredores y corredoras de más de 45 países diferentes, y se dispuso el despliegue más ambicioso que jamás habían realizado los organizadores, un sistema de *streaming* con diversos equipos de grabación y drones, los cuales, permitieron ver el evento sin necesidad de asistir, cumpliendo así las dificultosas restricciones impuestas estos años [2].

Estas carreras fueron alabadas por el público por la enorme satisfacción del resultado, especialmente en un contexto tan complejo.

Ilustración 1.2: Transgrancanaria celebrada en 2021, 22º edición [2]

Estas carreras son celebradas con diferentes modalidades, desde terrenos montañosos y para gente con experiencia hasta carreras costeras y de bajo kilometraje. Es común también realizarlas para recaudar incentivos en asociaciones sin ánimo de lucro o como competiciones lucrativas para los ganadores.

Por supuesto al ser una competición debe haber un ganador y un listado de posiciones, los organizadores del evento son los encargados de controlar el cumplimiento del reglamento, para así no solo evitar posibles trampas en la competición sino también para preservar la seguridad de los participantes.

Nos centramos en el apartado del seguimiento realizado por los organizadores, estos deben etiquetar a cada participante y asegurarse que siguen el transcurso de la vía marcada, ya sea mediante puntos de control o tecnologías más avanzadas como sensores y transpondedores.

Las soluciones propuestas a este seguimiento aún quedan lejos de la perfección, el uso de etiquetas o dorsales, es una de ellas. Dichas etiquetas tienden a estropearse a lo largo de la carrera debido a los bruscos movimientos que los participantes realizan, además estos suelen modificar su atuendo adaptándose a las temperaturas cambiando así la posición de la etiqueta o arrugándola.

Otra solución común es el uso de dispositivos de transmisión de señal, los cuales dan datos, en tiempo real o al pasar por un punto de control, de la posición del participante. Destacan los chips desechables de bajo coste y en carreras de mucha distancia las balizas GPS, las cuales no solo son más costosas sino que añaden un peso extra a cada corredor, por lo cual suele estar restringido a corredores de élite. Estos dispositivos pueden ser defectuosos o extraviarse en el transcurso de la carrera, lo que supone una gran frustración para los corredores ya que es descalificación automática. Además dependiendo de la precisión del dispositivo vemos variaciones en los datos recibidos e incluso pérdidas de puntos de control.

Ilustración 1.3: Ejemplo de un chip desechable utilizado en pruebas deportivas [3]

El principal problema de las técnicas comentadas es la identificación del usuario, ninguna nos asegura que el usuario que lleva el dorsal y el dispositivo es la misma persona durante toda la carrera. Pudiendo realizar trampas e intercambios de personas en tramos no controlados.

Tras el gran avance realizado estos años en la detección e identificación de personas y objetos se han propuesto múltiples soluciones o automatizaciones a este tipo de problemas. Se destacan soluciones como la sincronización de imágenes multimedia, obtenidos en los diferentes puntos de control del evento, con los sistemas de cronometraje. Así como reconocimiento de dorsales basados en sistemas automáticos, los cuales pueden ser problemáticos cuando se producen oclusiones en las imágenes, ya sea por vestimenta, ángulo, o factores externos como iluminación, suciedad o rotura del dorsal.

En este proyecto partimos con el objetivo de ser capaces de identificar a cada persona en un vídeo é etiquetarlas. Para, posteriormente, recopilar una serie de descriptores y definir las trayectorias de las personas identificadas.

Todos estos datos son extraídos con el objetivo de alimentar a un algoritmo de aprendizaje automático y comprobar si podemos catalogar a las personas de dichos vídeos, según sus trayectorias y comportamientos, en corredores o público. Hablamos de un primer paso necesario para la identificación de participantes, ya que si conseguimos un algoritmo capaz de catalogar a las personas podemos reducir en gran medida la carga de trabajo para una posible red de identificación y reconocimiento individual de corredores.

1.2. Plan de trabajo

Este apartado recogerá la distribución temporal realizada en este trabajo de fin de grado y sus distintas fases.

Las horas estimadas se han cumplido, pero con una distribución algo diferente, por ejemplo la tarea 2.1 “Implantación de un método de detección de personas” ha sido una tarea muy corta porque hemos acabado utilizando un código libre, el cual explicaremos más adelante en esta memoria. Por otro lado, nos ha llevado más tiempo la tarea 2.4 “Implementación de la aplicación de clasificación público/corredor”, ya que hemos realizado múltiples clasificadores.

Una tarea que ha consumido bastante tiempo ha sido la refactorización y limpieza del código así como la implementación de *scripts* con el único fin de visualizar los datos y sacar conclusiones.

No obstante, partiendo del cómputo general, se han cumplido los requisitos impuestos por la ULPGC de 300 horas, independientemente de las fluctuaciones temporales.

Fases	***Duración Estimada*** *(horas)*	***Tareas*** *(nombre y descripción, obligatorio al menos una por fase)*
Estudio previo / Análisis	80	Tarea 1.1: Estudio de métodos de detección de personas basadas en redes neuronales.
		Tarea 1.2: Estudio de métodos de seguimiento de multi-objetos basados en redes neuronales.
		Tarea 1.3: Recopilación de vídeos para la creación de un conjunto de datos para entrenamiento y validación.
Diseño / Desarrollo / Implementación	100	Tarea 2.1: Implementación de un método de detección de personas.
		Tarea 2.2: Implementación de un método de seguimiento multi-objetos aplicado a seguimiento de personas.
		Tarea 2.3: Definición e implementación de heurísticas y descriptores para trayectorias.
		Tarea 2.4: Implementación de la aplicación de clasificación público/corredor.
Evaluación / Validación / Prueba	80	Tarea 3.1: Evaluación del detector de personas en entornos de exterior.
		Tarea 3.2: Evaluación del seguimiento de personas en entornos de exterior.
		Tarea 3.3: Evaluación de la clasificación público/corredor.
Documentación / Presentación	40	Tarea 4.1: Redacción de la memoria del TFG
		Tarea 4.2: Realización y ensayos de la presentación del TFG

Ilustración 1.4: Estimación inicial del plan de trabajo

En este proyecto el apartado de evaluación y validación ha cobrado un importante papel, ya que el objetivo principal era ver el funcionamiento de los clasificadores y su capacidad de catalogación. Es por esto que hemos reincidido en esta parte múltiples veces, modificando los modelos y los diseños de los clasificadores según su evaluación.

Capítulo 2

Objetivos y competencias

2.1. Objetivos iniciales

El objetivo es validar si las trayectorias de las personas que aparecen en una escena de exterior, correspondiente a una prueba deportiva de carrera, permiten diferenciar entre espectadores y corredores. Por tanto, el resultado del trabajo de fin de título será una aplicación prototipo que admita un vídeo o secuencia de imágenes como entrada y que como salida muestre la misma secuencia de imágenes etiquetando a los espectadores y corredores si es posible. Para alcanzar este objetivo se deben realizar una serie de tareas cada una con sus objetivos asociados.

- ✔ Detección de las personas que aparecen en cada imagen.
- ✔ Obtención de las trayectorias de cada persona detectada.
- ✔ Definición de heurísticas sobre las trayectorias.
- ✔ Cálculo de descriptores a partir de las trayectorias.
- ✔ Clasificación de las trayectorias a partir de las heurísticas y/o descriptores.

2.2. Objetivos conseguidos

Finalmente hemos logrado completar los objetivos propuestos y hemos conseguido entrenar múltiples redes que clasifiquen entre corredores y público con muy buenos resultados.

Partiendo de un método de detección, hemos etiquetado a cada persona en los vídeos recopilados y hemos extraído una serie de características temporales en cada fotograma. Tras refinar dichas características y formar las trayectorias, logramos nuestro objetivo principal, alimentar a una red neuronal recurrente artificial y evaluar sus resultados.

El apartado que no hemos podido conseguir es el de mostrar las etiquetas que la red ha predicho en formato vídeo a tiempo real. Esto se debe a que tendríamos que re-identificar con otra red a cada usuario y su número identificador para posteriormente añadir la etiqueta corredor o público. Un proceso muy costoso para el beneficio visual que otorga.

2.3. Competencias específicas cubiertas y su justificación

Con el desarrollo del proyecto se han cubierto las siguientes competencias, todas ellas dispuestas por la escuela de Ingeniería Informática [15]:

CP04	Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

Ilustración 2.1: Competencia de computación 04

Gracias a los conocimientos obtenidos en el grado de ingeniería informática y a la supervisión de los tutores se ha podido investigar y conocer nuevos fundamentos, paradigmas y técnicas propias de los sistemas inteligentes como los múltiples algoritmos del paquete Sklearn o las diferentes redes convolutivas y se han aplicado con resultados fructuosos.

Esta era la principal competencia específica instaurada a la hora de certificar y aprobar el TFT pero tras la realización del mismo se han cumplido otras competencias.

CP07	Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.

Ilustración 2.2: Competencia de computación 07

Se realizaron múltiples scripts que calculan y extraen datos de los vídeos así como scripts que analizan dichos volúmenes de datos.

CII02	Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

Ilustración 2.3: Competencia común a la ingeniería informática 02

Se planteó desde cero toda la puesta en marcha del proyecto, sus aplicaciones y sus posibilidades. Por ejemplo se planteó su mejora tras finalizar el TFG utilizando técnicas de comparación para mejorar la identificación de las personas.

IS04	Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.

Ilustración 2.4: Competencia de ingeniería de software 04

Se analizó el problema de la identificación de personas y su clasificación, se probaron y diseñaron múltiples modelos y finalmente se verificó su eficacia y se documentó. En el apartado de desarrollo se puede ver especificado cada uno de estos pasos, así como en la planificación.

Capítulo 3

Estado del arte

3.1. Introducción

El siglo XXI ha avanzado con un pleno desarrollo tecnológico, automatizando procesos que hasta hoy en día realizaría una persona. De una manera progresiva se ha ido transformando nuestra manera de interactuar con el entorno, añadiendo sistemas inteligentes a nuestro ámbito cotidiano.

Uno de los campos con mayor índice de mejora es sin duda la fotografía y la grabación de imágenes. Pasamos de sacar fotos con armatostes y de grabar vídeos con muy pocos fotogramas, a tener una cámara profesional de bolsillo y multifunción. Esto facilita enormemente la generación de nuevos metrajes de fotogramas con fácil acceso. Por supuesto nuestro ámbito ha crecido mucho más que en solo este aspecto.

En los últimos años se ha priorizado la digitalización de procesos y herramientas, obteniendo avances increíbles en el apartado programático, así como en el campo del aprendizaje automático.

Aprendizaje automático

Se han conseguido importantes éxitos en múltiples pruebas realizadas en este campo y más concretamente al subárea denominada "aprendizaje profundo" [16] . Vemos software inteligente automatizar el trabajo rutinario, comprender el lenguaje o las imágenes hasta el punto de realizar diagnósticos en medicina y apoyar la investigación científica. El aprendizaje profundo es un conjunto de metodologías de "*machine learning*" basadas en algoritmos de redes neuronales artificiales con la finalidad de procesar grandes cantidades de datos. [17]

El adjetivo "profundo" se refiere al uso de múltiples capas en la red, lo que permite una aplicación práctica y una implementación optimizada. Por ejemplo, realizando el procesamiento de una imagen, las capas inferiores podrían identificar los bordes mientras que las capas superiores pueden identificar los conceptos relevantes, como personas, vehículos, etc.

Desde el descubrimiento de la neurona, el interés de saber como funciona el cerebro es cada vez mayor. Los primeros clasificadores se realizaron mediante perceptrones lineales y poco a poco se fueron creando implementaciones más complejas como las redes neuronales multicapa, hasta finalmente introducir las redes convoluciones, las redes neuronales recurrentes y su variante, la utilizada en este proyecto, "*Long-Short Term Memory*", de la cual hablaremos en capítulos posteriores.

Estos algoritmos han tenido múltiples aplicaciones en campos como el reconocimiento del habla, vehículos con conducción autónoma, detección de personas etc. Como en cualquier proyecto la fase inicial indispensable es el estudio del dominio y de los diferentes campos a utilizar. Durante estos meses de desarrollo se han realizado horas de análisis y estudio durante casi cada día del proyecto, ya que van surgiendo nuevos métodos o posibilidades para aplicar al proyecto. Trataré de explicar todo este extenso proceso.

3.2. Estudio del dominio

Comenzamos analizando el dominio de nuestro objetivo inicial, el cual es ser capaz de pasar un fotograma o vídeo a un clasificador y que este detecte a cada una de las personas que se encuentran en la imagen. Esta fase cobró real importancia en mi caso, ya que este tema esta orientado a la computación y detección, conceptos que apenas se aplican al Grado de Ingeniería Informática y mucho menos al área de software, de la cual yo provengo. En conclusión todo este dominio y el resto de campos utilizados en este proyecto fueron completamente nuevos para mí y fui instruyéndome sobre la marcha.

3.2.1. Detección de personas basadas en redes neuronales

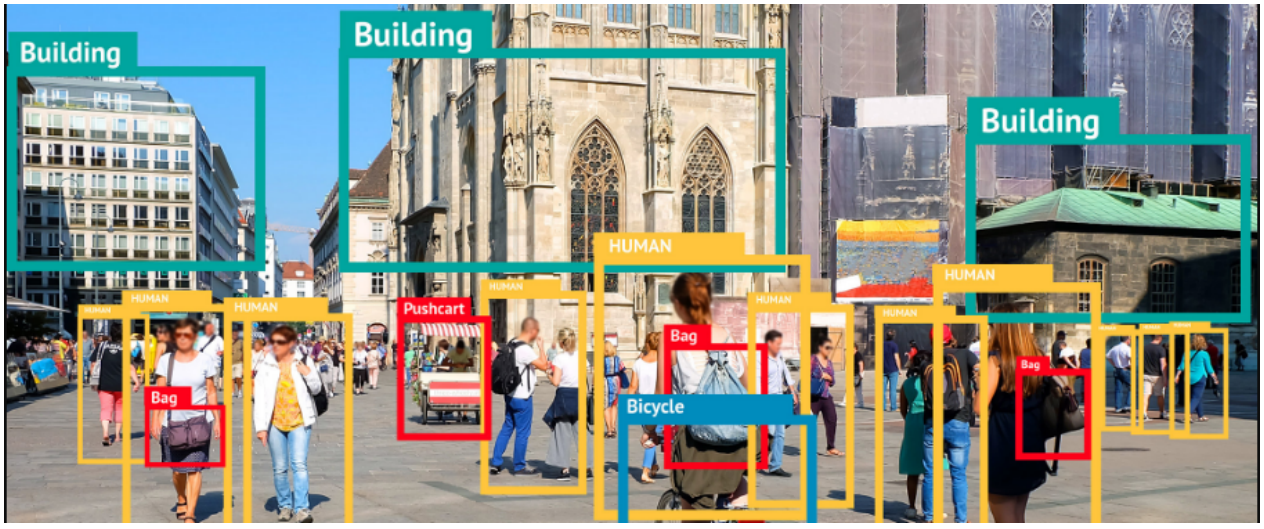

Ilustración 3.1: Ejemplo de detección [4]

Gran parte de la información mostrada a continuación ha sido obtenida y traducida de *Nanonets* [18].

Al iniciarte en este dominio los primeros términos que llegas a encontrar es "*Object Detection*" o detector de objetos y "*Object Tracking*" o rastreador de objetos. El detector de objetos tiene como objetivo identificar en un fotograma el objeto deseado y marcarlo utilizando alguna técnica, como puede ser, una malla alrededor o un recuadro. Por otro lado el rastreador se encarga de seguir dicho objeto fotograma a fotograma.

Estos complejos mecanismos presentan diferentes problemas en el mundo real, como por ejemplo, la oclusión de múltiples objetos en obstáculos, esto hace que el rastreador pierda al objetivo en una serie de fotogramas y puede llegar a asumir que es un nuevo objeto cuando lo vuelva a detectar. Lo cual generaría dos ID's para un mismo objeto.

Otros problemas típicos son los diferentes puntos de vista, vídeos grabados con diferentes ángulos de cámaras o cámaras no estacionarias, es decir, cámaras que tienen que moverse para seguir el objeto en cuestión.

Por último, uno de los mayores problemas a la hora de construir este tipo de herramientas es obtener los datos de entrenamiento necesarios, denominado conjunto de datos. Lo que requiere secuencias de vídeos donde en cada instancia, el objeto es identificado para cada fotograma.

3.3. Metodologías tradicionales

Meanshift

El Meanshift es una técnica no paramétrica del análisis del espacio de características, para localizar los máximos de una función de densidad, un algoritmo denominado búsqueda de modos ("*mode-seeking*") [19].

Los ámbitos de aplicación incluyen el análisis de conglomerados en la visión por ordenador y el procesamiento de imágenes. Este método fue presentado originalmente en 1975 por Fukunaga y Hostetler.

El objetivo de este algoritmo es encontrar todos los modos en una distribución de datos partiendo de un fotograma donde el objeto ha sido detectado, se aplica el algoritmo extrayendo las diferentes características de detección (color, textura, histogramas etc). Una vez se tiene una idea general del modo de la distribución en el estado actual del fotograma se pasa al siguiente fotograma, donde se buscan las mismas características.

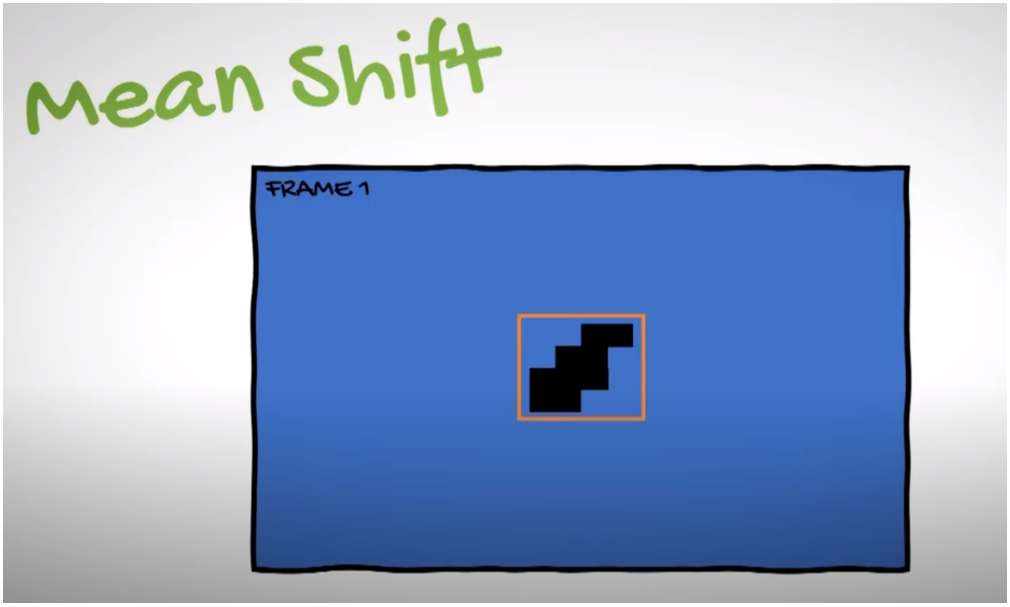

Ilustración 3.2: Vídeo explicativo [5]

La diferencia es que buscamos dichas características en una región de interés mucho mayor, que se conoce como el vecindario ("*The neighbourhood*").

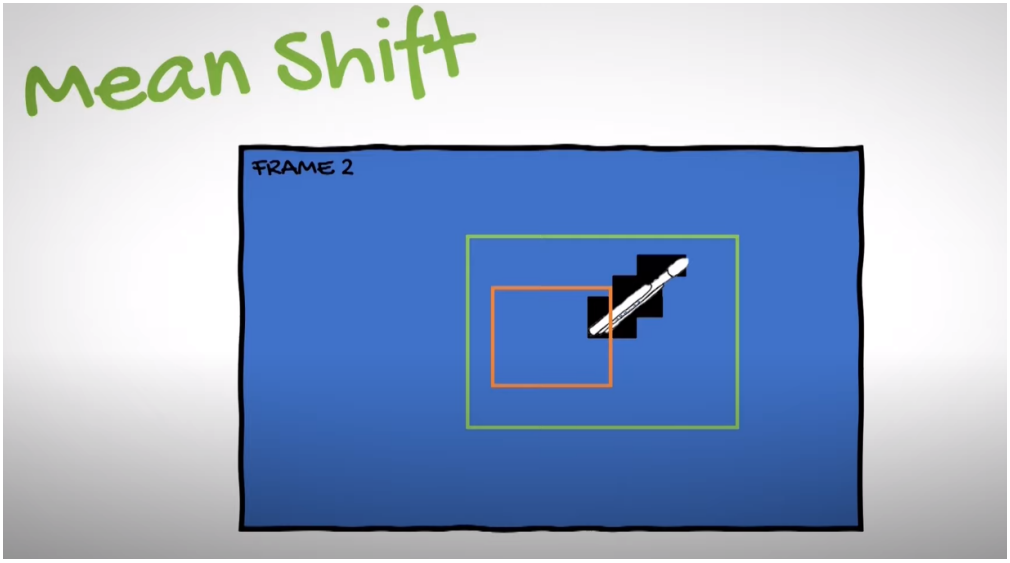

Ilustración 3.3: Vídeo explicativo mostrando la región del vecindario [5]

Pero por supuesto esto no es suficiente para encontrar a nuestro objetivo, por ello también se utiliza una técnica denominada "*Optical flow*" o Flujo óptico.

Flujo óptico

El flujo óptico es el patrón de movimiento aparente de objetos, superficies y bordes en una escena, causado por el movimiento relativo entre un observador y la propia escena. El flujo óptico también puede definirse como la distribución de las velocidades aparentes del movimiento del patrón de brillo en una imagen. El concepto de flujo óptico fue introducido por el psicólogo estadounidense James J. Gibson en la década de 1940 para describir el estímulo visual que reciben los animales que se desplazan por el mundo. Gibson destacó la importancia del flujo óptico para la percepción de la asequibilidad, la capacidad de discernir las posibilidades de acción dentro del entorno [20].

El método se centra en obtener un vector de desplazamiento para el objeto a través de cada fotograma teniendo en cuenta las siguientes características.

- ✔ Consistencia del brillo: En una pequeña región o píxel se asume que el brillo se mantendrá constante, aunque la localización de dicha región pueda cambiar.
- ✔ Coherencia espacial: Los puntos cercanos en la escena suelen pertenecer a la misma superficie y suelen mantener un movimiento similar.
- ✔ Persistencia temporal: Los cambios en movimientos se realizan de manera gradual.
- ✔ Movimiento limitado: Los puntos no se mueven demasiado lejos ni de manera azarosa. Se suele utilizar un método muy conocido llamado Lucas-Kanade, este obtiene la ecuación para la velocidad en un determinado punto. Mediante la ecuación y algunas técnicas de predicción se puede rastrear el objeto a lo largo del vídeo.

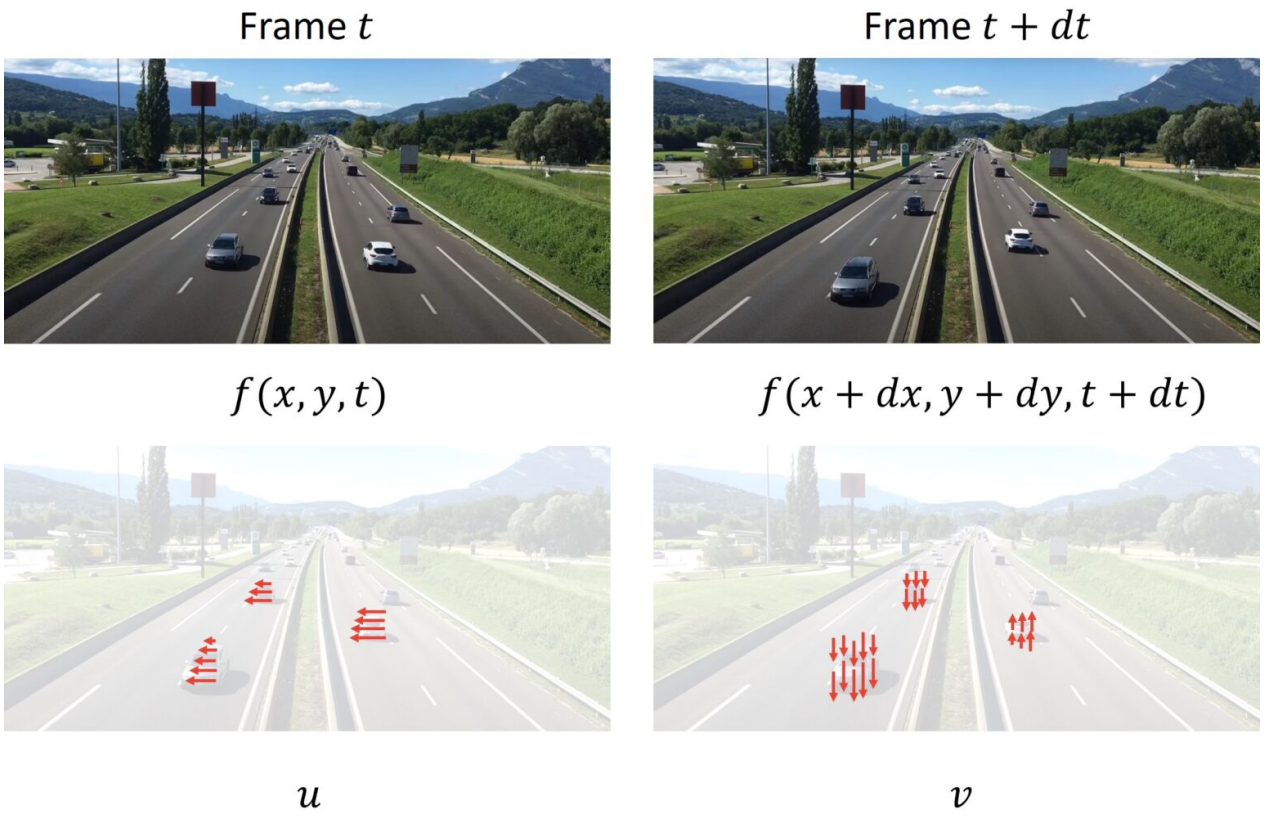

Ilustración 3.4: Vectores de velocidad [6]

Si solo utilizáramos estas dos técnicas clásicas nos daríamos cuenta rápidamente de que Meanshift no es fiable si el objetivo va demasiado rápido y sale de la zona de interés o vecindario. Y esto sin hablar que ocurría si en un fotograma tenemos alguna oclusión, es decir, algún otro objeto que tape a nuestro objetivo. Por este motivo aparece un nuevo método denominado 'Kalman filters' o Filtros de Kalman.

Filtros de Kalman

El filtro de Kalman se basa en aplicar constantes de velocidad a objetos ya detectados en un determinado fotograma. De manera que, si el objeto se pierde de vista, podemos predecir por donde se está moviendo en base a su modelo de velocidad constante. Por supuesto hay múltiples problemas en el mundo real, ya que no podemos esperar que los objetos mantengan siempre una velocidad constante. A esto se le llama "Process Noise".

El filtro de Kalman se utiliza de manera común en navegación de vehículos y en múltiples campos como por ejemplo la econometría.

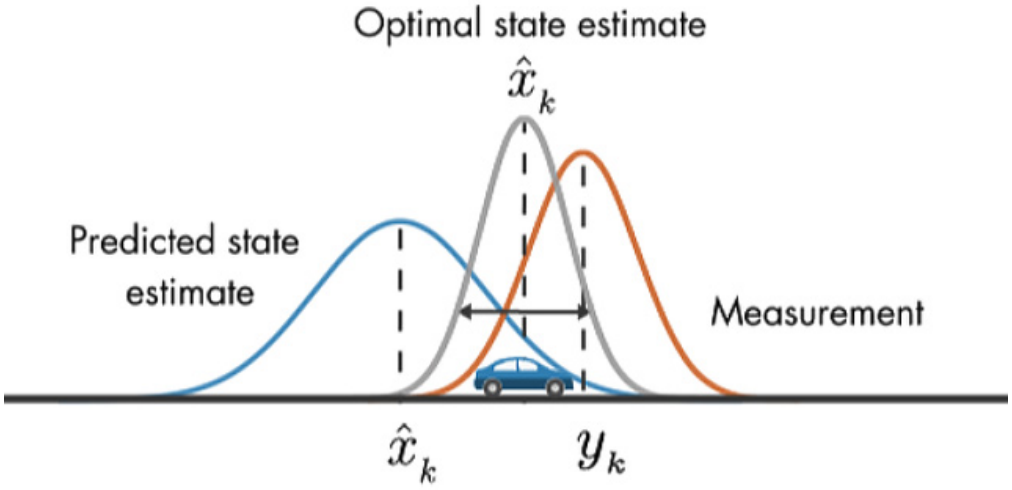

Ilustración 3.5: Filtro de Kalman [7]

El filtro de Kalman es un algoritmo recursivo, por lo cual actúa en tiempo real leyendo sus entradas y mediciones. De esta manera recalcula su estado y su matriz de incertidumbre sin necesidad de información adicional [21].

Combinando estos sistemas podemos predecir la posición del objetivo aunque este no este visible.

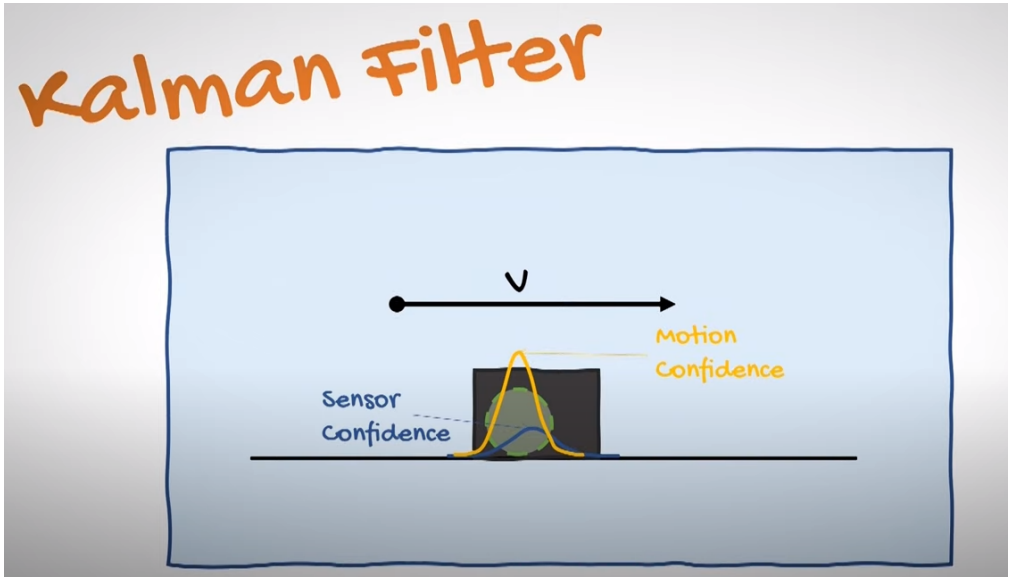

Ilustración 3.6: Vectores de velocidad [5]

SORT

El marco de seguimiento de objetos más popular y uno de los más utilizados y elegantes es DeepSORT, una extensión de SORT (Simple Real Time Tracker) [18].

El filtro de Kalman, es un componente crucial en el DeepSORT. Nuestro estado contiene 8 variables; (u,v,a,h,u',v',a',h') donde (u,v) son los centros de las cajas delimitadoras, a es la relación de aspecto y h, la altura de la imagen. Las demás variables son las velocidades respectivas de las variables.

Como hemos comentado anteriormente, las variables solo tienen factores de posición absoluta y velocidad, ya que estamos asumiendo un modelo de velocidad lineal simple. El filtro de Kalman nos ayuda a tener en cuenta el ruido en la detección y utiliza el estado previo para predecir un buen ajuste de los cuadros delimitadores o cajas.

Para cada detección, creamos un "*Track*" o pista, que tiene toda la información del estado necesaria. También tiene un parámetro para rastrear y eliminar las pistas que tuvieron su última detección exitosa mucho tiempo atrás, ya que esos objetos habrían salido de la escena. Además, para eliminar las pistas duplicadas, hay un umbral del número mínimo de detecciones para los primeros cuadros.

Una vez tenemos los nuevos cuadros delimitadores generados por el filtro de Kalman el siguiente problema reside en asociar las nuevas detecciones con las nuevas predicciones, ya que estas son procesadas de manera independiente. Para resolver esto, necesitamos dos cosas: una métrica de distancia para cuantificar la asociación y un algoritmo eficiente para asociar los datos.

Para la métrica de distancia se utiliza la distancia de Mahalanobis, la cual incorpora las incertidumbres del filtro de Kalman. Esta métrica resulta más precisa que la distancia euclídea y como algoritmo se utiliza el algoritmo estándar húngaro ya que presenta una mayor eficiencia.

Con lo cual tenemos el detector de objetos, tenemos el filtro de Kalman rastreando dichos objetos y dándonos pistas perdidas. Y el algoritmo húngaro resolviendo el problema de asociación. ¿Hace falta el aprendizaje profundo?

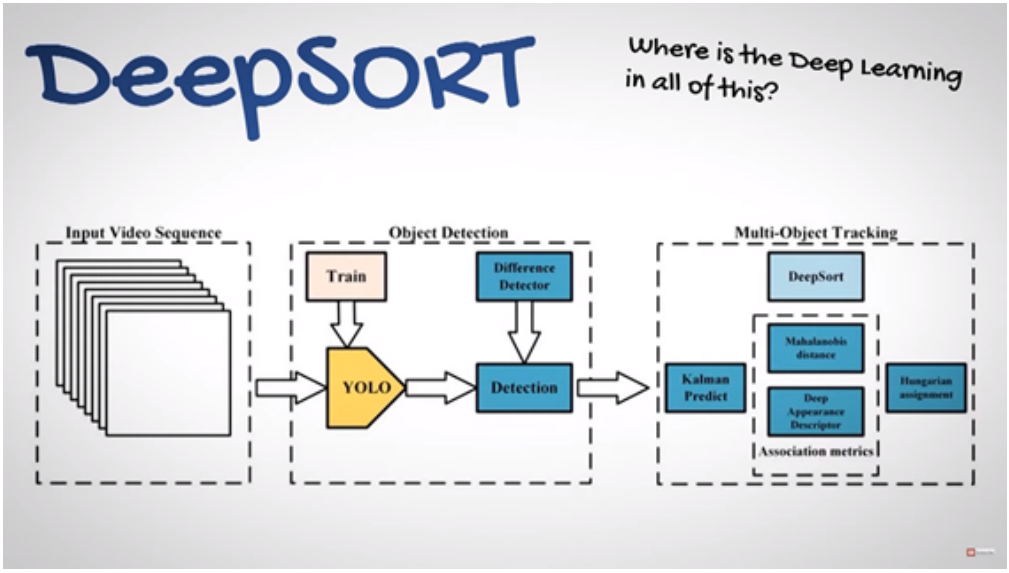

Ilustración 3.7: DeepSORT resumen [5]

Aprendizaje profundo

Si, aunque el filtro de Kalman es efectivo falla múltiples veces en escenarios reales debido a la oclusión o a diferentes puntos de vista etc. Para mejorar su funcionamiento se introduce una nueva métrica de distancia basada en la apariencia del objeto. La idea es obtener un vector que describa todas las características de una imagen dada. Primero construimos un clasificador sobre nuestro conjunto de datos, lo entrenamos hasta que posea una precisión razonable. Por último, nos quedamos con esa última capa clasificatoria. Lo que nos queda es una capa densa que produce un único vector característico esperando a ser clasificado.

Por supuesto hemos barajado otras opciones aparte de DeepSORT + aprendizaje profundo. Estudiamos la viabilidad de otras posibilidades como utilizar Tracktor++, el cual es preciso pero no viable en tiempo real debido a que tiene un promedio de ejecución de 3 fps. Por otro lado tenemos también TrackR-CNN, el cual da segmentación como bonus, pero posee un promedio de 1.6fps en tiempo real. Y por último JDE, con una actuación de 12fps promedio pero con una limitación de imagen de 1088x608, con lo cual debemos esperar menos fps en imágenes HD aunque es bastante preciso.

Sin embargo nuestro ganador, como se esperaba, es DeepSORT, con la mejor precisión y con unos 16fps promedios es el más rápido en ejecutarse.

Nuestro siguiente paso fue investigar que algoritmo de aprendizaje profundo nos podía dar los mejores resultados.

YOLO

Rápidamente destacamos YOLO [8](You Only Look Once) este algoritmo utiliza aprendizaje profundo y CNN para detectar objetos. Se caracteriza porque solo necesita analizar la imagen una sola vez, de esta manera consigue ser más rápido que sus competidores aunque pierda algo de precisión. Es capaz de detectar objetos en tiempo real, para realizar dicha detección primero segmenta la imagen en cuadros. Por cada celda predice N posibles "bounding boxes" o cajas. Por cada caja calcula el nivel de certidumbre. Una vez se obtienen estas predicciones se eliminan las cajas por debajo de cierto umbral. A las cajas restantes se les aplica un método denominado "non-max suppression", el cual elimina posibles objetos duplicados, dejando así el más exacto.

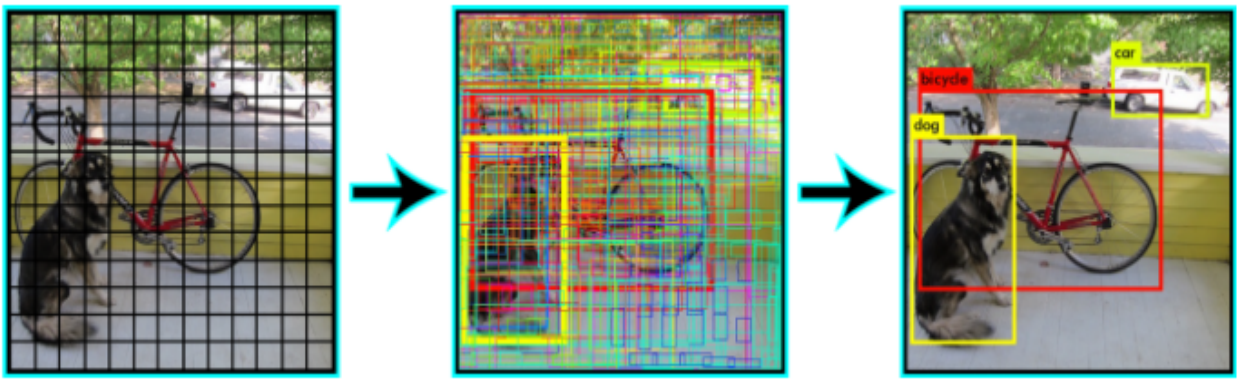

Ilustración 3.8: YOLO funcionamiento [8]

En una primera instancia íbamos a utilizar YOLOV3, cuyo desarrollo finalizó en febrero de 2020 tras dos años (Lanzamiento abril 2018). Su autor Joseph Redmon anunciaba que abandonaba el proyecto por miedo a un impacto negativo de su trabajo, ya que múltiples aplicaciones militares lo utilizaban.

Joseph Redmon: "Dejé de investigar el CV porque vi el impacto que tenía mi trabajo. Me encantaba el trabajo, pero las aplicaciones militares y los problemas de privacidad acabaron siendo imposibles de ignorar" [22].

Pero finalmente descubrimos que en abril de 2020 se lanzó una nueva versión de YOLO creada por Alexey Bochkovskiy, el desarrollador que creo la versión para Window de YOLO, Chien-Yao Wang, y Hong-Yuan Mark Liao ambos del Instituto de Ciencias de la Información Academia Sinica, Taiwán.

En comparación con el anterior YOLOv3, YOLOv4 tiene las siguientes ventajas:

- ✔ Es un modelo de detección muy potente y eficiente, además de su fácil capacidad de entrenamiento.

- ✔ Está influenciado por métodos de detección de objetos como “Bag-of-Freebies” y “Bag-of-Specials” del estado del arte durante el entrenamiento del detector.
- ✔ Los métodos modificados del estado del arte, incluyendo CBN (“Cross-iteration batch normalization”), PAN (“Path aggregation network”) son ahora más eficientes y adecuados para el entrenamiento en una sola GPU.

Para una información más detallada y concisa puede consultar el artículo original. [9]

En el documento oficial mostraron las siguientes tablas diferenciando claramente su nueva versión del detector de todos los demás.

Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	96 (V)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	83 (V)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	62 (V)	**43.5%**	**65.7%**	47.3%	**26.7%**	46.7%	53.3%
EfficientDet: Scalable and Efficient Object Detection [77]									
EfficientDet-D0	Efficient-B0	512	62.5 (V)	33.8%	52.2%	35.8%	12.0%	38.3%	51.2%
EfficientDet-D1	Efficient-B1	640	50.0 (V)	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
EfficientDet-D2	Efficient-B2	768	41.7 (V)	43.0%	62.3%	46.2%	22.5%	**47.0%**	**58.4%**
EfficientDet-D3	Efficient-B3	896	23.8 (V)	45.8%	65.0%	49.3%	26.6%	49.4%	59.8%
Learning Spatial Fusion for Single-Shot Object Detection [48]									
YOLOv3 + ASFF*	Darknet-53	320	60 (V)	38.1%	57.4%	42.1%	16.1%	41.6%	53.6%
YOLOv3 + ASFF*	Darknet-53	416	54 (V)	40.6%	60.6%	45.1%	20.3%	44.2%	54.1%
YOLOv3 + ASFF*	Darknet-53	608×	45.5 (V)	42.4%	63.0%	**47.4%**	25.5%	45.7%	52.3%
YOLOv3 + ASFF*	Darknet-53	800×	29.4 (V)	43.9%	64.1%	49.2%	27.0%	46.6%	53.4%
HarDNet: A Low Memory Traffic Network [4]									
RFBNet	HarDNet68	512	41.5 (V)	33.9%	54.3%	36.2%	14.7%	36.6%	50.5%
RFBNet	HarDNet85	512	37.1 (V)	36.8%	57.1%	39.5%	16.9%	40.5%	52.9%
Focal Loss for Dense Object Detection [45]									
RetinaNet	ResNet-50	640	37 (V)	37.0%	-	-	-	-	-
RetinaNet	ResNet-101	640	29.4 (V)	37.9%	-	-	-	-	-
RetinaNet	ResNet-50	1024	19.6 (V)	40.1%	-	-	-	-	-
RetinaNet	ResNet-101	1024	15.4 (V)	41.1%	-	-	-	-	-

Ilustración 3.9: Comparación de la velocidad y la precisión de diferentes detectores de objetos en el conjunto de datos MS COCO (test-dev 2017). (Aquí se destacan los detectores en tiempo real con fps 30 o superior. Comparamos los resultados con batch=1 sin utilizar tensorRT). [9]

Además mostraron una comparativa con los demás detectores del mercado.

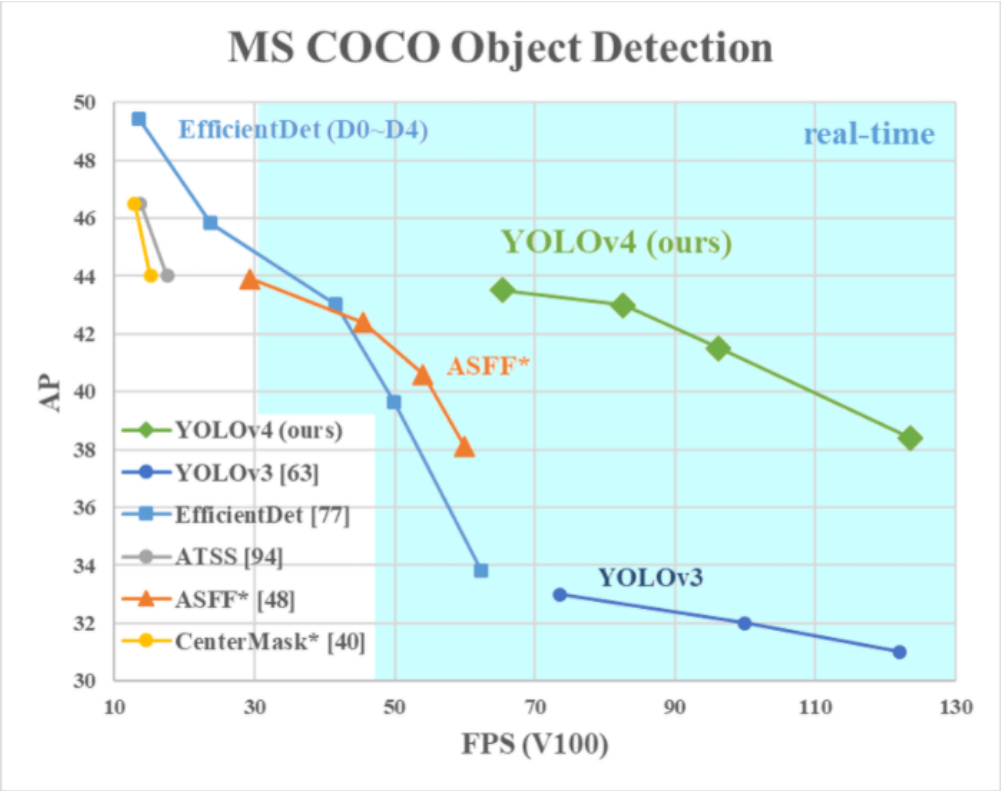

Ilustración 3.10: YOLOV4 comparación con otros detectores [9]

Comparación del YOLOv4 propuesto con otros detectores de objetos de última generación. YOLOv4 funciona dos veces más rápido que EfficientDet y el AP y el fps de YOLOv3 mejora en un 10 % y un 12 %, respectivamente.

Tras ver estos resultados decidimos utilizar YOLOv4 como nuestro detector principal.

3.3.1. Análisis, estudio tecnológico y recopilación de datos

Se determinó el uso de Python como lenguaje principal del proyecto, esto debido a que no solo el detector a utilizar (YOLOv4) se encontraba en dicho lenguaje sino también a las múltiples librerías o módulos de manejo de imágenes que este posee, como puede ser scikit-image o PIL, otras librerías de visión por computador de código abierto como OpenCV o Dlib. Además para la programación de las redes neuronales profundas, se destaca mucho este lenguaje gracias a su incorporación de Keras, modulo el cual facilita el desarrollo de este tipo de métodos usando como base Tensorflow.

Aparte se estudiaron los diferentes entornos de desarrollo que podíamos utilizar, destacando Pycharm o Visual Studio Code debido a su comodidad y soporte para el sistema de control de versiones Git.

Para el entrenamiento de nuestra futura red neuronal se debían recopilar una serie de vídeos de los que extraer los datos a utilizar en nuestro conjunto de datos. Por suerte, mis

tutores ya poseían estos vídeos. Ellos mismos realizaron una serie de grabaciones en la carrera celebrada en la isla el 24 de octubre de 2020, LPATrail y en la Transgrancanaria Classic que se celebró el 6 y 7 de marzo del mismo año.

Capítulo 4

Herramientas utilizadas

4.1. Herramientas organizativas y de entorno

Visual Studio Code

Visual Studio Code [23] es un editor de código fuente que ha sido desarrollado por Microsoft para diferentes sistemas operativos (Windows, Linux y macOS). En nuestro caso todo el desarrollo se realizó en Window 10 y no llegó a trasladarse el proyecto a ningún otro sistema operativo [24].

Se utilizó por su soporte para depuración, integración de Git, auto-completado inteligente de código etc. Además es gratuito y de código abierto. Todo esto junto con la ventaja del conocimiento acumulado por la múltiple utilización de este entorno, que he realizado, nos hizo elegirlo.

GitHub

GitHub [25] es una plataforma que permite alojar proyectos mediante un sistema de control de versiones. Esto permite a los desarrolladores llevar una organización controlada de sus proyectos y trabajar de forma cooperativa.

GitHub fue escrito en Ruby on Rails y ha sido un pilar en la comunidad para divulgar y difundir código libre. Se prefirió utilizar esta herramienta junto con su GitBash debido a la experiencia ya acumulada durante la carrera de Ingeniería Informática.

Overleaf

Overleaf [26] es un editor LaTeX colaborativo basado en la nube que se utiliza para escribir, editar y publicar documentos científicos [27].

Se asocia con una amplia gama de editores científicos para proporcionar plantillas LaTeX oficiales de revistas, y enlaces de presentación directa. En mi caso nunca había utilizado un editor LaTeX debido a la costumbre de utilizar los conocidos paquetes de Office, sin embargo mis tutores me recomendaron realizar la memoria en este editor y dio buenos resultados.

4.2. Herramientas de desarrollo

Python 3.7.0

Python [28] es un lenguaje de alto nivel y multipropósito. Su filosofía prioriza la legibilidad del código mediante el uso de sangría. Sus construcciones de lenguaje, así como su enfoque orientado a objetos, pretenden ayudar a los programadores a escribir un código claro para proyectos tanto de pequeña como de gran escala [29].

Python es de tipado dinámico y recolecta basura. Soporta múltiples paradigmas de programación, incluyendo la programación estructurada (particularmente, la procedimental), la orientada a objetos y la funcional. A menudo se describe a Python como un lenguaje "con pilas incluidas" debido a su completa biblioteca estándar, esta última característica fue un pilar a la hora de elegir el lenguaje. Python posee múltiples bibliotecas que nombraremos a continuación y que fueron base indispensable para el proyecto.

JSON

JSON (JavaScript Object Notation, "notación de objeto de JavaScript") [30] es un lenguaje de marcado que posee un formato de texto diseñado expresamente para el intercambio de datos de manera sencilla. Gracias a su fácil adaptación se utiliza como alternativa a XML y desde 2019 se considera un formato independiente al lenguaje JavaScript [30].

Una de sus ventajas es la facilidad para escribir analizadores sintácticos (parsers) para su formato. En nuestro caso utilizamos este lenguaje de marcado para almacenar los datos de cada fotograma. Posteriormente en la memoria se describirá su utilización y razonamiento.

Anaconda

Anaconda [31] es una distribución de lenguajes de programación que simplifica la gestión y despliegue de paquetes. Es de las herramientas más útiles para trabajar con Python, ya sea en Windows, Linux o macOs.

Su sistema de entornos permite mantener un control de versiones en los paquetes instalados para cada proyecto, algo fundamental hoy en día debido a los múltiples cambios que sufren estos paquetes. En nuestro proyecto se decidió utilizar Anaconda por ese expreso motivo, además el YOLOv4 que utilizamos recomienda también el uso de este programa [32].

Anaconda fue un pilar que facilito la instalación de todas las bibliotecas y paquetes utilizados. Aparte nos permitió, con su sistema de entornos, exportar un archivo yml con todo lo necesario para ejecutar nuestro proyecto.

Tensorflow

TensorFlow [33] es una librería de código abierto para proyectos de machine learning. Posee múltiples bibliotecas y herramientas que permite a los investigadores y desarrolladores realizar complejos proyectos de aprendizaje automático.

En nuestro caso utilizamos Tensorflow-gpu para una ejecución más rápida de los entrenamientos realizados.

Scikit-learn

Scikit-learn [34] es una biblioteca de código abierto diseñada para Python que posee múltiples algoritmos para el aprendizaje automático. Estos algoritmos han sido claves, Scikit-learn es uno de los motivos principales por los que se realizó el proyecto en Python. Se utilizó múltiples algoritmos clásicos de entrenamiento como "Gradient Boosting","Random Forest", "Logistic Regression" etc. También se destaca la utilización de los paquetes "train test split", para la generación de los conjuntos de entrenamiento y test, y el " KBinsDiscretizer" para la discretización de dichos datos [35].

Pandas

Pandas [36] es una biblioteca de código abierto preparada para trabajar junto con NumPy en la organización, manipulación y análisis de datos en Python. El nombre deriva del término "Panel Data", utilizado en econometría. Panda fue otra biblioteca indispensable para la creación de los dataframes, estructura de datos utilizada para organizar y encapsular toda la información recolectada fotograma a fotograma.

Opencv

Opencv [37] o "Open Computer Vision"(Visión Artificial Abierta) es otra biblioteca de código abierto destinada principalmente a la visión por ordenador y desarrollada por Intel. Es una de las bibliotecas más populares en su campo y es utilizada en una gran cantidad de aplicaciones para el reconocimiento de objetos, detección de movimiento, reconstrucción 3D etc.

OpenCV es la biblioteca principal utilizada por YOLOv4, nosotros la utilizamos para visualizar las trayectorias de los corredores, las cuales mostraremos más adelante en esta memoria.

4.3. Hardware de entrenamiento utilizado

Para YOLOv4 ya utilizamos unos pesos entrenados por sus autores, los cuales mostraremos en el apartado de desarrollo, sin embargo para el entrenamiento de nuestros algoritmos utilizamos en una primera instancia la CPU Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz, posteriormente cuando nuestros datos fueron cada vez mayores y la ejecución se convirtió en un proceso lento comenzamos a utilizar la GPU GeForce GTX 980 mediante la configuración de cudnn y cudatoolkit.

Capítulo 5

Desarrollo

El desarrollo de este proyecto ha pasado por múltiples fases de investigación y testeo, con un proceso iterativo semanal, definimos las diferentes posibilidades a tener en cuenta en la siguiente iteración y lo ponemos a prueba.

En esta sección solo se pondrán pequeños fragmentos del código realizado pero el proyecto completo está disponible en Github [38].

Muchos de los *scripts* realizados han sido para uso interno y comprobación de datos, por lo cual los nombraremos pero no explicaremos en mucho detalle, no obstante en el Github se puede encontrar un documento con la descripción de cada *script* y como utilizarlo.

5.1. Detección y recopilación de datos de trayectorias

Tras nuestro periodo de investigación inicial, determinamos que utilizar YOLOv4 era la mejor opción para la detección y recopilación de datos de las trayectorias de las personas en nuestros vídeos de carreras deportivas.

Encontramos un proyecto en Github con muy buenos resultados utilizando YOLOv4 para la detección de múltiples clases. Este proyecto fue realizado por The AI Guy, sobrenombre que utiliza su autor tanto en su canal de youtube, donde explica su proyecto paso a paso, como en Github [10] [39].

Este proyecto por defecto utiliza unos pesos ya entrenados para catalogar hasta 80 clases diferentes, por supuesto para nuestro proyecto solo nos interesa una clase, personas, esa fue la primera modificación que realizamos.

Ilustración 5.1: Detector YOLOv4 acción [10]

Tras esta pequeña modificación aprovechamos los datos que el detector recopila para imprimir las cajas en cada fotograma y almacenamos dichos datos.

Cada fotograma imprime una caja en cada persona detectada y nuestra modificación crea un archivo JSON almacenando dichos datos de la siguiente manera.


```
{"Frame": 3996, "Filename": "lpatrail-
clip01.mp4", "Persons": {"24": {"id": 24, "x": 270, "y": 304, "width": 551, "height": 1082, "Cen
troid(W,H)": [410, 693], "Type": "Runner"}, "33": {"id": 33, "x": 1097, "y": 425, "width": 1137,
 "height": 539, "Centroid(W,H)": [1117, 482], "Type": "Runner"}}}
```


Ilustración 5.2: JSONs generados

De cada persona se almacena: ID o etiqueta generada por el YOLOv4, posición X, Y de la caja, Ancho (*Width*) y Alto (*Height*) de la caja, tipo de persona (en este caso todos serán corredores de manera inicial y posteriormente se etiquetarán) y por último se realiza el cálculo del centroide o punto central de la caja de cada persona. Una vez recopiladas todas las personas dentro de ese fotograma, se almacena el número del fotograma y el nombre del vídeo junto con un array con todas las personas que aparecen.

Width y el Height no son realmente el ancho y el alto, sino que son la coordenada extrema de la caja. Esto es así para poder generar dichas cajas. El cálculo del centroide se realiza restando la posición X Y con su ancho y altura respectiva y posteriormente se divide entre dos para obtener el punto central.

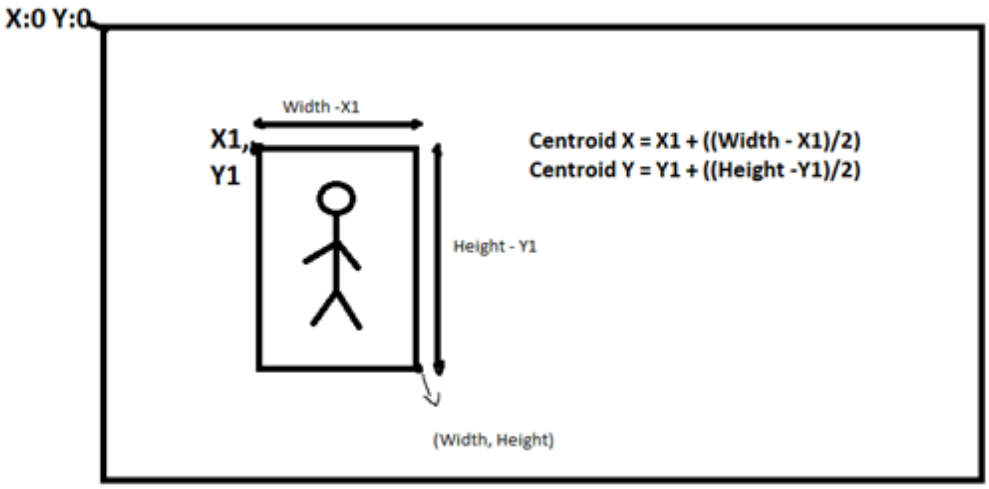

Ilustración 5.3: Explicación de los datos recopilados

Finalmente lo que obtenemos es un archivo JSON por cada fotograma del vídeo con todos los datos de cada persona que aparece en dicho fotograma. (este código se ha realizado modificando el archivo object_tracker.py del YOLOv4)

Para este proyecto se han utilizado unos 26 fragmentos de vídeos, todos ellos grabados a 50 fotogramas por segundo y con una duración variable entre 1 y 4 minutos. Tras procesar cada vídeo obtuvimos 46555 archivos JSON con todos estos datos y para comprobar veracidad decidimos realizar un mapa de calor por cada vídeo.

En estos mapas de calor podemos visualizar las trayectorias generadas por los integrantes del vídeo imprimiendo cada centroide como un punto individual.

Utilizamos dos representaciones distintas para visualizar los datos:

- ✔ Representar los centroides mediante un punto nos permite distinguir claramente a cada centroide individualmente. Sin embargo al representar las trayectorias con líneas se pueden dar uniones erróneas, al producirse un salto en el detector nos generaría una línea que cruzaría la pantalla. Es por esto que como primera instancia representamos los centroides con puntos y posteriormente con líneas pero controlando dichos saltos.

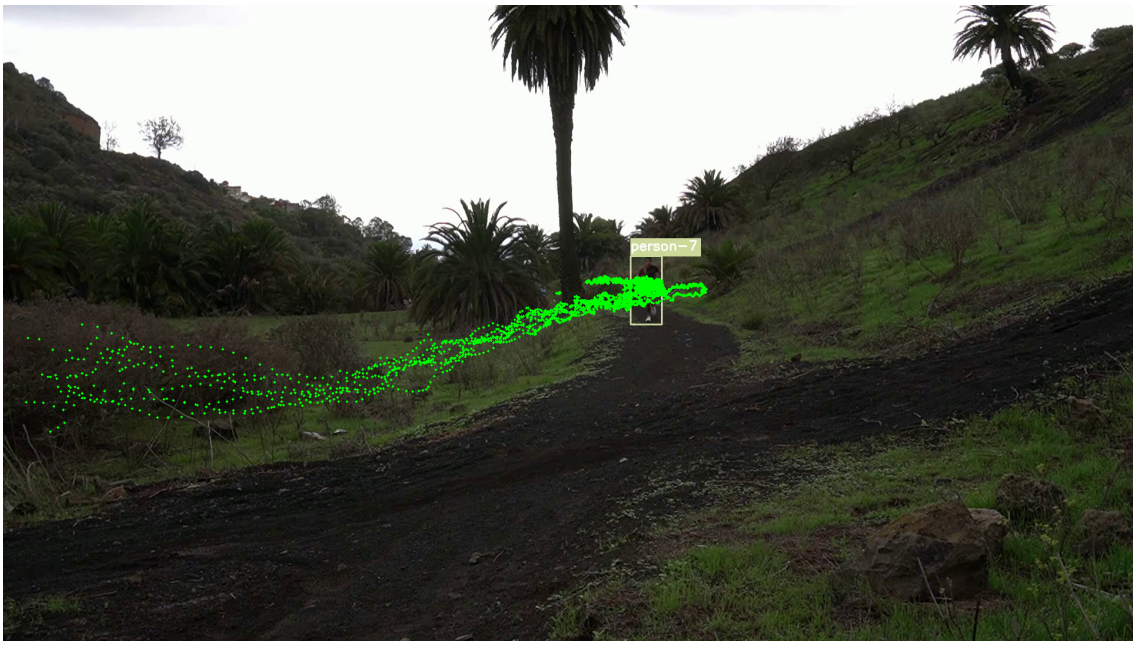

Ilustración 5.4: 24 de octubre de 2020, Carrera LPATrail, representación de trayectorias mediante círculos

- ✔ Representar las trayectorias mediante líneas nos permite distinguir los saltos y fallos del detector, cuando generamos la imagen por primera vez se podían observar líneas que cruzaban toda la imagen, esto se debía a los errores producidos por el detector, el cual perdía a su objetivo durante un fotograma.

  Para limpiar el mapa de calor de dichos errores recopilamos información de los movimientos de cada persona cada 10 fotogramas, si su movimiento actual no supera el triple de la media de desplazamiento de esos 10 fotogramas anteriores, imprimimos dicha línea y añadimos este movimiento para el cálculo de la siguiente media de desplazamiento.

  Así cada vez que se produce un salto repentino de un fotograma a otro, es completamente descartado. Es por ello que todas nuestras imágenes en los mapas de calor se ven así de limpias.

Ilustración 5.5: 24 de octubre de 2020, Carrera LPATrail, representación de trayectorias mediante líneas

De manera inicial todos los ID's recopilados en los archivos JSON están etiquetados como 'Runners', corredores. Con lo cual nuestro siguiente paso era visualizar cada vídeo y buscar una manera eficiente de etiquetar a cada persona.

5.1.1. Etiquetado

Para esta labor realizamos un *script* (JsonLabeller.py) el cual lee un archivo txt. Si el programa encuentra una serie de números, buscará dichos números (ID's) en los archivos JSON del vídeo correspondiente y cambiará sus etiquetas a 'Runner', además todo aquel ID que no se encuentre en el archivo txt será modificado a 'Public'.

Por otro lado, si encuentra la palabra 'all' etiquetará a todos los usuarios del vídeo como 'Runner' y si encuentra la palabra 'none' los etiquetara a todos como 'Public'.

Una vez realizado el *script* solo quedaba visualizar cada vídeo e ir anotando en cada txt los ID's correspondientes.

Se nos plantearon soluciones más directas pero esta nos permitía re-etiquetar fácilmente los JSONs.

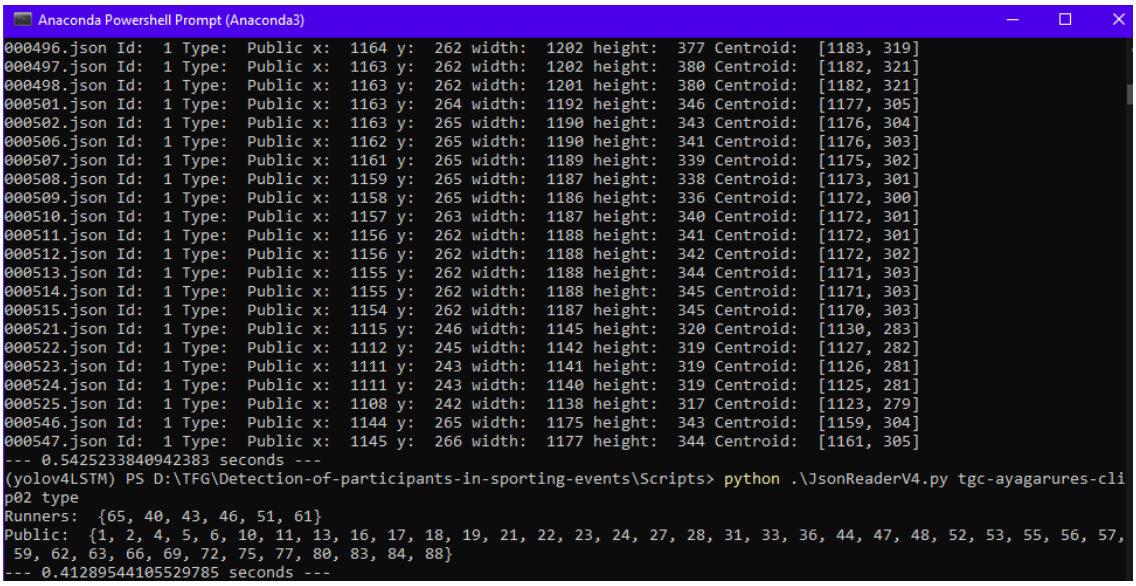

Ilustración 5.6: Visualización de los datos recopilados

Una vez etiquetados los 26 vídeos realizamos nuevamente los mapas de calor pero esta vez mostrando los corredores como verde y el público como rojo. Estos mapas de calor ha sido generados con el script JsonAnalyser.py.

Ilustración 5.7: Mapa de calor diferenciando puntos de público(rojo) y corredores(verde)

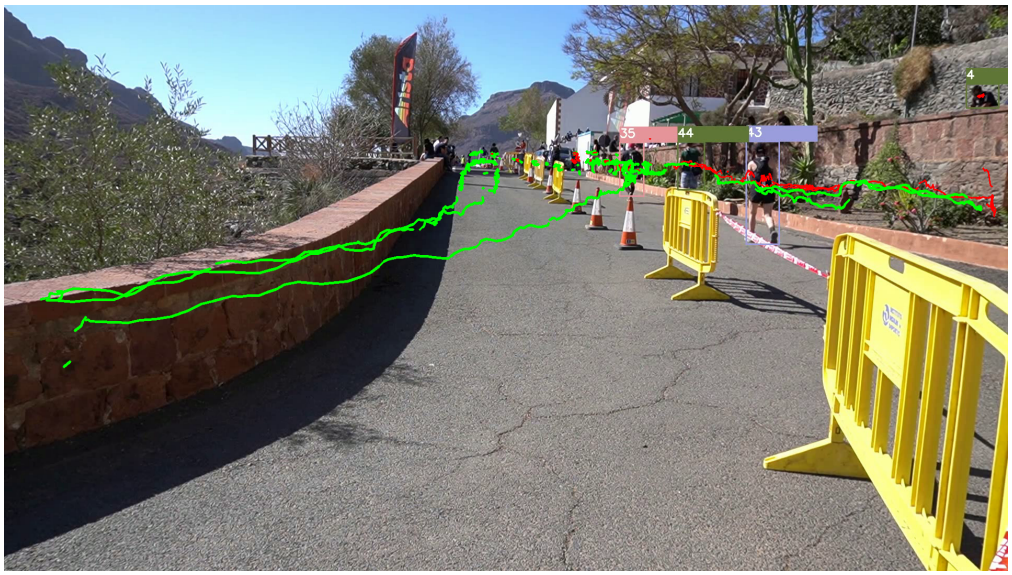

Ilustración 5.8: Mapa de calor diferenciando líneas de público (rojo) y corredores (verde)

5.1.2. Cálculo de características

Para este proyecto pensamos en múltiples características que podríamos sacar de los datos recopilados y que podrían ser útiles a la hora de entrenar los algoritmos.

Entre ellas destacamos:

- ✔ Velocidad, el cálculo de la velocidad se realiza en dos fotogramas, por lo cual el tiempo transcurrido será de 2 entre el número de fotogramas por segundo que tenga el vídeo. Tener en cuenta que no estamos calculando la velocidad real de la persona sino la velocidad en píxeles. La distancia entre dos puntos en el espacio la calculamos utilizando el teorema de Pitágoras y dividiendo esta entre el tiempo tenemos la velocidad. Calculamos también la velocidad promedio en 5 fotogramas. Esta característica nos permitirá identificar posibles saltos o fallos del detector, ya que si faltan fotogramas o el cambio de posición de un fotograma a otro es muy brusco se dará una velocidad desproporcionada. Más adelante comentaremos el estudio realizado para estos errores.
- ✔ Orientación, el cálculo de la orientación la realizamos utilizando la función atan2 o arcotangente de dos parámetros. De esta manera calculamos el ángulo de la trayectoria entre dos fotogramas, es decir, el ángulo que posee una persona en un fotograma con respecto al fotograma anterior. A continuación, sumamos 360^{o} a aquellos ángulos negativos. Nos permite determinar el ángulo de movimiento seguido por cada usuario.

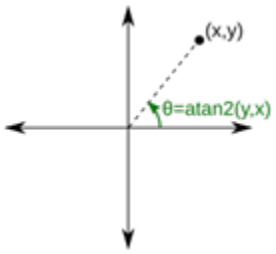

Ilustración 5.9: Función atan2 [11]

✔ Altura de la caja, este dato lo calculamos partiendo de los datos mostrados en la ilustración 8.3, restando su posición Y1 y el punto Height tenemos la altura de la caja. Este dato es muy importante, ya que nos permite distinguir entre personas al fondo de la imagen (alturas menores) y personas muy cerca de la cámara.

Mediante el *script* JsonCharacteristic.py podemos visualizar dichas características.

```
ID:  1
Type:  Runner
[ Frame:  000364.json  ]
Speed:  0.0 | Orientation:  0.0 | BoxHeight:  66 |SpeedAverange(5 Frames):  0.0
[ Frame:  000365.json  ]
Speed:  35.36 | Orientation:  315.0 | BoxHeight:  65 |SpeedAverange(5 Frames):  17.5
[ Frame:  000366.json  ]
Speed:  0.0 | Orientation:  0.0 | BoxHeight:  64 |SpeedAverange(5 Frames):  11.666666666666666
[ Frame:  000367.json  ]
Speed:  0.0 | Orientation:  0.0 | BoxHeight:  64 |SpeedAverange(5 Frames):  8.75
[ Frame:  000368.json  ]
Speed:  75.0 | Orientation:  0.0 | BoxHeight:  65 |SpeedAverange(5 Frames):  22.0
[ Frame:  000369.json  ]
Speed:  35.36 | Orientation:  45.0 | BoxHeight:  66 |SpeedAverange(5 Frames):  29.0
[ Frame:  000370.json  ]
Speed:  0.0 | Orientation:  0.0 | BoxHeight:  66 |SpeedAverange(5 Frames):  22.0
[ Frame:  000371.json  ]
Speed:  134.63 | Orientation:  291.8 | BoxHeight:  55 |SpeedAverange(5 Frames):  48.8
[ Frame:  000372.json  ]
Speed:  50.0 | Orientation:  270.0 | BoxHeight:  50 |SpeedAverange(5 Frames):  58.8
[ Frame:  000373.json  ]
Speed:  25.0 | Orientation:  0.0 | BoxHeight:  49 |SpeedAverange(5 Frames):  48.8
[ Frame:  000374.json  ]
Speed:  125.0 | Orientation:  90.0 | BoxHeight:  63 |SpeedAverange(5 Frames):  66.8
[ Frame:  000375.json  ]
Speed:  50.0 | Orientation:  90.0 | BoxHeight:  67 |SpeedAverange(5 Frames):  76.8
[ Frame:  000376.json  ]
Speed:  25.0 | Orientation:  90.0 | BoxHeight:  68 |SpeedAverange(5 Frames):  55.0
[ Frame:  000377.json  ]
Speed:  0.0 | Orientation:  0.0 | BoxHeight:  69 |SpeedAverange(5 Frames):  45.0
```

Ilustración 5.10: Visualización de las características del corredor con ID 1 en el vídeo de LPATrail

5.1.3. Análisis de las características y cálculos de error

Una vez obtenido todos estos nuevos datos comenzamos a analizar los posibles factores de error producidos por YOLOv4.

Tras ver los vídeos procesados por YOLOv4, es fácil distinguir pequeños errores a la hora de catalogar a las personas, por ejemplo, YOLO puede catalogar a una persona con el ID

1 y posteriormente catalogarla con el ID 30, ya sea por la oclusión de un objeto o de otra persona.

Ilustración 5.11: Ejemplo de error producido por YOLOv4

En la primera imagen de la figura 5.11 podemos distinguir al corredor de camiseta verde con ID 18, detrás de él, vemos a un corredor con camiseta verde sin identificar y a otro corredor con camiseta blanca e ID 19.

En la siguiente imagen vemos como YOLO ha confundido al corredor de camiseta blanca con el de camiseta verde, que no estaba identificado, y lo ha etiquetado como número 19. Por último en el siguiente fotograma vemos como YOLO a re-etiquetado al corredor de camiseta blanca con el ID 28.

Este error produce duplicados de corredores y grandes saltos en los cálculos de los datos. La peor parte de estos errores es cuando se producen entre un corredor y un espectador. Ya que el ID 26 puede ser en 200 fotogramas un corredor y en otros 200 un espectador.

Para entender cuanto de graves podían ser estos errores realizamos una serie de *scripts* definiendo unas métricas de error.

✔ Saltos de velocidad: Tras un análisis de todos los vídeos los mayores saltos en velocidad

que no presentaban errores eran de unos 800, con lo cual se estableció como error todos aquellos saltos de velocidad superiores a 1000.

- ✔ Saltos de velocidad promedio: Para el promedio de velocidad se estableció como error aquellas velocidades que fueran dos veces mayores que en su fotograma anterior.
- ✔ Altura de caja: Para la altura debemos tener en cuenta muchos más casos que para la velocidad, ya que la velocidad solo tenemos que controlar incrementos, pero en la altura debemos controlar que la progresión de la caja sea uniforme o sin saltos bruscos. Esto lo realizamos calculando la diferencia de altura entre el fotograma anterior y el actual. Si dicha diferencia es más del doble que la altura original anterior o actual sabemos que se ha producido un error.

Ilustración 5.12: Clip: tgc-ayagaures-02, Fotograma: 2552, Velocidad: 100.0, Orientación: 0.0, Altura: 327, Velocidad Promedio: 104.0

Ilustración 5.13: Clip: tgc-ayagaures-02, Fotograma: 2596, Velocidad: 13432.35, Orientación: 188.78, Altura: 66, Velocidad Promedio: 2768.2

Como podemos apreciar en las imágenes 5.12, 5.13 YOLO pierde al ID 10 durante múltiples fotogramas debido a que este ya ha salido del plano. Posteriormente confunde a una persona del fondo de la imagen con este ID 10.

En los datos esta confusión se traduce como un salto enorme en velocidad y en un decremento de la altura de la caja muy brusco.

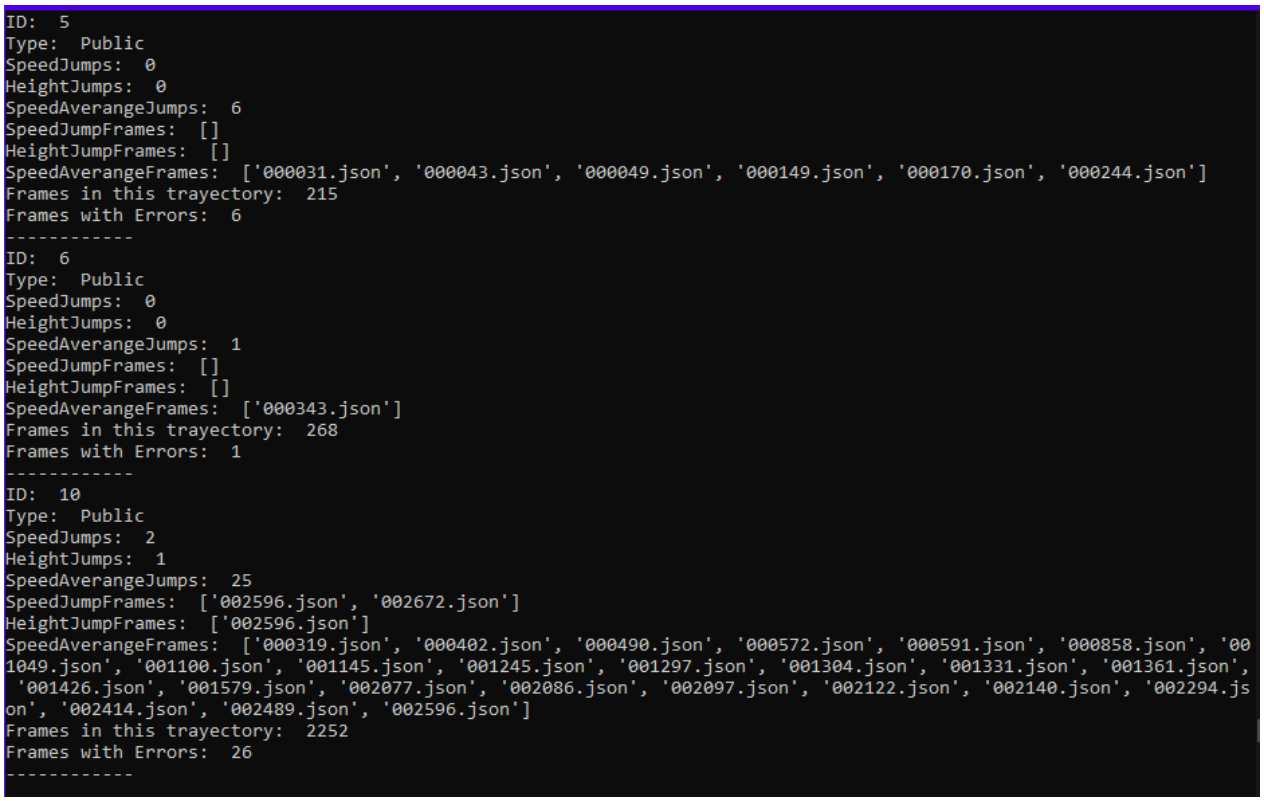

Ilustración 5.14: Recopilación de errores en el clip: Ayagaures-02

Nuestro *script* contabiliza todos estos errores producidos por cada ID y nos calcula el número de fotogramas con errores.

Finalmente realiza estas operaciones para todos los vídeos y da los porcentajes de error, los cuales podemos observar en la figura 5.15.

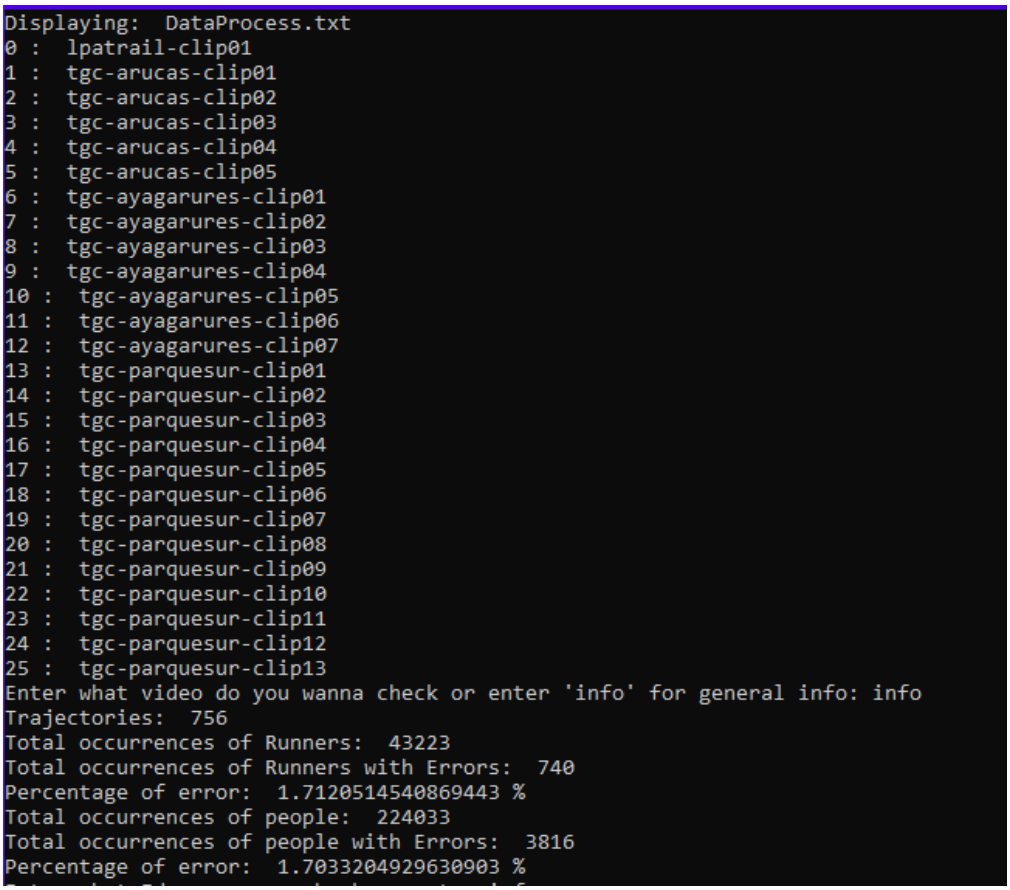

Ilustración 5.15: Cálculo de errores realizados en todos los vídeos

Como podemos observar en la imagen, tenemos 756 trayectorias de personas en nuestros 26 vídeos. En esas trayectorias poseemos 224.033 fotogramas de datos de los cuales solo 3.816 presentan errores. Hablamos un porcentaje de error del 1,7 %, un error muy bajo.

Además de esos 224.033 fotogramas 43.223 son de corredores, y de esos solo 740 son error, un 1,71 %.

Estos datos nos muestra como a pesar de los errores que pueda producir YOLOv4 y las alteraciones en los datos no es necesario ninguna intervención ni modificación a nuestro conjunto de datos.

Discretización

El siguiente paso fue discretizar estos datos para poder realizar un histograma de las trayectorias y probar diferentes entrenamientos.

La velocidad promedio y la velocidad por cada fotograma las discretizamos utilizando una función de sklearn llamada KBinsDiscretizer. Esta función nos permite discretizar los datos de aplicando una estrategia por cuantiles o que significa que cada bin tiene aproximadamente

el mismo número de muestras. Utilizando esta estrategia discretizamos la velocidad entre 0-8 y la velocidad promedio entre 0-10.La función KbinsDiscretizer tras procesar nuestros datos genera los siguientes límites para cada segmento.

Velocidad:

$0 \leq x < 25 = 0$

$25 \leq x < 35 = 1$

$35 \leq x < 50 = 2$

$50 \leq x < 55 = 3$

$55 \leq x < 75 = 4$

$75 \leq x < 103 = 5$

$103 \leq x < 175 = 6$

$175 \leq x < 3372 = 7$

$3372 \leq = 8$

Ilustración 5.16: Límites definidos por la función KbinsDiscretizer de sklearn para la velocidad

Velocidad promedio:

$0 \leq x < 5 = 0$

$5 \leq x < 10 = 1$

$10 \leq x < 15 = 2$

$15 \leq x < 20 = 3$

$20 \leq x < 25 = 4$

$25 \leq x < 32 = 5$

$32 \leq x < 43 = 6$

$43 \leq x < 62 = 7$

$62 \leq x < 105 = 8$

$105 \leq x < 8237 = 9$

$8237 \leq = 10$

Ilustración 5.17: Límites definidos por la función KbinsDiscretizer de sklearn para la velocidad promedio

Podemos observar como la velocidad promedio centra sus límites en niveles inferiores,

esto es lógico, ya que los valores están calculados como el promedio de 5 fotogramas, con lo cual, los valores son menores en general, de hecho aunque este el nivel 10 ninguna trayectoria llega a dicho límite y si llegara se consideraría error debido a su alto valor.

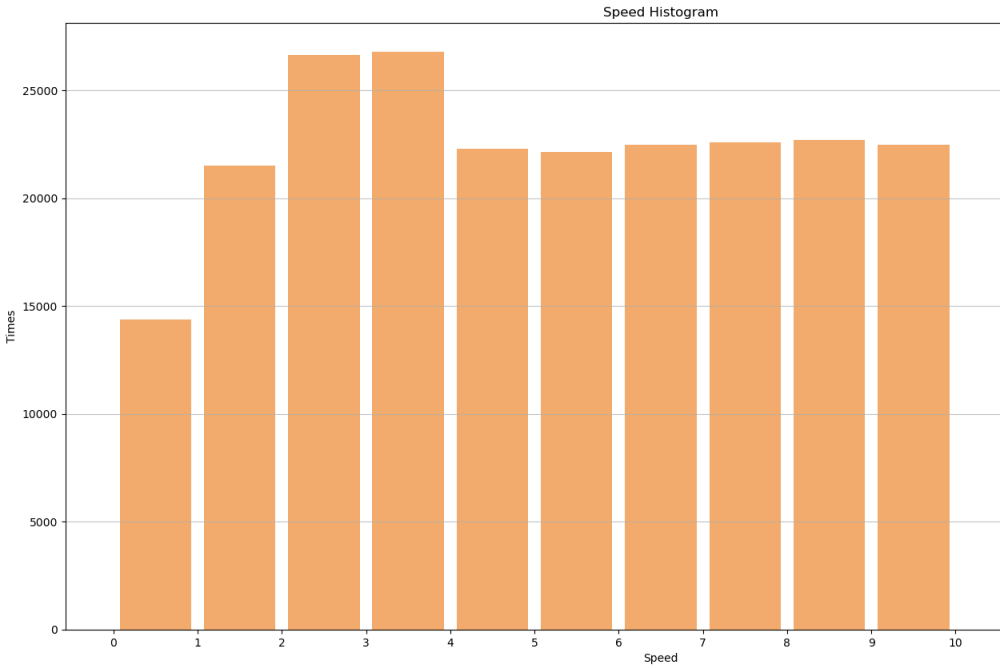

Ilustración 5.18: Histograma de la velocidad promedio

En el caso de la altura la discretizamos entre 100, de manera que los datos nos quedan entre 0-10

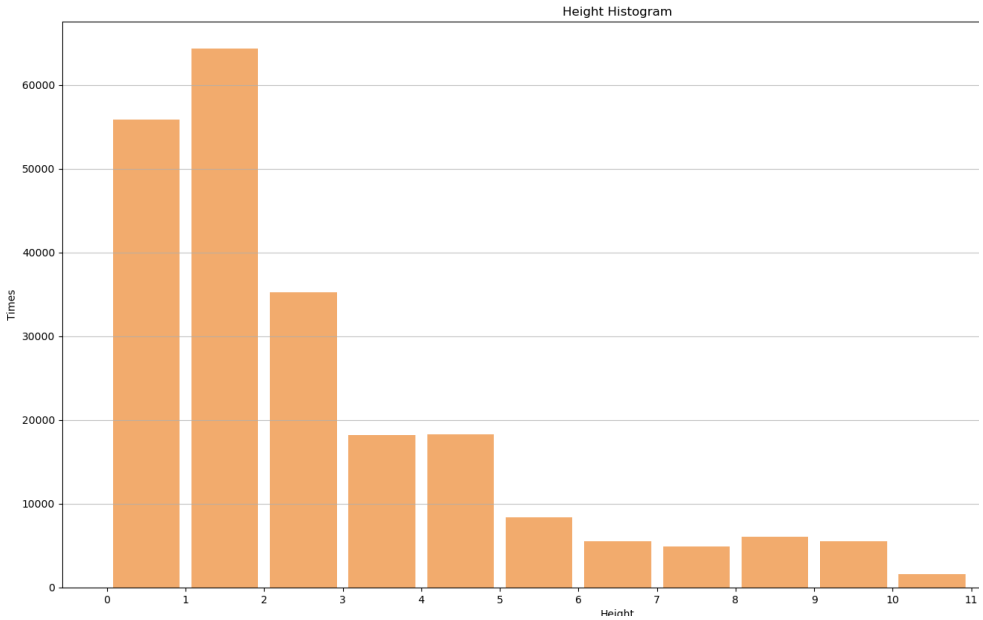

Ilustración 5.19: Histograma de la altura

Para la orientación realizamos una discretización algo más compleja, diferenciando los ocho puntos cardinales clásicos y en porciones definidas por los siguientes rangos:

Norte: 337.5 º> x ≤22.5º

Noreste: 22. 5º < x≤67.5º(45+22.5)

Este: 67. 5º <x≤112.5º(90+22.5)

Sureste: 112.5º<x≤157.5º(135+22.5)

Sur: 157.5º<x≤202.5º(180+22.5)

Suroeste: 202.5º<x ≤247.5º(225+22.5)

Oeste: 247.5º<x≤292.5(270+22.5)

Noroeste:292. 5<x<337.5º (315+22.5)

Ilustración 5.20: Rangos definidos para la orientación

Estos rangos fueron calculados dividiendo los 360 en segmentos de 8 y estos a su vez entre dos segmentos lo que nos da divisiones de 22.5, que corresponden a sectores de tamaño 45º. En el siguiente dibujo 5.21 se puede apreciar las divisiones fácilmente.

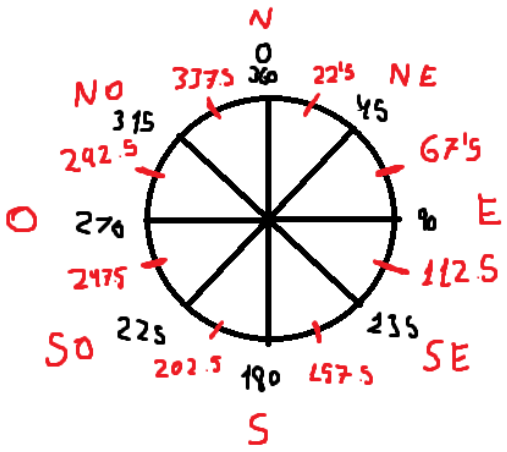

Ilustración 5.21: Dibujo explicativo de los rangos elegidos

Ilustración 5.22: Fotograma 915 de la carrera de LPATrail

Todos nuestros vídeos están grabados a 50 fotogramas por segundo, lo que nos facilita el acceso rápido a los minutos de cada vídeo. Por ejemplo este es el fotograma 915 ubicado en el segundo 18.30 donde podemos observar al corredor etiquetado con el número 7 por YOLOv4 y los datos que hemos extraído de este fotograma son los siguientes:

-Sin discretización

Velocidad: 25.0 - Orientación: 180.0 - Altura: 322 - Velocidad Promedio: 57.6

-Discretizado

Velocidad: 1 - Orientación: Sur - Altura: 3 - Velocidad Promedio: 7

5.2. Primeras pruebas mediante algoritmos clásicos

Como aproximación inicial decidimos realizar una serie de pruebas aprovechando los múltiples algoritmos que posee la biblioteca SKlearn, todas las pruebas realizadas se han hecho con los datos discretizados.

Creación de conjunto de datos:

El primer paso es crear el dataframe con los datos recopilados:

```
Int64Index: 224033 entries, 0 to 224032
Data columns (total 8 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Video          224033 non-null  object
 1   Frame          224033 non-null  object
 2   ID             224033 non-null  int64
 3   Speed          224033 non-null  int64
 4   Orientation    224033 non-null  object
 5   Height         224033 non-null  int64
 6   SpeedAverange  224033 non-null  int64
 7   Type           224033 non-null  object
dtypes: int64(4), object(4)
memory usage: 15.4+ MB
DataSet:  None
```

Ilustración 5.23: Dataframe sin modificar

	Video	Frame	ID	Speed	Orientation	Height	SpeedAverange	Type
0	lpatrail-clip01	000364.json	1	0	North	0	0	Runner
1	lpatrail-clip01	000365.json	1	2	NorthWest	0	3	Runner
2	lpatrail-clip01	000366.json	1	0	North	0	2	Runner
3	lpatrail-clip01	000367.json	1	0	North	0	1	Runner
4	lpatrail-clip01	000368.json	1	5	North	0	4	Runner
...	...	...	..	...	...	...	...	...
224028	tgc-parquesur-clip13	001597.json	36	0	North	0	1	Public
224029	tgc-parquesur-clip13	001598.json	36	0	North	0	1	Public
224030	tgc-parquesur-clip13	001599.json	36	0	North	0	0	Public
224031	tgc-parquesur-clip13	001600.json	36	0	North	0	0	Public
224032	tgc-parquesur-clip13	001601.json	36	0	North	0	0	Public

Ilustración 5.24: Dataframe sin modificar, ejemplo de datos

Ahora debemos adaptar este dataframe para que pueda ser procesado por los algoritmos, como es lógico debemos eliminar los campos que no aportan información y modificar los que no estén en el formato apropiado.

Eliminamos la columna Vídeo y Frame (Fotograma). La columna Type (Tipo) será también eliminada de nuestro dataframe pero esta será nuestra función a comprobar, además binarizaremos esta columna de manera que Public = 0 y Runner = 1, de esta manera mantendremos todo en el mismo formato numérico.

Para la columna Orientation (Orientación) se utilizará la función de pandas “get dummies” la cual genera tantas columnas como opciones posee Orientación, es decir, insertara 8 columnas, una para cada punto cardinal y pondrá todas a 0 menos la que se encontraba en la columna original.

```
Int64Index: 224033 entries, 0 to 224032
Data columns (total 13 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   ID             224033 non-null  int64
 1   Speed          224033 non-null  int64
 2   Height         224033 non-null  int64
 3   SpeedAverange  224033 non-null  int64
 4   Type           224033 non-null  object
 5   East           224033 non-null  uint8
 6   North          224033 non-null  uint8
 7   NorthEast      224033 non-null  uint8
 8   NorthWest      224033 non-null  uint8
 9   South          224033 non-null  uint8
 10  SouthEast      224033 non-null  uint8
 11  SouthWest      224033 non-null  uint8
 12  West           224033 non-null  uint8
dtypes: int64(4), object(1), uint8(8)
memory usage: 12.0+ MB
```

Ilustración 5.25: Dataframe con la columna orientación modificada y sin las columnas Video y Frame

Tras la figura número 5.25 solo faltaría eliminar la columna 4, Type (tipo) y tendríamos listo los datos. A continuación contabilizamos el número de corredores y de público, posteriormente mediante la función “train test split” dividimos los datos en dos conjuntos, el conjunto de entrenamiento la cual pasará por los diversos procesos de los algoritmos y el conjunto de prueba o test, el cual se utilizará para comprobar el nivel de efectividad.

Los siguientes datos son los usados en todos los algoritmos, utilizamos exactamente los mismos datos para poder sacar conclusiones sobre que algoritmo es más eficaz en nuestro proyecto.

Tenemos 224.033 ocurrencias de personas, de esas, 180.810 son Público y 43.223 son corredores. Dividimos nuestro set cogiendo un 30 % para tests, de manera que nos quedan 156.823 ocurrencias para el entrenamiento y 67.210 para tests. De esas 156.823, 126.584 son público y 30239 son corredores. De los 67.210, 54.226 son público y 12984 son corredores. Es importante tener clara las proporciones utilizadas en el entrenamiento, ya que no queremos entrenar con pocos datos de corredores.

Estos primeros entrenamientos vamos a realizarlos utilizando todos los datos en crudo, pero más adelante en la memoria explicaremos la segunda metodología donde agrupamos los datos en sus respectivas trayectorias realizando un histograma para cada una.

Decision Tree

Los árboles de decisión son una estructura en árbol donde cada nodo interno representa una prueba sobre un atributo, cada rama representa el resultado de dicha prueba y cada nodo de la hoja la etiqueta, la cual se calcula tras probar todos los atributos. De esta manera Decision Tree halla la estrategia de mayor probabilidad. Es un sistema de predicción basado en reglas, muy similar a un diagrama de flujo y muy utilizado en inteligencia artificial y economía.

Decidimos utilizar este algoritmo debido a su sencilla preparación, ya que no requiere ninguna normalización ni escalado previo en los datos y por su fama de dar resultados de fácil interpretación.

Tras entrenar con este algoritmo obtenemos los siguientes datos:

```
--- DECISION TREE ---
-------------------------------------
--- Confusion matrix ---
[[53591   635]
 [10220  2764]]
--- Classification report ---
              precision    recall  f1-score   support

           0       0.84      0.99      0.91     54226
           1       0.81      0.21      0.34     12984

    accuracy                           0.84     67210
   macro avg       0.83      0.60      0.62     67210
weighted avg       0.83      0.84      0.80     67210

--- Accuracy score ---
0.8384912959381045
```

Ilustración 5.26: Resultados tras procesar los datos mediante Decision Tree

Obtenemos un 83 % de precisión, el cual es un valor aceptable pero creemos que se puede superar con modelos más complejos, ya que el árbol de decisión presenta una serie de desventajas.

Un pequeño cambio en los datos puede provocar un gran cambio en la estructura del árbol causando inestabilidad. Este algoritmo tampoco es bueno para predecir valores continuos a través de regresión, ya que son modelos estrictamente de clasificación [40].

Random Forest

También conocidos como bosques de decisión o bosques aleatorios es un algoritmo de aprendizaje automático para la clasificación y regresión. Se implementa mediante la construcción de múltiples árboles de decisión y corrigen la tendencia a sobre-ajustarse (*overfitting*)

que estos tienen, además suelen dar mejores resultados. Por estos motivos decidimos probar nuestro conjunto de datos en este algoritmo [41].

```
--- RANDOM FOREST ---
-----------------------------------
--- Confusion matrix ---
[[51664  2562]
 [ 4821  8163]]
--- Classification report ---
              precision    recall  f1-score   support

           0       0.91      0.95      0.93     54226
           1       0.76      0.63      0.69     12984

    accuracy                           0.89     67210
   macro avg       0.84      0.79      0.81     67210
weighted avg       0.88      0.89      0.89     67210

--- Accuracy score ---
0.8901502752566582
```

Ilustración 5.27: Resultados tras procesar los datos mediante Random Forest

Como podemos observar ya en nuestro segundo intento hemos mejorado considerablemente nuestro resultado de precisión. En concreto pasamos a un 89 %.

Gradient Boosting

La potenciación del gradiente o "Gradient Boosting" es otra de las técnicas clásicas utilizadas para el aprendizaje automático para diversos problemas de regresión y clasificación. Este modelo produce un conjunto de modelos de predicción más débil, típicamente árboles de decisión y los refuerza construyéndolos por etapas, y generalizándolos, permitiendo optimizaciones con funciones de pérdidas diferenciales arbitrarías [42].

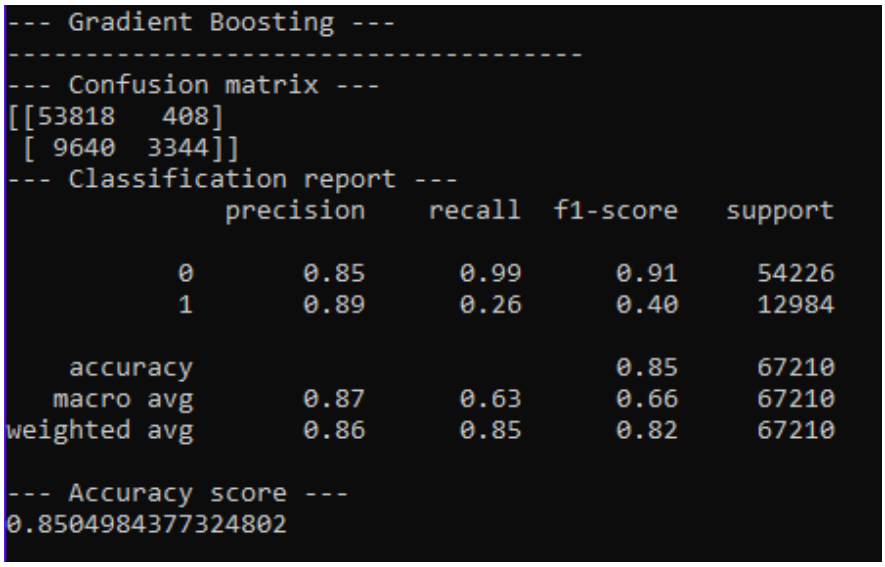

Ilustración 5.28: Resultados tras procesar los datos mediante Gradient Boosting

Obtenemos un 85 % de precisión mejorando lógicamente los árboles de decisión pero sin superar ese 89 % del Random Forest.

Naive Bayes

Los clasificadores Bayes simples (*Naive Bayes*) son conocidos especialmente en estadística como uno de los clasificadores probabilísticos más sencillos basados en la aplicación del teorema de Bayes. Estos clasificadores son altamente escalables pero requieren de un número de parámetros lineal al número de variables y de un tiempo lineal en lugar de las costosas aproximaciones iterativas utilizadas por otros clasificadores [43].

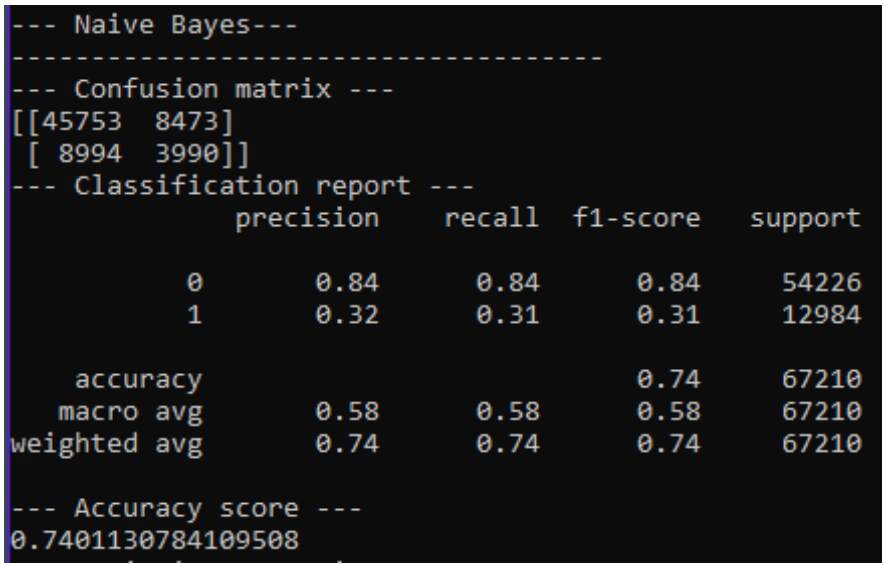

Ilustración 5.29: Resultados tras procesar los datos mediante Naive Bayes

Como podemos observar, con un 74 % , Naive Bayes es un algoritmo demasiado simple como para mejorar nuestros resultados obtenidos con otros clasificadores.

Logistic Regression

La regresión logística es un modelo estadístico que utiliza una función logística para modelar una variable dependiente binaria, aunque existen extensiones más complejas. Una de las grandes ventajas que posee esta metodología es que cuando se tiene un problema lineal complicado y no hay muchos datos aún puede producir buenas predicciones [44].

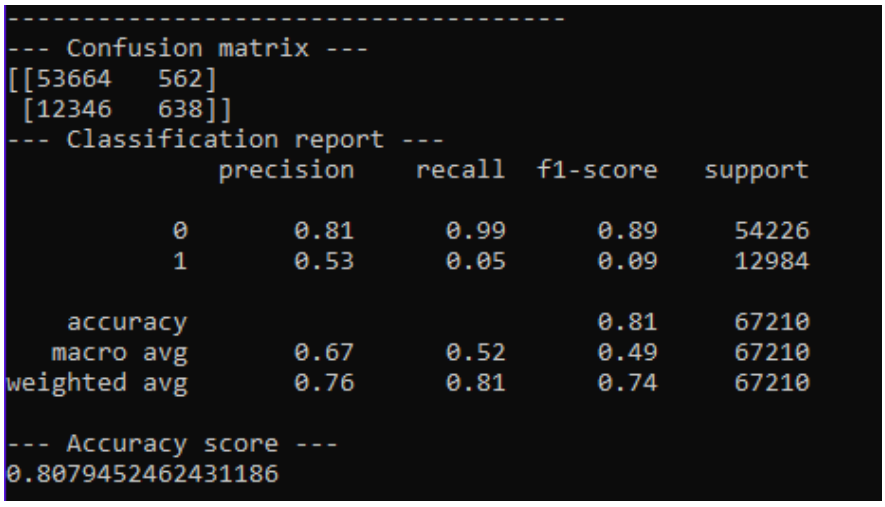

Ilustración 5.30: Resultados tras procesar los datos mediante Logistic Regression

Utilizando esta metodología conseguimos un 80 % de precisión, mejorando el clasificador de Bayes pero no a los demás clasificadores.

Conclusiones iniciales

Tras probar estos algoritmos con el conjunto de datos en bruto vemos resultados muy prometedores para lo que debería ser esta prueba. Mis tutores ya me habían comentado que, por su experiencia, sabían que los valores podían mejorar contundentemente al agrupar los datos por trayectorias, sin embargo consideré realizar estas primeras pruebas de esta manera para ver la viabilidad del proyecto.

Random Forest ha dado los mejores resultados en esta primera tanda con un 89 % y le sigue de cerca Gradient Boosting con un 85 %. El siguiente paso a seguir no es otro que preparar los datos tal y como mis tutores me recomendaban y comprobar las diferencias obtenidas.

5.2.1. Entrenamiento de algoritmos clásicos mediante trayectorias

Para esta segunda prueba agruparemos todos los datos por trayectorias, como habíamos dicho antes tenemos 756 trayectorias de ID's. Con lo cual nuestro dataframe será un histograma de los datos de cara ID, como se muestra en la siguiente imagen.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 756 entries, 0 to 755
Data columns (total 33 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Video      756 non-null    object
 1   Frame      756 non-null    int64
 2   ID         756 non-null    int64
 3   0          756 non-null    int64
 4   1          756 non-null    int64
 5   2          756 non-null    int64
 6   3          756 non-null    int64
 7   4          756 non-null    int64
 8   5          756 non-null    int64
 9   6          756 non-null    int64
 10  7          756 non-null    int64
 11  8          756 non-null    int64
 12  9          756 non-null    int64
 13  h0         756 non-null    int64
 14  h1         756 non-null    int64
 15  h2         756 non-null    int64
 16  h3         756 non-null    int64
 17  h4         756 non-null    int64
 18  h5         756 non-null    int64
 19  h6         756 non-null    int64
 20  h7         756 non-null    int64
 21  h8         756 non-null    int64
 22  h9         756 non-null    int64
 23  h10        756 non-null    int64
 24  North      756 non-null    int64
 25  East       756 non-null    int64
 26  West       756 non-null    int64
 27  South      756 non-null    int64
 28  NorthEast  756 non-null    int64
 29  NorthWest  756 non-null    int64
 30  SouthEast  756 non-null    int64
 31  SouthWest  756 non-null    int64
 32  Type       756 non-null    object
dtypes: int64(31), object(2)
memory usage: 200.8+ KB
```

Ilustración 5.31: Columnas del dataframe con las trayectorias agrupadas

Para este dataframe únicamente utilizaremos la velocidad promedio, como explicamos anteriormente, esta está discretizada de 0-10, siendo el 10 una cifra de error, debido a su alto valor solo puede ser generada mediante saltos producidos por YOLOv4. Con lo cual insertamos una columna por cada nivel de velocidad para realizar el histograma. Hacemos lo mismo para la altura y volvemos a aplicar la función “get dummies” en la orientación como en el caso anterior.

	Video	Frame	ID	0	1	2	...	South	NorthEast	NorthWest	SouthEast	SouthWest	Type
0	lpatrail-clip01	254	1	1	5	12	...	8	11	12	10	15	Runner
1	lpatrail-clip01	383	7	1	5	12	...	101	16	13	56	41	Runner
2	lpatrail-clip01	715	10	6	14	36	...	96	35	42	79	54	Runner
3	lpatrail-clip01	175	11	1	0	5	...	3	20	20	5	1	Runner
4	lpatrail-clip01	233	14	1	0	0	...	84	7	7	49	31	Runner
5	lpatrail-clip01	2	16	1	0	0	...	0	0	1	0	0	Runner
6	lpatrail-clip01	391	17	1	0	5	...	104	17	7	74	42	Runner
7	lpatrail-clip01	404	18	2	8	26	...	20	36	36	15	14	Runner
8	lpatrail-clip01	15	19	1	0	1	...	5	0	1	1	0	Runner
9	lpatrail-clip01	118	20	1	2	3	...	11	12	11	4	1	Runner
10	lpatrail-clip01	461	22	1	4	4	...	116	17	17	81	59	Runner
11	lpatrail-clip01	961	24	2	12	23	...	218	46	38	151	90	Runner
12	lpatrail-clip01	51	25	1	0	0	...	12	1	3	8	9	Runner
13	lpatrail-clip01	157	26	1	7	3	...	60	3	6	32	11	Runner
14	lpatrail-clip01	121	33	3	8	16	...	4	8	4	1	6	Runner
15	lpatrail-clip01	108	34	2	5	4	...	2	4	6	5	3	Runner
16	lpatrail-clip01	19	36	3	2	1	...	0	0	1	0	0	Runner
17	tgc-arucas-clip01	431	1	1	1	3	...	94	32	24	49	61	Public
18	tgc-arucas-clip01	157	2	1	0	1	...	25	10	13	13	10	Public
19	tgc-arucas-clip01	512	3	1	2	4	...	66	44	38	48	37	Public
20	tgc-arucas-clip01	408	4	3	8	38	...	32	24	18	24	22	Public

Ilustración 5.32: Datos del dataframe con las trayectorias agrupadas

Finalmente nos queda un dataframe con más columnas que el anterior pero con muchísimas menos filas. Esta agrupación es clave para identificar las características de un corredor, es decir, si nos fijamos en la figura 5.32 podemos ver que el ID 1 de LPAtrail ha tenido una velocidad de 0 una sola vez, una velocidad de 1 cinco veces y una velocidad de 2 doce veces. Estos datos nos indican las propiedades de su movimiento, ha estado más en movimiento que quieto en los fotogramas que ha sido grabado y esa podría ser una característica identificativa para un corredor. Estas agrupaciones permitirán a los algoritmos ver estos patrones.

Algo que nos preocupó al realizar esta agrupación son aquellos ID's con pocos fotogramas, ya que podrían confundir a los algoritmos y distorsionar el resultado. Con lo cual añadimos un umbral para el número de fotogramas, todo aquel ID con menos de 100 fotogramas, recordemos que nuestros vídeos están a 50 fotogramas por segundo por lo que hablamos de 2 segundos, son eliminados del dataframe.

Tras realizar este corte en los datos pasamos de tener 756 filas x 31 columnas a 385 filas x 31 columnas. Hablamos de 31 columnas y no 33 como en la imagen porque al igual que en la primera prueba eliminamos la columna vídeo y la columna tipo será nuestra etiqueta, sin embargo esta vez la columna Fotograma(Frame) determina el número de fotogramas que posee dicho ID, aportando información extra sobre el estado de dicha persona, con lo cual mantenemos dicha variable.

Este será nuestro dataframe tras las modificaciones realizadas:

```
Int64Index: 385 entries, 0 to 755
Data columns (total 31 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Frame      385 non-null    int64
 1   ID         385 non-null    int64
 2   0          385 non-null    int64
 3   1          385 non-null    int64
 4   2          385 non-null    int64
 5   3          385 non-null    int64
 6   4          385 non-null    int64
 7   5          385 non-null    int64
 8   6          385 non-null    int64
 9   7          385 non-null    int64
 10  8          385 non-null    int64
 11  9          385 non-null    int64
 12  h0         385 non-null    int64
 13  h1         385 non-null    int64
 14  h2         385 non-null    int64
 15  h3         385 non-null    int64
 16  h4         385 non-null    int64
 17  h5         385 non-null    int64
 18  h6         385 non-null    int64
 19  h7         385 non-null    int64
 20  h8         385 non-null    int64
 21  h9         385 non-null    int64
 22  h10        385 non-null    int64
 23  North      385 non-null    int64
 24  East       385 non-null    int64
 25  West       385 non-null    int64
 26  South      385 non-null    int64
 27  NorthEast  385 non-null    int64
 28  NorthWest  385 non-null    int64
 29  SouthEast  385 non-null    int64
 30  SouthWest  385 non-null    int64
dtypes: int64(31)
```

Ilustración 5.33: Dataframe final

Ahora que tenemos los datos preparados solo nos queda contabilizarlos y dividirlos en nuestros sets de entrenamiento y testeo. De 385 trayectorias, 301 son de público y 84 son de corredores. Dividimos los datos en un 30 % como en la prueba anterior de manera que nos quedan 269 trayectorias de entrenamiento y 116 de test. De esas 269 trayectorias, 209 son público y 60 son corredores, y de las 116 trayectorias, 92 son público y 24 son corredores. Ahora que ya tenemos claro nuestros datos es hora de ver los resultados.

Pasamos este nuevo dataframe por todos los algoritmos comentados en la sección anterior, y obtenemos los siguientes resultados en precisión:

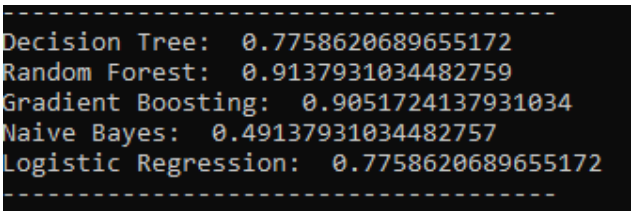

Ilustración 5.34: Resultados del entrenamiento mediante trayectorias en los algoritmos clásicos

Como podemos ver todos los resultados difieren de nuestra primera prueba, los algoritmos más básicos como Decision Tree, Naive Bayes y Logistic Regression dan resultados peores. Sin embargo los dos algoritmos que mejor daban, Random Forest y Gradient Boosting, mejoran aún más. Pasan de un 89 % y 85 % a un 91 % y 90 % respectivamente.

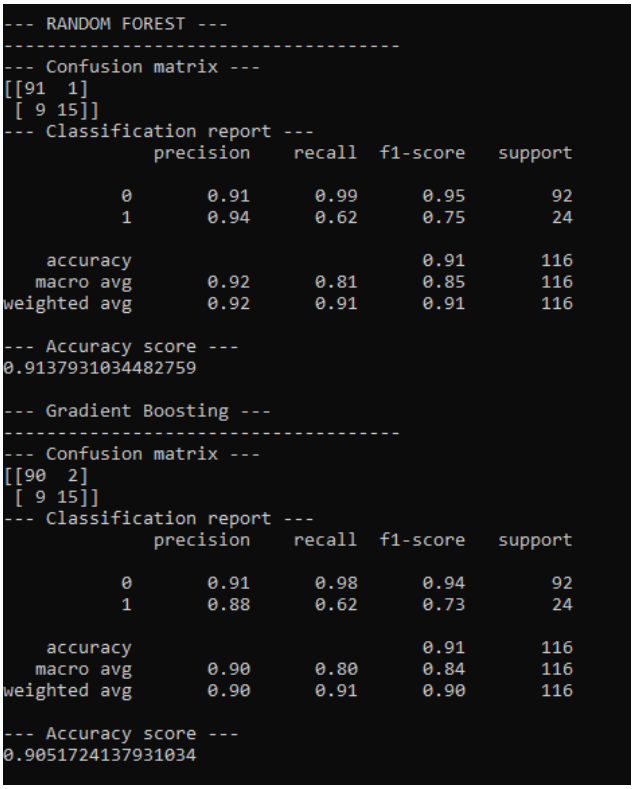

Ilustración 5.35: Resultados del entrenamiento mediante trayectorias en Random Forest y Gradient Boosting

5.2.2. Conclusiones obtenidas de los algoritmos clásicos

Tras nuestras múltiples pruebas con estos algoritmos hemos comprobado como Random Forest y Gradient Boosting se adaptan mejor a nuestros datos y a su vez la importancia de agrupar estos mediante trayectorias. Finalmente nuestro mejor resultado ha sido 91 % en

Random Forest, un muy buen resultado, sin duda superior a lo esperado en estas primeras pruebas lo que nos anima a probar con modelos más complejos como por ejemplo las LSTM (*Long short-term memory*), de las cuales hablaremos a continuación.

5.3. Pruebas mediante arquitectura LSTM

LSTM, *Long short-term memory* es una arquitectura de red neuronal recurrente artificial utilizada para el aprendizaje automático. Se caracterizan por tener conexiones de retroalimentación, siendo capaces de procesar secuencias enteras de datos para aplicaciones como reconocimiento del habla, de la escritura etc.

Una LSTM común está compuesta por una celda, una puerta de entrada, una puerta de salida y una puerta de olvido o *forget gate*. La celda recuerda los valores en intervalos de tiempo arbitrarios mientras las tres puertas regulan el flujo de información. Estas redes son muy útiles para procesar, clasificar y realizar predicciones mediante series de datos temporales, ya que controlan los desfases de duración desconocida entre los eventos importantes de una serie temporal [45].

Para poder utilizar esta red debemos modificar la estructura de nuestros datos y adaptarla. Como primer paso, escalamos los datos entre 0 y 1 con la función MinMaxScaler de Sklearn además para esta prueba solo utilizaremos aquellas personas con al menos 50 fotogramas, es decir, un 1 segundo en nuestros vídeos.

Nuestro objetivo es formar un *array* tridimensional con agrupaciones de 400 fotogramas por fila (xfilas, 400, 11 columnas), lo que se traduce a 8 segundos en nuestros vídeos grabados a 50 fotogramas por segundo.

Cada fila contendrá los primeros 400 fotogramas del ID con sus datos agrupados en las siguientes 11 columnas.

```
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Speed          136 non-null    int64
 1   Height         136 non-null    int64
 2   SpeedAverange  136 non-null    int64
 3   East           136 non-null    uint8
 4   North          136 non-null    uint8
 5   NorthEast      136 non-null    uint8
 6   NorthWest      136 non-null    uint8
 7   South          136 non-null    uint8
 8   SouthEast      136 non-null    uint8
 9   SouthWest      136 non-null    uint8
 10  West           136 non-null    uint8
```

Ilustración 5.36: Dataframe para cada fila del *array* tridimensional

Como podemos ver, hemos vuelto a utilizar la función “dummies” para la orientación y

hemos eliminado las columnas Vídeo, Frame y Tipo que será nuestra variable “Y”. Aparte de estas agrupaciones de datos mis tutores me recomendaron hacer una extensión de los datos la cual explicaré paso a paso.

El proceso realizado es el siguiente, accedemos al primer ID del primer vídeo.

Menos de 50 fotogramas:

-Se descarta el ID

Más de 50 fotogramas:

-Si posee menos de 400 fotogramas de datos añadimos ceros hasta llegar a 400.

-Sí posee más de 400 fotogramas lo dividimos en las inserciones necesarias y añadimos ceros en la última inserción si es necesario para llegar a 400.

-A la hora de insertar realizamos una extensión de los datos, insertando los datos múltiples veces de la siguiente manera. Si un ID posee 790 fotogramas, añadimos 10 datos conformados por ceros e insertamos los primeros 400 fotogramas. A continuación en lugar de insertar los siguientes 400 insertamos los datos en intervalos de 50(50-450, 100-500, 150-550, 200-600, 250-650, 300-700, 350-750, 400-800). De esta manera conseguimos extender los datos en gran medida, en lugar de obtener un *array* (2,400,11) obtenemos uno mucho mayor (9,400,11).

La primera pregunta a realizar es, ¿cómo afecta la inserción de ceros?, ya que estamos añadiendo datos “falsos”. Si lo pensamos detenidamente estamos añadiendo público, ya que las velocidades insertadas son cero. Con lo cual lo importante es intentar mantener un balance entre público y corredores, a su vez mediante la extensión de los datos con inserciones en intervalos de 50 estamos modificando ese balance.

Nuestro *array* final contiene 3218 filas con 400 fotogramas cada fila y 11 columnas cada fotograma, lo que se traduce a 1.287.200 fotogramas de los 224033 que teníamos antes de extender.

```
TRAINX:  (3218, 400, 11)
TRAINY:  (3218. 1)
Train:  (2252, 400, 11) (2252, 1)
Test: (966, 400, 11) (966, 1)
```

Ilustración 5.37: División de los datos

	Trayectorias(Personas)		
	Publico	Corredores	Total
Entrenamiento	1801	451	2252
Pruebas	796	170	966
Datos	2597	621	3218

Ilustración 5.38: Cantidad de público y corredores para cada apartado

En esta ilustración podemos ver las distribuciones realizadas en los datos mediante la función split de sklearn. Se ha utilizado un 70 % de los datos para entrenar y el 30 % restante para los test. En nuestros datos de 3218 trayectorias, 621 son de corredores, casi un 24 % de los datos. Al dividir entre los conjuntos de entrenamiento y testeo tenemos 2252 trayectorias para entrenar, de esas, 451 son trayectorias de corredores. Por último para los test tenemos 966 trayectorias de las cuales 170 son corredores.

A priori sin realizar pruebas ni entrenamientos no sabemos si poseemos un balance suficiente como para obtener buenos resultados, con lo cual comenzamos la tanda de entrenamientos.

Comenzamos con una arquitectura simple, una sola LSTM con 32 neuronas junto a una capa densa de otras 32 neuronas y dos dropout de 0.5, por último una capa final densa de salida con la función sigmoide. Como optimizador utilizamos "Adam" y para la pérdida utilizamos "*binary crossentropy*".

```
public = (y == 0).sum()
runners = (y == 1).sum()
print("Trajectories of public: ", public)
print("Trajectories of runners: ", runners)
x_train , x_test, y_train, y_test = train_test_split(x, y,test_size=0.30, random_state=42)

publicTrain = (y_train == 0).sum()
runnersTrain = (y_train == 1).sum()
publicTest = (y_test == 0).sum()
runnersTest = (y_test == 1).sum()
print("Trajectories of public in training: ", publicTrain)
print("Trajectories of runners in training: ", runnersTrain)
print("Trajectories of public in testing: ", publicTest)
print("Trajectories of runners in testing: ", runnersTest)
print("Train: ",x_train.shape, y_train.shape)
print("Test:",x_test.shape, y_test.shape)
model = Sequential()

model.add(LSTM(32, input_shape=(x_train.shape[1],x_train.shape[2]), return_sequences=False))
model.add(Dropout(0.5))
#model.add(LSTM(32,return_sequences=False))
 #model.add(Dropout(0.5))
model.add(Dense(32))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

name = folder

checkpoint = ModelCheckpoint("models/"+name+"/"+name+"MaxValAccuracy.h5", monitor='val_accuracy', verbose=0, save_best_only=True, mode='max')

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=int(epoch), batch_size = 32, validation_data=(x_test,y_test), callbacks=[checkpoint])
```

Ilustración 5.39: Primer modelo realizado

Aunque en la imagen aparezcan más capas nótese que están comentadas, esto es debido a que ya había teorizado con arquitecturas más complejas.

Los datos obtenidos son los siguientes:

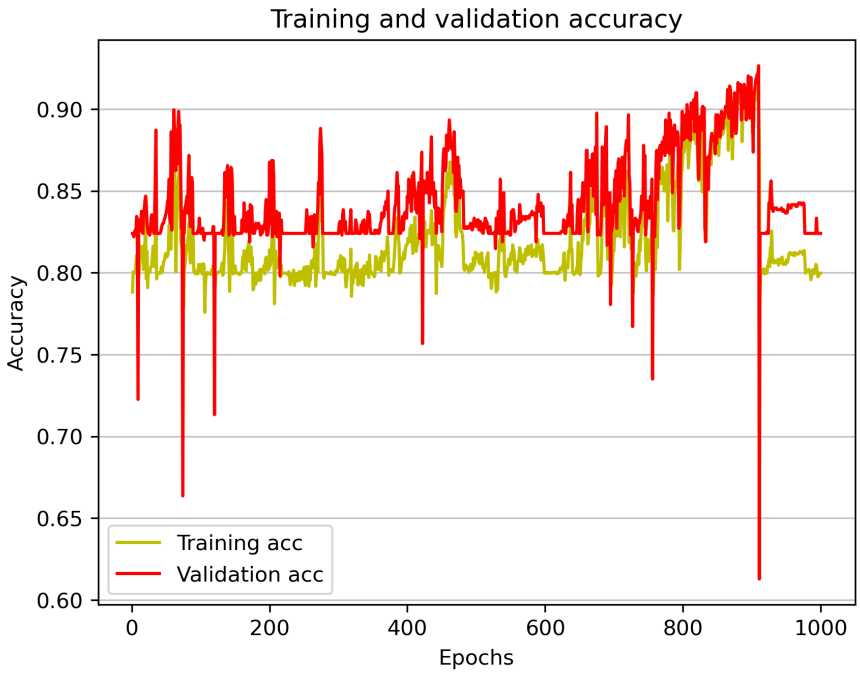

Ilustración 5.40: Resultados de precisión obtenidos mediante una LSTM de 32 neuronas

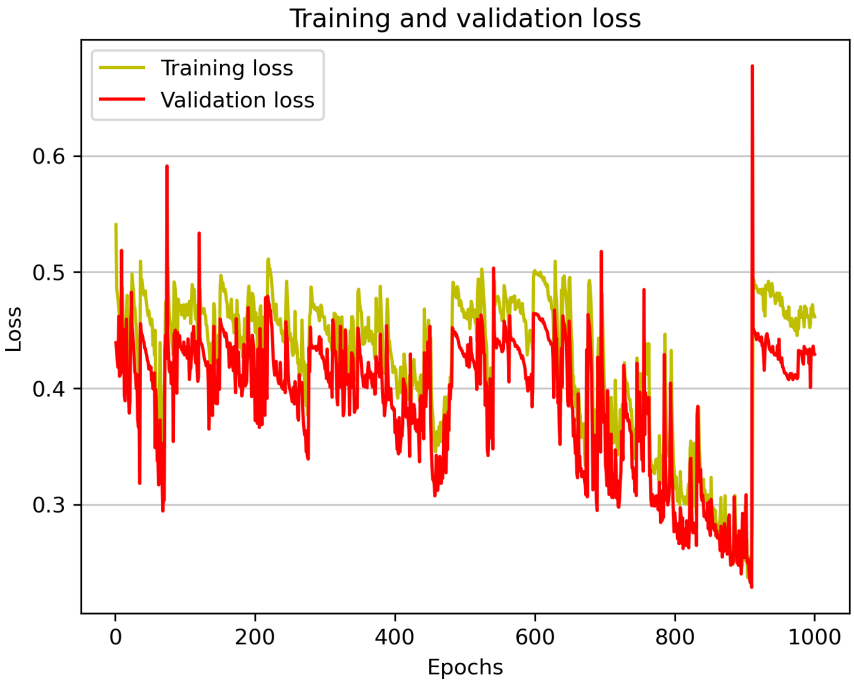

Ilustración 5.41: Resultados de pérdidas obtenidos mediante una LSTM de 32 neuronas

Como podemos ver en las gráficas realizadas mediante la librería Matplotlib, hemos realizado mil épocas con nuestros datos. De estas mil épocas hemos extraído aquella con mejor precisión mediante la utilización de la función checkpoint de Tensorflow.

En las gráficas podemos ver como el conjunto de validación (en rojo) se ajusta debidamente a los resultados de entrenamiento (en verde), con múltiples saltos y variaciones probablemente debido a la gran diferencia que hay en nuestros datos.

El mejor resultado obtenido con este modelo es de una pérdida en validación de 0.2288 y una precisión en validación de 92 %, mejorando ligeramente nuestro mejor resultado en la utilización de los algoritmos clásicos.

Tras estos datos determinamos que un modelo más complejo puede dar mayores resultados, por lo cual utilizamos una doble LSTM he incrementamos el número de capas y neuronas.

```
public = (y == 0).sum()
runners = (y == 1).sum()
print("Trajectories of public: ", public)
print("Trajectories of runners: ", runners)
x_train , x_test, y_train, y_test = train_test_split(x, y,test_size=0.30, random_state=42)

publicTrain = (y_train == 0).sum()
runnersTrain = (y_train == 1).sum()
publicTest = (y_test == 0).sum()
runnersTest = (y_test == 1).sum()
print("Trajectories of public in training: ", publicTrain)
print("Trajectories of runners in training: ", runnersTrain)
print("Trajectories of public in testing: ", publicTest)
print("Trajectories of runners in testing: ", runnersTest)
print("Train: ",x_train.shape, y_train.shape)
print("Test:",x_test.shape, y_test.shape)
model = Sequential()

model.add(LSTM(128, input_shape=(x_train.shape[1],x_train.shape[2]), return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(64,return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(32))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

name = folder

checkpoint = ModelCheckpoint("models/"+name+"/"+name+"MaxValAccuracy.h5", monitor='val_accuracy', verbose=0, save_best_only=True, mode='max')

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=int(epoch), batch_size = 32, validation_data=(x_test,y_test), callbacks=[checkpoint])


_, acc = model.evaluate(x_test, y_test)
```

Ilustración 5.42: Segundo modelo realizado

En este modelo utilizamos una primera LSTM de 128 neuronas seguido de un DropOut de 0.5 para evitar un posible sobreentrenamiento. Una segunda LSTM de 64 neuronas con su correspondiente DropOut y dos capas densas, una de 64 neuronas y la última con la función de salida sigmoide. No modificamos ningún otro aspecto del modelo para poder identificar las variaciones en los resultados fácilmente.

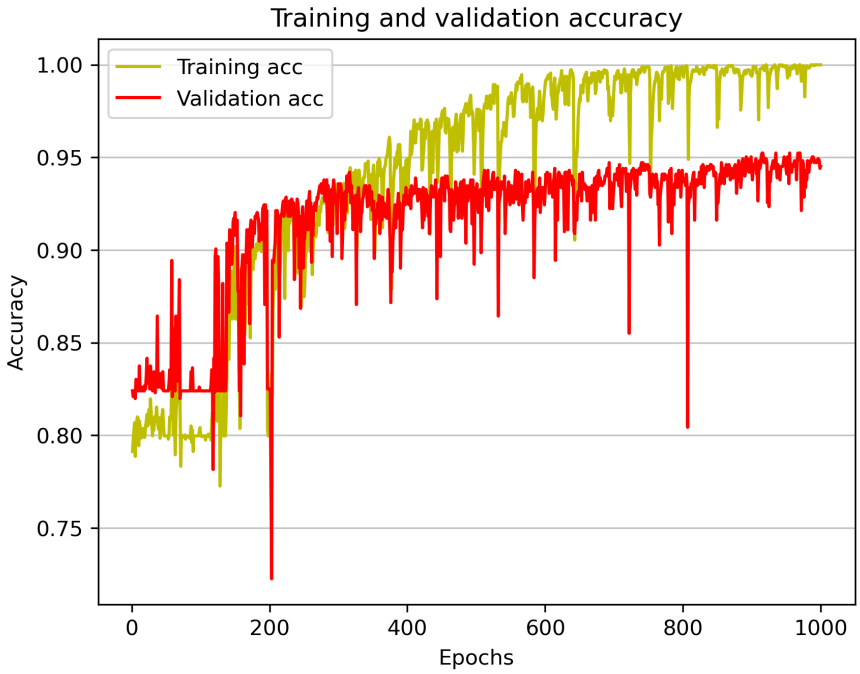

Ilustración 5.43: Resultados de precisión obtenidos mediante una doble LSTM de 128-64

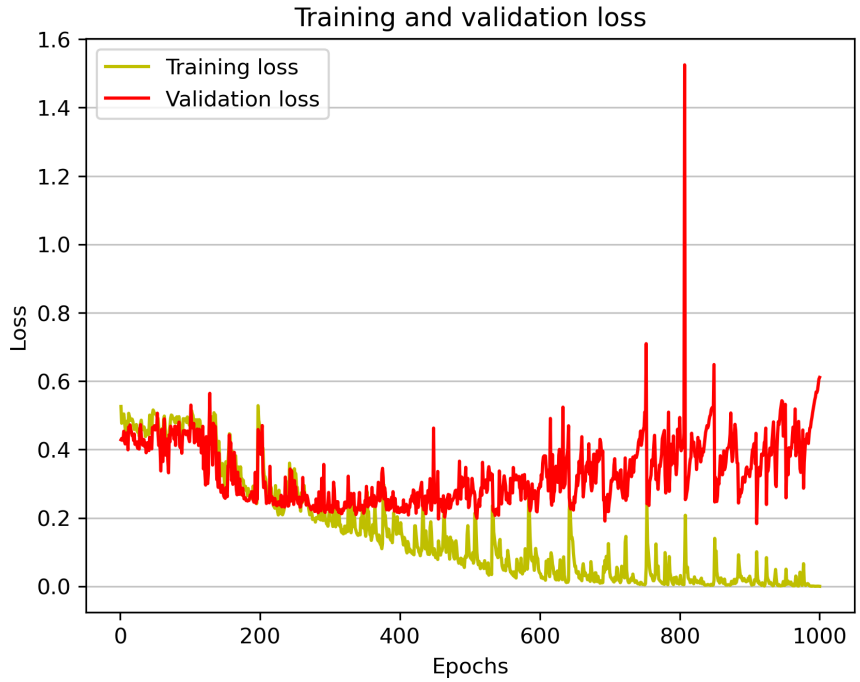

Ilustración 5.44: Resultados de pérdidas obtenidos mediante una doble LSTM de 128-64

Una vez más extraemos el mejor resultado de precisión en validación mediante nuestro checkpoint y obtenemos unos resultados de una pérdida en validación de 0.3235 y una

precisión en validación de un 95 %.

A su vez se puede observar en los gráficos como la validación y el entrenamiento comienza a separarse, muy poco como para que indique algún problema pero algo a denotar.

Tras estos resultados tan espléndidos realizamos múltiples pruebas modificando el número de neuronas, modificando la función de activación con una Relu en lugar de sigmoid, decrementando las funciones de DroptOut, incrementando el número de épocas etc. En esta memoria no introduciremos todas estas pruebas para no alargarla indebidamente y solo destacamos las más representativas.

Tras muchas pruebas nuestro mejor resultado se obtuvo mediante el siguiente modelo.

```
public = (y == 0).sum()
runners = (y == 1).sum()
print("Trajectories of public: ", public)
print("Trajectories of runners: ", runners)
x_train , x_test, y_train, y_test = train_test_split(x, y,test_size=0.30, random_state=42)

publicTrain = (y_train == 0).sum()
runnersTrain = (y_train == 1).sum()
publicTest = (y_test == 0).sum()
runnersTest = (y_test == 1).sum()
print("Trajectories of public in training: ", publicTrain)
print("Trajectories of runners in training: ", runnersTrain)
print("Trajectories of public in testing: ", publicTest)
print("Trajectories of runners in testing: ", runnersTest)
print("Train: ",x_train.shape, y_train.shape)
print("Test:",x_test.shape, y_test.shape)
model = Sequential()

model.add(LSTM(64, input_shape=(x_train.shape[1],x_train.shape[2]), return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(32,return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(32))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

name = folder

checkpoint = ModelCheckpoint("models/"+name+"/"+name+"MaxValAccuracy.h5", monitor='val_accuracy', verbose=0, save_best_only=True, mode='max')

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=int(epoch), batch_size = 32, validation_data=(x_test,y_test), callbacks=[checkpoint])
```

Ilustración 5.45: Mejor modelo realizado

Nuestro mejor modelo está compuesto de una LSTM de 64 neuronas, tres DropOut de 0.5, otra LSTM de 32 neuronas, una capa densa de 32 neuronas y por último una capa densa de salida de una neurona con activación sigmoide.

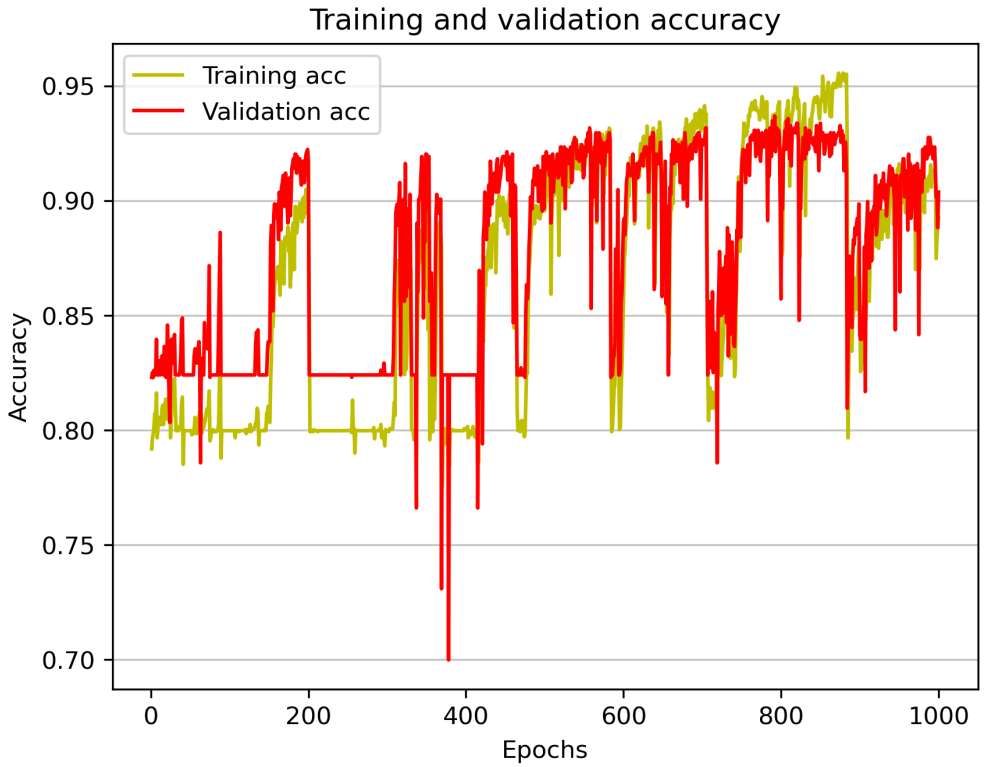

Ilustración 5.46: Resultados de precisión obtenidos mediante una doble LSTM de 64-32

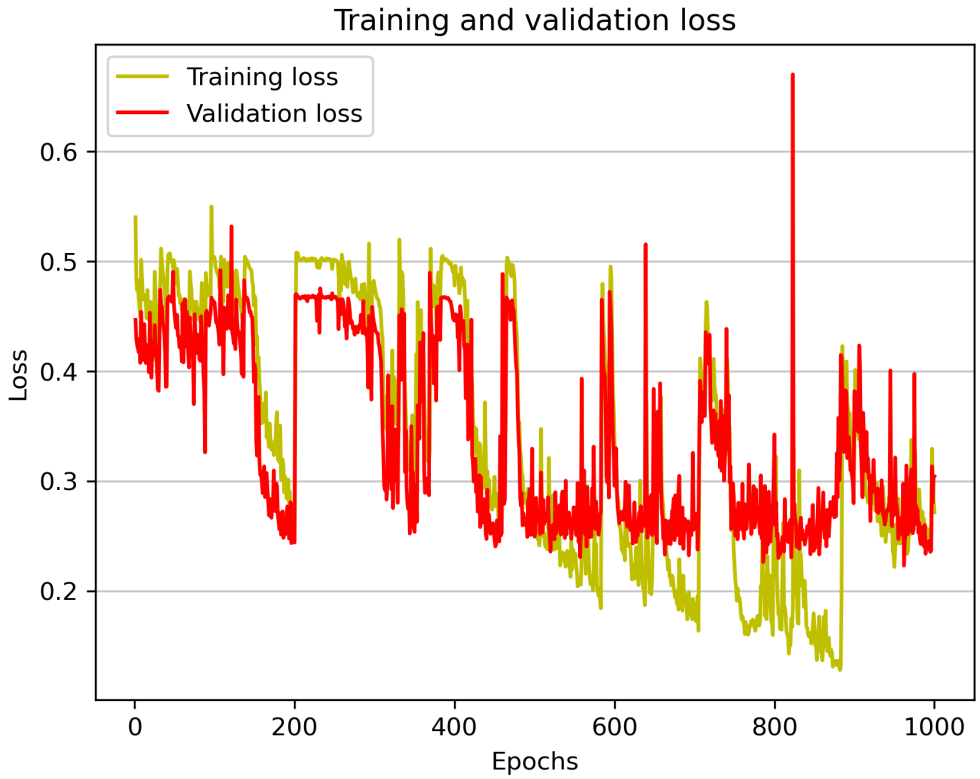

Ilustración 5.47: Resultados de pérdidas obtenidos mediante una doble LSTM de 64-32

Como podemos observar en las gráficas este modelo nos presentó los mejores resultados, una pérdida muy inferior a nuestro último modelo, pérdida en validación de 0.1542 y una precisión en validación de 98.03 %

Además, tanto la validación como el entrenamiento se mantienen por encima de un 90 % de precisión.

Capítulo 6

Conclusiones y trabajo futuro

Comenzamos este proyecto analizando los problemas dados en pruebas deportivas, viendo como los métodos de detección y seguimiento de corredores de hoy en día solventan los problemas en la medida de lo posible pero sin resultados perfectos ni mucho menos.

Tras esto vimos algunos de los avances realizados en el campo del aprendizaje automático y sus metodologías. Estudiamos el uso de diferentes detectores y terminamos utilizando YOLOv4 con una serie de modificaciones.

YOLOv4 a pesar de los errores de detección que puede presentar, acabó funcionando de manera prometedora, nos generó una serie de archivos JSON con información sobre cada persona detectada en cada fotograma. Con estos datos y tras muchas pruebas, calculamos los descriptores a utilizar en los algoritmos de aprendizaje automático.

Una vez obtenidos los primeros descriptores alimentamos a algoritmos clásicos del dominio, y analizamos los primeros resultados.

	Precisión	
Algoritmo	Sin trayectorias	Con trayectorias
Decision tree	83%	77%
Random forest	89%	91%
Gradient boosting	85%	90%
Naive bayes	74%	49%
Logistic regression	80%	77%

Ilustración 6.1: Resultados obtenidos utilizando algoritmos clásicos

Gradient Boosting y Random Forest destacaron notablemente en ambas pruebas, siendo los algoritmos con mejores resultados y a la vez los más complejos de la lista. Comenzamos con Decision Tree, un algoritmo muy simple compuesto de una serie de pasos secuenciales diseñado para dar respuesta a una pregunta basándose en probabilidades. Por otro lado

Random Forest utiliza una colección de estos árboles de manera individual para producir una misma respuesta, un único árbol de decisión es una predicción muy débil pero muy rápida de computar, sin embargo utilizar más árboles te ofrece un modelo más robusto y evita problemas como el Overfitting. En nuestro caso, el tiempo de ejecución no ha sido un problema debido a la sencillez de nuestros datos, con lo cual Random Forest ha dado unos resultados muy superiores a Desicion Tree.

En cuanto a Gradient boosting, al igual que Random Forest, es una composición de árboles de decisión pero diferenciados por su manera de construir el modelo. Random Forest construye cada árbol de manera independiente mientras que Gradient boosting construye cada árbol a la vez, además Random forest combina los resultados al final del proceso mientras que Gradient boosting va combinando cada resultado mientras los va obteniendo [46].

Según los parámetros utilizados Gradient boosting suele dar un mayor rendimiento a no ser que en los datos tengamos mucho ruido, lo cual resultaría en un mayor *overfitting*. Estos además suelen ser más difíciles de afinar que Random forest, sin embargo nuestros resultados son muy similares en ambos algoritmos y bastante prometedores.

Quizás ajustando una serie de parámetros podríamos mejorar el resultado obtenido con Gradient boosting, pero decidimos dar un paso adelante.

Tras analizar y comparar los resultados obtenidos, vemos como los algoritmos clásicos de mayor complejidad mejoraban al agrupar los datos en trayectorias y hacer un histograma de sus descriptores, con lo cual utilizamos esta aproximación para algoritmos más complejos.

Pasamos a las long-short term memory o LSTM y construimos múltiples modelos, desde los más simples a algunas variaciones más complejas y extraemos las mejores épocas de cada modelo.

Capas	Precisión en validación	Pérdida en validación
LSTM-32-Dense-32-Sigmoid	92.65%	0.2288
LSTM-64-LSTM-32-Dense-32-Sigmoid	98.03%	0.1542
LSTM-64-LSTM-64-Dense-64-Sigmoid	95.55%	0.2724
LSTM-128-LSTM-64-Dense-32-Sigmoid	95.24%	0.3235

Ilustración 6.2: Resultados obtenidos utilizando diferentes modelos de LSTM y extrayendo la mejor época.

Comenzamos con modelos con pocas neuronas y obtuvimos resultados parecidos a los de los algoritmos clásicos, incrementamos la cantidad de neuronas hasta que los resultados comenzaron a variar significativamente y a reducir su precisión. Realizamos este proceso hasta encontrar un terminó medio donde nuestros resultados eran superiores, ya no solo por obtener la mejor época en cuanto a precisión y pérdida en validación sino por mantenerse de manera constante sobre el 90 % de precisión en la mayoría de épocas.

Finalmente obtenemos resultados muy prometedores, teniendo en cuenta que estamos

utilizando descriptores diseñados y escogidos por el usuario. Por supuesto aun que da mucho por mejorar y experimentar pero hablamos de un TFG con un tiempo limitado y con el objetivo de adquirir esta serie de resultados iniciales.

Como trabajo futuro nos podemos aventurar en múltiples opciones, por ejemplo en la distribución utilizada para entrenar las LSTM, dividimos las trayectorias de todos las personas en pistas de 400 fotogramas, y la LSTM debe predecir cada pista. Como capa extra podríamos controlar esas predicciones a un mismo ID, y como si de una votación se tratara, elegir la predicción que más se repita.

De esta manera, aunque la red falle en algunas predicciones es menos probable que se equivoque en la decisión final. Además, al dividir las trayectorias de las personas en 400 fotogramas se pueden dar ciertos errores como la siguiente situación. Imaginemos que un corredor está pasando por el camino donde estamos grabando y este se para 10 segundos a coger aire porque está agotado. 10 segundos en nuestros vídeos son 500 fotogramas en los que sus descriptores de velocidad y velocidad promedio están completamente a 0 y su altura también se mantendrá sin variaciones aparentes (Un corredor pasa desde el fondo de la imagen hasta pasar la cámara, con lo cual su altura varía a lo largo de la trayectoria).

Probablemente la red catalogara esa pista de "público" y el resto, si está bien entrenada, de corredor. Al realizar esta capa extra de selección por mayoría evitamos este tipo de errores. El ejemplo puede parecer muy concreto pero esto también se estipula a los fallos de identificación que posee YOLOv4, pasa múltiples veces que identifica a una persona con el ID 1 y esa persona es corredor. Posteriormente, cuando esa persona ya ha dejado la imagen, el detector se confunde y etiqueta a otra persona con el ID 1. Con lo cual tenemos pistas con datos de un corredor y pistas con datos de público en una misma trayectoria.

Existen múltiples soluciones que podemos intentar para refinar los datos obtenidos de YOLOv4, una de las ideas que estuvo presente era utilización de re-identificación para determinar si varias personas habían sido etiquetadas con un mismo ID.

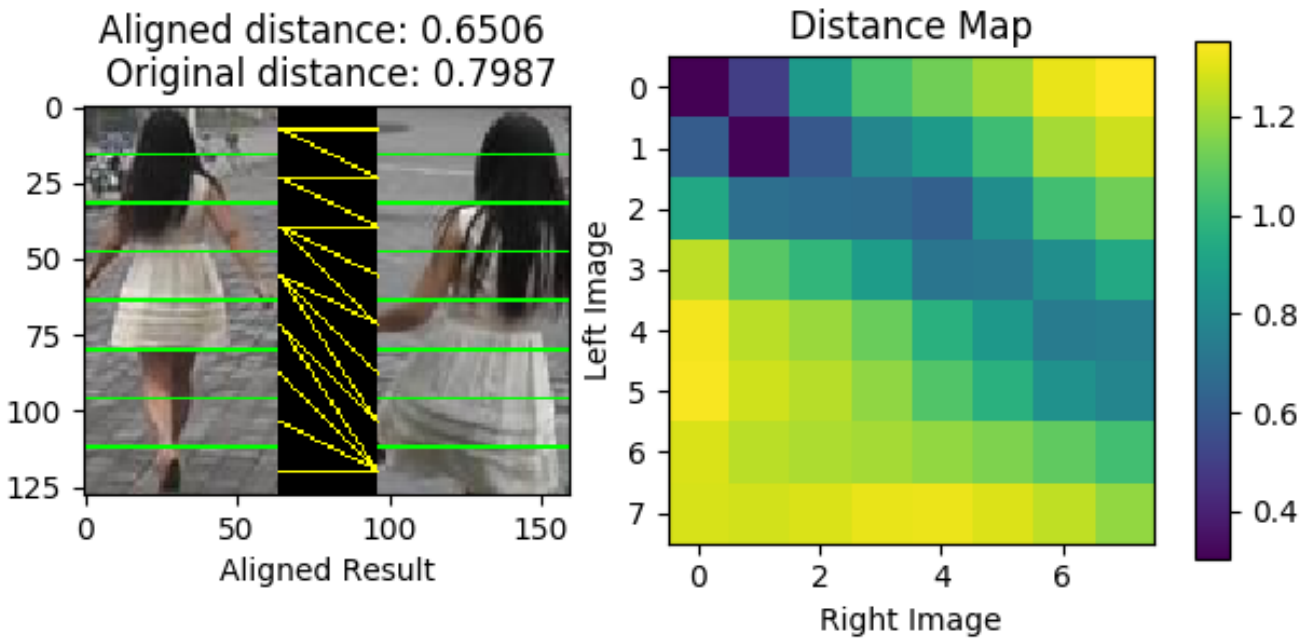

Ilustración 6.3: Ejemplo de utilización de AlignedReID para determinar si dos fotogramas contienen a la misma persona [12].

Por ejemplo la imagen mostrada es una re-identificación de dos fotogramas tratando de determinar si la chica es la misma persona, utilizando AlignedReID [47]. De hecho mis tutores poseen múltiples alumnos con trabajos de fin de grado y trabajos de fin de máster probando todo tipo de algoritmos en este campo, con lo cual fue una idea presente pero el tiempo siempre fue un límite.

Para utilizar este tipo de software requiere realizar múltiples pruebas sobre las imágenes, extraer capturas y analizar el tipo de detección que necesitamos, además todos estos algoritmos son de nueva índole y su eficacia no está asegurada ni mucho menos, de hecho no paran de actualizarse y sacar nuevas versiones. No obstante es algo a tener en cuenta.

Otra posible mejora a realizar es modificar los descriptores diseñados y substituirlos por algún tipo de descriptor automático generado por redes profundas, por ejemplo de detección de acciones.

En este campo hay múltiples algoritmos que determinan las acciones y movimientos de las personas, podríamos realizar todos los descriptores basados en estos datos generados automáticamente, por ejemplo, una característica que suelen calcular estos algoritmos es el "wondering" o cuando una persona merodea por la imagen, lo cual nos podría indicar que es público y no corredor, ya que estos siguen el itinerario de la carrera.

Otra opción son las redes de reconocimiento cinético que determinen si una persona está corriendo o caminando. Existen múltiples ejemplos en este campo pero todo muy innovador y con resultados muy variables.

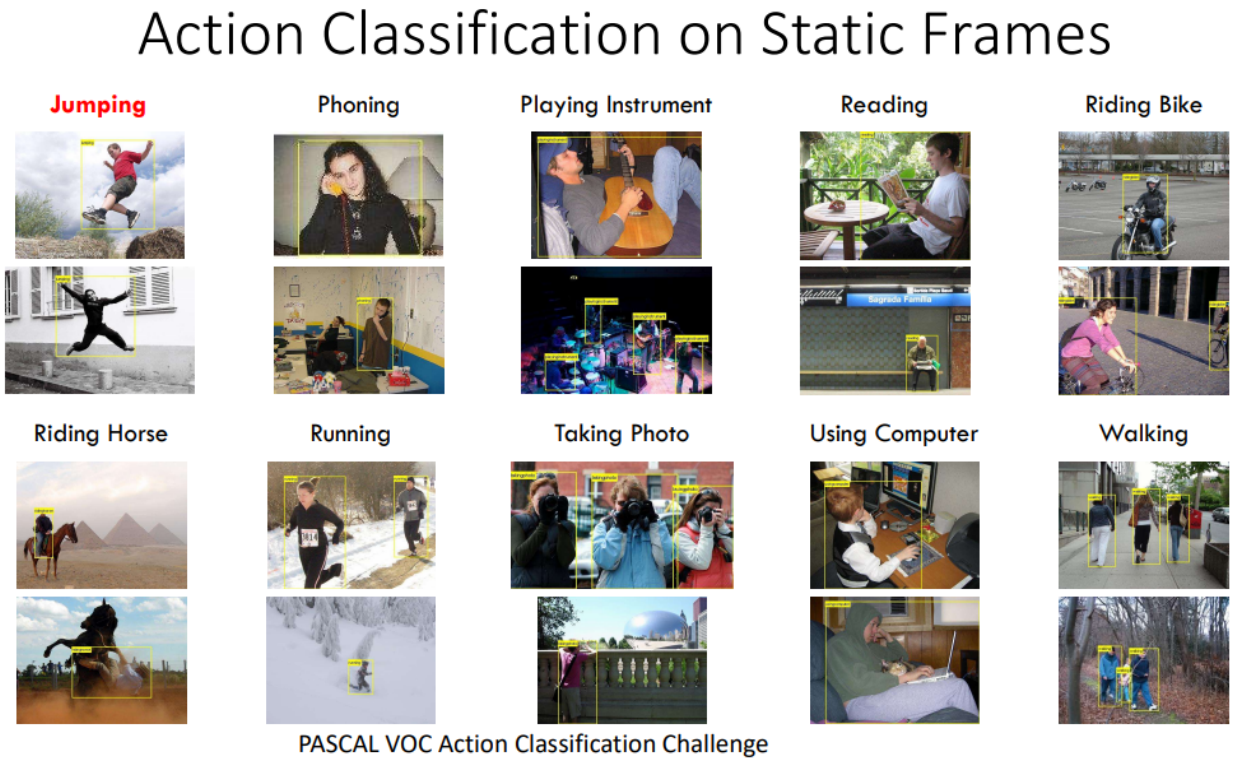

Ilustración 6.4: Reconocimiento de movimientos cinéticos [13].

También tenemos ejemplos de reconocimiento de poses y formas que podríamos utilizar para mejorar nuestros descriptores, por supuesto utilizar todos estos algoritmos llevaría un tiempo extenso de análisis y pruebas.

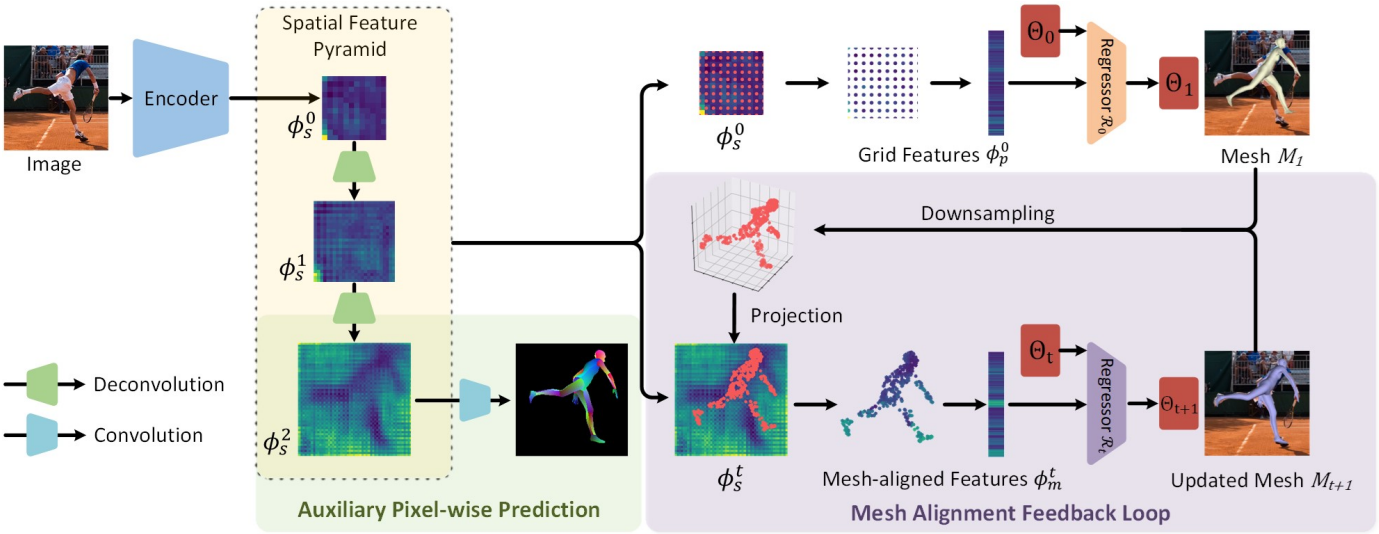

Ilustración 6.5: Ejemplo de utilización de reconocimiento de poses en personas mediante métodos de regresión [14].

Mejoras más sencillas a realizar sin utilizar software completamente nuevo podría ser simplemente tratar de ajustar la red, con algún algoritmo extra o ir comprobando el peso que posee cada descriptor elegido en nuestra predicción. De esta manera podríamos modificar los descriptores o incluso eliminar alguno si determinamos que no aporta información.

Estamos ante un proyecto de índole innovadora en un campo de dominio creciente donde es muy importante continuar con este tipo de investigaciones y avances. Solo hay que observar el claro crecimiento que lleva este dominio en los últimos años para llegar a la conclusión de que habrán avances muy significativos.

Este proyecto es un pequeño grano de arena con resultados prometedores y un mar de posibilidades a tener en cuenta.

Capítulo 7

Código y scripts disponibles

Todo el código de este proyecto son múltiples scripts de uso interno, para analizar los descriptores y determinar cuáles son los más indicados, y los segmentos a destacar serían las pruebas ya explicadas con los algoritmos de aprendizaje profundo. No obstante, todo el código está disponible en github, junto al YOLOv4 modificado.

Los vídeos no han sido subidos a github, por un lado por su extenso tamaño y porque han sido cedidos expresamente para realizar el trabajo. Los mapas de calor realizados para cada vídeo tampoco han sido subidos todos por su extenso tamaño pero si hay ejemplos disponibles, al igual que en la memoria.

En el "readme" se explica como ejecutar un entrenamiento y los pasos que se deben realizar, por último aclarar que todos los scripts incluidos en la carpeta ./Scripts han sido realizados por mí y que hay un documento que explica su utilización en la carpeta ./documentos/describiendoScripts.

Repositorio:

https://github.com/InDeX696/Detection-of-participants-in-sporting-events

Bibliografía

[1] "Resultados del maraton de gran canaria 2019." [Online]. Available: https://runealo.es/resultados-gran-canaria-maraton-2019/

[2] "Transgrancanaria." [Online]. Available: https://www.transgrancanaria.net/la-transgrancanaria-hg-2021-se-despide-con-un-sueno-para-el-proximo-ano/

[3] "Chip desechables y diferentes equipos utilizados en pruebas deportivas." [Online]. Available: https://macsha.com/one4all

[4] Imagen utilizada de medium.com (Ultimo acceso el 14 abril 2021). [Online]. Available: https://medium.com/@vanessabogado/competencia-en-kaggle-google-ai-open-images-detecci%C3%B3n-de-objetos-4ffaa389cb1c

[5] A. Startups, "Deepsort - deep learning applied to object tracking." [Online]. Available: https://www.youtube.com/watch?v=LbyqsoLJu5Q&ab_channel=AugmentedStartups

[6] S. Zivkovic, "Optical flow using horn and schunck method," solo se utilizo la imagen. [Online]. Available: http://datahacker.rs/013-optical-flow-using-horn-and-schunck-method/

[7] "Kalman filter photo." [Online]. Available: https://es.mathworks.com/videos/understanding-kalman-filters-part-3-optimal-state-estimator--1490710645421.html

[8] enrique a, "Detección de objetos con yolo: implementaciones y como usarlas." [Online]. Available: https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246

[9] H.-Y. M. L. Alexey Bochkovskiy, Chien-Yao Wang, "Yolov4: Optimal speed and accuracy of object detection." [Online]. Available: https://arxiv.org/pdf/2004.10934.pdf

[10] TheAIGuy, "yolov4-deepsort." [Online]. Available: https://github.com/theAIGuysCode/yolov4-deepsort

[11] "Arcotangente de dos parámetros." [Online]. Available: https://es.wikipedia.org/wiki/Arcotangente_de_dos_par%C3%A1metros

[12] "Alignedreid." [Online]. Available: https://github.com/michuanhaohao/AlignedReID

[13] "Reconocimiento de cinética." [Online]. Available: https://static.googleusercontent.com/media/research.google.com/es//youtube8m/workshop2018/p_i01.pdf

[14] "3d human pose and shape regression with pyramidal mesh alignment feedback loop." [Online]. Available: https://hongwenzhang.github.io/pymaf/

[15] Competencias de la ULPGC grado de ingeniería informática. [Online]. Available: https://www2.ulpgc.es/archivos/plan_estudios/4008_40/ObjetivosyCompetenciasdelGII.pdf

[16] Crisvill, "Avances en redes neuronales," 2016, (Ultimo acceso el 12 abril 2021). [Online]. Available: https://medium.com/espanol/avances-en-redes-neuronales-705c2efe53d2

[17] "Deep learning." [Online]. Available: https://en.wikipedia.org/wiki/Deep_learning

[18] S. R. Maiya, "Deepsort: Deep learning to track custom objects in a video." [Online]. Available: https://nanonets.com/blog/object-tracking-deepsort/

[19] "Meanshift." [Online]. Available: https://en.wikipedia.org/wiki/Mean_shift

[20] "Optical flow." [Online]. Available: https://en.wikipedia.org/wiki/Optical_flow

[21] "Kalman filter." [Online]. Available: https://en.wikipedia.org/wiki/Kalman_filter

[22] J. Redmon. [Online]. Available: https://twitter.com/pjreddie/status/1230524770350817280?s=20

[23] "Visual studio code." [Online]. Available: https://code.visualstudio.com/

[24] "Visual studio code wiki." [Online]. Available: https://es.wikipedia.org/wiki/Visual_Studio_Code

[25] "Github." [Online]. Available: https://github.com/

[26] "Overleaf." [Online]. Available: https://www.overleaf.com/

[27] "Overleaf wiki." [Online]. Available: https://en.wikipedia.org/wiki/Overleaf

[28] "Python." [Online]. Available: https://www.python.org/

[29] "Python wiki." [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language)

[30] "Javascript object notation." [Online]. Available: https://es.wikipedia.org/wiki/JSON

[31] "Anaconda." [Online]. Available: https://www.anaconda.com/

[32] "Anaconda wiki." [Online]. Available: https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)

[33] "Tensorflow." [Online]. Available: https://www.tensorflow.org/?hl=es-419

[34] "scikit-learn." [Online]. Available: https://scikit-learn.org/stable/

[35] "scikit-learn wiki." [Online]. Available: https://es.wikipedia.org/wiki/Scikit-learn

[36] "Pandas." [Online]. Available: https://pandas.pydata.org/

[37] "Opencv." [Online]. Available: https://es.wikipedia.org/wiki/OpenCV

[38] J. A. S. Ramirez, "Detection-of-participants-in-sporting-events." [Online]. Available: https://github.com/InDeX696/Detection-of-participants-in-sporting-events

[39] T. A. Guy, "Object tracking using yolov4, deep sort, and tensorflow." [Online]. Available: https://www.youtube.com/watch?v=FuvQ8Melz1o&t=64s&ab_channel=TheAIGuy

[40] "Decision tree." [Online]. Available: https://holypython.com/dt/decision-tree-pros-cons/

[41] "Random forest." [Online]. Available: https://en.wikipedia.org/wiki/Random_forest#:~:text=Random%20forests%20or%20random%20decision,average%20prediction%20(regression)%20of%20the

[42] "Gradient boosting." [Online]. Available: https://en.wikipedia.org/wiki/Gradient_boosting

[43] "Naive bayes." [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[44] "Logistic regression." [Online]. Available: https://en.wikipedia.org/wiki/Logistic_regression#:~:text=Logistic%20regression%20is%20a%20statistical,a%20form%20of%20binary%20regression).

[45] "Lstm." [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory

[46] "Gradient boosting vs random forest vs desicion tree." [Online]. Available: https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained

[47] "Alignedreid articulo." [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0031320319302031?via%3Dihub#!