Re-identificación basada en caras en tiempos de pandemia

Grado en Ingeniería Informática

Ignacio José Marín Reyes

Tutorizado por:

José Javier Lorenzo Navarro
Pedro Antonio Marín Reyes

07/2021

Agradecimientos

A mis amigos, por estos maravillosos cuatro años realizando juntos Ingeniería Informática.

A Modesto Fernando Castrillón Santana, por su colaboración en este trabajo.

A mi tutor, José Javier Lorenzo Navarro, por orientarme y ayudarme en la realización de este proyecto.

A mi cotutor, mi hermano/Pedro Antonio Marín Reyes, por aguantarme de lunes a domingo, guiarme en la realización de este trabajo, y mostrarme lo apasionante que puede ser la informática.

A mi familia, por apoyarme y confiar en este nuevo camino.

Abstract

The pandemic has completely changed the way that we behave. An obvious case are popular races, the risk of becoming infected with SRAS-CoV-2 has caused the mandatory of using mask in many popular races. For this reason, in this work, we verify how different detection and re-identification techniques behave in a popular race. In this Final Degree Project, we have some images of participants in races to analize, so images of the same people with and without a mask are available to evaluate how detection and re-identification methods are affected.

Resumen

La pandemia a cambiado por completo la forma en que nos comportamos. Un caso evidente, son las carreras populares, el riesgo de infectarse del SRAS-CoV-2 ha provocado el uso obligatorio de mascarillas en muchas carreras populares. Por eso, en este trabajo comprobamos como distintas técnicas de detección y re-identificación se comportan en una carrera popular. En este Trabajo de Fin de Grado, poseemos imágenes de distintos corredores para analizar, por lo que se disponen de imágenes de las mismas personas con y sin mascarilla para evaluar cómo se ven afectados los métodos de detección y re-identificación.

Índice general

Índice de figuras

Índice de Algoritmos

Índice de tablas

Capítulo 1

Descripción del proyecto

1.1. Introducción

En estos tiempos de pandemia del COVID-19, hemos elaborado un trabajo de fin de grado enfocado a analizar algunos problemas de la visión artificial. Estos problemas surgen como consecuencia de esta nueva situación de pandemia provocada por el SRAS-CoV-2, caracterizada por una serie de medidas de prevención frente al virus. En el caso de las carreras populares, aparecen nuevos problemas derivados de la necesidad de llevar mascarilla en algunos tramos de la competición. Por ello, en este trabajo nos centramos en identificar a un individuo entre múltiples cámaras (re-identificación de corredores), y determinar si un individuo lleva o no mascarilla (detección de mascarillas), para comprobar si el uso de mascarilla afecta a la re-identificación de corredores.

Para poder llevar a cabo este trabajo, se aplican todos los conocimientos adquiridos a lo largo del Grado en Ingeniería Informática, en la especialidad de Ingeniería del Software. Adicionalmente, estos conocimientos se complementan con información buscada de forma más autodidacta y no tan reglada. Principalmente, recalco la asignatura de Fundamentos de Sistemas Inteligentes para entender ciertos conceptos que utilizamos en este trabajo. Entre estos conceptos, podemos destacar las nociones básicas que son necesarias para en-

tender ciertos algoritmos que utilizan redes neuronales. A lo largo de este trabajo, nos apoyamos en algoritmos de re-identificación y detección que se basan principalmente en modelos neuronales.

Dentro de la disciplina de visión por computador, disciplina encargada de adquirir, analizar, comprender y procesar imágenes por parte de un ordenador, con el objetivo de interpretar dichas imágenes. Uno de los principales retos es la detección de caras. A lo largo de la literatura científica, los investigadores han implementado redes neuronales y otras técnicas, y escrito artículos para lograr dicho objetivo. Entre estos artículos podemos citar [8, 9]. Además, otro de los retos más importantes es la re-identificación de personas, destacamos artículos como [10].

En el campo de la visión por computador, sus inicios fueron en las principales universidades que estaban a la cabeza en los avances en inteligencia artificial, se empezaba a diferenciar del análisis de imágenes de aquella época porque se comenzaba a obtener características que describían los tres canales de la imagen, y que nos ofrecían una descripción completa. En la década de 1990, se comenzaban a usar modelos estadísticos para el reconocimiento de caras en las imágenes, se empezaban a utilizar las autocaras [11], conjunto de vectores de características que nos permitían resolver problemas de detección facial. Este campo se ha nutrido para su evolución en otras disciplinas como el reconocimiento de patrones, ingeniería de información, física del estado sólido, neurobiología, procesamiento de señales, matemáticas, estadísticas, etc.

La visión por computadores ha tenido una gran cantidad de aplicaciones, entre ellas podemos destacar, en los vehículos autónomos para que puedan reconocer los obstáculos a su alrededor. De esta manera, pueden trazar la ruta a seguir sin que exista ningún tipo de accidente. En la fabricación de toda clase de objetos, cada vez más se utilizan técnicas de visión por computadores para automatizar los procesos de fabricación. En la agricultura, por ejemplo para diferenciar los alimentos que se deben de cosechar. Y especialmente en el reconocimiento facial y la re-identificación de personas, aplicaciones en las que nos centramos en nuestro trabajo.

En el título de este trabajo, re-identificación basada en caras en tiempos de pandemia resumimos de forma muy precisa el sentido de este trabajo. Este trabajo tiene como contexto la pandemia del COVID-19 y las carreras populares celebradas durante este periodo, que han llevado a la situación del uso de mascarillas al competir.

1.2. Motivación

Las consideraciones más importantes para realizar este trabajo acerca de la re-identificación o detección de personas en tiempos de pandemia se deben a los inconvenientes que ocasiona el uso de mascarilla en estos procesos automáticos.

En la seguridad, comprobar si una persona es la misma en distintas imágenes tiene una gran importancia en las investigaciones de posibles delitos, por ejemplo, para el análisis de grandes cantidades de imágenes que han sido grabadas en distintos puntos de un robo. El personal de seguridad tiene el deber de comprobar si esa persona se corresponde con la misma en distintos puntos grabados por cámaras de videovigilancia, la re-identificación permite comprobar esta ingente cantidad de información de forma automatizada. Con estas comprobaciones los investigadores pueden trazar la posible ruta de escape del delincuente, su comportamiento previo al delito, etc. Además, no solo es utilizada en labores de investigación sino también de vigilancia, como puede ser la vigilancia de hogares. Actualmente, algunas cámaras de videovigilancia traen software de detección de personas, con el objetivo de no disparar la alarma por movimientos de cosas o de mascotas, sino por personas, que podrían ser potenciales ladrones.

En el sector de las ventas minoristas y mayoristas, ha empezado a tener una gran aplicación el uso de la identificación o re-identificación de personas, un ejemplo que demuestra esta utilidad son las tiendas de Amazon Go, tiendas que hacen uso de la tecnología de reconocimiento facial para trazar como el usuario se desplaza a través de la tienda. Primero, el usuario se identifica con

su móvil a través de la aplicación de Amazon al acceder a la tienda, solamente tiene que mostrar el código QR que le ofrece la aplicación en el lector de la entrada. Luego, las redes neuronales procesan la información y determinan si la persona es la misma en distintos momentos y puntos de la tienda. Esta tecnología permite al sistema averiguar la localización del usuario a lo largo de la tienda, que junto con el reconocimiento facial inicial en el escaneo del código QR, permite reconocer a ese individuo en la distintas imágenes grabadas por las cámaras de la tienda, con el objetivo de saber que individuo añade un producto a la cesta de la compra. Por tanto, como el individuo está asociado a una cuenta de usuario, el sistema permite añadir el producto elegido al carrito de la aplicación, con la finalidad de realizar el cobro de los productos al salir de la tienda, todo este proceso sin pasar por ninguna caja con un empleado que te atienda.

Otro campo de aplicación del reconocimiento facial es el entretenimiento. En los últimos años, han surgido multitud de aplicaciones que modifican nuestro rostro, por ejemplo, los filtros de Instagram [1] o TikTok [2] que nos permiten colocarnos cuernos, gafas, coronas de ángel, etc, es decir, añadir distintos elementos a nuestro rostro. Muchas de estas aplicaciones usan el reconocimiento facial para detectar la cara y determinar en que posición de la imagen se deben de colocar estos elementos en todo momento. Cualquier movimiento de la cara permite detectarla en una nueva posición recolocando los elementos. De esta manera, da la sensación de que somos un ángel, un diablo o tenemos gafas nuevas. Otra aplicación que estuvo de moda durante un tiempo fue FaceApp [3], principalmente porque podías envejecer a cualquier individuo, solamente tenías que ofrecer al sistema una imagen de un joven y el sistema a través de sus algoritmos la avejentaba. Estos algoritmos se basan principalmente en la visión por computador, particularmente en toda esta rama que se encarga de la detección de rostros, ya que la aplicación detecta el rostro del individuo y le aplica una máscara en 3D para mapear tu cara, con el fin de trazar los rasgos y manipularlos según las especificaciones del algoritmo de envejecimiento.

En cambio, en el ámbito de las carreras populares no se ha estandarizado el

[1] https://www.instagram.com
[2] https://www.tiktok.com/es
[3] https://www.faceapp.com

uso de la detección de caras para obtener una utilidad práctica. La mayoría de las carreras los únicos controles que existen son a través de personas o con dispositivos como chip que permiten registrar el tiempo de la carrera realizada o comprobar que el corredor efectivamente atraviesa todos los puntos claves del recorrido. Actualmente, la principales aplicaciones son en la investigación, como en este trabajo de fin de grado (TFG), que nos va a permitir determinar como algunos algoritmos de detección de caras y mascarillas junto con los de re-identificación actúan en el contexto de las carreras populares.

Personalmente, siempre he tenido un gran interés en todo este tipo de técnicas que considero que están provocando una revolución tecnológica en muchos sectores, y que definirán el futuro de muchos empleos en los cuáles el uso de estas tecnologías será necesario. Por tanto, la automatización de multitud de tareas debido a la visión por computadores y concretamente al reconocimiento facial, me motivan a aprender sobre estos temas y ha desarrollar este TFG.

Con respecto al ámbito de las carreras populares, la motivación personal se deriva del interés y la participación en carreras populares, por ejemplo, la carrera popular de "NightRun Las Palmas". Tras participar en ediciones anteriores y comprobar como se están llevando a cabo en estos tiempos de pandemia, me permiten realizar una comparativa e identificar una serie de problemas que aparecen por el uso obligatorio de la mascarilla, y que la visión por computador podría darles una solución. Por ejemplo, si es obligatorio el uso de la mascarilla en la salida, la detección de caras y mascarillas nos permiten determinar el número de individuos que realmente cumplen el reglamento. Desde el punto de vista de la re-identificación, el determinar si un mismo individuo es el mismo en distintos puntos de la carrera, nos permite conocer si ese corredor realmente ha realizado todo el recorrido de la carrera, o si en cambio se ha saltado algún punto. Aunque todos estos problemas la visión por computadores todavía no nos brinda una solución con exactitud, si considero que con su evolución y en un futuro, esta tecnología permita automatizar las tareas que actualmente realizan los voluntarios y empleados de la carrera.

1.3. Objetivo

El objetivo principal del TFG es comprobar si el uso de mascarilla afecta a los métodos de detección e identificación/re-identificación utilizando como escenario un evento deportivo. Para ello se deberán seleccionar una serie de métodos tanto de detección para comprobar su fiabilidad en este escenario y posteriormente en las caras detectadas y comprobar el rendimiento de algunos métodos de caracterización de caras basados en aprendizaje profundo.

1.4. Planificación temporal

En total se han planificado 300 horas para la realización de este TFG. En la Tabla 1.1 se muestra la distribución horaria.

Esta distribución se divide en cuatro fases: estudio previo y análisis, diseño, desarrollo e implementación, evaluación, validación y prueba, y documentación y presentación. Cada una de estas fases tiene una duración estimada que engloba una serie de tareas a realizar. Esta planificación no se ha modificado a lo largo de la realización de este TFG. Además, se ajusta al tiempo dedicado en la ejecución de este proyecto.

1.5. Justificación de la competencia

En el presente trabajo cubrimos una serie de competencias que se recogen en el título del Grado en Ingeniería Informática [12], entre estas competencias indicamos una competencia de cada tipo: general, formación básica, común en ingeniería informática, y específica de ingeniería del software. Algunas de estas competencias se han cubierto con la realización de este trabajo.

Con respecto a las competencias generales, señalamos:

Fases	**Duración Estimada**	**Tareas**
Estudio previo / Análisis	90 horas	Tarea 1.1: Revisión de métodos de detección de caras.
		Tarea 1.2: Revisión de métodos de representación de caras basados en redes neuronales.
		Tarea 1.3: Creación del conjunto de datos a partir de los vídeos de la LPATrail2020.
Diseño / Desarrollo / Implementación	110 horas	Tarea 2.1: Implementación de métodos de detección de caras.
		Tarea 2.2: Implementación de métodos de representación de caras.
		Tarea 2.3: Implementación de un esquema de re-identificación basado en caras.
		Tarea 2.4: Implementación de métricas de evaluación de rendimiento para detección y re-identificación.
Evaluación / Validación / Prueba	60 horas	Tarea 3.1: Comparación de métodos de detección de caras.
		Tarea 3.2: Comparación de métodos de representación de caras para re-identificación.
Documentación / Presentación	40 horas	Tarea 4.1: Redacción de la memoria del TFG.
		Tarea 4.2: Realización y ensayos de la presentación del TFG.

Tabla 1.1: Planificación temporal.

"**T3:** Capacidad para diseñar, desarrollar, evaluar y asegurar la accesibilidad, ergonomía, usabilidad y seguridad de los sistemas, servicios y aplicaciones informáticas, así como de la información que gestionan."

En el presente trabajo se trabaja con un sistema informático compuesto por imágenes, documentos y algoritmos que tiene que funcionar adecuadamente para obtener las métricas deseadas, por tanto, aseguramos su usabilidad, accesibilidad y ergonomía.

De formación básica destacamos:

"**FB03:** Capacidad para comprender y dominar los conceptos básicos de matemática discreta, lógica, algorítmica y complejidad computacional, y su aplicación para la resolución de problemas propios de la ingeniería."

Para el entendimiento y utilización de los algoritmos de detección y re-identificación es necesario comprender una serie de conceptos básicos de matemáticas que se enseñan a lo largo de la formación básica como ingeniero.

Con respecto a las comunes a la ingeniería informática, remarcamos:

"**CII015:** Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica."

La mayor parte de este proyecto utiliza sistemas inteligentes como pueden ser las redes neuronales para la identificación y re-identificación de personas.

Por último, destacamos en particular una competencia del Grado en Ingeniería Informática, en la especialidad de Ingeniería del Software:

"**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."

Consideramos que identificamos problemas que surgen en las carreras populares por el uso de las mascarillas, y el conocimiento de como son afectadas las técnicas de detección y re-identificación utilizadas en este trabajo, nos pueden permitir encontrar soluciones a estos problemas originados por esta nueva normativa. Para dar solución a dichos problemas llevamos a cabo el diseño, desarrollo, implementación y verificación de dicha solución software.

Por tanto, en la realización de este proyecto adquirimos o mejoramos las competencias anteriores que se detallan en el título del Grado en Ingeniería Informática.

1.6. Legislación

La normativa es un aspecto importante para la realización de este proyecto, existen cuestiones legales que se han tenido en cuenta para la legalidad de este proyecto. Por ejemplo, los derechos de imagen de las personas inscritas en las carreras populares. En particular, para este trabajo se trabajó con los derechos de imagen de los corredores de la carrera popular LPA Trail 2020.

Tenemos que tener en cuenta que, en el artículo 18, apartado 1 de la Constitución Española [13] se garantiza el derecho a la imagen, al honor y a la intimidad personal y familiar. Por tanto, existen una serie de supuestos derivados de dicha normativa. En nuestro proyecto, nos afecta principalmente el siguiente supuesto [14]:

"La captación, reproducción o publicación por fotografía, filme o cualquier otro procedimiento, de la imagen de una persona en lugares o momentos de su vida privada o fuera de ellos, salvo los casos previstos en el artículo 8.2."

En nuestro caso, queda prohibido la captación o filmación de dichos corredores en la carrera popular. Por ello, para salvaguardar dicho derecho individual tenemos que solicitar el consentimiento de dichos corredores. Este consentimiento lo hemos conseguido de la siguiente manera. En el artículo 6 del regla-

mento de LPA Trail 2020 se habla de dichas cuestiones, comprobamos que en el apartado 6.3 de dicho reglamento [15] los corredores dan el consentimiento a la organización para el tratamiento de dichas imágenes de carácter personal, por tanto, podrán ser tratadas por la organización con fines deportivos, promocionales o comerciales.

En definitiva, la organización que tiene el consentimiento para el tratamiento de esos datos de carácter personal de la manera que estime oportuna dentro de los fines citados anteriormente, nos da el consentimiento para el tratamiento de dichas imágenes en el presente trabajo.

1.7. Estructura del documento

La memoria descrita en este documento sigue la siguiente estructura: resumen, abstract, índice general, de figuras, de tablas y de algoritmos, los seis capítulos, los cuáles explicamos con mayor detalle a continuación, y la bibliografía.

1.7.1. Descripción del proyecto

En el primer capítulo, Descripción del proyecto, ofrecemos una introducción, motivación, objetivos, planificación temporal, justificación de la competencia, y legislación y la presente sección estructura del documento.

En la *Introducción*, ubicamos el trabajo en un contexto que es el de la pandemia del COVID-19, detallamos donde hemos adquirido los conocimientos necesarios para realizar este proyecto, recordamos brevemente que es el campo de la visión por computadores y su relación con este trabajo, explicamos un poco de historia acerca de esta disciplina, detallamos las aplicaciones que ha tenido la visión por computador y damos a entender el sentido del título ”Re-identificación basada en caras en tiempos de pandemia”.

En la *Motivación*, ofrecemos una serie de consideraciones qué nos han motivado a realizar dicho trabajo. Para ello, explicamos una serie de ejemplos de determinados sectores que hacen uso de tecnologías de identificación o re-identificación como la usada en este trabajo para los corredores. Además, damos una motivación más personal desde el punto de vista de los propios gustos del autor del trabajo. Y por último, una motivación derivada de la observación y detección de problemas que han surgido en este contexto de pandemia en las carreras populares.

En los *Objetivos*, detallamos el fin de este trabajo y qué queremos conseguir con su realización, principalmente, comprobar si el uso de mascarilla afecta a los métodos de detección y re-identificación en un evento deportivo.

En la *Planificación temporal*, comentamos como hemos distribuido las 300 horas destinadas a este proyecto, asignándolas a cada fase del proyecto con sus correspondientes tareas.

En la *Justificación de la competencia*, argumentamos por qué este trabajo nos permite adquirir o mejorar algunas competencias que se detallan en el conjunto de competencias de la titulación del Grado en Ingeniería Informática.

En la *Legislación*, hablamos sobre los aspectos legales como pueden ser los derechos de imagen de los participantes en la carrera y cómo conseguimos el consentimiento para utilizar la galería de imágenes que tenemos de esta carrera.

En la *Estructura del documento*, ofrecemos un breve resumen de cada uno de los apartados de cada capítulo.

1.7.2. Marco teórico

En el segundo capítulo, Marco teórico y estado del arte hablamos de estos dos términos.

En el *Marco teórico* explicamos los conceptos fundamentales que están relacionados con el campo de la visión por computador y ubicamos al lector en este marco teórico para un mejor entendimiento de nuestro proyecto.

En el *Estado del arte* mencionamos los artículos de algunos autores que investigan en el campo de la visión computacional y realizan trabajos relacionados con la detección y re-identificación.

1.7.3. Estudio previo y análisis

En el tercer capítulo, Estudio previo y análisis, planteamos el problema, analizamos y evaluamos el conjunto de datos y estudiamos las librerías, algoritmos basados en Redes Neuronales, métricas, herramientas y tecnologías utilizadas.

En el *Planteamiento del problema* exponemos el problema que nos surge en este proyecto y queremos solucionar.

En el *Análisis y evaluación del conjunto de datos* analizamos el conjunto de datos que obtenemos en la competición que mencionamos en este trabajo.

En el *Estudio de herramientas y tecnologías* analizamos todas herramientas y tecnologías que hemos utilizado en este trabajo y describimos brevemente cada una de ellas.

En el *Estudio de las librerías* analizamos las librerías más convenientes para utilizar en este proyecto y detallamos sus principales características.

En el *Estudio de los algoritmos basados en redes neuronales* estudiamos las características de cada uno de los algoritmos, que herramientas nos ofrecen, y su idoneidad para alcanzar los objetivos de este proyecto.

En el *Estudio de las métricas de detección de mascarillas y re-identificación*

analizamos las medidas cuantitativas que nos permiten determinar el grado de consecución de nuestros objetivos tanto para la detección como la re-identificación.

1.7.4. Metodología e implementación

En el cuarto capítulo, Metodología e implementación, exponemos la metodología y la implementación desarrollada.

En la *Metodología* explicamos las distintas fases de la metodología utilizada para desarrollar este TFG.

En la *Implementación* detallamos cada uno de los pasos que realizamos para alcanzar las estadísticas y las métricas y cómo hemos implementado la solución software.

1.7.5. Experimentos y resultados

En el quinto capítulo, Experimentos y resultados, mostramos los experimentos y resultados de la detección de mascarillas y re-identificación de corredores.

En la *Detección de mascarillas* explicamos los experimentos realizados y los resultados que hemos obtenido en la línea de trabajo de la detección de mascarillas.

En la *Re-identificación de corredores* detallamos las pruebas realizadas y las métricas obtenidas en la línea de trabajo de la re-identificación de corredores.

1.7.6. Conclusiones

En el sexto y último capítulo, Conclusiones, explicamos si hemos alcanzado el objetivo de este TFG y su grado de consecución, detallamos brevemente la implementación llevada a cabo, explicamos los resultados más relevantes y su por qué, detallamos una línea de trabajo futura y realizamos una apreciación personal acerca del proyecto.

Capítulo 2

Marco teórico y estado del arte

2.1. Marco teórico

La visión por computador es un campo que se encarga de extraer información del mundo físico a partir de imágenes, utilizando para ello un computador [16].

Para la realización de este trabajo, se pretende obtener información a partir de imágenes en color. Estas imágenes se definen como coordenadas (x, y), las cuáles tienen una terna de valores asociados, es decir, cada punto de la imagen tiene tres valores que describen una intensidad de rojo, verde y azul. Estos valores van desde 0 hasta 255 para cada color. Por ejemplo, el azul es (0, 0, 255).

Las redes neuronales artificiales [17] se basan principalmente en el concepto de neurona, es decir, unidades de proceso cuya función es recibir entradas y calcular un valor de salida en base a esas entradas. Cada neurona está asociada a una función de salida y a un estado de activación que determina el valor de salida como mostramos en la Figura 2.1. Esta función de salida está compuesta por una serie de pesos que son de gran importancia a la hora de entrenar la neurona.

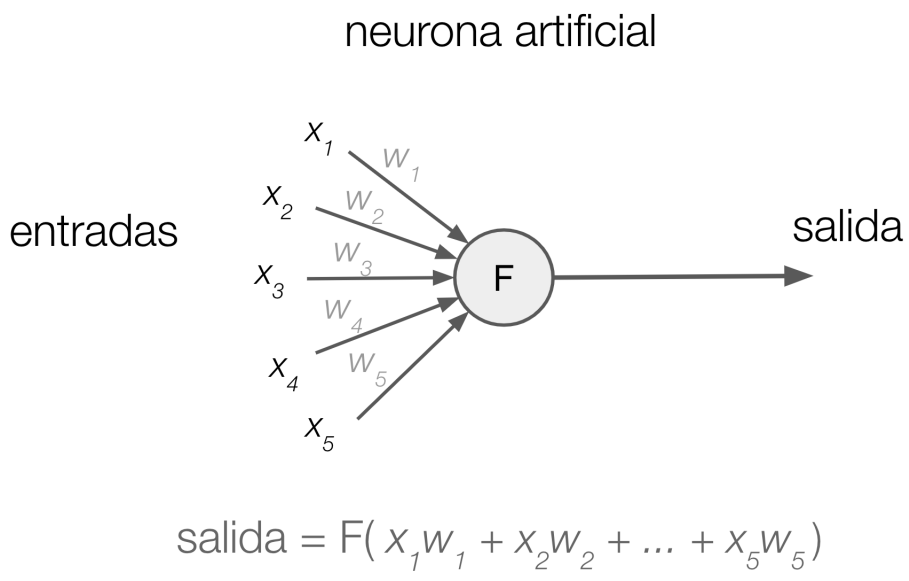

Figura 2.1: Neurona artificial.

Normalmente, un conjunto de neuronas reciben el nombre de capa. El encargado de la red procura configurar una red con los hiperparámetros que más le interesan, estos pueden ser el número, organización o tipo de capas, el tamaño del lote, el método de optimización elegido, etc. Una vez establecida la configuración de la red neuronal se procede a entrenarla, se le suministran datos a la red y a través de optimizaciones, como por ejemplo, las optimizaciones que realiza el método del descenso por el gradiente, la red neuronal es capaz de ajustar sus pesos para alcanzar los valores óptimos de estos. Una vez que la red se entrena, se pueden realizar pruebas, entre las que incluimos las re-identificaciones y detecciones que tratamos en este trabajo.

Para el tratamiento de imágenes se suele utilizar un caso concreto de red, las redes neuronales convolucionales [18]. Estas redes se suelen caracterizar porque las entradas a una neurona solamente se corresponden con un conjunto de salidas de la capa anterior. Estas redes suelen tener tres tipos de capas: de entrada, extracción de características y clasificación. La capa de entrada recibe las imágenes a color de los corredores en nuestro problema, por tanto, la capa de entrada está formada por tres canales, uno para el rojo, verde y azul que ofrecen la información de 0 a 255 de la intensidad de su correspondiente color, normalmente, normalizado de 0 a 1 para cada píxel. En la capa de extracción de características encontramos las capas convolucionales,

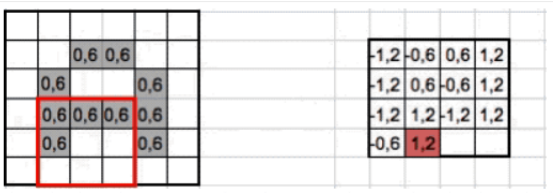

Figura 2.2: Obtención capa convolucional. Imagen obtenida desde [1].

de agrupación, aplanamiento y totalmente conectada. La capa convolucional se obtiene al utilizar el operador matemático convolución que se define según la Ecuación 2.1.

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(x) \cdot g(t - x)dx \tag{2.1}$$

En forma discreta, se definen las funciones anteriores como funciones que describen el kernel y la imagen, por tanto, con un tamaño n del kernel, obtenemos la capa convolucional con la Ecuación 2.2.

$$conv2D(i, j) = \sum_{y=0}^{n-1} \sum_{x=0}^{n-1} kernel(x, y) \cdot imagen(i + x, j + y) \tag{2.2}$$

Entonces, la capa de convolución a través de su definición se obtiene como muestra la Figura 2.2, donde el kernel representado por el cuadro rojo se va desplazando a lo largo de la imagen y calculando la convolución elemento a elemento, como se especifica en la Ecuación 2.2.

Según el kernel definido obtenemos una característica local de la imagen, por ello, utilizamos varios kernels para obtener muchas características locales del objeto. De esta forma, generamos la capa de convolución como un tensor.

La capa de agrupación (término del inglés pooling) disminuye las dimensiones bidimensionales de la capa anterior, concretamente con la de máxima agrupación (término del inglés max pooling), nos quedamos con el valor mayor de una zona.

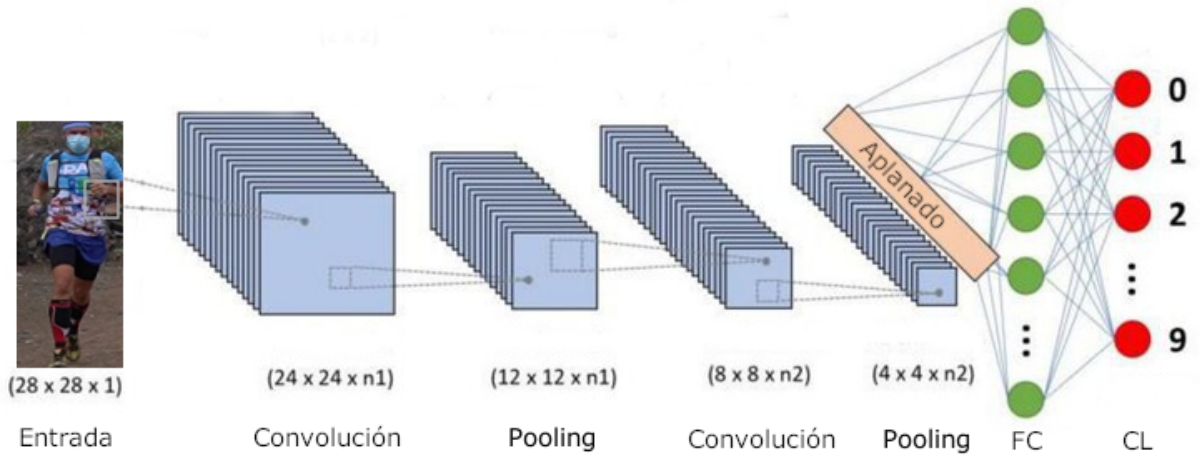

Figura 2.3: Arquitectura de red neuronal.

La capa de aplanamiento se encarga de convertir en un vector con una única dimensión el tensor de las capas anteriores caracterizado por sus tres dimensiones. Este vector es fundamental para los problemas de re-identificación e identificación que usamos en este trabajo, ya que, por ejemplo a partir de este vector de características somos capaces de calcular las distancias que se utilizan para determinar la similitud de los individuos.

En los problemas de clasificación, la capa de clasificación es primordial ya que nos permite determinar a que clase pertenece un objeto, por ejemplo, si es una persona, coche, mochila, etc. Algunas funciones que se utilizan en esta capa son la sigmoide o softmax, entre otras muchas.

A modo de resumen, mostramos en la Figura 2.3 un ejemplo de cómo obtenemos el vector de características de una imagen de un corredor, y como está estructurada una red neuronal en diversas capas. Adicionalmente, con la capa de clasificación comprobamos cómo se clasificaría esa imagen.

Estas redes neuronales son ampliamente usadas en problemas de re-identificación de personas, que emparejan a personas a través de diferentes cámaras en diversos escenarios que generalmente son de videovigilancia, ver la Figura 2.4. Existen dos términos en este tipo de problemas, galería (término del inglés gallery) y referencia (término del inglés probe). La referencia es un individuo concreto de una cámara en particular que utilizamos para realizar

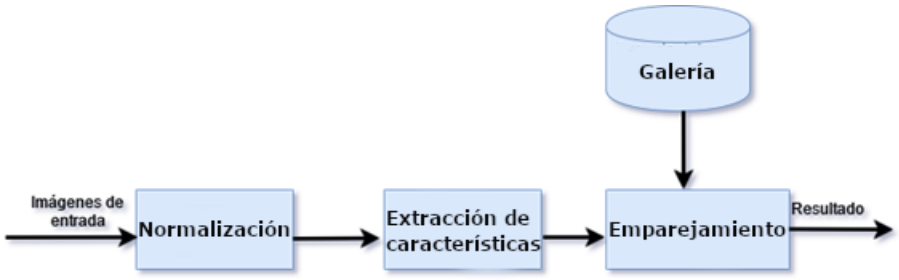

Figura 2.4: Proceso de re-identificación.

las comparaciones contra el conjunto de galería. Mientras que, la galería se refiere a todo el conjunto de identidades de todas las cámaras menos la cámara donde aparece el individuo en particular.

Los problemas de re-identificación pueden oscilar en diferentes complejidades, dependiendo del tipo de problema que se quiera afrontar. Se puede tratar desde una única imagen por persona a múltiples imágenes por persona, como también la temporalidad del problema, no es lo mismo re-identificar a una persona en un corto período de tiempo que en uno de larga duración, donde los rasgos faciales pueden cambiar. Asimismo, existen variantes en los problemas de re-identificación, denominándolos problemas de mundo abierto y mundo cerrado [19]. En el primero no tenemos la certeza de que todas las identidades hayan sido capturadas por todas las cámaras. Por tanto, es necesario identificar cuando una identidad no existe en la galería para posteriormente crear esa nueva identidad. En cambio, en el segundo si tenemos la certeza de que todas las identidades existen en la galería.

2.2. Estado del arte

Los estudios de detección y re-identificación son numerosos en el campo de la visión por computador. En algunos casos se estudian problemáticas diferentes que están relacionadas con nuestro problema, la detección y re-identificación en el ámbito de carreras populares. Estos problemas aparentemente diferentes, de fondo pueden albergar cierta similitud, es decir, muchos hacen usos de redes

neuronales convolucionales pero con una aplicación distinta. A continuación, citamos algunos artículos que se basan en los mismos principios que usamos para la detección y re-identificación en nuestro trabajo, aunque en algunos casos en un ámbito totalmente distinto al nuestro.

Uno de estos ámbitos son los problemas de diarización, que buscan averiguar quien y cuando un individuo se pronuncia, se sustentan en las re-identificaciones para averiguar quien vuelve a hablar [20], estos autores estudian la viabilidad de la diarización automática en sesiones parlamentarias desde una problemática de mundo abierto. Proponen el uso de la transformación de la tasa logarítmica isométrica basadas en las probabilidades a posteriori de la existencia de un individuo en el sistema.

Otro sistema es el propuesto por los autores en [21] que hace un recorrido por los problemas de mundo cerrado y proponen una capa de agrupación denominada GeM (acrónimo en inglés de media generalizada agrupada) que es empleada en una arquitectura Resnet50 [22] en escenarios de mundo abierto para conjuntos de datos del estado del arte.

No solamente los trabajos tienden por la línea del tipo de problema mundo abierto, sino que también, se siguen estudiando los problemas del tipo de mundo cerrado. Entre ellos, encontramos a los autores [23], que analizan la re-identificación en entornos de videovigilancia de viandantes. Para ello proponen un algoritmo basado en la supresión del fondo para luego dividir el cuerpo de la persona en varias proporciones y extraer unos vectores de características de estos.

Si nos centramos en problemas específicos del ámbito de las carreras populares, nos encontramos estudios como [24] cuyos autores pretenden identificar al corredor a través de la detección y reconocimiento del número de dorsal del participante, para ello, se utilizan técnicas de reconocimiento de texto en escenarios naturales, como son las R-CNN (acrónimo del inglés red neuronal convolucional basada en regiones) de donde extraen los vectores de características.

Figura 2.5: Transgrancanaria 2020. Imagen obtenida de [2].

También, existe algún estudio del ámbito de las carreras populares que afronta un objetivo similar de re-identificación que nosotros [2], los autores analizan con una serie de técnicas a participantes del evento deportivo, Transgrancanaria 2020, competición de 128km tanto de día como de noche y en seis puntos diferentes de la competición. En este trabajo, se tuvieron en cuenta las particularidades del mundo real, como por ejemplo, diferencias de tiempo, diferentes situaciones climáticas y de iluminación, o cambios de indumentaria por parte de los corredores. En la Figura 2.5 se muestran los puntos de la competición.

Por otro lado, también nos gustaría citar otro artículo [25] muy relacionado con la re-identificación de corredores, en el cuál evalúan la forma de moverse del corredor (término del inglés running gait) como atributo para el proceso de re-identificación en eventos de carreras de larga distancia.

Con respecto a nuestro objetivo de detección de mascarillas, existen menos estudios debido a la actualidad de la pandemia del COVID-19, pero podemos mencionar algún estudio reciente [26], los autores usan un modelo híbrido que usa aprendizaje automático profundo y clásico para detectar mascarillas. La implementación se basa en extraer características usando Resnet50 [22], mientras que para el proceso de clasificación usan árboles de decisión, máquinas de soporte vectorial y algoritmos ensamblados. Los conjuntos de datos son obtenidos de: RMFD, SMFD y LFW. Se lograron medidas de evaluación entorno a los 99,64 % de exactitud con el clasificador SVM.

Otro estudio reciente relacionado con la detección de mascarillas [27], es el de los autores del citado estudio que intentan ayudar a las autoridades a mitigar, evaluar, prevenir y actuar ante el SRAS-CoV-2. Ellos plantean un método basado en aprendizaje automático, concretamente el clasificador de imágenes MobileNetV2 [28], una red neuronal convolucional que fue desarrollada por Google. El modelo consiste en recolectar, pre-procesar y dividir datos, con el objetivo de probar e implementar el modelo. Con este modelo podemos destacar si las personas usan o no mascarilla con una exactitud del 96,85 %.

Capítulo 3

Estudio previo y análisis

3.1. Planteamiento del problema

El problema que planteamos en este trabajo lo exponemos a continuación, los métodos de identificación y re-identificación se ven afectados por el uso de mascarilla. Establecimos el objetivo de determinar la influencia de la mascarilla en un contexto de participantes que debían hacer uso de ella, estableciendo una serie de métricas para evaluar dicha influencia.

Para dar una solución al problema planteado, se realiza una fase de análisis para determinar que algoritmos debemos aplicar, cómo debemos desarrollarlos o que riesgos existen a la hora de diseñar distintas alternativas. Por tanto, se lleva a cabo un proceso de estudio y análisis de una serie de algoritmos existentes en la comunidad científica, en aras de determinar si son apropiados para nuestro problema y qué limitaciones pueden tener en este ámbito. Obviamente, para concluir sobre las métricas que vamos a utilizar con el fin de alcanzar nuestro objetivo, se necesita también de un estudio y análisis que nos aconseje que herramientas debemos utilizar. Además, debemos analizar la adecuación de ciertas tecnologías, por ejemplo librerías, o la adecuación de las fuentes de datos, imágenes a utilizar en nuestro contexto. Estas imágenes es necesario investigar cómo vamos a obtenerlas para crear el conjunto

Modalidades	Nº asistentes	Hombres (%)	Mujeres (%)	Veteranos (%)
Corta (5km)	25	10 (40)	15 (60)	11 (44)
Media (10km)	115	86 (74)	29 (26)	64 (55)
Larga (20km)	60	52 (86)	8 (14)	26 (43)
Total	200	148 (74)	52 (26)	101 (69)

Tabla 3.1: Información asistentes.

de datos. Todas estas herramientas, librerías, galerías de imágenes y tecnologías que utilizamos en este proyecto las detallamos con mayor claridad en los siguientes apartados.

3.2. Análisis y evaluación del conjunto de datos

Para obtener las métricas que deseamos para nuestro proyecto, hemos partido de una galería de imágenes obtenida de la carrera popular de LPA Trail 2020 [29].

Se ha decidido elegir esta competición porque alberga las condiciones necesarias para probar nuestra hipótesis de si se ven afectados los métodos de detección y re-identificación con el uso de mascarilla. Entonces, necesitamos de escenarios con mascarillas y sin mascarillas. Esta competición obliga al uso de mascarilla en la salida de la competición y en los puntos de avituallamiento. En cambio, en otros puntos de la carrera ya no es necesaria, por eso, la mayoría de los corredores no la llevan puesta en esos tramos.

Esta competición fue celebrada el sábado 24 de octubre de 2020 y se inscribieron alrededor de 529 personas, de las cuáles solamente asistieron un 39 %, es decir, 200 personas. Posiblemente, esta alta tasa de abandono se debe a la situación de pandemia del COVID-19. En la Tabla 3.1 mostramos información acerca de los participantes que asistieron a la competición, ya que son los que nos interesaran de cara a análisis posteriores.

Esta competición está compuesta por tres modalidades, corta (5km), media

(10km) y larga (20km). En modalidad corta compitieron alrededor de 25, en media 115 y en larga 60 corredores. Según el género, alrededor del 10 %, 86 % y 52 % de los participantes son hombres respectivamente a las modalidades mencionadas anteriormente. En cambio, según la edad, podemos incluir al 44 %, 55 % y 43 % de los participantes en el grupo de veteranos respectivamente a las modalidades mencionadas anteriores, es decir, existe un número significativo de personas mayores. Esta información se ha obtenido gracias al portal que tienen disponible para todos los usuarios [30] la clasificación de la competición. Podemos observar que la muestra a analizar está parcialmente distribuida según el género o la edad, por tanto, el resultado de los algoritmos es más idóneo para generalizar.

Para obtener las imágenes necesarias para crear el conjunto de datos, se han posicionado diferentes personas en distintos puntos de la carrera, grabando a los corredores cuando han pasado por dicho tramo. En nuestro proyecto, usamos concretamente las imágenes de dos puntos de la carrera y en la modalidad media (10km), por tanto, la galería de imágenes utilizada para la detección y re-identificación se ha obtenido en un punto que llamamos Tafira, y en otro punto que llamamos Universidad, como mostramos en la Figura 3.1.

Esta figura nos enseña el recorrido realizado por los corredores en la modalidad media y los puntos donde se han realizado las grabaciones gracias a las marcas de las chinchetas. La carrera comienza en El Fondillo y termina en la plaza de Santa Ana en Vegueta, son exactamente 13,5 km con una desnivel positivo de 267 m y un desnivel negativo de 485 m. Por tanto, tras la salida nos encontramos con el primer punto de grabación, Tafira, cerca de Tafira Alta, en la Figura 3.2 observamos su apariencia.

Tras atravesar ese punto, los corredores siguen corriendo hasta el siguiente punto, Universidad, que está próximo al Campus Universitario de Tafira. Además, en esta localización se encuentra un avituallamiento de la competición como mostramos en la Figura 3.3.

La elección de estos lugares para realizar la grabación se debe a un motivo muy importante en nuestro trabajo. En la primera localización que se corresponde

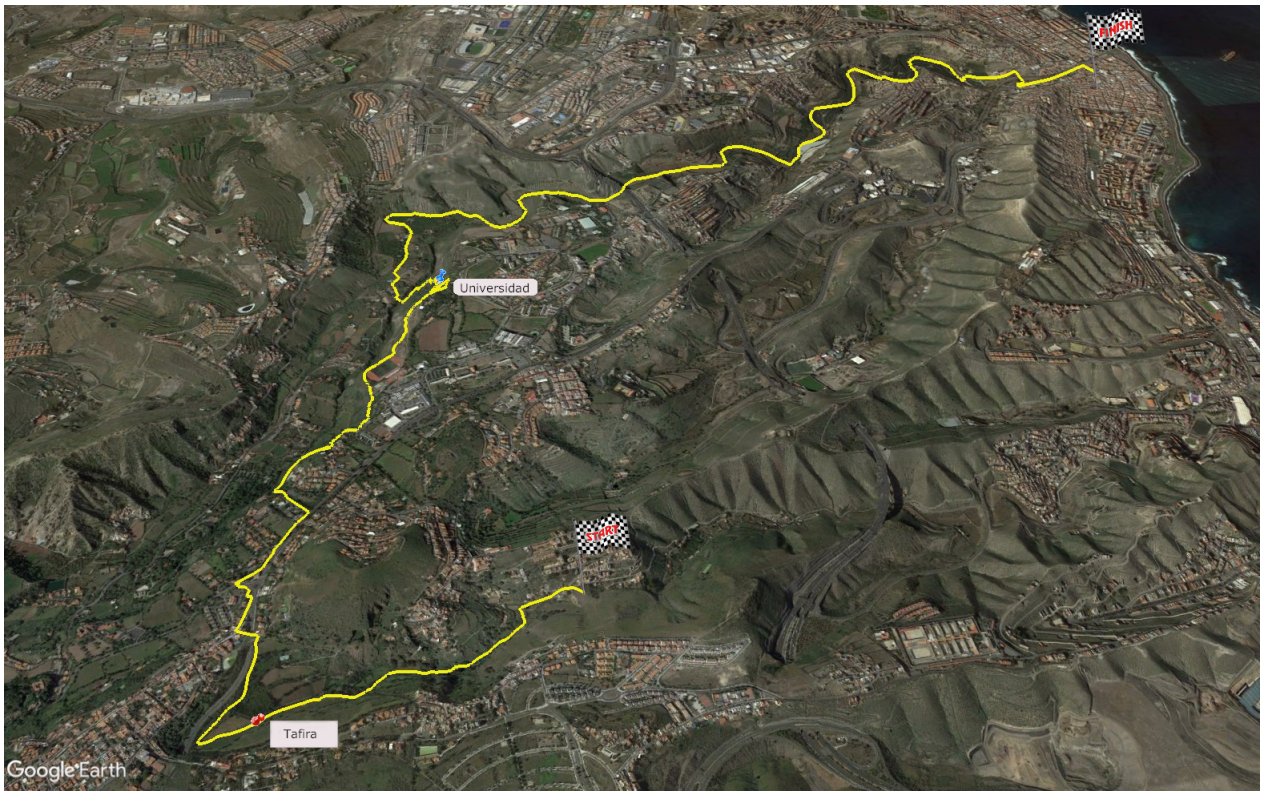

Figura 3.1: LPA Trail 2020 media (10 km).

Figura 3.2: Punto de grabación Tafira.

Figura 3.3: Punto de grabación Universidad.

con el punto de Tafira no es obligatorio el uso de mascarilla, por eso, la mayoría de los competidores no la llevan puesta. En cambio, en la segunda localización, Universidad, están obligados a llevarla si se detienen en el avituallamiento. Por tanto, tras visualizar todas las imágenes comprobamos que alrededor del 50 % de los participantes no la llevan puesta. Este hecho en nuestro trabajo, nos va a permitir tener un grupo con mascarilla y otro grupo sin mascarilla en la Universidad, esto tiene implicaciones importantes en el proceso de re-identificación. En definitiva, tenemos dos lugares con características distintas, en Tafira, todos van sin mascarilla mientras que en la Universidad mitad si llevan y la otra mitad no. Esto de cara al proceso de re-identificación y detección de mascarillas nos permite calcular unas métricas, para averiguar si los métodos de detección y re-identificación se ven afectados por el uso de mascarilla, en la Sección 5.2 y la Sección 5.1 lo explicamos con mayor detalle.

3.3. Estudio de herramientas y tecnologías

Para la ejecución o edición de los algoritmos anteriores se necesitan de una serie de herramientas y de un entorno de trabajo apropiado. Además, necesitamos herramientas para escribir la presente memoria, por eso, a continuación describimos estas tecnologías y herramientas: Python, Java, Netbeans, Google Colab, Jupyter Notebook, Pycharm, Drive, Copia de Seguridad y Sincronización Google Drive, LaTeX, Overleaf, Git, GitHub y Gimp.

Python

Python es un lenguaje de programación interpretado debido a su capacidad de traducir el código a lenguaje máquina a medida que es necesario, multiparadigma porque utiliza los paradigmas de la programación orientada a objetos, programación imperativa y programación funcional, y dinámicamente tipado porque la comprobación del tipo la realiza durante la ejecución del programa. La característica más destacable de este lenguaje es su legibilidad, ya que se ha desarrollado con el objetivo de que sea fácilmente comprensible. Su creador fue Guido van Rossum y decidió que fuera un lenguaje de código abierto.

La elección de este lenguaje se debe a la gran aceptación que tiene en la comunidad científica para temas de visión por computadores, ya que tiene una gran cantidad de librerías estadísticas y numéricas que facilitan la tarea de investigación. Además, en nuestro caso, todos los algoritmos mencionados en la Sección 3.5 han sido codificados mayormente en Python, por lo que el uso de Python es fundamental en nuestro trabajo.

Java

Java es un lenguaje de programación compilado debido a su capacidad de traducir el código a lenguaje máquina a partir del código fuente, multipara-

digma ya que utiliza paradigmas como la orientación a objetos, programación imperativa y funcional. El tipado es estático, es necesario especificar el tipo de variable a utilizar. Y su creador fue James Gosling a través de la empresa Sun Microsystems, que posteriormente fue adquirida por Oracle. La licencia utilizada es una licencia pública general de GNU, por lo que puede ser considerado un lenguaje de código libre y abierto.

Este lenguaje se utiliza en menor medida en nuestro trabajo, para el único caso que es utilizado es para el diseño de un programa de etiquetado que es de utilidad para el proceso de identificación de mascarillas que describimos en mayor detalle en la Sección 4.2.2.

Netbeans

Netbeans es un entorno de desarrollo integrado (IDE) especialmente diseñado para el lenguaje de programación Java. Esto se debe a que fue diseñado por la empresa Sun Microsystems, que posteriormente fue adquirida por Oracle, creadora de Java. Netbeans es un software de código libre y abierto.

En nuestro trabajo, es utilizado para el desarrollo de la aplicación de etiquetado. Especialmente se elige este entorno de desarrollo ya que nos ofrece una serie de herramientas para automatizar el proceso de diseño de interfaces de usuario haciendo uso de Swing.

Google Colab

Es una herramienta muy utilizada por estudiantes e investigadores, consiste en un cuaderno en el que puedes combinar código ejecutable, texto enriquecido, imágenes, HTML, LaTeX, etc. Este código ejecutable puede ser código Python. Por otro lado, nos permite ejecutar comandos de Shell como si utilizamos una interfaz de línea de comandos. Google Colab es una aplicación

gratuita que se basa en el software libre Jupyter Notebook y que trabaja en los servidores de Google, por lo que, nuestras implementaciones no serán ejecutadas de forma local sino en servidores externos. Además, otra característica importante es la integración con la nube de almacenamiento de Google Drive que nos permite almacenar los datos necesarios para la ejecución de los algoritmos.

En nuestro caso, estudiamos la posibilidad de ejecutar los algoritmos de forma local en mi ordenador, un MacBook Pro de principios de 2011, pero tuvimos que buscar otra alternativa como Google Colab, ya que existían limitaciones en cuanto a los requisitos de los algoritmos de redes neuronales profundas y a las prestaciones de la tarjeta gráfica. Por ejemplo, algunos de estos algoritmos necesitan utilizar CUDA para su ejecución. Por tanto, para utilizar esta plataforma de computación en paralelo es necesaria una tarjeta gráfica de NVIDIA y mi ordenador dispone de una Intel HD Graphics 3000 512MB. Entonces, tanto la necesidad de potencia de cómputo y las incompatibilidades nos obligaron a utilizar las gráficas de Google. Google te ofrece en su cuenta de Colab Pro una gráfica NVIDIA Tesla T4 o NVIDIA Tesla P100. También, nos vimos obligados a utilizar la versión de pago de 9,99€ al mes, ya que existen restricciones al uso de las tarjetas gráficas en la versión gratuita que nos impedían ejecutar los algoritmos. Por ejemplo, tras un uso continuado de su potencia de cómputo las tarjetas dejan de estar operativas hasta que la empresa lo considere oportuno.

Jupyter Notebook

Jupyter Notebook es una herramienta gratuita y una aplicación web. Esta aplicación nos permite crear documentos Jupyter con extensión ipynb. Estos documentos consisten en una lista de celdas que pueden tener código, texto, texto enriquecido, imágenes, etc. Por tanto, a través de estos cuadernos podremos ejecutar el código.

Pycharm

Es un IDE desarrollado por la empresa JetBrains y de licencia de código libre y abierto. Entre sus principales usos, se encuentra la programación en lenguaje Python.

Google Drive con copia de seguridad y sincronización

Drive es una espacio de almacenamiento que nos ofrece la empresa Google para guardar nuestra información como pueden ser imágenes, documentos, textos, etc. En su versión gratuita tenemos una capacidad de almacenamiento de 15GB, mientras que en sus planes premium podemos alcanzar los 100GB, 200GB y 2TB. Además, cuenta con una aplicación para la sincronización de tus ficheros locales con los de la nube.

Para nuestro trabajo, hemos tenido que adquirir la versión premium de 100GB que cuesta alrededor de 1,99€ al mes, debido a que la galería de imágenes tiene un peso muy superior a los 15GB gratuitos. La idea es que toda la información necesaria para este trabajo este en la nube, por ejemplo, la galería de imágenes que contiene las imágenes de la competición de LPA Trail 2020 está en los servidores de Google. Esto es una condición necesaria si queremos ejecutar los algoritmos de detección y re-identificación. Además, con la aplicación de sincronización hemos podido trabajar cómodamente, editando el código Python en el entorno de desarrollo Pycharm, sincronizando el código fuente con la nube de Google automáticamente, y ejecutándolo en Google Colab.

LaTeX y Overleaf

LaTeX consiste en un sistema de composición de textos muy utilizado en el ámbito científico y para la realización de trabajos de fin de grado. Fue creado

por Donald Knuth. La gran ventaja e inconveniente de este sistema es que se basa en instrucciones para la generación de los textos, muy diferente a otras alternativas como Microsoft Word que son de tipo lo que ves es lo que obtienes (WYSIWYG). Entonces, la gran ventaja de este sistema es que el usuario focaliza su esfuerzo en el contenido en vez del formato. Además, nos ofrece una gran cantidad de herramientas para representar fórmulas, ilustraciones, tablas, etc. Por otro lado, nos permite estructurar el trabajo en capítulos, secciones, bibliografía, índices, etc. En la mayoría de los trabajos científicos, se exige un determinado formato que con LaTex el sistema se encarga de cumplir con ese requerimiento. Como complemento, Overleaf es un servicio web basado en un editor online y colaborativo que usa LaTex.

Para el desarrollo de la presente memoria nos decantamos por estas herramientas porque te permiten escribir mucho contenido despreocupándote del formato necesario para el trabajo, ya que el sistema se encarga de darle el formato necesario. Otros motivos fueron la gran aceptación que tiene en el ámbito científico, la facilidad de referenciar (tablas, ilustraciones, secciones, etc) y citar bibliografías mediante BibTeX, y el ambiente colaborativo que nos permite que los tutores realicen correcciones a través de las anotaciones.

Git y GitHub

Git es un sistema de control de versiones gratuito diseñado por Linus Torvalds que nos permite gestionar un repositorio con el fin de almacenar nuestro proyecto. Este repositorio cuenta con un historial de cambios y almacena los cambios que realizamos en nuestros ficheros. Por tanto, podemos tener distintas versiones de una aplicación, estas versiones se pueden eliminar o restaurar. Además, Git cuenta con una arquitectura distribuida porque el repositorio se guarda de forma local en cada dispositivo. Con respecto a GitHub, es un servicio web que ofrece un repositorio remoto donde almacenar nuestros repositorios locales. También, nos permite trabajar de forma colaborativa.

Para la realización de este trabajo es necesario un sistema de control de ver-

siones como Git con GitHub, ya que se han tenido que implementar código propio, realizar modificaciones en los códigos existentes y en definitiva desarrollar una solución software que necesita de un historial de cambios. También, se ha utilizado este sistema porque todos los algoritmos desarrollados por la comunidad científica están almacenados en GitHub para su clonación y uso libre. El último motivo es por la necesidad de colaboración con los tutores, ya que es necesario un repositorio común donde todos los participantes del trabajo puedan visualizar el trabajo que se realiza en el código.

Gimp

Gimp es un programa de edición de imágenes gratuito que está disponible para los sistemas operativos Linux, Windows y MacOS. Para nuestro trabajo nos va a permitir editar imágenes para la memoria y las pruebas del código.

3.4. Estudio de las bibliotecas

Para realizar las líneas de trabajo que detallamos en la Sección 4.2 hemos utilizado algunas bibliotecas que citamos a continuación.

La primera es **OpenCV**, es una librería libre de visión artificial originalmente desarrollada por Intel, su significado es visión artificial abierta. Las ventajas de OpenCV son las siguientes, librería popular utilizada en muchos proyectos de visión artificial, amplia documentación para facilitar su uso, publicado bajo una licencia BSD, es decir, se puede utilizar para propósitos de investigación y comerciales, software libre, desarrollada en C++, como consecuencia provoca una gran eficiencia en su ejecución, multiplataforma, es decir, existe para MacOS, Android, Windows, iOS y GNU/Linux, y conectores para distintos lenguajes de programación como Java, Python, Javascript, MATLAB y Octave. Nosotros hemos utilizado el conector para Python. Principalmente, OpenCV es utilizado para el tratamiento de imágenes, en nuestro proyecto lo

utilizamos para leer, redimensionar o recortar las imágenes de los corredores almacenadas en disco. Sin embargo, OpenCV ofrece más funcionalidades como pueden ser el reconocimiento facial, gestual o de objetos. También, tiene funcionalidades de seguimiento, realidad aumentada, etc.

La segunda es **Scikit-learn**, es una biblioteca utilizada para aprendizaje automático y desarrollada para su uso en Python. Algunos de sus algoritmos son de clasificación, regresión, agrupamiento y reducción de dimensionalidad. Esta librería está implementada con Python, aunque también usa Cython en algunas partes de su código. Una de sus principales ventajas es su condición de código abierto. En nuestro proyecto, la utilizamos para el cálculo de las métricas en el proceso de re-identificación, esto nos permite conocer si nuestro sistema de re-identificación es fiable. Esta librería se suele utilizar bastante en la disciplina de ciencia de datos que está en auge actualmente.

La tercera es **NumPy**, es una librería que ofrece las herramientas necesarias para crear vectores o matrices de gran tamaño. Además, nos brinda una colección de funciones matemáticas para usar en distintos problemas, y es un software de código libre en el que participan una gran cantidad de colaboradores, por lo que está muy extendido. Esta librería está programada en Python, C y Fortran. En nuestro trabajo, es usada principalmente para realizar operaciones en los vectores y es indispensable para el funcionamiento de otros proyectos que utilizamos.

La cuarta es **TensorFlow**, librería de código abierto muy popular en el aprendizaje automático, una de sus funciones más importantes es la creación y entrenamiento de redes neuronales que se encargan de detectar y descifrar patrones. Su creador fue Google Brain Team. TensorFlow está diseñado principalmente para hacer uso de la CPU en la ejecución de sus algoritmos. Sin embargo, con otras extensiones como CUDA, es posible obtener los recursos necesarios de la GPU. CUDA es una plataforma de computación en paralelo, es decir, nos permite delegar la ejecución de ciertos hilos en la GPU en vez de la CPU. Esta plataforma aprovecha las ventajas que ofrecen las tarjetas gráficas, por ejemplo, ejecutar varios hilos independientes de forma eficiente cuando utilizamos redes neuronales. Su desarrollador principal es la empre-

sa Nvidia. En nuestro proyecto, TensorFlow es la base de los algoritmos de detección y re-identificación que utilizamos.

La quinta es **Keras**, una librería de código abierto utilizada para el uso de redes neuronales, está escrita en Python y suele ejecutarse sobre TensorFlow. En nuestro trabajo, se suele utilizar Keras y TensorFlow conjuntamente para ejecutar algunos de estos algoritmos.

3.5. Estudio de algoritmos basados en redes neuronales

La base de este trabajo se fundamenta en una serie de algoritmos basados en redes neuronales profundas. Estos algoritmos son reconocidos por la comunidad científica y han sido publicado en distintas revistas científicas de reconocido prestigio. En las dos líneas de trabajo que describimos en la Sección 4.2 hacemos uso de estos algoritmos que explicamos a continuación, citando previamente sus nombres: Yolov4-DeepSort [31], InsightFace [32, 33, 34, 35, 36, 37, 38], DeepFace [39] y AlignedReID [40, 41]. Los dos primeros se utilizan en tareas de detección de caras y mascarillas de los participantes de la competición. En cambio, los dos últimos su finalidad es la de cumplir con el objetivo de la re-identificación de corredores.

3.5.1. Yolov4-DeepSort

El algoritmo Yolov4-DeepSort[1] está desarrollado con TensorFlow, YOLOv4 y DeepSort, por tanto, a través de la colaboración de los distintos módulos podemos obtener un resultado similar al de la Figura 3.4. Por tanto, las principales funcionalidades de Yolov4-DeepSort son detectar y seguir objetos, en nuestro caso, de corredores.

Para comprender mejor el funcionamiento del algoritmo, estudiamos por se-

[1] https://github.com/theAIGuysCode/yolov4-deepsort

Figura 3.4: Resultado Yolov4-Deepsort Imagen obtenida desde [3].

parado cada módulo. El primer módulo es YOLOv4 [42], es el encargado de la detección de objetos, para ello, hace uso de redes neuronales profundas. Este detector está compuesto por varias partes, un modelo preentrenado, un componente que permite predecir clases y otro componente para calcular el contenedor (bounding box). Además, su implementación está basada en CSP-Darknet53 y YOLOv3, destacamos a su creador Joseph Redmon que permitió el desarrollo de este algoritmo de detección de objetos. El segundo módulo es DeepSort [43], su interés reside en que podemos seguir a varios objetos en tiempo real, inclusive si existen oclusiones.

Tras el estudio de los distintos módulos, comprendemos que Yolov4-DeepSort se basa en obtener el resultado de detección de YOLOv4 e ir transmitiendo a DeepSort las detecciones de objetos devueltas, con la finalidad de que realice un seguimiento. En nuestro trabajo, lo utilizamos para realizar las detecciones de los corredores y su seguimiento. Por tanto, obtenemos información en cada imagen del contenedor, identificadores y tipos de clases. Ya que este algoritmo puede detectar distintos tipos de objetos, como por ejemplo, personas, coches, etc. En la competición solamente nos interesan las personas, por lo tanto, en el algoritmo se establece que solo encuentre personas.

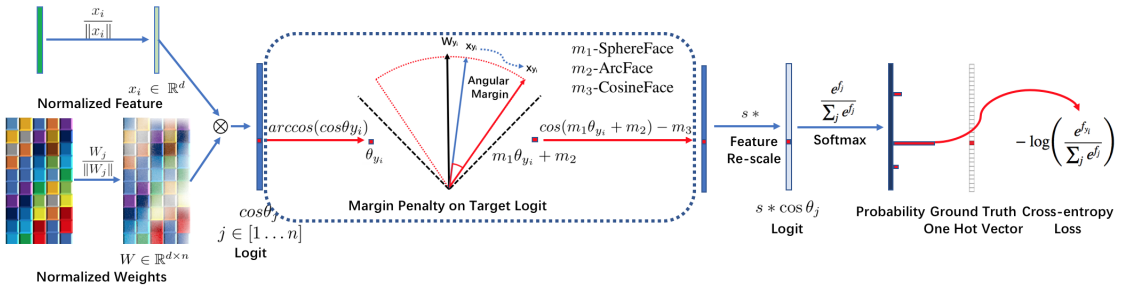

Figura 3.5: Funcionamiento ArcFace. Imagen obtenida desde [4].

3.5.2. InsightFace

Con respecto al algoritmo InsightFace[2], este consiste en un conjunto de herramientas que nos permiten realizar análisis de caras. Estas herramientas son DeepFaceRecognition, FaceDetection y FaceAlignment. Estas herramientas fueron desarrolladas por Jiankang Deng, Jia Guo, Niannan Xue y Stefanos Zafeiriou.

En el módulo de DeepFaceRecognition se proporciona conjuntos de entrenamiento, configuraciones de red y funciones de pérdida. Como conjuntos de entrenamiento ofrece las bases de datos MS1M, VGG2 y CASIA-Webface. Las configuraciones de red incluyen en la arquitectura a ResNet [22] , MobilefaceNet [44], MobileNet [28], InceptionResnetv2 [45], DenseNet [46], etc. Y por último, como funciones de pérdida a Softmax, SphereFace, CosineFace, ArcFace, SubCenter, ArcFace y Triplet. Entre estos, destacamos ArcFace [38], ya que fue elaborado por los autores de InsightFace y puede ser utilizado como función de pérdida para la detección de caras. La idea principal del funcionamiento de la pérdida de margen angular aditivo (ArcFace) se muestra en la Figura 3.5

El módulo de FaceDetection nos permite detectar caras y realizar un análisis de una imagen, como resultado nos ofrece información como la posición, ancho y alto del contenedor de la cara y cinco puntos que describen la posición de los ojos, nariz y comisuras de los labios. Tenemos que tener en cuenta que, este módulo se divide en dos submódulos, RetinaFace y RetinaFaceAntiCovid. Por

[2]https://github.com/deepinsight/insightface#retinafaceanticov

Figura 3.6: RetinaFace. Imagen obtenida desde [4].

un lado, RetinaFace nos ofrece conjuntos de datos de entrenamiento, código de entrenamiento, modelos preentrenados y scripts de evaluación, por tanto, el objetivo de este módulo es la detección de caras como mostramos en la Figura 3.6. Observamos que el resultado son los contenedores que delimitan todas las caras de los individuos. También, se contemplan los puntos de ojos, nariz y comisuras de labios.

Por otro lado, RetinaFaceAntiCovid es un submódulo experimental diseñado para la situación de la pandemia del COVID-19 que se basa en las mismas herramientas que RetinaFace, pero con algunas modificaciones para la detección de las mascarillas. Por tanto, nos detecta la cara y nos indica si llevamos o no mascarilla, en la Figura 3.7 mostramos un ejemplo.

Comprobamos que nos ofrece información acerca del contenedor de la cara y de los cinco puntos mencionados anteriormente como el otro módulo. Adicionalmente, nos indica con un cuadro rojo si llevamos mascarilla y con un cuadro verde si no la llevamos. Este algoritmo es utilizado en la línea de trabajo descrita en la Sección 4.2.2 para la detección de las mascarillas y las caras de los corredores.

El último módulo de InsightFace encontramos FaceAlignment, este módulo

Figura 3.7: RetinaFaceAntiCovid.

genera una malla de puntos en el rostro que describen los bordes de los ojos, nariz, rostro, boca y cejas. Con estos puntos conseguimos obtener información acerca de la forma de la cara de un individuo como vemos en la Figura 3.8.

3.5.3. DeepFace

La herramienta DeepFace[3] es utilizada principalmente para el reconocimiento de rostros y análisis de biometría blanda a partir de los rasgos faciales, como pueden ser el género, las emociones, la edad, aunque también nos da la opción de realizar tareas de re-identificación gracias a las utilidades que nos ofrece como complemento. Esta librería está basada principalmente en Keras y TensorFlow y está diseñada para Python. Los distintos modelos que utiliza esta herramienta son VGG-Face [47], FaceNet [48], OpenFace [49], DeepFace [50], DeepId [51], ArcFace [38] y Dlib [52].

Esta herramienta está compuesta por diversas utilidades que describimos a continuación, verificación facial, análisis facial de atributos, análisis en tiempo real y streaming, reconocimiento facial para aprendizaje embebido, API, detectores faciales y pila tecnología.

[3]https://github.com/serengil/deepface

Figura 3.8: FaceAlignment. Imagen obtenida desde [4].

La verificación facial de esta librería consiste en una función que nos permite analizar dos imágenes e indicarnos si es la misma persona. Esta verificación la realiza a través de los atributos de la cara del individuo, por tanto, previamente ha realizado una detección, alineamiento y representación del rostro. El resultado de esta utilidad lo mostramos en la Figura 3.9.

Podemos observar, como la herramienta nos indica que son la misma persona, para llegar a esta conclusión se basa en la distancia obtenida, cuanto menor sea la distancia, mayores similitudes existen en el rostro. En cambio, distancias mayores nos indican que existen más diferencias en el rostro. Para determinar, si es la misma persona o no, se debe de establecer un umbral a la distancia. En este caso, como la distancia está por debajo de 0,40 justificamos que es la misma persona. Esta distancia ha sido calculada utilizando el modelo VGG-Face y la métrica de similitud de la distancia coseno, existen otras métricas utilizadas en DeepFace como son la distancia euclídea y la euclídea L2. En nuestro trabajo, esta utilidad nos permite saber si un corredor es el mismo en distintos puntos de la competición haciendo uso de las distancias nombradas anteriormente.

Además, DeepFace te ofrece más funcionalidades como es el análisis de atributos faciales para determinar una serie de características de un individuo,

Figura 3.9: DeepFace. Imagen obtenida desde [5].

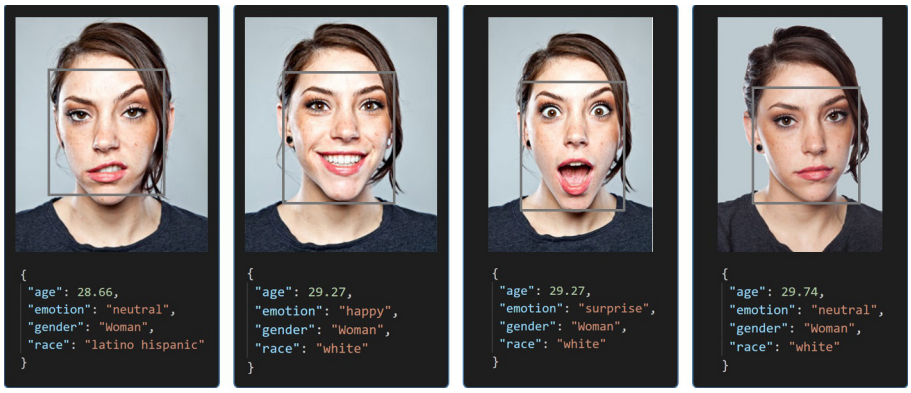

Figura 3.10: Análisis de atributos faciales. Imagen obtenida desde [5].

como pueden ser la edad, género, expresión facial y etnia. En las expresiones faciales es capaz de detectar si un individuo está enfadado, neutral, triste, asqueado, con miedo, feliz o sorprendido. En cambio, en la raza nos permite afirmar si es asiático, blanco, del medio oriente, indio, latino o negro. En la Figura 3.10 mostramos un ejemplo. Las estimaciones son: edad entre 28 a 30 años, estado de ánimo neutral, feliz, sorprendida y neutral, género mujer, y raza blanca aunque también considera que puede ser latina en una imagen.

Una de las características para mejorar el rendimiento del algoritmo es la posibilidad de guardar las caras ya reconocidas en una base de datos. De esta manera, conseguimos que los procesos de re-identificación o análisis de los atributos faciales puedan utilizar la imagen almacenada. Por tanto, el proce-

dimiento de detección del rostro solo se tiene que realizar una vez para cada imagen, ya que se guarda en la base de datos la imagen procesada. Esto nos permite verificar la imagen más veces con otros modelos sin volver a realizar la detección del rostro, en consecuencia, el rendimiento mejora considerablemente.

También, nos ofrece la opción de cambiar el detector por defecto MTCNN utilizado en la fase temprana del reconocimiento facial y usar otro detector como OpenCV, SSD, Dlib o RetinaFace.

Otra utilidad destacable de DeepFace es el streaming y análisis en tiempo real, es decir, podemos activar nuestra webcam y cada cinco fotogramas realiza un análisis facial para indicarnos el género, edad, expresión facial y raza. Por tanto, si antes realizábamos el análisis a través de imágenes que suministrábamos al sistema, ahora el sistema obtiene las imágenes del vídeo generado por la webcam en tiempo real para mostrarnos sobre esa propia visualización las características mencionadas anteriormente.

La mejora de la exactitud en las métricas se puede conseguir a través de un método de DeepFace, el reconocimiento del rostro ensamblando en el aprendizaje. Este método consiste en controlar por varios modelos y métricas de similitud la tarea de reconocimiento facial, para mejorar la exactitud de las métricas. El inconveniente radica en que el tiempo de ejecución del algoritmo de reconocimiento facial es mayor que utilizando un único modelo.

Para el uso de DeepFace de forma remota podemos generar una API. DeepFace nos ofrece la opción de crear un servicio REST, para ello, nos ofrece un fragmento de código para crear dicho servicio. Por tanto, solamente tendremos que consumirlo a través de un método http post como http://127.0.0.1:5000/verify. Esto nos permite realizar un reconocimiento facial a través de la red, las imágenes deben de ser pasadas como una string codificada en base64.

Por último, una utilidad muy importante para nuestro proyecto es la representación facial de imágenes como vectores, que consiste en una utilidad de DeepFace que nos devuelve el vector de características de la imagen facial.

De esta forma, no tenemos que obtener dicho vector continuamente para cada prueba que realizamos. Este vector puede ser almacenado en una base de datos o en ficheros. En nuestro caso, almacenamos para cada imagen un vector de características, permitiendo reducir considerablemente los tiempos a la hora de probar distintos modelos. Por tanto, no tenemos que volver a realizar un reconocimiento facial completo para obtener dicho vector, sino leemos la información del vector almacenado en disco.

En nuestro proyecto, DeepFace es utilizado para la re-identificación de corredores en distintos puntos de la competición. Para ello, utilizamos las utilidades de la verificación facial para obtener las distancias y determinar si dos imágenes en diferentes puntos representan al mismo corredor. Además, para mejorar el rendimiento utilizamos la representación facial de imágenes como vectores.

3.5.4. AlignedReID

Otro método que utilizamos para la re-identificación de los competidores es AlignedReID[4]. Este algoritmo surge como consecuencia de los problemas de re-identificación derivados de parejas de imágenes no alineadas. Por tanto, pretende a través de su método información local enlazada dinámicamente (DMLI) alinear información local sin requerir una supervisión extra. La ventaja de este método es que logra un mejor rendimiento en comparación a sus competidores en los casos que la pose de la persona no está alineada, normalmente ocasionado por inexactitud en la detección de personas y trazado de los contenedores. La esencia de este método radica en el DMLI, que explicamos a continuación haciendo uso de la Figura 3.11.

Existen dos tipos de distancias, la distancia alineada y la distancia original. La distancia alineada es la distancia local después del DMLI mientras que la distancia original es la distancia local sin el alineamiento. Por tanto, un valor menor de la distancia nos indica una mayor posibilidad de que sean la misma persona. En los pares de imágenes analizados, comprobamos como el par de la izquierda existe una gran diferencia entre la distancia alineada y la distancia

[4]https://github.com/michuanhaohao/AlignedReID

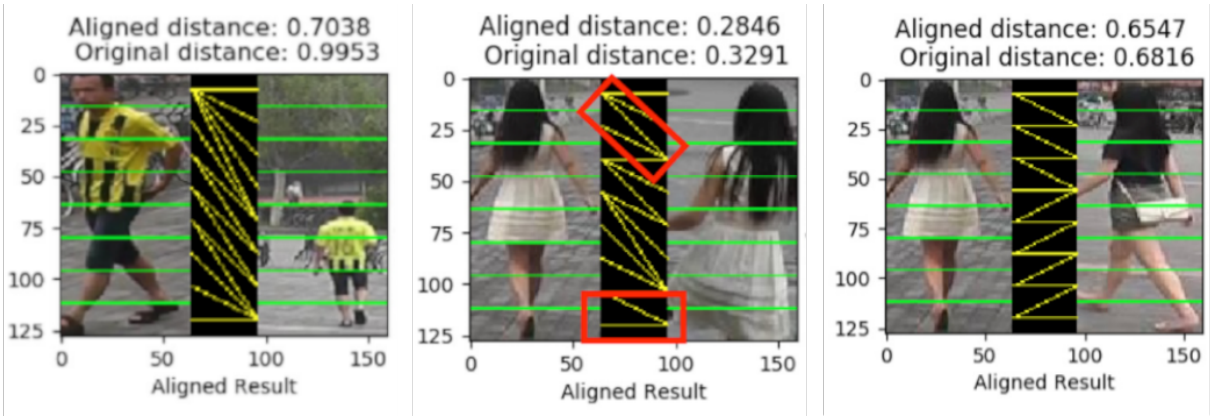

Figura 3.11: AlignedReID. Imagen obtenida desde [6].

original, 0,7038 y 0,9953 respectivamente. Esto nos demuestra que el método ante diferencias de altura de la persona con respecto a la imagen funciona aceptablemente, ya que la distancia alineada es menor que la distancia original, es decir, al considerar la información local alineada conseguimos mejorar la distancia. Entonces, como son la misma persona se comprueba fácilmente que el método está funcionando correctamente. En el par central, el método es eficaz ante cambios en la orientación vertical de la persona, el desajuste ocasionado por la inclinación a la derecha de la individuada no provoca que la distancia alineada (0,2846) sea mayor valor que la distancia original (0,3291). Por tanto, al tratarse de la misma persona, la distancia obtenida después del DMLI es más fiable que sin el DMLI. Por último, el par de la derecha nos demuestra que ante imágenes sin diferencias ante la altura o orientación vertical del individuo, ambas distancias son muy similares, distancia alineada de 0,6547 y distancia original de 0,6816. Este hecho es esperado, ya que el DMLI surge para solucionar los problemas de imágenes no alineadas, y en este caso, están alineadas.

En la tarea de re-identificación existe una diferencia destacable entre DeepFace y AlignedReID, mientras que el primero se basa en las características faciales para calcular las correspondientes distancias que nos permiten determinar el grado de similitud del individuo, en AlignedReID se utilizan las características de los cuerpos del individuo, no solamente de la cara.

Esta utilidad nos ofrece una serie de modelos entrenados para realizar la re-identificación, como pueden ser Cuhk03-Resnet50, DukeMTMCReID-Resnet50,

Market1501-Resnet50 y MSMT17-Resnet50.

3.6. Estudio de métricas de detección de mascarillas y re-identificación de corredores

Hemos estudiado una serie de métricas para determinar cuál es la más apropiada a la hora de llegar a una respuesta sobre nuestro objetivo del trabajo de fin de grado. Entonces, nos hemos decantado para la detección de mascarillas por la métrica de la exactitud, mientras que para la re-identificación por la mAP (acrónimo en inglés de media de la precisión promedio) y el rank 1.

La **exactitud** es el ratio entre las predicciones correctas y las predicciones totales como mostramos en la Ecuación 3.1.

$$exactitud = \frac{PC}{PT} \tag{3.1}$$

PC = predicciones correctas
PT = predicciones totales

Las predicciones correctas están formadas por los verdaderos positivos y los verdaderos negativos. Un verdadero positivo es un corredor que realmente lleva mascarilla y el algoritmo dicta que lleva mascarilla. En cambio, un verdadero negativo es un corredor que no lleva mascarilla y nuestro método nos confirma que no la lleva. Por otro lado, las predicciones totales se definen como los dos verdaderos anteriores, los falsos positivos y los falsos negativos, es decir, los casos fallidos del algoritmo. El falso positivo, se da cuando el corredor no lleva mascarilla y la predicción indica que si la lleva. Mientras que el falso negativo, el participante lleva mascarilla y la predicción indica que no. Cuanto más cercano sea el valor a 1, más muestras acierta el algoritmo. En cambio, valores más lejanos a 1, indican una mayor tasa de fallos. En nuestro trabajo, hacemos uso de la librería sklearn con su método sklearn.metrics.accuracy_score para calcular la exactitud, esta librería desa-

rrolla la Ecuación 3.1 transformándola en la Ecuación 3.2.

$$exactitud(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} 1(y_i = \hat{y}_i) \tag{3.2}$$

y = valores reales
$\hat{y}$ = predicciones
n = número de muestras

Este método puede ser utilizado para clasificar dos clases o varias, en nuestro caso, solamente clasificamos dos, es decir, si llevan mascarilla o no llevan mascarilla, dando valores de 1 o 0 respectivamente. En la Sección 4.2.2 detallamos la integración de esta métrica en nuestro problema.

La **media de la precisión promedio** (mAP) es una media que se calcula sobre las precisiones medias (AP) de todas las clases [7] como indica la Ecuación 3.3.

$$mAP = \frac{1}{n} \sum_{i=1}^{n} AP_i \tag{3.3}$$

n = numero total de precisiones medias calculadas

Donde AP es el área bajo la curva precision-recall desde $r = 0$ a $r = 1$ como se indica en la Ecuación 3.4.

$$AP = \int_{r=0}^{r=1} p(r)dr \tag{3.4}$$

Esta curva de precision-recall ($p(r)$) de forma discreta se calcula hallando los precision y recall de cada contenedor predicho. La precision es el ratio entre verdaderos positivos y los verdaderos positivos más falsos positivos como indica la Ecuación 3.5. En cambio, el recall es el ratio entre verdaderos positivos y verdaderos positivos más falsos negativos como mostramos a continuación

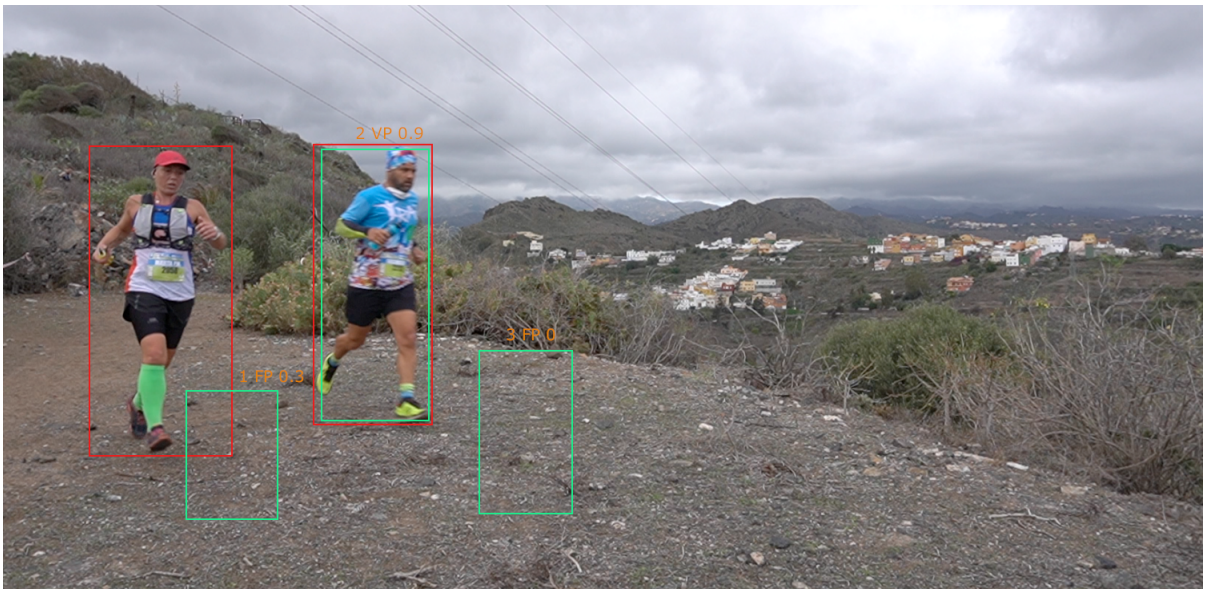

Figura 3.12: Imagen con verdaderos y falsos positivos.

en la Ecuación 3.6.

$$precision = \frac{VP}{VP + FP} \tag{3.5}$$

$$recall = \frac{VP}{VP + FN} \tag{3.6}$$

En la Figura 3.12 mostramos una serie de imágenes con las situaciones de verdaderos y falsos positivos.

El cuadro rojo representa el contenedor real y el cuadro verde el contenedor predicho. El primer valor de cada contenedor predicho es su identificador, el segundo valor el tipo de positivo, y el tercer valor la intersección a través de la unión (IoU), es decir, la probabilidad de que un objeto esté en el contenedor de la predicción, en la Figura 3.13 se representa su definición. En nuestro ejemplo, un verdadero positivo es cuando el IoU $>$ 0,5, en caso contrario, es un falso positivo. Existen otras situaciones que no contemplamos para este ejemplo.

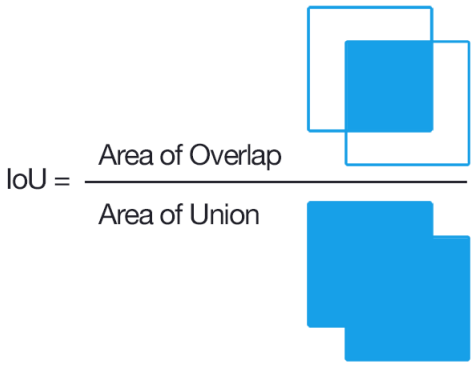

Figura 3.13: Intersección a través de la unión (IoU). Imagen obtenida desde [7].

Contenedor predicho	IoU	TP o FP	Precision	Recall
1	0,9	TP	1/1	1/1
2	0,3	FP	1/2	1/1
3	0	FP	1/3	1/1

Tabla 3.2: Ejemplo cálculo precision y recall.

Entonces, para trazar la curva precisión-recall ordenamos por IoU las predicciones y calculamos los ratios anteriores como mostramos en la Tabla 3.2.

Finalmente, con estos valores podemos trazar la curva de precisión-recall, siendo el eje x el recall y el eje y la precisión, con el fin de hallar el área bajo la curva para obtener el AP de la clase persona. Si existen más tipos de clases en las predicciones se deberían de incluir las otras clases en el cálculo de la media. En la Sección 4.2.3 integramos mAP en nuestro proyecto haciendo uso del método sklearn.metrics.average_precision_score de la librería sklearn que se apoya en las distancias calculadas, pero cuyo fundamento es el anterior.

El **rank 1** es el ratio entre el número de imágenes de la consulta (término del inglés query), cuya mejor predicción de la distancia corresponde con el mismo corredor, y el número de imágenes del conjunto referencia. Por tanto, los pasos para calcular el rank 1 son:

1. Calcular la distancia de cada imagen referencia con el resto de imágenes de la galería con distinta ubicación.

2. Si el corredor de la imagen referencia es el mismo que el corredor comparado con la menor distancia, entonces contamos dicha imagen referencia en nuestro ratio, en caso contrario, no la contamos.

3. Repetir los pasos 1 y 2 con el resto de imágenes del conjunto referencia.

En definitiva, el rank 1 nos determina el porcentaje de imágenes del conjunto referencia cuya mejor re-identificación corresponde al mismo corredor. En la Sección 4.2.3 integramos el rank 1 en nuestro proyecto.

Capítulo 4

Metodología e implementación

4.1. Metodología

La estrategia llevada a cabo para alcanzar nuestro objetivo de comprobar si el uso de mascarilla afecta a los métodos de detección y re-identificación utilizando como escenario un evento deportivo, se basa en una metodología basada en prototipos ya que podemos implementar distintos prototipos de forma rápida, con el fin de probar, revisar y mejorarlo. La metodología basada en prototipos se divide en las siguientes fases como mostramos en la Figura 4.1:

Identificación de requisitos básicos

Realizamos una búsqueda en la bibliografía existente para determinar que algoritmos son apropiados para nuestro problema, como son los algoritmos de Yolov4-DeepSort e InsightFace para la detección de mascarillas y DeepFace y AlignedReID para la re-identificación de corredores. También, determinamos que requisitos son necesarias en nuestro equipo para ejecutar estos algoritmos, por ejemplo, una gráfica Nvidia de altas prestaciones. Por último, consideramos que arquitectura nos interesa utilizar en este proyecto, decantándonos por una arquitectura en la nube de Google, en la que todos los ficheros, modelos

Figura 4.1: Metodología de prototipado.

e imágenes estarán almacenados en un servidor.

Desarrollo del prototipo

Con la información disponible en los repositorios remotos de los distintos algoritmos se configura el entorno de trabajo y se realizan las modificaciones necesarias para adaptar los algoritmos a nuestro contexto. Además, se realizan unas pruebas con muestras pequeñas para probar que los algoritmos cumplen su cometido, para ello, se elige algún modelo entrenado. Estas pruebas pueden consistir en detectar la mascarilla de un solo corredor o re-identificar un par de corredores en distintos lugares.

Evaluación

Se desarrolla una serie de métricas para determinar si los resultados son los esperados y para cuantificar el rendimiento de los algoritmos. Por ejemplo,

en la detección de mascarillas se mide la exactitud, mientras que en la re-identificación se evalúa la mAP y el rank 1. De esta forma, podemos evaluar si los algoritmos funcionan correctamente y alcanzar una conclusión en este trabajo.

Revisión y mejora del prototipo

Revisamos y mejoramos el prototipo hasta alcanzar el que nos permita concluir acerca de nuestro objetivo. Para ello, aumentamos la muestra de corredores y realizamos modificaciones en el algoritmo para contemplar distintas restricciones o limitaciones que son necesarias. Por ejemplo, limitar el tamaño mínimo de la detección. También en el proceso de mejora, corregimos errores o mejoramos el código para que las métricas sean más fiables.

4.2. Implementación

En el presente trabajo existen dos líneas de trabajo fundamentales, una para el procedimiento de detección de mascarillas en los corredores y otra para la re-identificación de competidores en diferentes puntos de la competición. Por tanto, a continuación explicamos el desarrollo por separado de estas líneas de trabajo que conllevan la implementación de una solución software para conseguir nuestro objetivo, explicando primero el procedimiento inicial y común a estas dos vertientes. En la Figura 4.2 mostramos un diagrama de trabajo del procedimiento común y de las líneas de trabajo.

Esta solución software la podemos encontrar en el repositorio de GitHub de "Re-identificación basada en caras en tiempos de pandemia"[1] de mi perfil. A continuación, explicamos algunos detalles de la implementación y su funcionamiento, pero si queremos visualizar el código completo de los scripts desarrollados en Python para la detección y re-identificación, la aplicación Java de etiquetado o el cuaderno de Jupyter Notebook utilizado para ejecutar los algoritmos en Google Colab, los podemos encontrar en el repositorio

[1] https://github.com/ignaciomarinreyes/Re-identificacion_basada_en_caras_en_tiempos_de_pandemia

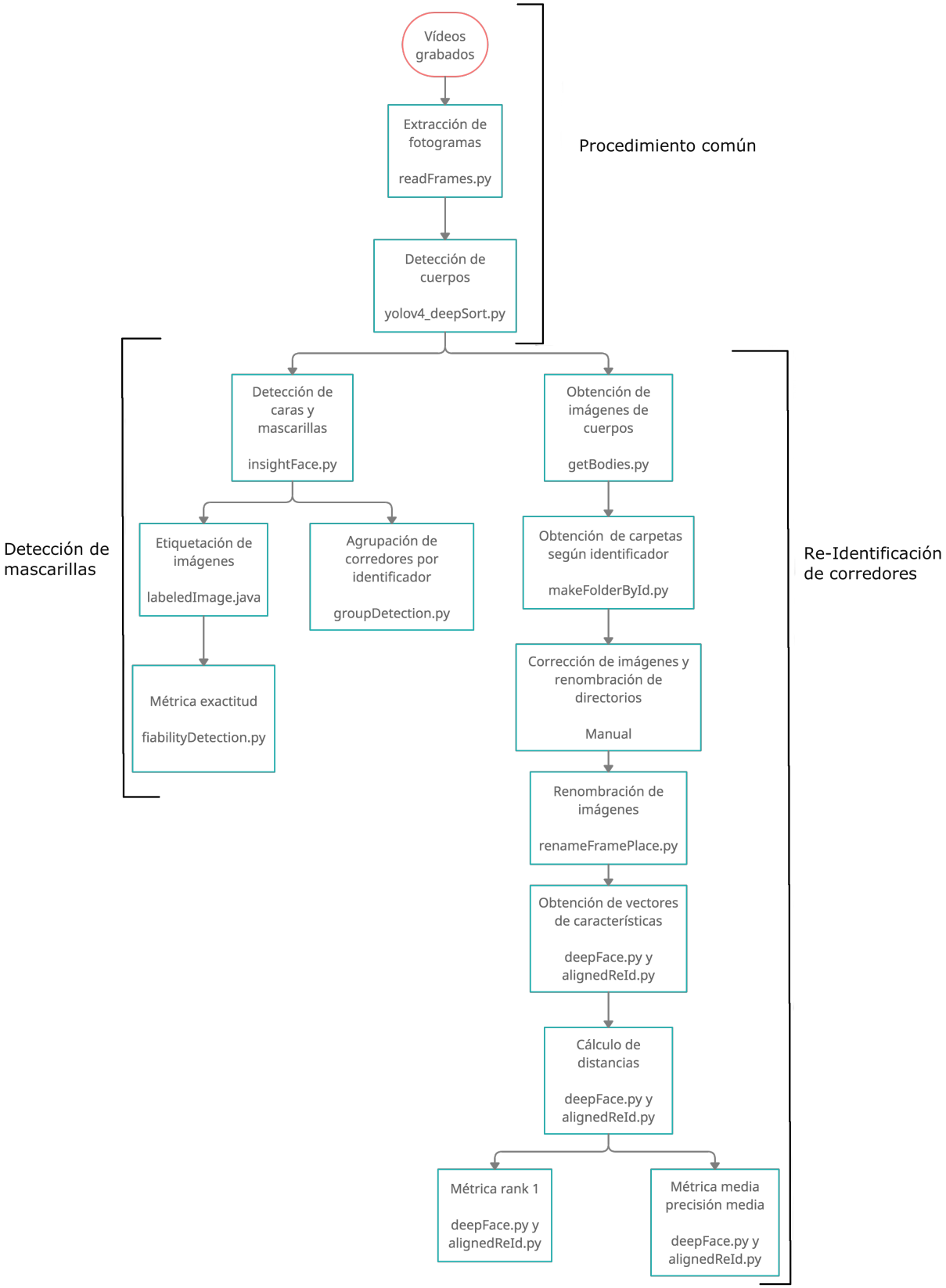

Figura 4.2: Diagrama de trabajo.

anterior.

4.2.1. Procedimientos comunes

Comenzamos procesando los vídeos de la competición de LPA Trail 2020, para ello, usamos la script *readFrames.py* que albergamos en el repositorio, en la que enfatizamos la función del Algoritmo 4.1.

Este código se encarga de obtener los fotogramas de los vídeos de la competición. Debemos de tener en cuenta que las videocámaras graban con una tasa de 50 FPS (fotogramas por segundo). Por tanto, si para nuestro proyecto queremos 5 FPS, calculamos un incremento por fotogramas de 10 fotogramas, es decir, cada 10 fotogramas obtenemos una imagen del vídeo que almacenamos en disco con su correspondiente formato, por ejemplo: Salida frame_10_13_49_400.jpg, cada 200 milisegundos obtenemos una imagen, 1 segundo son 1000 milisegundos y 1000/5 son 200 milisegundos por fotograma.

Después de obtener todas las imágenes de todos los vídeos de la competición, ejecutamos el algoritmo Yolov4-DeepSort del script *yolov4.py*. Su función es la de detectar, seguir, etiquetar y asignar un identificador a los cuerpos de los corredores como mostramos en la Figura 4.3.

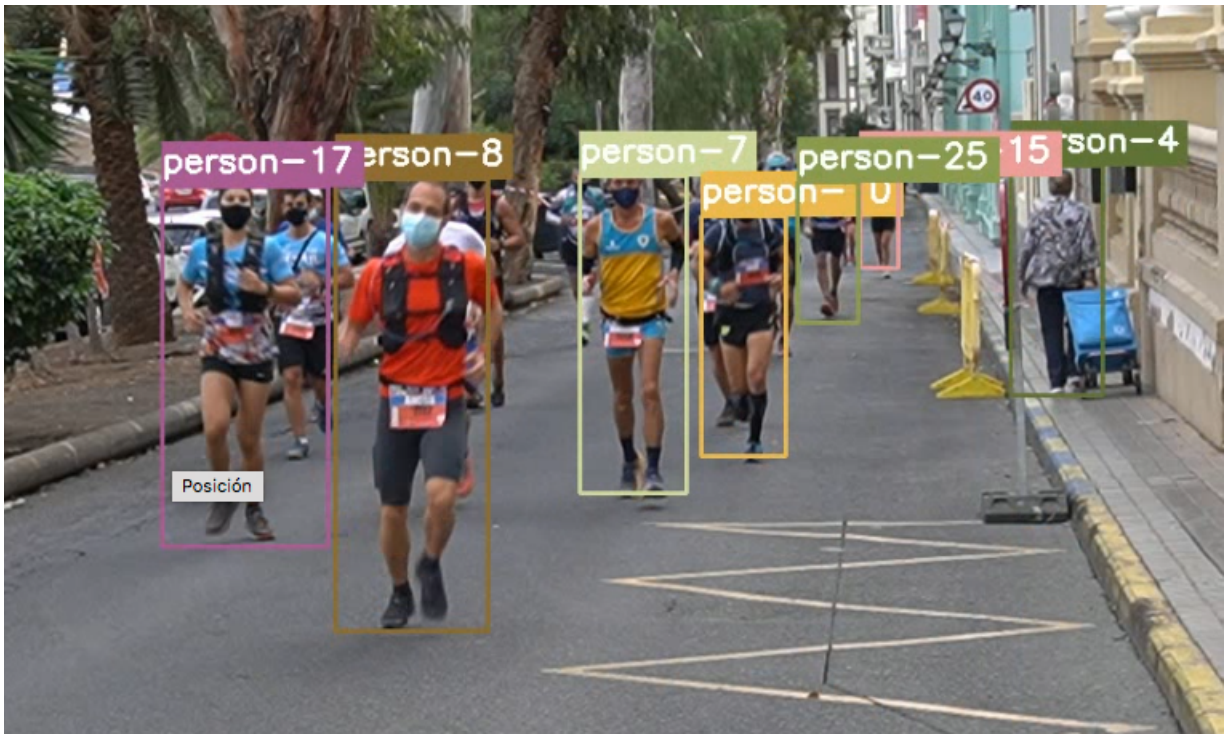

Figura 4.3: Yolov4_DeepSort, LPA Trail 2020 Vegueta.

```
def extract_frames(videofile, frames_folder, start_time, num_frames, frames_per_sec, frame_prefix):
    vid = cv2.VideoCapture(videofile)
    if 50 % frames_per_sec != 0:
        print('ERROR: Number of frames per second must be divisor of 50')
        return False
    inc_frames = int(50 / frames_per_sec)
    frame_time = start_time
    i = 0
    while i < num_frames:
        ret, frame = vid.read()
        if not ret:
            print('ERROR reading video {}'.format(videofile))
            return False
        if i % inc_frames == 0:
            frame_name = '{}frame_{}_{:03d}.jpg'.format(frame_prefix,
                                                        frame_time.strftime('%H_%M_%S'),
                                                        int(frame_time.microsecond / 1000))
            print(os.path.join(frames_folder, frame_name))
            cv2.imwrite(os.path.join(frames_folder, frame_name), frame)
        frame_time = frame_time + datetime.timedelta(hours=0, minutes=0, seconds=0.02, milliseconds=0)
        i = i + 1
    return True
```

Algoritmo 4.1: Código creación de fotogramas

Observamos que el algoritmo es capaz de detectar los cuerpos de los corredores, crear unos contenedores que nos ofrecen información acerca de la posición y tamaño de los cuerpos en la imagen, etiquetar el cuerpo como una persona, en nuestro caso, tenemos limitado el algoritmo para que solo detecte personas, ya que solo nos interesa detectar corredores, pero puede detectar otro tipo de clases, y asignar un identificador que no es único. La no unicidad del identificador generado por Yolov4-DeepSort nos genera un problema en la re-identificación. Principalmente, porque las métricas del rank 1 y mAP serían erróneas, para usar estas métricas es condición necesaria que los identificadores sean únicos en cada vídeo. Este problema, la lectura de las imágenes y la escritura en ficheros lo solucionamos adaptando el algoritmo Yolov4-DeepSort.

Por tanto, como mostramos en el Algoritmo 4.2, se comienza leyendo las imágenes de disco y a cada imagen se le ejecuta un conjunto de instrucciones que dan como resultado varios contenedores correspondientes a cada corredor de la imagen, hemos ocultado un gran número de instrucciones para explicarlo en la memoria. Los contenedores son guardados por el algoritmo en la variable *bbox*, donde se almacena la información de dicho contenedor.

```
for pathJpg in sorted(glob.glob(path + "*.jpg")):
        listIdImagen = []
        init = pathJpg.find("Salida_frame_") + 13
        timeJpg = pathJpg[init: init + 12]
        frame = cv2.imread(pathJpg)
        .
        . # Code Lines
        .
        bbox = track.to_tlbr()
        .
        . # Code Lines
        .
        if heightBox > 250:
            newId =""
            if str(track.track_id) in dIncrement:
                newId = str(int(dIncrement[str(track.track_id)]) + int(track.track_id))
            else:
                newId = str(track.track_id)
            listIdImagen.append(newId)
            dNorepetition[newId] = 0
            fileOutput.write(str(newId) + " " +
                             ("1" if int(bbox[0]) < 0 else str(int(bbox[0]))) + " " +
                             ("1" if int(bbox[1]) < 0 else str(int(bbox[1]))) + " " +
                             str(widthBox) + " " + str(heightBox) + " \n")
        .
        . # Code Lines
```

```
    .
for newId in dNorepetition:
    if newId not in listIdImagen:
        dNorepetition[newId] = dNorepetition[newId] + 1
        if dNorepetition[newId] == umbral:
            oldId = str(int(newId) % 1000)
            if oldId not in dIncrement:
                dIncrement[oldId] = 0
            dIncrement[oldId] = dIncrement[oldId] + 1000
```

Algoritmo 4.2: Código Yolov4_DeepSort

Esta información es la posición, ancho y alto del contenedor, e identificador del corredor. Por tanto, almacenamos esta información en un fichero de texto, donde cada fila del fichero de texto representa los datos de un corredor detectado por el algoritmo. Por ejemplo: ”id x y ancho alto”. Y cada fichero de texto corresponde con una imagen analizada. Este proceso se repite reiteradamente para cada imagen hasta que se analizan todas las imágenes del vídeo. Además, establecemos una restricción al alto del contenedor de 250 píxeles, esto lo realizamos para descartar todas las detecciones que no corresponden con personas y para elegir a los corredores que están cerca del foco de la cámara, ya que los más próximos tendrán un contenedor mayor y una mejor calidad de imagen, condición indispensable para detecciones de caras y re-identificaciones correctas. Recordemos, que el identificador de *bbox* no es único, por lo que se realiza el procedimiento del Algoritmo 4.2 para convertirlo en único. El procedimiento consiste en una serie de arrays asociativos que actuando conjuntamente provocan que cuando Yolov-DeepSort devuelva un identificador repetido, este identificador se incrementa una cantidad múltiplo de 1000. Por tanto, existe un array asociativo encargado de almacenar los identificadores de todos los corredores para comprobar si se repite en un futuro, y otro array asociativo encargado de gestionar la cantidad que se incrementa el identificador cuando se vaya a repetir, de tal manera, que no se repita el incremento para cada identificador generado por el algoritmo. Por último, existe otro aspecto importante a destacar, la adicción de un umbral que controla el número de fotogramas seguidos que consideramos que el identificador debe ser el mismo. Esto es debido a que un corredor aparece en fotogramas seguidos, y no nos interesa que el algoritmo cambie el identificador por ser repetido, ya que es el mismo corredor. Por tanto, establecemos un umbral que provoca que si un corredor no se detecta en un número de foto-

gramas seguidos superior al umbral, se genere un nuevo identificador cuando se vuelva a repetir, ya que probablemente es otro corredor. El valor establecido no puede ser ni muy alto ni muy bajo, sino un valor alcanzado a través de la experimentación, valores altos provocan que distintos corredores tengan el mismo identificador, y valores bajos que un corredor tenga más de un identificador. A través de pruebas hemos establecido el valor de 5, es decir, permisividad de 5 fotogramas seguidos.

A continuación, explicamos la línea de detección de mascarillas.

4.2.2. Detección de mascarillas

En la línea de detección de mascarillas se realizan los siguientes pasos y se usa el algoritmo InsightFace.

Como hemos utilizado Yolov4-DeepSort en el paso anterior, ya tenemos un conjunto de ficheros para cada imagen cuyo contenido alberga en cada fila el identificador del corredor y el contenedor del cuerpo. Entonces, con esta información utilizamos el algoritmo *insightface.py*. que visualizamos en el Algoritmo 4.3 destacando las líneas más importantes y ocultando el resto.

```
detector = RetinaFaceCoV("/content/gdrive/My
    Drive/TFG/insightface/detection/RetinaFaceAntiCov/model/mnet_cov2", 0, gpuid, 'net3l')
for pathTxtBody, pathPng in zip(sorted(glob.glob(path + "*bodies.txt")), sorted(glob.glob(path + "*.jpg"))):
    .
    . # Code Lines
    .
    fileBody = open(pathTxtBody)
    img = cv2.imread(pathPng)
    fileOutput = open(path + "Salida_frame_" + timeFile + "_faces.txt", "w")
    for line in fileBody:
        values = line.split(" ")
        beginXBody, endXBody, beginYBody, endYBody = int(values[1]), int(values[1]) + int(values[3]),
            int(values[2]), int(
            values[2]) + int(values[4])
        heihtImageBody = int(endYBody) - int(beginYBody)
        endYBodyHead = int(beginYBody) + int((1 / 3 * heihtImageBody))
        crop_imgBodyHead = img[beginYBody:endYBodyHead, beginXBody:endXBody]
        .
        . # Code Lines
        .
```

```
faces, landmarks = detector.detect(crop_imgBodyHead,
                                   thresh,
                                   scales=scales,
                                   do_flip=flip)
    .
    . # Code Lines
    .
    if faces.shape[0] == 0:
        fileOutput.write(values[0] + " ND ND ND ND ND ND ND ND ND ND ND ND ND ND ND ND \n")
    else:
        areaBoxFace = []
        for i in range(faces.shape[0]):
            face = faces[i]
            boxFace = face[0:4].astype(np.int)
            widthXBoxFace = boxFace[2] - boxFace[0]
            heightYBoxFace = boxFace[3] - boxFace[1]
            areaBoxFaceValue = widthXBoxFace * heightYBoxFace
            areaBoxFace.append(areaBoxFaceValue)
        maxValue = max(areaBoxFace)
        j = areaBoxFace.index(maxValue)
        landmark5 = landmarks[j].astype(np.int)
        faceMax = faces[j]
        .
        . # Code Lines
        .
        fileOutput.write(str(values[0]) + " " + str(colorBox) + " " + str(beginXBoxFace) + " " +
             str(beginYBoxFace) + " " + str(widthXBoxFace) + " " + str(heightYBoxFace) + "\n")
.
. # Code Lines
.
```

Algoritmo 4.3: Código InsightFace

Consiste en un fichero python que haciendo uso de un detector de InsightFace, concretamente el del modelo 'mnet_cov2', nos permite detectar la cara de un corredor y determinar si lleva o no mascarilla. Primero, leemos todos los ficheros creados por Yolov4-DeepSort para obtener los contenedores de los cuerpos. Segundo, aplicamos el detector solamente dentro de los contenedores y al tercio superior del contenedor que corresponde con la cara, para ello, leemos la imagen completa y con la información del fichero generamos en memoria una imagen cortada ($crop\_imgBodyHead$) según lo indicado por el contenedor y obteniendo el tercio superior. Es importante aplicárselo solamente a los cuerpos, ya que si intentamos detectar una cara en la imagen completa, en la mayoría de los casos detectaría caras erróneamente o no detectaría ninguna cara, ya que es mucho más fácil para el algoritmo encontrar una cara en un cuerpo que en la imagen completa. Por otro lado, solo hay

Figura 4.4: Casos de error InsightFace.

Id	Detección Mascarilla	x Cara	y Cara	Ancho Cara	Alto Cara
1007	1	840	312	41	61
13	0	922	355	29	39

Tabla 4.1: Fichero de caras.

que aplicárselo a la parte superior del cuerpo porque en muchos casos el algoritmo detecta el dorsal del corredor como si fuera una cara, en la Figura 4.4 mostramos un ejemplo de los dos casos anteriores que generan fallos y en la Figura 4.5 un resumen del procedimiento realizado.

Finalmente, llamando a la función *detector.detect(...)* ejecutamos el detector de InsightFace que nos devuelve el contenedor de la cara y un indicador de si lleva o no mascarilla. Esta información junto con los identificadores y contenedores que nos ofrecía Yolov4-DeepSort son almacenados en un nuevo fichero que representamos en la Tabla 4.1. Si lleva mascarilla se representa con un 1 y en la imagen con un cuadrado rojo. En cambio, si no lleva mascarilla indicamos un 0 y un cuadrado verde en la imagen. Además, en los casos que no se puede detectar ninguna cara en el área seleccionada, se asigna toda la fila con no disponible (ND) menos el identificador.

Un problema que hemos solucionado es el siguiente, en el proceso de selección del cuerpo y aplicación del algoritmo InsightFace, comprobamos que existe la

Figura 4.5: Procedimiento InsightFace.

Figura 4.6: Error detección de varias caras.

posibilidad de que existan varias caras en el área seleccionada, provocando un error ya que al seleccionar otro cuerpo distinto podríamos detectar la misma cara debido a la duplicidad de caras anteriores como mostramos en la Figura 4.6.

La solución consiste en usar una heurística que selecciona la cara de cada área seleccionada cuya área de la cara es mayor, descartando el resto de caras como mostramos en el Algoritmo 4.3. De esta manera, solamente tenemos una cara por área seleccionada. No es 100 % efectivo pero nos permite corregir la mayoría de estos errores, ya que el corredor más próximo al objetivo de la cámara suele ser el que tiene el área de la cara mayor, y dicho corredor es el que nos interesa.

La métrica de la exactitud la implementamos haciendo uso de la librería sklearn, por tanto, desarrollamos el fichero *fiabilityDetection.py* que alberga tres funciones: *accuracyWearOrNotMask()*, *accuracyWearMask()* y *accuracyNotWearMask()*. Con estas funciones calculamos la precisión del algoritmo InsightFace en distintos grupos de corredores, todos los corredores independientemente del uso de mascarilla, corredores que llevan mascarilla y corredores sin mascarilla. Además, mostramos la matriz de confusión para visualizar mejor los resultados en cada grupo. A continuación, mostramos un ejemplo de la función *accuracyWearOrNotMask()* en el Algoritmo 4.4, ya que las demás funciones son muy similares y únicamente cambia el grupo elegido.

```
def accuracyWearOrNotMask():
    print("========WearOrNotMask========")
    vPred = []
    vTrue = []
    for pathTxtResult, pathTxtTagged in zip(sorted(glob.glob(path + "*result.txt")), sorted(glob.glob(path +
        "*tagged.txt"))):
        fileResult = open(pathTxtResult)
        fileTagged = open(pathTxtTagged)
        for lineResult, lineTagged in zip(fileResult, fileTagged):
```

```
            valuesLineResult = lineResult.split(" ")
            valuesLineTagged = lineTagged.split(" ")
            if valuesLineTagged[4] != "ND":
                vTrue.append(int(valuesLineTagged[4]))
                vPred.append(int(valuesLineResult[4]))
    print("======Metrics=====")
    print(mt.accuracy_score(vTrue, vPred))
    print("===Confusion matrix =====")
    print(mt.confusion_matrix(vTrue,vPred))
```

Algoritmo 4.4: Código métrica exactitud

Para utilizar la función *mt.accuracy_score(...)* y *mt.confusion_matrix(...)* de la librería sklearn que nos permite calcular la exactitud y la matriz de confusión respectivamente, debemos de construir los parámetros necesarios, estos son una lista vPred, compuesta por las etiquetas predichas por InsightFace, y una lista vTrue, formada por las etiquetas de la verdad fundamental.

La lista vPred se construye leyendo cada uno de los ficheros que generamos con *insightFace.py*, ya que contienen la información predicha del uso de mascarilla de cada corredor. En cambio, para formar la lista vTrue necesitamos realizar un procedimiento manual de etiquetado, para agilizar el etiquetado de tantas imágenes, hemos desarrollado una aplicación Maven en Java que nos etiqueta las imágenes y almacena el resultado en ficheros. Ese procedimiento de etiquetado consiste en asignar a todas las detecciones de corredores que realizo Yolov4-DeepSort, un 1 si visualmente comprobamos que un corredor lleva mascarilla, un 0 en caso contrario, y ND si no existe ninguna cara. Esta información la almacenamos en ficheros para cada imagen.

Esta aplicación se llama LabeledImage y consiste en un visor de imágenes que nos permite ir etiquetando imágenes al pulsar una serie de teclas, en la Figura 4.7 vemos la interfaz de usuario de la aplicación desarrollada con Swing.

El uso es muy sencillo ya que se basa en seguir una secuencia de teclas, es decir, aparece la imagen a etiquetar y se dibuja un cuadro rojo en la posible cara del corredor, decidimos visualmente si lleva o no mascarilla, pulsamos la tecla 'A' si no lleva mascarilla, la tecla 'S' si lleva mascarilla o la tecla 'D' si la detección de la cara no es correcta. Si existe otro corredor detectado en la

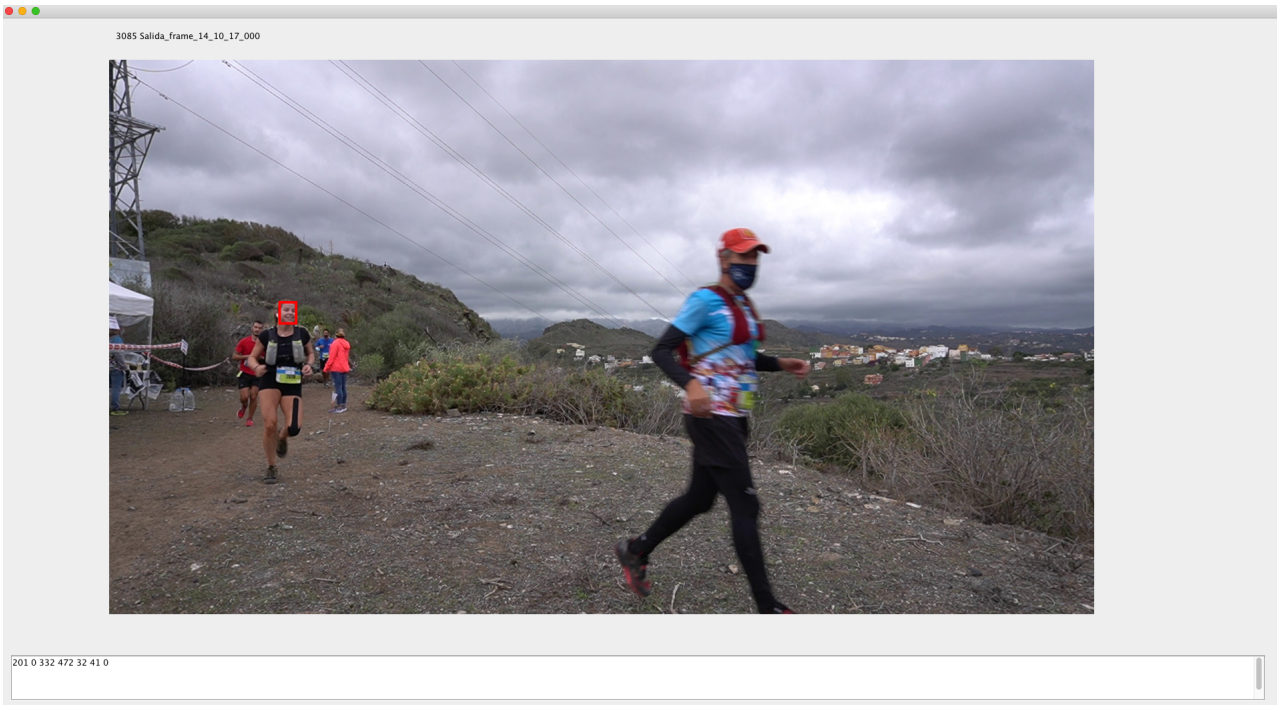

Figura 4.7: Interfaz de usuario LabeledImage.

imagen, se marca con el cuadro rojo otro corredor y lo etiquetamos a través de las teclas anteriores. En caso contrario, debemos de pulsar 'N' para pasar a la siguiente imagen y la información etiqueta se almacena en un fichero. Adicionalmente, la interfaz de usuario nos ofrece un título para visualizar el nombre de la imagen actual, y un área de texto para ir visualizando los valores que se almacenarán en disco. Finalmente, tendremos un conjunto de ficheros que representan el etiquetado de todas las imágenes y que nos sirven para el cálculo de las métricas.

Tras haber etiquetado todas las imágenes, podemos calcular la métrica de la exactitud y hallar la matriz de confusión.

Otra utilidad desarrollada para este trabajo la podemos encontrar en el fichero *groupDetection.py* que mostramos en el Algoritmo 4.5. Con esta herramienta, somos capaces de realizar una estadística que nos muestra para cada corredor, el instante de inicio y final que aparece en el vídeo, el número de fotogramas que fue detectado el cuerpo por el tracker, el número de detecciones de caras con y sin mascarilla, y el número de veces que no detecta ninguna cara en la

Id	Instante inicial	Instante final	ND	NM	NSM	NND
1	12_00_19_600	12_00_22_000	13	7	5	1
5	12_00_37_400	12_00_42_000	20	5	12	3
18	12_00_38_800	12_00_41_400	14	9	4	1

Tabla 4.2: Fichero GroupDetection. Las cuatro últimas columnas represantan el número de detecciones de cuerpos (ND), número de detecciones de caras con mascarilla (NM), número de detecciones de caras sin mascarilla (NSM) y el número de caras sin detectar (NND).

detección del cuerpo. En la Tabla 4.2 mostramos un fragmento del fichero.

```
for pathTxtFace in sorted(glob.glob(path + "*faces.txt")):
    init = pathTxtFace.find("Salida_frame_") + 13
    timeFile = pathTxtFace[init: init + 12]
    dIdsFace = getIdsFile(pathTxtFace)
    for id in dIdsFace:
        if id in dGroup:
            vContent = dGroup[id].copy()
            if dIdsFace[id][6] == '1':
                vContent[4] += 1
            if dIdsFace[id][6] == '0':
                vContent[5] += 1
            if dIdsFace[id][6] == 'ND':
                vContent[6] += 1
            vContent[3] += 1
            vContent[2] = timeFile
            dGroup[id] = vContent
        else:
            dGroup[id] = [id, timeFile, timeFile, 1, 1 if dIdsFace[id][6] == '1' else 0, 1 if dIdsFace[id][6]
                == '0' else 0, 1 if dIdsFace[id][6] == 'ND' else 0]

fileGroupDetection = open(path + "groupDetection.txt", "w")
for id in dGroup:
    fileGroupDetection.write(str(dGroup[id][0]) + " " + str(dGroup[id][1]) + " " + str(dGroup[id][2]) + " " +
        str(dGroup[id][3]) + " " + str(dGroup[id][4]) + " " + str(dGroup[id][5]) + " " + str(dGroup[id][6]) +
        " \n")
```

Algoritmo 4.5: Código GroupDetecion

A continuación, explicamos la línea de re-identificación de corredores.

4.2.3. Re-identificación de corredores

Tras la utilización del algoritmo Yolov4-DeepSort procedemos a realizar el siguiente paso, con un fichero para cada imagen cuyo contenido alberga en cada fila el identificador del corredor y el contenedor del cuerpo. Con esta información, utilizamos *getBodies.py* que mostramos en el Algoritmo 4.6.

```
for pathTxtBody in sorted(glob.glob(path + "*bodies.txt")):
    init = pathTxtBody.find("Salida_frame_") + 13
    timeFile = pathTxtBody[init: init + 12]
    fileFace = open(pathTxtBody)
    img = cv2.imread(path + "Salida_frame_" + timeFile + ".jpg")
    for line in fileFace:
        values = line.split(" ")
        beginX, endX, beginY, endY = int(values[1]), int(values[1]) + int(values[3]), int(values[2]),
            int(values[2]) + int(values[4])
        body_img = img[beginY:endY, beginX:endX]
        cv2.imwrite(pathDestiny + timeFile + "_" + values[0] + "_body.jpg", body_img)
    fileFace.close()
```

Algoritmo 4.6: Código obtención de cuerpos

Este algoritmo lee cada uno de los ficheros generados por Yolov4-DeepSort, recorta y almacena en un directorio todos los cuerpos de los corredores, nombrando cada cuerpo con su marca de tiempo e identificador. Por ejemplo, 14_50_39_200_1605_body.jpg.

Con toda esta colección de cuerpos de competidores procedemos a utilizar *makeFolderById.py* que visualizamos en el Algoritmo 4.7.

```
for place, body in zip(places, bodies):
    if not os.path.exists(pathDestiny + place):
        os.makedirs(pathDestiny + place)

    for pathPng1 in sorted(glob.glob(pathOrigin + body + "/" + "*.jpg")):
        pathWithOutBaseName, baseName = os.path.split(pathPng1)
        id = baseName.split("_")[4]
        if not os.path.exists(pathDestiny + place + "/" + id):
            os.makedirs(pathDestiny + place + "/" + id)
        shutil.copy(pathPng1,pathDestiny + place + "/" + id + "/" + baseName)
```

Algoritmo 4.7: Código creación carpetas por identificador

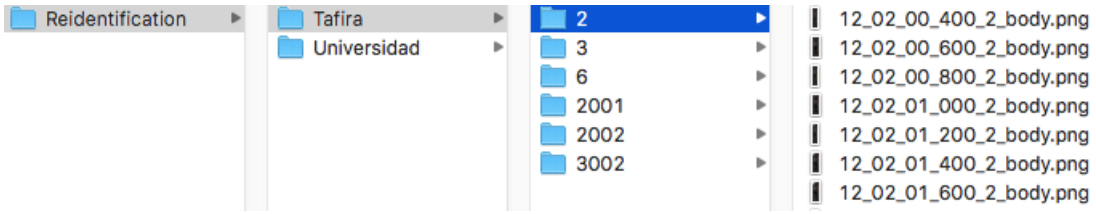

Figura 4.8: Resultado *makeFolderById.py*.

Este código nos permite agrupar todas las imágenes de corredores en carpetas según su identificador. Por tanto, tendremos tantas carpetas como corredores se han detectado, en su interior todo el conjunto de imágenes de cuerpos que corresponden a un corredor y el nombre de cada carpeta es el identificador del corredor almacenado en su interior, en la Figura 4.8 se contempla el resultado.

A continuación, realizamos un procedimiento manual de verificación y corrección, para que todas las carpetas realmente tengan cuerpos de corredores válidos para el algoritmo de re-identificación. Ya que existen situaciones que generan imágenes inválidas, por ejemplo, yolov4-DeepSort detecta una parte del cuerpo porque el corredor está en los bordes de la imagen pero el cuerpo no está completo, o existen varios cuerpos en la misma detección, provocado porque se detecta un corredor y detrás hay otra persona que no pertenece a la detección, pero sí a la imagen. En estos casos que mostramos en la Figura 4.9 eliminamos las imágenes.

También, existe otro problema que debemos de resolver manualmente, en algunos casos, Yolov4-DeepSort genera dos identificadores para el mismo corredor, esto a veces ocurre cuando se cruzan entre ellos como mostramos en la Figura 4.10.

Comprobamos que la persona con identificador 34 al cruzarse por delante del 46 el tracker se confunde con los identificadores y le asigna un número nuevo, el 9. De todas formas, en la mayoría de los casos el seguimiento es correcto y mantienen el identificador, son casos aislados que ocurren con más frecuencia cuando hay muchedumbre de corredores.

Por tanto, debemos de juntar las carpetas del mismo corredor pero que tienen

Figura 4.9: Casos de imágenes inválidas.

Figura 4.10: Caso de confusión de identificadores.

identificadores distintos, para ello, hemos generado una script con las órdenes mv y rm para agilizar el proceso como mostramos en 4.8.

```
!mv "Reidentification/Universidad_Tafira/601/"* "Reidentification/Universidad_Tafira/599"
!rm -f -d "Reidentification/Universidad_Tafira/601/"
```

Algoritmo 4.8: Script para mover corredores

Una vez tengamos un punto de la carrera con las imágenes preparadas para la re-identificación, realizamos los procedimientos anteriores con otro punto de la carrera, detectar los cuerpos, obtener los contenedores, crear los ficheros, crear las imágenes de los cuerpos, agruparlas en carpetas según su identificador, y verificar y corregir las imágenes. De esta forma, ya tenemos dos puntos de la carrera en distintos lugares e instantes, ideal para realizar la re-identificación. Sin embargo, todavía debemos de solucionar otro problema, al ejecutar Yolov4-DeepSort en dos vídeos distintos, el identificador de un mismo corredor en un vídeo y en otro no va a coincidir, y para obtener las métricas de re-identificación necesitamos que coincidan. Por tanto, realizamos un procedimiento manual para renombrar el nombre de la carpeta y que los identificadores de los corredores coincidan en ambos lugares. Para ello, elegimos una carpeta del primer sitio y visualizamos una imagen del corredor, después, buscamos entre las carpetas del segundo sitio a ese corredor, una vez encontrado, renombramos la carpeta para que coincidan los identificadores. Repetimos este proceso hasta elegir todas las carpetas del primer sitio. Una vez finalizado, podemos usar el código *renameFramePlace.py* que nos renombra los nombres de las imágenes contenidas en cada carpeta, cambiándoles el identificador y asignándoles el que tiene la carpeta. De esta manera, los identificadores de las carpetas coinciden con los de las imágenes en su interior.

Para nuestro proyecto nos interesa generar dos conjuntos de corredores con mascarilla y sin mascarilla, para poder realizar comparaciones y determinar con las métricas si los proceso de re-identificación se ven afectados por el uso de mascarilla. Además, establecemos que nuestro problema es de mundo cerrado, por lo que, todos los corredores tienen que estar en ambos puntos, no pueden existir corredores en un único lugar. Para lograr lo anterior, en el proceso reiterativo de renombrar las carpetas también movemos a cuatro di-

rectorios distintos las carpetas de los corredores. Primera carpeta, corredores con mascarilla del lugar B, segundo directorio, corredores sin mascarilla del lugar B, tercera carpeta, individuos del lugar A que coincidan con los corredores con mascarilla del lugar B, y cuarto directorio, individuos del lugar A que coinciden con los corredores sin mascarilla del lugar B. De esta forma, cumplimos las dos condiciones anteriores, agrupamos según el uso de mascarilla y no existen corredores en un único lugar. Para realizar esa tarea hemos desarrollado la siguiente script que mostramos en el Algoritmo 4.9

```
referencia="217"
comparado="1605"

!mkdir "Reidentification/Universidad_Tafira_Con_Mascarilla/$referencia"
!cp "Reidentification/Universidad_Tafira/$comparado/"*
    "Reidentification/Universidad_Tafira_Con_Mascarilla/$referencia"
!mkdir "Reidentification/Tafira_Universidad_Con_Mascarilla/$referencia"
!cp "Reidentification/Tafira_Universidad/$referencia/"*
    "Reidentification/Tafira_Universidad_Con_Mascarilla/$referencia"
```

Algoritmo 4.9: Script para juntar corredores

Con esta estructura y conjunto de datos ya estamos en condiciones de ejecutar los algoritmos de re-identificación, *deepFace.py* y *AlignedReId.py* que utilizan como base los códigos de deepFace y AlignedReId y añaden funcionalidades adicionales. Ambos algoritmos se han implementado de forma similar para tratar las imágenes, es decir, ambos leen las imágenes de dichos directorios, obtienen el vector de características de la imagen y lo almacenan en un fichero, aplican el algoritmo base para hallar las distancias y guardarlas en disco, y calculan las métricas del rank 1 y mAP para determinar la efectividad del algoritmo. Además, hemos desarrollado un código que permite elegir la opción deseada según el parámetro utilizado gracias a argparse. A continuación, explicamos con mayor detalle cada algoritmo.

El primer código, *deepFace.py* ofrece distintas opciones: -v obtiene los vectores de características, -die guarda en disco las distancias, -ap calcula el mAP y -r1 obtiene el rank 1.

Empezamos con el parámetro -v que lee las imágenes de la galería y genera los vectores de características como mostramos en el Algoritmo 4.10 llamando

a la función *DeepFace.represent()*. Estos vectores los almacenaremos en disco con la extensión .npy.

```
def deepFaceVector():
    model_names = ["VGG-Face", "Facenet", "OpenFace", "DeepFace", "DeepID", "Dlib", "ArcFace"]
    for dirpath, dirnames, filenames in os.walk(params.path):
        filenames = [f for f in filenames if not f[0] == '.' and f[-3:] == 'jpg']
        for file in sorted(filenames):
            pathWithOutBaseName, id = os.path.split(dirpath)
            timeFile = file[0: 12]
            for modelName in model_names:
                fileObj = Path(dirpath + "/" + timeFile + "_" + id + "_" + modelName + "_deepFaceVector.npy")
                if not fileObj.exists():
                    try:
                        print(dirpath + "/" + timeFile + "_" + id + "_body.jpg" + " ===> " + modelName)
                        result = DeepFace.represent(dirpath + "/" + timeFile + "_" + id + "_body.jpg",
                             model_name=modelName, enforce_detection=False)
                    except:
                        print("No se puede aplicar algoritmo" + dirpath + "/" + timeFile + "_" + id +
                             "_body.jpg" + " ===> " + modelName)
                    np.save(dirpath + "/" + timeFile + "_" + id + "_" + modelName + "_deepFaceVector", result)
                else:
                    print("Existe " + dirpath + "/" + timeFile + "_" + id + "_" + modelName + "_deepFaceVector")
```

Algoritmo 4.10: Código DeepFace obtención vectores de características

Después, guardamos las distancias como visualizamos en el Algoritmo 4.11. Se calcularon tres distancias, la coseno, euclídea y euclídea L2 y las almacenamos en disco, para ello, leemos los ficheros generados anteriormente que contienen los vectores de características. Para el cálculo de las distancias elegimos un corredor del lugar A y calculamos sus distancias con todos los competidores del lugar B y repetimos este proceso hasta elegir todos los identificadores del lugar A. Por tanto, tendremos un fichero por cada iteración que almacena en cada fila los siguientes valores importantes: idElegido idAComparar distanciaCoseno distanciaEuclidea distanciaEuclídeaL2, es decir, cada fila es la comparación con cada par de imágenes. Este cálculo lo repetimos para los distintos modelos que nos ofrece DeepFace (VGG-Face, Facenet, OpenFace, DeepFace, DeepId, Dlib y ArcFace) con el objetivo de obtener más resultados para comparar.

```
def distanceDeepFaceInterVideo():
    model_names = ["VGG-Face", "Facenet", "OpenFace", "DeepFace", "DeepID", "Dlib", "ArcFace"]
    for model in model_names:
        print("============= " + model + " ==================")
        for dirpath1, dirnames1, filenames1 in os.walk(params.path):
            filenames1 = [f for f in filenames1 if not f[0] == '.' and f[-18:] == 'deepFaceVector.npy' and
```

```
        f.split("_")[5] == model]
    for file1 in sorted(filenames1):
        pathWithOutBaseName1, id1 = os.path.split(dirpath1)
        x, place1 = os.path.split(pathWithOutBaseName1)
        timeFile1 = file1[0: 12]
        vectorFeature1 = np.load(dirpath1 + "/" + timeFile1 + "_" + id1 + "_" + model
            +"_deepFaceVector.npy")
        fileOutput = open(dirpath1 + "/" + timeFile1 + "_" + id1 + "_" + model +
            "_deepFaceInterDistance.txt", "w")
        for dirpath2, dirnames2, filenames2 in os.walk(params.path):
            filenames2 = [f for f in filenames2 if not f[0] == '.' and f[-18:] == 'deepFaceVector.npy'
                and f.split("_")[5] == model]
            for file2 in sorted(filenames2):
                pathWithOutBaseName2, id2 = os.path.split(dirpath2)
                y, place2 = os.path.split(pathWithOutBaseName2)
                timeFile2 = file2[0: 12]
                if place1 != place2:
                    vectorFeature2 = np.load(dirpath2 + "/" + timeFile2 + "_" + id2 + "_" + model +
                        "_deepFaceVector.npy")
                    print(dirpath1 + "/" + file1 + " ===> " + dirpath2 + "/" + file2 + " ===> " )
                    distanceCosine, distanceEuclidean, distanceEuclidean_l2 =
                        distanceVectorFeature(vectorFeature1, vectorFeature2)
                    fileOutput.write(place1 + " " + timeFile1 + " " + id1 + " " + place2 + " " +
                        timeFile2 + " " + id2 + " " + str(distanceCosine) + " " + str(distanceEuclidean)
                        + " " + str(distanceEuclidean_l2) + " \n")
```

Algoritmo 4.11: Código DeepFace cálculo distancias

Con las distancias en disco, ya podemos calcular las métricas necesarias para el trabajo, por ejemplo, la mAP como mostramos en el Algoritmo 4.12.

Lugar A	Instante A	Id A	Lugar B	Instante B	Id B	Coseno	Euclídea	Euclídea L2
Tafira	12_02_09_400	6	Universidad	14_02_00_400	2	0,2533	1,3918	0,2251
Tafira	12_02_09_400	6	Universidad	14_02_00_600	6	0,0025	0,0044	0,0035
Tafira	12_02_09_400	6	Universidad	14_02_00_800	6	0,0024	0,0045	0,0035

Tabla 4.3: Fichero distancias DeepFace.

```
def averagePrecisionDeepFace():
    model_names = ["VGG-Face", "Facenet", "OpenFace", "DeepFace", "DeepID", "Dlib", "ArcFace"]
    for model in model_names:
        allAPCosine = []
        allAPEuclidean = []
        allAPEuclidean_l2 = []
        for dirpath1, dirnames1, filenames1 in os.walk(params.path):
            filenames1 = [f for f in filenames1 if not f[0] == '.' and f[-25:] == 'deepFaceInterDistance.txt'
                 and f.split("_")[5] == model]
            for file1 in sorted(filenames1):
                gallery_coincidences, similarityQueryGalleryCosine, similarityQueryGalleryEuclidean,
                     similarityQueryGalleryEuclidean_l2 = getParameterAveragePrecision(dirpath1, file1)
                apCosine = average_precision_score(gallery_coincidences, similarityQueryGalleryCosine,
                     average='macro', pos_label=1)
                apEuclidean = average_precision_score(gallery_coincidences, similarityQueryGalleryEuclidean,
                     average='macro', pos_label=1)
                apEuclidean_l2 = average_precision_score(gallery_coincidences,
                     similarityQueryGalleryEuclidean_l2, average='macro', pos_label=1)
                allAPCosine.append(apCosine)
                allAPEuclidean.append(apEuclidean)
                allAPEuclidean_l2.append(apEuclidean_l2)
        mAPCosine = np.mean(allAPCosine)
        mAPEuclidean = np.mean(allAPEuclidean)
        mAPEuclidean_l2 = np.mean(allAPEuclidean_l2)
        print("============ mAP " + model + " ====================")
        print("mAP Coseno " + str(mAPCosine))
        print("mAP Euclidea " + str(mAPEuclidean))
        print("mAP Euclidea_l2 " + str(mAPEuclidean_l2))
        print("=================================================")
```

Algoritmo 4.12: Código DeepFace cálculo mAP

En este código calculamos el mAP para cada distancia: coseno, euclidea y euclidean L2, y a su vez para cada modelo. Para calcular esta métrica necesitamos una lista de verdaderos con la misma dimensión que el tamaño de la galería, cuyas posiciones coinciden con la identidad de la imagen query a 1 y el resto a 0. Y otras tres listas para cada distancia. Estas listas son obtenidas de cada fichero de distancias generado en el paso anterior, mostramos un fragmento del contenido del fichero en la Tabla 4.3 para mejorar la compresión.

Por tanto, las tres últimas columnas del fichero coinciden con esas distancias que necesitamos, mientras que la lista de verdaderos puede ser calculado a través de la columna "Id B" conociendo el identificador de la imagen query dado por la columna "Id A". Si nos centramos en una única distancia de las anteriores, obtenemos una lista de similitudes a partir de la fórmula 1 - $\frac{distancia}{max(distancia)}$ de cada imagen query gracias a la función $getParameterAverage(...)$ que visualizamos en el Algoritmo 4.13. Además, esta función nos calcula la lista de verdaderos mencionado anteriormente.

```
def getParameterAveragePrecision(dirpath1, file1):
    gallery_coincidences = []
    distQueryGalleryCosine = []
    distanceEuclidean = []
    distanceEuclidean_l2 = []
    fileOutput = open(dirpath1 + "/" + file1)
    for line in fileOutput:
        if line.split(" ")[2] == line.split(" ")[5]:
            gallery_coincidences.append(1)
        else:
            gallery_coincidences.append(0)
        distQueryGalleryCosine.append(float(line.split(" ")[6]))
        distanceEuclidean.append(float(line.split(" ")[7]))
        distanceEuclidean_l2.append(float(line.split(" ")[8]))
    similarityQueryGalleryCosine = 1 - distQueryGalleryCosine / np.amax(distQueryGalleryCosine)
    similarityQueryGalleryEuclidean = 1 - distanceEuclidean / np.amax(distanceEuclidean)
    similarityQueryGalleryEuclidean_l2 = 1 - distanceEuclidean_l2 / np.amax(distanceEuclidean_l2)
    return gallery_coincidences, similarityQueryGalleryCosine, similarityQueryGalleryEuclidean,
         similarityQueryGalleryEuclidean_l2
```

Algoritmo 4.13: Código DeepFace obtención vectores para cálculo mAP

Finalmente, utilizamos la función $average\_precision\_score(...)$ con la lista de verdaderos y con la lista de similitudes y añadimos el AP calculado a un array. Repetimos los pasos anteriores con toda la galería y obtenemos el valor promedio de AP, que es nuestra mAP. Este proceso se realiza en paralelo para cada distancia.

También, calculamos el rank 1 en el conjunto de imágenes, para ello, volvemos a utilizar el fichero de distancias representado en la Tabla 4.3 y llamamos a la función del Algoritmo 4.14.

```
def rank1DeepFace():
    model_names = ["VGG-Face", "Facenet", "OpenFace", "DeepFace", "DeepID", "Dlib", "ArcFace"]
    for model in model_names:
```

```
numeradorRank1Cosine = 0
numeradorRank1Euclidean = 0
numeradorRank1Euclidean_l2 = 0
denominadorRank1 = 0
print("============= " + model + " ==================")
for dirpath1, dirnames1, filenames1 in os.walk(params.path):
    filenames1 = [f for f in filenames1 if not f[0] == '.' and f[-25:] == 'deepFaceInterDistance.txt'
         and f.split("_")[5] == model]
    for file1 in sorted(filenames1):
        distCosineList = []
        distEuclideanList = []
        distEuclidean_l2List = []
        id2List = []
        fileOutput = open(dirpath1 + "/" + file1)
        id1 = file1.split("_")[4]
        for line in fileOutput:
            id2List.append(line.split(" ")[5])
            distCosineList.append(line.split(" ")[6])
            distEuclideanList.append(line.split(" ")[7])
            distEuclidean_l2List.append(line.split(" ")[8])
        positionMinValueRank1CosineList = distCosineList.index(min(distCosineList))
        positionMinValueRank1EuclideanList = distEuclideanList.index(min(distEuclideanList))
        positionMinValueRank1Euclidean_l2List = distEuclidean_l2List.index(min(distEuclidean_l2List))
        if(id1 == id2List[positionMinValueRank1CosineList]):
            numeradorRank1Cosine+=1
        if(id1 == id2List[positionMinValueRank1EuclideanList]):
            numeradorRank1Euclidean+=1
        if(id1 == id2List[positionMinValueRank1Euclidean_l2List]):
            numeradorRank1Euclidean_l2+=1
        denominadorRank1+=1
rank1Cosine = numeradorRank1Cosine/denominadorRank1
rank1Euclidean = numeradorRank1Euclidean / denominadorRank1
rank1Euclidean_l2 = numeradorRank1Euclidean_l2 / denominadorRank1
print("========= RANK1 " + model + " =============")
print("Cosine " + str(rank1Cosine))
print("Euclidean " + str(rank1Euclidean))
print("Euclidean_l2 " + str(rank1Euclidean_l2))
```

Algoritmo 4.14: Código DeepFace cálculo rank 1

Esta función nos permite calcular el rank 1 para todas las distancias y modelos. La idea fundamental para calcular el rank 1 de una distancia y modelo es buscar en el fichero anterior en una de las columnas de distancias ”Coseno”, ”Euclídea” o ”Euclídea L2” el valor mínimo y comprobar si ese valor corresponde a la imagen query haciendo uso de las columnas ”Id A” y ”Id B”. Si coinciden aumentamos en 1 el numerador del Rank1, en caso contrario, no incrementamos. Repetimos el proceso hasta recorrer todas las imágenes de la galería del Lugar A para finalmente calcular el rank 1 como $\frac{numeradorRank1}{denominadorRank1}$,

siendo el denominador el número total de imágenes del Lugar A.

El segundo código, *AlignedReId.py* ofrece distintas opciones: -v obtiene los vectores de características, -d guarda en disco las distancias, -ap calcula el mAP y -r1 obtiene el rank 1, comprobamos que las opciones son muy parecidas a las de *deepFace.py* y la implementación también es muy similar, por lo tanto, este código lo explicamos de forma más resumida debido al gran parecido con DeepFace que ya explicamos anteriormente.

Con la opción -v obtenemos los vectores de características y llamamos al algoritmo 4.15.

```
def alignedReIdVector():
    pathModel = "/content/gdrive/My Drive/TFG/AlignedReID/data/"
    model_names = ["Cuhk03_Resnet50", "DukeMTMCReID_Resnet50", "Market1501_Resnet50"]
    num_classes_models = [767, 702, 751]
    for modelName, num_classes in zip(model_names, num_classes_models):
        os.environ['CUDA_VISIBLE_DEVICES'] = "0"
        use_gpu = torch.cuda.is_available()
        model = models.init_model(name='resnet50', num_classes=num_classes, loss={'softmax', 'metric'},
             use_gpu=use_gpu,aligned=True)
        checkpoint = torch.load(pathModel + modelName + ".tar", encoding = "ISO-8859-1", )
        model.load_state_dict(checkpoint['state_dict'])
        img_transform = transforms.Compose([
            transforms.Resize((256, 128)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])
        exact_list = ['7']
        myexactor = FeatureExtractor(model, exact_list)
        for dirpath, dirnames, filenames in os.walk(params.path):
            filenames = [f for f in filenames if not f[0] == '.' and f[-8:] == 'body.png']
            for file in sorted(filenames):
                pathWithOutBaseName, id = os.path.split(dirpath)
                timeFile = file[0: 12]
                pathImage = dirpath + "/" + timeFile + "_" + id + "_body.png"
                print(pathImage)
                img1 = read_image(pathImage)
                img1 = img_to_tensor(img1, img_transform)
                if use_gpu:
                    model = model.cuda()
                    img1 = img1.cuda()
                model.eval()
                f1 = myexactor(img1)
                a1 = normalize(pool2d(f1[0], type='max'))
                np.savez(dirpath + "/" + timeFile + "_" + id + "_" + modelName +
                     "_AlignedReId",a1[0],a1[1],a1[2],a1[3],a1[4],a1[5],a1[6],a1[7])
```

Algoritmo 4.15: Código AlignedreId obtención vectores de características

Por tanto, leemos las imágenes de la galería y utilizamos AlignedReId para que nos devuelve en a1 un conjunto de vectores que representan al vector de características de la imagen, y lo almacenamos en un fichero con extensión .npz. Esta extensión permite guardar un conjunto de vectores en un fichero. Esta función está desarrollada para que calcule los vectores con distintos modelos: "Cuhk03_Resnet50", "DukeMTMCReID_Resnet50" y "Market1501_Resnet50".

Con la opción -d calculamos las distancias como mostramos en el Algoritmo 4.16.

```
def distanceAlignedReIdVector():
    model_names = ["Cuhk03_Resnet50", "DukeMTMCReID_Resnet50", "Market1501_Resnet50"]
    for modelName in model_names:
        print("======= " + modelName + " ==========")
        for dirpath1, dirnames1, filenames1 in os.walk(params.path):
            filenames1 = [f for f in filenames1 if not f[0] == '.' and f[-15:] == 'AlignedReId.npz' and
                 f.split("_")[5] == modelName.split("_")[0]]
            for file1 in sorted(filenames1):
                pathWithOutBaseName1, id1 = os.path.split(dirpath1)
                x, place1 = os.path.split(pathWithOutBaseName1)
                timeFile1 = file1[0: 12]
                arrays1 = np.load(dirpath1 + "/" + timeFile1 + "_" + id1 + "_" + modelName + "_AlignedReId.npz")
                fileOutput = open(dirpath1 + "/" + timeFile1 + "_" + id1 + "_" + modelName +
                     "_AlignedReIdDistance.txt", "w")
                for dirpath2, dirnames2, filenames2 in os.walk(params.path):
                    filenames2 = [f for f in filenames2 if not f[0] == '.' and f[-15:] == 'AlignedReId.npz' and
                         f.split("_")[5] == modelName.split("_")[0]]
                    for file2 in sorted(filenames2):
                        pathWithOutBaseName2, id2 = os.path.split(dirpath2)
                        y, place2 = os.path.split(pathWithOutBaseName2)
                        timeFile2 = file2[0: 12]
                        if place1 != place2:
                            arrays2 = np.load(dirpath2 + "/" + timeFile2 + "_" + id2 + "_" + modelName +
                                 "_AlignedReId.npz")
                            print(dirpath1 + "/" + file1 + " ===> " + dirpath2 + "/" + file2 + " ===> " )
                            dist = np.zeros((8, 8))
                            for i in range(8):
                                temp_feat1 = arrays1["arr_" + str(i)]
                                for j in range(8):
                                    temp_feat2 = arrays2["arr_" + str(j)]
                                    dist[i][j] = euclidean(temp_feat1, temp_feat2)
                            alignDistance, originalDistance = calculateAlignedAndOriginalDistance(dist)
                            fileOutput.write(place1 + " " + timeFile1 + " " + id1 + " " + place2 + " " +
                                 timeFile2 + " " + id2 + " " + alignDistance + " " + originalDistance + " \n")
```

Algoritmo 4.16: Código AlignedreId cálculo distancias

Entonces, leemos los ficheros anteriores que representan cada imagen para su correspondiente modelo, y calculamos las distancias para cada imagen comparándola con todas las imágenes del otro lugar. Las distancias calculadas son la distancia alineada y la distancia original. Las almacenamos en un fichero con el siguiente formato que es representado en la Tabla 4.4, es un ejemplo de un fragmento de un fichero.

A continuación, invocamos al algoritmo *AlignedReid.py* con la opción -ap para calcular el mAP, mostramos en el Algoritmo 4.17 el fragmento de código ejecutado. El algoritmo es muy similar al de DeeFace, a partir de los datos del fichero de distancia calculamos el mAP para cada distancia y modelo. Por último con la opción -r1 obtenemos el rank1 ejecutando la función del Algoritmo 4.18.

Lugar A	**Instante A**	**Id A**	**Lugar B**	**Instante B**	**Id B**	**Alineada**	**Original**
Tafira	12_02_09_400	6	Universidad	14_02_00_400	2	0,3533	1,3918
Tafira	12_02_09_400	6	Universidad	14_02_00_600	6	0,0015	0,0064
Tafira	12_02_09_400	6	Universidad	14_02_00_800	6	0,0014	0,0015

Tabla 4.4: Fichero distancias AlignedReId.

```
def averagePrecisionAlignedReId():
    model_names = ["Cuhk03_Resnet50", "DukeMTMCReID_Resnet50", "Market1501_Resnet50"]
    for model in model_names:
        allAPAlign = []
        allAPOriginal = []
        for dirpath1, dirnames1, filenames1 in os.walk(params.path):
            filenames1 = [f for f in filenames1 if not f[0] == '.' and f[-23:] == 'AlignedReIdDistance.txt' and
                 f.split("_")[5] == model.split("_")[0]]
            for file1 in sorted(filenames1):
                gallery_coincidences, similarityQueryGalleryAlign, similarityQueryGalleryOriginal =
                     getParameterAveragePrecision(dirpath1, file1)
                apAlign = average_precision_score(gallery_coincidences, similarityQueryGalleryAlign,
                     average='macro', pos_label=1)
                apOriginal = average_precision_score(gallery_coincidences, similarityQueryGalleryOriginal,
                     average='macro', pos_label=1)
                allAPAlign.append(apAlign)
                allAPOriginal.append(apOriginal)
        mAPAlign = np.mean(allAPAlign)
        mAPOriginal = np.mean(allAPOriginal)
        print("============ mAP " + model + " ====================")
        print("mAP Align " + str(mAPAlign))
        print("mAP Original " + str(mAPOriginal))
        print("================================================")
```

Algoritmo 4.17: Código AlignedreID cálculo mAP

```
def rank1AlignedReId():
    model_names = ["Cuhk03_Resnet50", "DukeMTMCReID_Resnet50", "Market1501_Resnet50"]
    for modelName in model_names:
        print("======= " + modelName + " ==========")
        numeradorAlignRank1 = 0
        numeradorOriginRank1 = 0
        denominadorRank1 = 0
        for dirpath1, dirnames1, filenames1 in os.walk(params.path):
            filenames1 = [f for f in filenames1 if not f[0] == '.' and f[-23:] == 'AlignedReIdDistance.txt' and
                 f.split("_")[5] == modelName.split("_")[0]]
            for file1 in sorted(filenames1):
                distAlignList = []
                distOriginList = []
                id2List = []
                fileOutput = open(dirpath1 + "/" + file1)
                id1 = file1.split("_")[4]
                for line in fileOutput:
                    id2List.append(line.split(" ")[5])
                    distAlignList.append(line.split(" ")[6])
                    distOriginList.append(line.split(" ")[7])
                positionMinValueAlignRank1List = distAlignList.index(min(distAlignList))
                positionMinValueOriginRank1List = distOriginList.index(min(distOriginList))
                if(id1 == id2List[positionMinValueAlignRank1List]):
                    numeradorAlignRank1+=1
                if(id1 == id2List[positionMinValueOriginRank1List]):
                    numeradorOriginRank1+=1
                denominadorRank1+=1
        rank1Align = numeradorAlignRank1/denominadorRank1
```

```
rank1Origin = numeradorOriginRank1/denominadorRank1
print("========= RANK1 " + modelName + " =============")
print("Aligned " + str(rank1Align))
print("Original " + str(rank1Origin))
```

Algoritmo 4.18: Código AlignedreId cálculo rank 1

Capítulo 5

Experimentos y resultados

En este capítulo ejecutamos las implementaciones expuestas en la Sección 4.2 sobre un conjunto de datos correspondientes a dos puntos de la competición LPA Trail 2020, en Tafira y la Universidad. Además, realizamos algunas pruebas con muestras de tamaño reducido para comprobar el funcionamiento de los algoritmos. Para la línea de detección de mascarillas ejecutamos el algoritmo solamente en la Universidad. En cambio, para la línea de la re-identificación de corredores utilizamos los dos lugares anteriores.

5.1. Experimentos y resultados detección de mascarillas

Para realizar estos experimentos debemos de realizar el procedimiento común y la línea de detección de mascarillas descrita en la Sección 4.2.

El primer experimento consiste en una prueba con una muestra de tamaño reducido para testear si el funcionamiento del algoritmo InsightFace junto con DeepSort y Yolov4 es el adecuado. El experimento consta de dos imágenes de la Universidad, una imagen con un corredor con mascarilla y otra imagen con un participante sin ella, de esta forma, comprobamos si el algoritmo

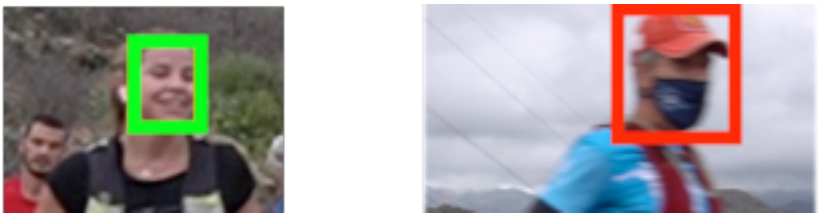

Figura 5.1: Detección de mascarillas Universidad.

InsightFace

	Con y sin mascarilla (%)	**Con mascarilla (%)**	**Sin mascarilla (%)**
Exactitud	78,02	**85,68**	70,73

Tabla 5.1: Métrica InsightFace.

acierta en ambos casos. Para ello, comprobamos si las predicciones realizadas por InsightFace se corresponden con la realidad, es decir, en las detecciones de corredores efectuadas por DeepSort_Yolov4, InsightFace acierta en la detección de la cara y el uso de la mascarilla. Esto nos permite comprobar el correcto funcionamiento de la línea de trabajo desarrollada, pero no si el uso de mascarilla afecta a la detección. Sabemos que una muestra tan pequeña no es representativa de la población, pero es una buena aproximación a un experimento más ambicioso. En la Figura 5.1 mostramos los resultados del experimento.

Comprobamos que en ambos casos el algoritmo InsightFace acierta la predicción de si llevan o no mascarilla ambos corredores, pero todavía no estamos en condiciones de contrastar nuestra hipótesis. Recordamos que un cuadro verde indica que no lleva mascarilla, y un cuadro rojo que si la lleva, precisamente lo que ocurre en la Figura 5.1.

Para alcanzar una conclusión, en el segundo experimento seleccionamos una muestra con un mayor número de individuos y más representativa, concretamente, con un tamaño de 31233 imágenes y 115 corredores aproximadamente, de estas 31233 imágenes obtenemos 1073 individuos sin mascarilla y 1020 individuos con mascarilla. La muestra está formada por el conjunto de individuos de la Universidad con y sin mascarilla. Por tanto, realizamos la línea de trabajo común y de detección de mascarillas y obtenemos las métricas de la Tabla 5.1.

Entonces, el método se ve afectado por el uso de mascarilla, como esperábamos es más fácil acertar un individuo que lleve mascarilla que un participante sin ella, como demostramos en el experimento, de los individuos que llevan mascarilla, acierta el 85,68 % como que realmente llevan mascarilla. En cambio, en los individuos sin mascarilla solamente acierta el 70,73 %. Observamos que existe una diferencia importante entre las dos mediciones, siendo a favor de los individuos con mascarilla. Por tanto, el uso de mascarilla afecta al método de detección de mascarillas, y el método tendrá una mayor fiabilidad cuando el individuo lleve mascarilla. También, observamos que si agrupamos todos los individuos, tanto con mascarilla como sin ella, el 78,02 % es superior al 70,73 % pero inferior al 85,68 %, valor también esperado, ya que al estar mezclado ambos grupos, debe ser mejor detectando que el grupo completo sin mascarilla, pero peor que el grupo con mascarilla. Una hipótesis de por qué ocurre este hecho, puede deberse a que el método acierta con mayor facilidad en una detección donde existe una figura de un polígono con un color de fondo homogéneo como puede ser una mascarilla, que acertar una detección de una cara con boca, nariz, ojos, color de piel, etc, que suelen tener más particularidades que un polígono regular.

Como información adicional, calculamos la matriz de confusión en el conjunto de individuos de la Universidad con y sin mascarilla que mostramos en la Tabla 5.2.

		Clases predichas	
		Sin	Con
Clases reales	Sin	759	314
	Con	146	874

Tabla 5.2: Matriz de confusión.

Alcanzamos la misma conclusión que con la métrica anterior, en este caso, se demuestra porque observamos que de 1073 individuos sin mascarilla acierta

Id	Instante inicial	Instante final	ND	NM	NSM	NND
4	14_00_02_400	14_00_05_600	17	6	0	11
10	14_00_17_200	14_00_18_000	5	1	3	1
11	14_00_20_200	114_00_20_400	2	1	0	1
6	114_00_20_400	14_00_20_800	3	0	2	1
1010	14_00_21_400	14_00_21_600	2	0	0	2

Tabla 5.3: Estadística GroupDetection. Las cuatro últimas columnas representan el número de detecciones de cuerpos (ND), número de detecciones de caras con mascarilla (NM), número de detecciones de caras sin mascarilla (NSM) y el número de caras sin detectar (NND).

759 y falla 314, y de 1020 con mascarilla acierta 874 y falla 146, es decir, en el grupo con mascarilla acierta más y falla menos que en el grupo sin mascarilla. Por tanto, demostramos lo anterior, el uso de mascarilla afecta a la detección, y concretamente, en los casos que el individuo lleve mascarilla, la detección se facilita.

Por último, mostramos los primeros cinco resultados de la herramienta desarrollada para generar una estadística de las detecciones de mascarilla en la Tabla 5.3, realmente el fichero contiene un mayor número de resultados, pero consideramos que no es necesario mostrarlos todos debido a su extensión. De todas formas, podemos encontrar el fichero groupDetection.txt completo en el repositorio de GitHub[1] en el directorio informes. Recordamos que esta estadística muestra el instante inicial y final de cada corredor junto con el número de detecciones de cuerpos, de caras con mascarilla, de caras sin mascarilla y de caras sin detectar.

5.2. Experimentos y resultados re-identificación de corredores

En estos experimento debemos de realizar el procedimiento común y la línea de trabajo re-identificación de corredores descrita en la Sección 4.2.

Primero, realizamos un experimento en una muestra muy pequeña para com-

[1]https://github.com/ignaciomarinreyes/Re-identificacion_basada_en_caras_en_tiempos_de_pandemia

Tafira \ Universidad	5123	9123	2105	166
5123	X	X		
2105			X	X

Tabla 5.4: Comparaciones re-identificación Tafira-Universidad

probar el correcto funcionamiento del procedimiento común y la línea de trabajo re-identificación de corredores sin tener en cuenta las métricas. El experimento se basa en cuatro comparaciones que mostramos en la Figura 5.2 y en la Tabla 5.4. Entonces, si nos centramos en la parte de la re-identificación, es decir, en AlignedReID y DeepFace. La teoría nos indica que, si las distancias son más próximas a 0, mayor posibilidad de que sean el mismo individuo. En cambio, si las distancias son más lejanas a 0, mayor posibilidad que sean distintos.

Por tanto, teniendo en cuenta que el valor superior de cada comparación de la Figura 5.2 corresponde a DeepFace con el modelo Dlib y usando la distancia coseno, y el valor inferior corresponde a AignedReID con el modelo CuhK03_Resnet50 y calculando la distancia alineada. Podemos observar en este caso que las distancias calculadas por DeepFace no son discriminantes, todos los valores son muy próximos unos a otros, solamente se percibe una ligera reducción de sus valores al no llevar mascarilla. Por otra parte, AlignedReID si discrimina eficazmente una persona distinta a la comparada, observamos que si se compara un corredor distinto su valor es más próximo a 1 que la comparación con el mismo corredor. Y también observamos que AlignedReID parece que se aproxima más a 0 cuando el corredor no lleva mascarilla. Todavía, no estamos en condiciones de alcanzar ninguna conclusión, por ello, en el segundo experimento realizamos una prueba con muestras mayores y más representativas. Este experimento nos ha permitido comprobar que la línea de trabajo común y de re-identificación funciona correctamente.

El segundo experimento consiste en 4 muestras que representamos en la Figura 5.3. Comprobamos que del conjunto de datos de Tafira obtenemos dos muestras, grupo 1 (Tafira sin mascarilla) y grupo 3 (Tafira sin mascarilla), mientras que del conjunto de datos de la Universidad obtenemos otras dos muestras, grupo 2 (Universidad con mascarilla) y grupo 4 (Universidad sin

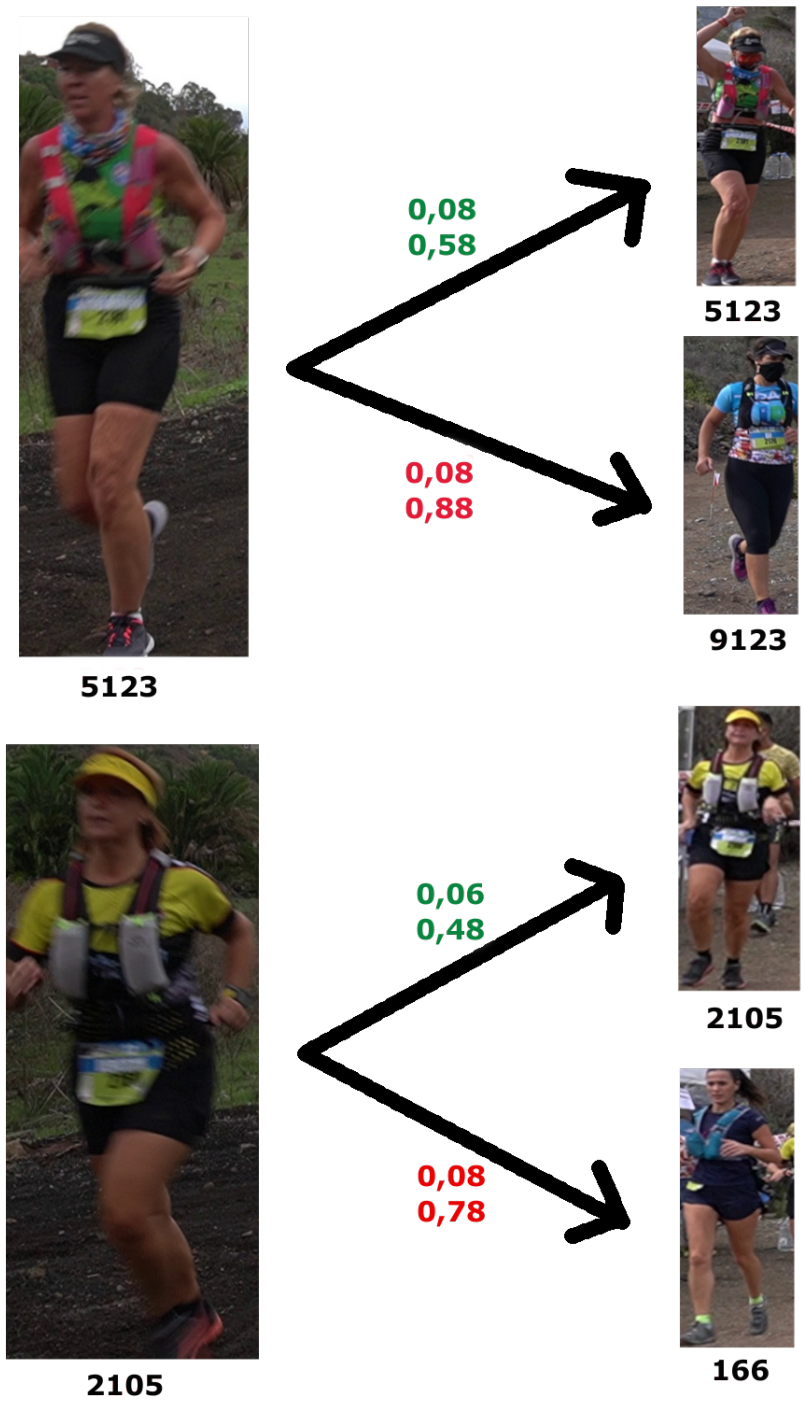

Figura 5.2: Re-identificación Tafira-Universidad.

Figura 5.3: Muestras de la prueba.

mascarilla). Las muestras obtenidas de los conjuntos de datos de Tafira y la Universidad son independientes, es decir, las re-identificaciones son realizadas en cuatro conjuntos de imágenes distintas. Las re-identificaciones se realizan entre el grupo 1 y 2, y el grupo 3 y 4. Por tanto, el conjunto de datos de Tafira es la referencia que nos permite comparar con otros grupos, mientras que el conjunto de datos de la Universidad al dividirlo entre corredores con y sin mascarilla nos permite analizar cómo se ven afectados los métodos de re-identificación por el uso de mascarilla. Además, los corredores que aparecen en el grupo 1 son los mismos que aparecen en el grupo 2 pero en distintas imágenes, sin existir algún corredor que solo aparezca en un grupo. El mismo hecho sucede con el grupo 3 y 4. De esta forma, cumplimos la condición de problema de re-identificación de mundo cerrado y agruparlos por el uso de mascarilla

Estas muestras tienen un número de imágenes y de inscritos que representamos en la Tabla 5.5, comprobamos que en total son 3607 imágenes y 113 identidades, ya que tanto el grupo 1 y 2 como el grupo 3 y 4 comparten el mismo identificador. Por término medio, cada corredor tiene alrededor de 30 imágenes.

Tras realizar cada uno de los pasos de la línea de trabajo común y re-identificación obteniendo las 4 muestras anteriores. Las métricas nos dan una serie de resultados que mostramos en la Tabla 5.6.

	Grupo 1	Grupo 2	Grupo 3	Grupo 4	Total
Imágenes	1585	864	749	409	3607
Identidades	72		41		113

Tabla 5.5: Tamaño muestras re-identificación.

DeepFace

Sin mascarilla

mAP

	VGG-Face (%)	Facenet (%)	OpenFace (%)	DeepFace (%)	DeepID (%)	Dlib (%)	ArcFace (%)
Coseno	11,89	6,10	9,07	13,74	11,14	16,59	14,92
Euclidea	9,02	6,10	9,07	12,31	6,67	**17,12**	7,51
Euclidea L2	11,89	6,10	9,07	13,74	11,14	16,59	14,92

Rank 1

	VGG-Face (%)	Facenet (%)	OpenFace (%)	DeepFace (%)	DeepID (%)	Dlib (%)	ArcFace (%)
Coseno	15,75	6,68	10,55	22,16	14,02	26,70	**27,90**
Euclidea	9,88	6,81	10,55	2,27	2,80	27,64	8,28
Euclidea L2	15,75	6,68	10,55	22,16	14,02	26,70	**27,90**

Con mascarilla

mAP

	VGG-Face (%)	Facenet (%)	OpenFace (%)	DeepFace (%)	DeepID (%)	Dlib (%)	ArcFace (%)
Coseno	nan	3,88	3,81	**7,17**	4,56	6,64	5,80
Euclidea	nan	3,82	3,81	5,94	4,30	6,70	3,27
Euclidea L2	nan	3,88	3,81	**7,17**	4,56	6,64	5,80

Rank 1

	VGG-Face (%)	Facenet (%)	OpenFace (%)	DeepFace (%)	DeepID (%)	Dlib (%)	ArcFace (%)
Coseno	nan	4,53	3,92	11,51	5,51	**12,82**	8,74
Euclidea	nan	4,08	3,92	2,65	1,63	12,66	4,57
Euclidea L2	nan	4,53	3,92	11,51	5,51	**12,82**	8,74

Tabla 5.6: Métricas re-identificación DeepFace.

Como podemos visualizar, utilizamos el algoritmo DeepFace que realizaba re-identificaciones basadas en los rasgos de las caras de los corredores, es decir, realizaba una detección de la cara para extraer el vector de características para continuar con la re-identificación. En el experimento se utilizan los modelos: VGG-Face, Facenet, OpenFace, DeepFace, DeepID, Dlib y ArcFace, las distancias: coseno, euclidea y euclidea L2, y se calcula el mAP y el rank 1 tanto para el grupo con mascarilla como sin ella. En este experimento comprobamos que el método de re-identificación se ve afectado por el uso de mascarilla, ya que tanto el rank 1 como el mAP en el grupo sin mascarilla nos ofrece valores mayores que en el grupo con mascarilla. Por tanto, el experimento cumple lo esperado, es decir, si un corredor usa mascarilla, la re-identificación se dificulta, como demuestran los valores menores en el grupo con mascarilla. Posiblemente, este hecho ocurre porque si un corredor oculta con una mascarilla la mitad de su cara, las características realmente valiosas para la re-identificación de este algoritmo se basan en la parte de la cara descubierta. Entonces, cuanta menos área de cara obtengamos en la imagen debido a que está oculta por la mascarilla, peores resultados obtendremos. También, podemos destacar que los modelos que nos ofrecen mejores resultados en ambos casos son Dlib y ArcFace, que la distancia coseno parece que ofrece mejores resultados que la euclidea en la mayoría de los casos, que la distancia euclídea L2 tiene una gran similitud con respecto a la coseno, probablemente porque en su implementación son muy parecidas, que con el rank 1 se obtienen mejores resultados que con el mAP y que VGG-Face en los casos que lleva mascarilla no es capaz ni de detectar la cara del corredor (nan) en la mayoría de los casos. Sin embargo, un hecho muy relevante es que la mayoría de los valores son bajos, inferiores a un 30 %, posiblemente porque las caras que suministramos a la red son muy pequeñas, y esto no permite obtener mucha información valiosa acerca de la cara del individuo que ayude en la re-identificación. Este hecho es normal, nuestro contexto son corredores que atraviesan un punto corriendo, las caras son diminutas, no es una imagen de un retrato con un área mayor de cara en la imagen. Estos bajos resultados nos demuestran que el algoritmo falla muchísimo en las re-identificaciones y que no sería válido para esta tarea y contexto, pero nos permite demostrar nuestro objetivo del proyecto, ya que apreciamos la diferencia entre los valores con y sin mascarilla, que nos permite concluir acerca de la afectación del método de re-identificación. Por tanto, si un problema son las caras di-

minutas, vamos a probar con un algoritmo que analice una mayor área del corredor, por ello, mostramos la Tabla 5.7 con los resultados de AlignedReID.

Este algoritmo muestra mejores resultados que el anterior, principalmente, porque obtiene las características que utiliza en la re-identificación a partir del cuerpo del corredor, cuya área es mucho mayor que solamente la cara. Observamos que los resultados rondan entre el 72,01 % y el 94,57 %. En este experimento se utilizan los modelos: Cuhk03_Resnet50, DukeMTMCReID_Resnet50 y Market1501_Resnet50, la distancia alineada y la original, y también se utilizan las métricas del mAP y del rank 1. En este algoritmo también sucede que el uso de mascarilla afecta al método de re-identificación, ya que los valores del grupo sin mascarilla son mayores que los del grupo con mascarilla. Esto ocurre por lo mismo que comentamos anteriormente, el algoritmo falla en mayor medida si el corredor usa mascarilla, posiblemente porque existe un área oculta con la mascarilla con información más relevante para la re-identificación. Solamente en el caso del rank 1 vemos que los valores no son los esperados, creemos que puede deberse a las diferencias entre el mAP y el rank 1, mientras que el rank 1 se centra en la distancia mínima en su procedimiento y el mAP en múltiples factores como el IoU, puede ocurrir en el rank 1 no discrimine tan bien cuando se analiza el cuerpo entero como una cara, por ejemplo, distintos corredores con la misma camiseta podrían confundirse y encima solamente tenemos en cuenta un único corredor, el que tiene distancia mínima. Por otro lado, verificamos que la diferencia con y sin mascarilla es menor en AlignedReID que en DeepFace, posiblemente porque el área relativa de la mascarilla con respecto al cuerpo es menor que el área que ocupa la mascarilla con respecto a la cara, ya que AlignedReID lo realiza sobre el cuerpo, mientras que DeepFace sobre la cara. Además, observamos que las métricas para la distancia alineada y la original son muy similares en la mayoría de los casos. Esto es normal ya que utilizamos imágenes de los cuerpos que en la mayoría de los casos están alineadas.

AlignedReId

Sin mascarilla

mAP

	Cuhk03_Resnet50 (%)	**DukeMTMCReID_Resnet50 (%)**	**Market1501_Resnet50 (%)**
Alineada	72,01	85,61	**88,21**
Original	73,98	86,74	87,64

Rank 1

	Cuhk03_Resnet50 (%)	**DukeMTMCReID_Resnet50 (%)**	**Market1501_Resnet50 (%)**
Alineada	80,91	90,79	**92,79**
Original	83,71	92,12	91,99

Con mascarilla

mAP

	Cuhk03_Resnet50 (%)	**DukeMTMCReID_Resnet50 (%)**	**Market1501_Resnet50 (%)**
Alineada	69,63	82,69	**85,60**
Original	71,81	83,04	84,88

Rank 1

	Cuhk03_Resnet50 (%)	**DukeMTMCReID_Resnet50 (%)**	**Market1501_Resnet50 (%)**
Alineada	81,70	90,34	**94,57**
Original	84,54	91,79	94,32

Tabla 5.7: Métricas re-identificación AlignedReID.

Capítulo 6

Conclusiones

Los experimentos realizados durante este proyecto en la competición de LPA Trail 2020 han permitido alcanzar el objetivo buscado. Comenzamos el trabajo intentando determinar si los métodos de detección de mascarillas y re-identificación de corredores son afectados por el uso de la mascarilla en los participantes de la competición.

Para ello, hemos implementado una solución software. Basada en una línea común que extrae los fotogramas y detecta los cuerpos con Yolov4_DeepSort, y dos líneas de trabajo dependientes de la común. La primera, la de detección de mascarillas que detecta las caras y mascarillas con InsightFace, etiqueta las imágenes con la aplicación de etiquetado 'labeledImage', calcula la métrica de exactitud y agrupa los corredores por identificadores para mostrar una estadística. La segunda, la línea de re-identificación de corredores, que obtiene las imágenes de cuerpos, obtiene las carpetas según el identificador del corredor, corrige las imágenes y renombra los directorios, renombra las imágenes, obtiene los vectores de características y calcula las métricas del rank 1 y del mAP.

Tras aplicar esta solución, en la detección de mascarillas comprobamos que se ve afectada ya que en la métrica de la exactitud existe una diferencia significativa entre el grupo con mascarilla (85,68 %) y el grupo sin mascarilla

(70,73 %), claramente el uso de la mascarilla mejora la fiabilidad de InsightFace, es más fácil detectar un polígono regular como es una mascarilla que una cara con distintas particularidades.

Con respecto a la re-identificación de corredores también contrastamos que existen diferencias significativas entre los participantes con y sin mascarilla, por ejemplo, en DeepFace el mAP se sitúa entre el 6 % y el 17 % en el grupo sin mascarilla. En cambio, en el grupo con mascarilla esta métrica ronda alrededor del 3 % y 9 %. Por tanto, concluimos que la re-identificación se ve afectada por el uso de mascarilla, y concretamente si un corredor lleva mascarilla el método falla en mayor medida. La misma conclusión la alcanzamos a través de la métrica del rank 1, sin mascarilla alrededor del 2 % al 28 % y con mascarilla alrededor del 3 % al 13 %. En el caso de AlignedReID, concluimos de igual manera en el mAP, las diferencias entre ambos grupos también son significativas, sin mascarilla alrededor de 72 % al 89 % y con mascarilla alrededor del 68 % al 86 %. La re-identificación se dificulta al llevar puesta el corredor la mascarilla, posiblemente porque las características valiosas para la re-identificación las obtenemos de la cara del participante, y si se oculta una parte de ella con una mascarilla, obtendremos menos características relevantes para extraer.

Por otro lado, confirmamos que el método idóneo para la re-identificación es AlignedReID en este contexto antes que DeepFace, debido a las altas tasas de éxito que alcanza AlignedReID en comparación a DeepFace, ya que AlignedReID se basa en el análisis del cuerpo del corredor mientras que DeepFace solamente en la cara. En este contexto, es más interesante realizar un análisis del cuerpo que de la cara, debido a la diminuta área que ocupa la cara en comparación al cuerpo del corredor en la imagen.

Con el desarrollo de este proyecto se abren varias líneas de trabajo futuro, proponemos normalizar las imágenes en base a los ojos del corredor para mejorar los resultados de DeepFace, ya que los ojos son una parte de la cara que siempre están descubiertos tanto si el corredor usa mascarilla como si no la lleva puesta. Además, nos gustaría realizar un análisis similar pero suprimiendo el fondo de las imágenes, ya que las áreas con información relevante para

los métodos de re-identificación y detección son la cara y el cuerpo, y no el fondo de la imagen que lo que puede ocasionar es una mayor tasa de fallos.

En definitiva, hemos alcanzado el objetivo de este trabajo y alcanzado las métricas que esperábamos y con un porcentaje de aciertos muy elevado. Desde un punto de vista más personal, considero que ha sido muy enriquecedor conocer todas estas herramientas y tecnologías basadas en inteligencia artificial y su posible aplicabilidad al mundo real, considero que la inteligencia artificial está creciendo de forma exponencial y su utilidad práctica es cada vez mayor. Por tanto, en un futuro con el avance de esta tecnología se podría resolver el problema de controlar que los corredores cumplan el reglamento de una carrera popular sin revisar manualmente las imágenes.

Bibliografía

[1] "Redes neuronales convolucionales," https://www.juanbarrios.com/redes-neurales-convolucionales/, (Accedido el 06/26/2021).

[2] A. Penate-Sanchez, D. Freire-Obregon, A. Lorenzo-Melián, J. Lorenzo-Navarro, and M. Castrillón-Santana, "Tgc20reid: A dataset for sport event re-identification in the wild," *Pattern Recognition Letters*, vol. 138, pp. 355–361, 2020.

[3] "theaiguyscode/yolov4-deepsort: Object tracking implemented with yolov4, deepsort, and tensorflow." https://github.com/theAIGuysCode/yolov4-deepsort, (Accedido el 06/24/2021).

[4] "deepinsight/insightface: Face analysis project on pytorch and mxnet," https://github.com/deepinsight/insightface#retinafaceanticov, (Accedido el 06/24/2021).

[5] "serengil/deepface: A lightweight deep face recognition and facial attribute analysis (age, gender, emotion and race) framework for python," https://github.com/serengil/deepface, (Accedido el 06/24/2021).

[6] "michuanhaohao/alignedreid: Alignedreid+: Dynamically matching local information for person re-identification." https://github.com/michuanhaohao/AlignedReID, (Accedido el 06/24/2021).

[7] "Breaking down mean average precision (map)," https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52, (Accedido el 06/26/2021).

[8] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 1, pp. 23–38, 1998.

[9] R.-L. Hsu, M. Abdel-Mottaleb, and A. K. Jain, "Face detection in color images," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 696–706, 2002.

[10] S. Tang, M. Andriluka, B. Andres, and B. Schiele, "Multiple people tracking by lifted multicut and person re-identification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3539–3548.

[11] B. K. Gunturk, A. U. Batur, Y. Altunbasak, M. H. Hayes, and R. M. Mersereau, "Eigenface-domain super-resolution for face recognition," *IEEE transactions on image processing*, vol. 12, no. 5, pp. 597–606, 2003.

[12] "Objetivos y competencias del grado en ingeniería informática, web de la escuela de ingeniería informática de la ulpgc," https://www.eii.ulpgc.es/tb_university_ex/?q=objetivos-y-competencias-del-gii, (Accedido el 06/07/2021).

[13] "Boe.es - boe-a-1978-31229 constitución española." https://www.boe.es/buscar/act.php?id=BOE-A-1978-31229, (Accedido el 06/07/2021).

[14] "El derecho a la propia imagen," https://www.jraulfernandez.es/el-derecho-a-la-propia-imagen/, (Accedido el 06/07/2021).

[15] "Reglamento lpatrail.com," http://www.lpatrail.com/reglamento-2/, (Accedido el 06/07/2021).

[16] J. F. V. Serrano, A. B. M. Díaz, Ángel Sánchez Calle, and J. L. E. Sánchez-Marín, *Visión por Computador, 2ta Edicion.* Dykinson, 2003, ch. 1.3, p. 16.

[17] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] A. Bedagkar-Gala and S. K. Shah, "A survey of approaches and trends in person re-identification," *Image and Vision Computing*, vol. 32, no. 4, pp. 270–286, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885614000262

[20] P. A. Marín Reyes, I. Irigoien, B. Sierra, J. Lorenzo-Navarro, M. Castrillón Santana, and C. Arenas, "Ilra: Novelty detection in face-based intervener re-identification," *Symmetry*, vol. 11, no. 9, p. 1154, 2019.

[21] Q. Leng, M. Ye, and Q. Tian, "A survey of open-world person re-identification," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 4, pp. 1092–1108, 2020.

[22] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," *arXiv preprint arXiv:1603.08029*, 2016.

[23] L. Bazzani, M. Cristani, and V. Murino, "Symmetry-driven accumulation of local features for human characterization and re-identification," *Computer Vision and Image Understanding*, vol. 117, no. 2, pp. 130–144, 2013.

[24] P. Hernández-Carrascosa, A. Penate-Sanchez, J. Lorenzo-Navarro, D. Freire-Obregón, and M. Castrillón-Santana, "Tgcrbnw: A dataset for runner bib number detection (and recognition) in the wild," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 9445–9451.

[25] Y. Choi, Y. Napolean, and J. C. van Gemert, "The arm-swing is discriminative in video gait recognition for athlete re-identification," *arXiv preprint arXiv:2106.11280*, 2021.

[26] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the covid-19 pandemic," *Measurement*, vol. 167, p. 108288, 2021.

[27] S. A. Sanjaya and S. Adi Rakhmawan, "Face mask detection using mobilenetv2 in the era of covid-19 pandemic," in *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI)*, 2020, pp. 1–5.

[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[29] "Lpa trail, trail en las palmas de gran canaria," http://www.lpatrail.com/#, (Accedido el 06/08/2021).

[30] "Lpa trail 2020 online," https://www.chronorace.es/carreras/g-live.html?f=2020-002/LPA%20TRAIL%202020.clax, (Accedido el 06/08/2021).

[31] S. Kumar, Vishal, P. Sharma, and N. Pal, "Object tracking and counting in a zone using yolov4, deepsort and tensorflow," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 2021, pp. 1017–1022.

[32] J. Guo, J. Deng, A. Lattas, and S. Zafeiriou, "Sample and computation redistribution for efficient face detection," *arXiv preprint arXiv:2105.04714*, 2021.

[33] X. An, X. Zhu, Y. Xiao, L. Wu, M. Zhang, Y. Gao, B. Qin, D. Zhang, and F. Ying, "Partial fc: Training 10 million identities on a single machine," in *Arxiv 2010.05222*, 2020.

[34] J. Deng, J. Guo, T. Liu, M. Gong, and S. Zafeiriou, "Sub-center arcface: Boosting face recognition by large-scale noisy web faces," in *Proceedings of the IEEE Conference on European Conference on Computer Vision*, 2020.

[35] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, "Retinaface: Single-shot multi-level face localisation in the wild," in *CVPR*, 2020.

[36] J. Guo, J. Deng, N. Xue, and S. Zafeiriou, "Stacked dense u-nets with dual transformers for robust face alignment," in *BMVC*, 2018.

[37] J. Deng, A. Roussos, G. Chrysos, E. Ververas, I. Kotsia, J. Shen, and S. Zafeiriou, "The menpo benchmark for multi-pose 2d and 3d facial landmark localisation and tracking," *IJCV*, 2018.

[38] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690–4699.

[39] S. I. Serengil and A. Ozpinar, "Lightface: A hybrid deep face recognition framework," in *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE, 2020, pp. 23–27.

[40] H. Luo, W. Jiang, X. Zhang, X. Fan, J. Qian, and C. Zhang, "Alignedreid++: Dynamically matching local information for person re-identification," *Pattern Recognition*, vol. 94, pp. 53–61, 2019.

[41] X. Zhang, H. Luo, X. Fan, W. Xiang, Y. Sun, Q. Xiao, W. Jiang, C. Zhang, and J. Sun, "Alignedreid: Surpassing human-level performance in person re-identification," *arXiv preprint arXiv:1711.08184*, 2017.

[42] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[43] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017.

[44] S. Chen, Y. Liu, X. Gao, and Z. Han, "Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices," in *Chinese Conference on Biometric Recognition*. Springer, 2018, pp. 428–438.

[45] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[46] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[47] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[48] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[49] T. Baltrušaitis, P. Robinson, and L.-P. Morency, "Openface: an open source facial behavior analysis toolkit," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2016, pp. 1–10.

[50] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708.

[51] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C.-C. Loy, *et al.*, "Deepid-net: Deformable deep convolutional neural networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2403–2412.

[52] D. E. King, "Dlib-ml: A machine learning toolkit," *The Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.