UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Diseño de un umbral dinámico para la mejora de algoritmos de verificación facial a diferentes distancias

Autor:

D. Julio Díez Tomillo

Tutor:

Dr. D. Juan Carlos Aguado Manzano

VALLADOLID, SEPTIEMBRE 2022

TRABAJO FIN DE GRADO

TÍTULO:	**Diseño de un umbral dinámico para la mejora de algoritmos de verificación facial a diferentes distancias**
AUTOR:	**D. Julio Díez Tomillo**
TUTOR:	**Dr. D. Juan Carlos Aguado Manzano**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Ignacio de Miguel Jiménez**
VOCAL:	**Ramón José Durán Barroso**
SECRETARIO:	**Ramón de la Rosa Steinz**
SUPLENTE:	**Patricia Fernández Reguero**
SUPLENTE:	**Javier Aguiar Pérez**
SUPLENTE:	**María Jesús Verdú Pérez**

RESUMEN

Los sistemas de verificación facial están presentes en multitud de situaciones cotidianas que se encuentran en el día a día como por ejemplo desbloquear el teléfono móvil. Las verificaciones faciales desde drones son una posibilidad para no interferir o invadir la privacidad de la persona siendo verificada. Pero para ello, hace falta un sistema que pueda realizarlas desde largas distancias, lo que conlleva una resolución muy baja de la cara de la persona. En este proyecto se propone un sistema con umbrales dinámicos que varían en función de la distancia del dron a la persona para mejorar la precisión de los algoritmos de verificación facial sin tener que reentrenarlos. Se ha conseguido mejorar la precisión de todos los algoritmos de verificación facial comparados hasta en un 30% en distancias entre 2 y 30 metros, alcanzando como máximo un 96,8%. Además, se han usado dos métricas diferentes para poder compararlas y analizar cuál funciona mejor con cada algoritmo y a qué distancias. Con ello se ha logrado un sistema de verificación facial que puede funcionar en drones de bajo presupuesto sin tener que usar una cámara de muy alta resolución y coste. Con nuestro sistema se pueden realizar verificaciones de una cara teniendo una resolución únicamente de aproximadamente 15 pixeles de tamaño.

PALABRAS CLAVE

Verificación facial, Reconocimiento facial*, Detección facial, Distancia de coseno, Distancia euclídea, Umbral, Dron*

ABSTRACT

Face verification systems are used in everyday situations such as unlocking a smartphone. Face verifications from drones are a possibility to avoid invading the privacy of the person being verified. For that, a system that can verify at long distances with low resolution faces is needed. This project proposes a system with dynamic thresholds that vary according to the distance from the drone to the person to improve the accuracy of the face verification algorithms without further training. The accuracy has been improved by 30% in some cases between 2 and 30 meters achieving a maximum of 96.8%. Two metrics has been used to compare which one is better for each distance and algorithm. A face verification system has been achieved that can work in low-cost drones without a high resolution and expensive camera. Our system can perform face verification with a face resolution of only 15 pixels.

KEYWORDS

Face verification, Face recognition, Face detection, Cosine distance, Euclidean distance, Threshold, UAV

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mi tutor Juan Carlos por su paciencia conmigo durante el tiempo que he estado trabajando en este proyecto. Además de todas las correcciones y consejos que me ha dado para mejorar no solo en esta memoria, sino también en futuras.

En segundo lugar, a mis tutores de la University of the West of Scotland (UWS), Jose María Alcaraz y Qi Wang por darme la idea de este trabajo, además de proporcionarme los medios materiales para poder realizarlo.

También me gustaría agradecer a mis compañeros Ignacio y Gelayol por su ayuda en la realización de este proyecto, además de proporcionarme todas las explicaciones y consejos que podían darme.

Quiero agradecer a mi familia por su apoyo y ánimos durante el tiempo que estuve trabajando en este proyecto. En especial mis padres Elisa y Carlos, y mis hermanos Patricia y Javier.

Finalmente, me gustaría dar un agradecimiento especial también a todos mis amigos que han estado apoyándome, que me comprendían cuando no podía quedar por tener que escribir la memoria y que me ofrecieron toda la ayuda que podían: Ángel, Jimena, Moha, Kike, Nacho y Luis.

A todos ellos, muchas gracias.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO I

INTRODUCCIÓN

Este primer capítulo del documento tiene como objetivo definir la motivación del proyecto y cuáles son los objetivos que se esperan conseguir con su realización. También se listarán los medios disponibles para cumplir esos objetivos y cuál es la estructura de todo el documento.

Motivación del proyecto

Actualmente, el reconocimiento facial se puede encontrar en casi cualquier lugar de nuestra vida cotidiana. Cuando vamos a usar nuestro teléfono móvil, podemos usar nuestro rostro para desbloquearlo sin tener que introducir ninguna contraseña. Si nos vamos a ir de viaje, en el aeropuerto en todos los controles se realizan tareas de reconocimiento, ya sea de forma manual con un policía comprobando nuestra identidad o de forma automática como viene siendo cada más habitual mediante una cámara y un algoritmo de verificación facial. También, mientras andamos por el aeropuerto las cámaras CCTV nos van reconociendo en tiempo real por motivos de seguridad.

En algunas ciudades, como por ejemplo en Londres, la policía está empezando a usar tecnologías de reconocimiento facial en las cámaras repartidas por la ciudad para reconocer a delincuentes y poder detenerles [1]. En esta misma ciudad se decidió probar un sistema de reconocimiento facial portátil de la policía. Sin embargo, ganó más detractores que simpatizantes al tener numerosos falsos positivos que llevaron que la policía realizara identificaciones innecesarias de muchos peatones [2]. La mayor queja surgió de que la mayoría de los falsos positivos eran de personas de color, lo que llevó a tildar al sistema de racista. Esto se debe a que el algoritmo no fue entrenado correctamente, ya que no se introdujeron el número suficiente de imágenes de todo tipo de etnias para hacer que el sistema pudiera reconocer a cualquier persona de forma correcta. Por ello, se hace necesaria la creación de un *dataset* con una variedad étnica suficiente como para evitar que los algoritmos cometan más falsos positivos en algunas etnias que en otras.

Además, la mayoría de los reconocimientos faciales se suelen hacer a largas distancias, ya que son menos intrusivas para el usuario, al no tener que acercar la cámara hasta la cara. Debido a esto, se hace necesario entrenar a los diferentes algoritmos para poder reconocer a las personas desde largas distancias. El problema que se plantea es la baja resolución de los rostros a medida que nos alejamos del usuario, por ello se hacen necesarias técnicas para mejorar la precisión de estos algoritmos minimizando el número de falsos positivos. Al no haber tampoco una gran cantidad de *datasets* de caras a largas

distancias es muy difícil reentrenar los algoritmos para permitir reconocer ese tipo de imágenes, por lo que las técnicas usadas deberán usarse modificando el diseño de reconocimiento facial. Con eso conseguimos evitar tener que reentrenar los algoritmos.

Lo más importante es intentar minimizar los falsos positivos para cualquier sistema de reconocimiento facial. Un falso positivo puede ser muy peligroso ya que puede llevar a detener a una persona equivocada, desbloquear el móvil con un rostro diferente o dejar pasar a alguien en un aeropuerto con un pasaporte robado. Por ello, es preferible un falso negativo que un falso positivo. Además, al no haber muchos estudios sobre el reconocimiento facial desde largas distancias, se hace necesario no solo explorar este campo, sino también mejorar los algoritmos en estas situaciones.

Objetivos

En este Trabajo de Final de Máster (TFM) se han planteado los siguientes objetivos:

- Realizar un análisis sobre el estado del arte actual de los sistemas de detección, reconocimiento y verificación facial. Para ello, se explicarán los algoritmos más usados actualmente y se compararán comentando ventajas y desventajas de cada uno de ellos.
- Creación de un *dataset* para el entrenamiento y *testing* de algoritmos de verificación facial a diferentes distancias y alturas desde el objetivo a la cámara. Este *dataset* deberá ser lo más completo posible para lograr unos resultados lo más fieles a la realidad, incluyendo personas de diferentes géneros, etnias y edades.
- Comparación de la precisión de los principales algoritmos de verificación facial en función de la distancia y la altura a la persona que se quiere verificar. También se compararán otros parámetros como por ejemplo su tiempo de inferencia.
- Propuesta de un nuevo diseño de verificación facial para mejorar la precisión de los algoritmos a diferentes distancias y alturas del objetivo. El nuevo diseño se basa en modificar dinámicamente el umbral de verificación en función de la distancia al objetivo. También se probará el nuevo diseño comparando la precisión obtenida con el nuevo diseño frente al antiguo con umbrales estáticos previamente definidos.

Metodología

El desarrollo del proyecto se va a dividir en tres fases diferentes:

1. **Fase preparatoria:** En esta primera fase se analizará en primer lugar el estado del arte de los diferentes algoritmos de detección y verificación facial para pasar a elegir únicamente los que tienen las mejores características. A continuación, se reunirá a un grupo de personas para obtener un *dataset* desde un dron para poder

realizar una comparativa entre los diferentes algoritmos a diferentes distancias. En esta comparativa se analizarán diferentes parámetros como el *accuracy* o el tiempo de inferencia entre otros.

2. **Fase de implementación:** En la siguiente fase se pasará a implementar en el código la propuesta de diseño. Partiendo de los resultados obtenidos en la fase anterior se analizarán usando octave para poder obtener los mejores umbrales para nuestros algoritmos y así poderlos implementar en nuestro nuevo diseño.
3. **Fase de análisis:** Finalmente, en esta última fase se pondrá a prueba nuestro diseño usando de nuevo el *dataset* creado anteriormente. Se analizarán los mismos parámetros de los algoritmos que en la primera fase y se compararán para ver que mejora se ha obtenido usando el nuevo diseño.

Medios

Para cumplir los objetivos previamente definidos, se va a hacer uso de los siguientes medios:

- Para obtener el *dataset* a diferentes distancias y alturas se usará un dron DJI Mini 2, que incorpora una cámara 4K (3840x2160 px) capaz de grabar video a 30 FPS. Además, los videos se grabarán en un campo apartado de la ciudad fuera de las zonas restringidas de vuelo. El usuario que ha pilotado el dron disponía de las licencias necesarias para poder realizar los vuelos.
- Para la ejecución de los algoritmos se usará un ordenador con las siguientes características:
    - **Tarjeta gráfica:** NVIDIA GeForce GTX TITAN X con 12 GB de memoria integrada
    - **Almacenamiento:** 3 TB en disco SSD
    - **Procesador: Intel Core i7-3930K**
    - **Sistema Operativo:** Focal Ubuntu 20.04.3
    - **Plataforma de programación:** Pycharm
- Para el desarrollo del proyecto se han utilizado los siguientes lenguajes de programación:
    - Python versión 3.8
    - GNU Octave
- Para el *testing* de los diferentes algoritmos de verificación facial se usarán las librerías de TensorFlow versión 2.9.1 y Keras [3].
- Para el cálculo de los nuevos umbrales dinámicos y el manejo de los resultados obtenidos de los algoritmos se usará la aplicación GNU Octave.

Organización del documento

El documento está organizado en seis capítulos diferentes:

- En el primer capítulo se hace una introducción de todo el proyecto, explicando las motivaciones para realizarlo y los objetivos que se esperan conseguir con la realización de este junto con los medios disponibles para cumplirlos.
- En el segundo capítulo se explicarán y definirán diferentes conceptos teóricos sobre la verificación facial que son necesarios para entender el resto del documento.
- En el tercer capítulo se analizará el estado del arte de la verificación facial. Se explicarán los *datasets*, algoritmos de detección y verificación facial existentes actualmente. También se explicarán las métricas que se pueden usar para el cálculo de las distancias de los vectores de características faciales.
- En el cuarto capítulo se propondrá y explicará el nuevo diseño de verificación facial, así como su programación y cómo se han obtenido los diferentes parámetros para el mismo. También se explicará el *dataset* creado para poder obtener estos resultados.
- En el quinto capítulo se calcularán los nuevos umbrales para nuestro diseño usando el algoritmo explicado en el capítulo anterior y se mostrarán los que se han obtenido usando nuestro *dataset*.
- En el sexto capítulo se compararán los resultados experimentales obtenidos con el nuevo diseño frente al anterior usando los diferentes algoritmos y métricas.
- Finalmente, en el capítulo siete se expondrán unas conclusiones sobre el trabajo realizado y posibles líneas de trabajo futuro.

CAPÍTULO II

MARCO TEÓRICO

En este segundo capítulo se van a explicar algunos conceptos teóricos necesarios para poder entender el resto del documento. Entre otros se explicarán las diferencias entre el reconocimiento y la verificación facial o cuales son las etapas más habituales a la hora de realizar verificaciones faciales.

Reconocimiento facial y Verificación Facial

Habitualmente se suele confundir reconocimiento y verificación facial, usando muchas veces el primer término incorrectamente. Verificación facial consiste en comprobar que dos personas son la misma. Por ejemplo, cuando se quiere desbloquear un teléfono móvil usando la cara, se está haciendo una verificación facial, ya que solo se está comprobando, si la persona que quiere desbloquear el teléfono es la misma que está almacenada como propietario en la base de datos del teléfono.

Por otro lado, cuando se habla de reconocimiento facial se trata de identificar a una persona. Para ello, se hace uso de bases de datos con una gran cantidad de identidades y se compara la persona con las de la base de datos intentando encontrar una coincidencia. Un ejemplo de su uso es identificar personas en aeropuertos o por la calle, como se ha comentado en la introducción.

En la **Fig. 1** se puede ver el proceso de verificación facial de una forma muy simplificada. Lo primero que debemos tener es una cara almacenada en una base de datos que es la cara que usaremos para comparar. Después necesitaremos la cara que va a ser verificada como entrada a nuestro proceso. Esta cara se pasará a través de nuestro flujo de verificación facial para extraer sus características o *features* por su denominación en inglés. Estas *features* se extraen habitualmente a través de redes neuronales convolucionales. Una vez sacadas se comparan con las de la otra cara que usamos para comparar. En función de esta comparación se obtiene un resultado del proceso, que puede ser positivo o negativo, es decir, si son la misma persona o personas diferentes.

Por otro lado, en la **Fig. 2** se puede apreciar el proceso de reconocimiento facial de forma simplificada. Como se puede ver, la única diferencia entre los dos procesos es que en vez de comparar la cara a ser reconocida o verificada con una única persona se compara con una base de datos de caras. Entonces, el proceso empieza de la misma manera extrayendo las *features* de la cara a ser reconocida. A continuación, serán comparadas con las *features* de todas las caras contenidas en la base de datos. Finalmente, el resultado del proceso de reconocimiento facial será la identidad de la persona en la base de datos que coincide con la persona a reconocer. También se puede dar el caso de que la persona a reconocer no

coincida con ninguna de las personas de la base de datos por lo que el resultado será que la persona no ha podido ser reconocida.

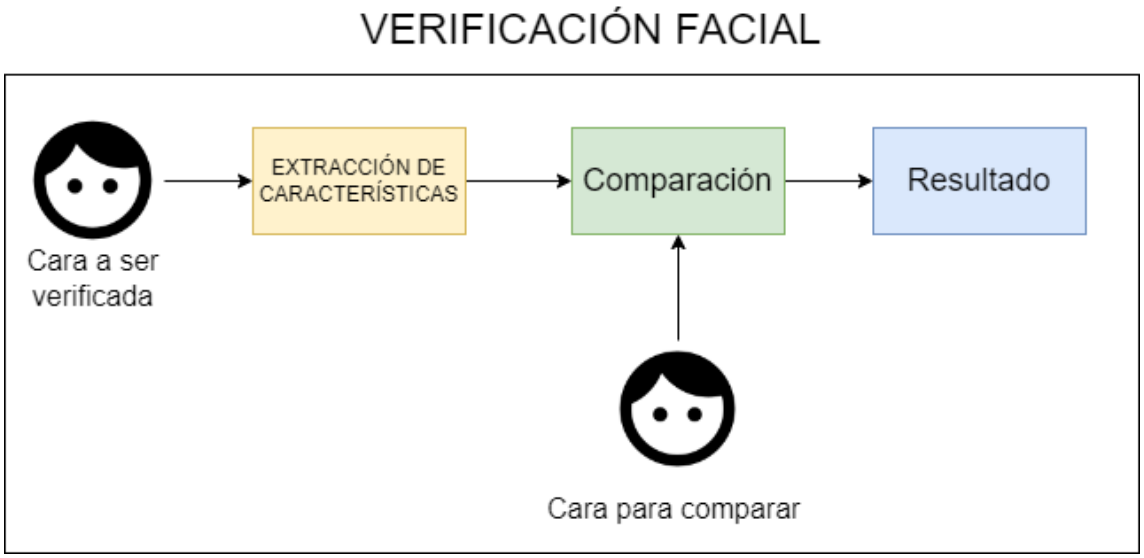


Fig. 1. *Proceso simplificado de verificación facial*

Como se ha explicado, la principal diferencia entre los dos métodos es que en uno se compara si dos caras son de la misma persona, mientras que en el reconocimiento facial se dice a qué identidad pertenece una cara. Cada uno de los métodos tiene utilidades diferentes y cada uno se utilizará en el caso de uso que sea necesario.

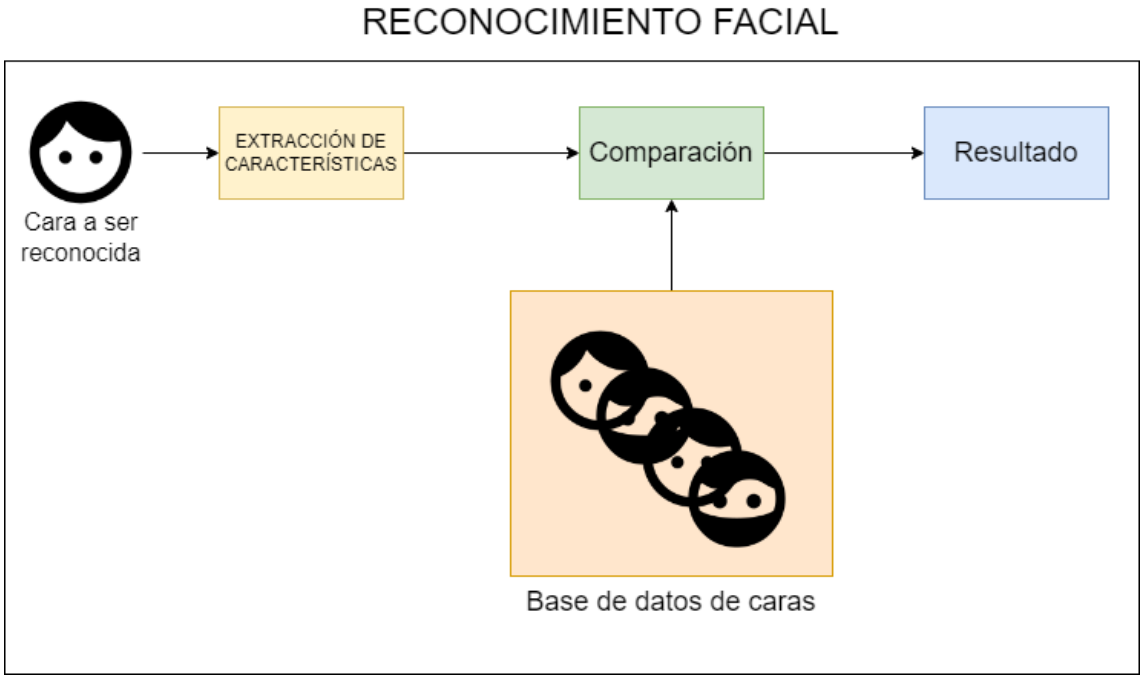


Fig. 2. *Proceso simplificado de reconocimiento facial*

El reconocimiento facial además es un proceso mucho más lento que la verificación facial, ya que hay que comparar con todas las imágenes presentes en la base de datos. A mayor tamaño de la base de datos mayor será el tiempo que llevará hacer el reconocimiento facial, por lo que el tiempo que se tarda en realizar el reconocimiento facial es variable en función de la base de datos.

Por otro lado, en la verificación facial, al solo tener que comparar con una cara, el tiempo que tarda es más o menos estable, variando únicamente en función del preprocesamiento que hay que hacer en la imagen de entrada. Con ello es más fácil conseguir el objetivo de realizar verificación facial en tiempo real, ya que es un proceso más rápido que el reconocimiento y con un tiempo más o menos constante.

Otra diferencia entre los dos métodos es la manera en la que se toma la decisión final para obtener los resultados. En el reconocimiento facial se puede usar un algoritmo KNN (K *Nearest Neighbour* o K vecinas más cercanas en español) [3]. Este almacena todas las clases disponibles (todas las caras disponibles en la base de datos) y clasifica los datos (las nuevas caras) en función de su similaridad. Es decir, asocia la cara que queremos reconocer con la cara más cercana en nuestra base de datos. A eso se puede añadir un umbral mínimo para el caso en el que la cara a identificar no está contenida en la base de datos.

Por otro lado, en la verificación facial el procedimiento para obtener el resultado suele ser más sencillo. Para ello, únicamente se calcula la distancia entre las características de las dos caras a comparar mediante alguna métrica y si supera un umbral mínimo se identifican como la misma persona y si se encuentra por debajo serán personas diferentes. Por ello, es crítico encontrar un umbral óptimo, ya que nos permitirá minimizar el número de falsos positivos de nuestros algoritmos a la vez que aumentamos su precisión. Si el umbral no se elige de forma correcta puede llevar a que se verifique erróneamente a muchos usuarios, ya sea por falsos positivos (al poner un umbral muy laxo) o falsos negativos (al poner un umbral demasiado restrictivo).

Pipeline usado para verificación facial

Este proyecto se va a centrar únicamente en la verificación facial, por lo que solo trataremos de identificar si dos caras pertenecen a la misma persona o no. El proceso de verificación facial necesita unas etapas previas para preprocesar las imágenes que van a ser utilizadas para maximizar la precisión de los algoritmos. En función de qué algoritmos se usen, los *pipelines* (el proceso) de verificación facial pueden cambiar. Además, cada autor usa uno diferente pudiendo añadir, quitar o unir etapas en función de lo que se necesite. A pesar de ello, en la **Fig. 3** se puede ver un *pipeline* básico completo de verificación facial con las etapas típicas más usadas. Este *pipeline* se puede dividir en cuatro partes básicas bien diferenciadas [4]:

- **Detección facial:** La primera etapa casi siempre será la de detección facial. Su objetivo, como bien dice su nombre es detectar una cara en la imagen. Habitualmente, en las imágenes que se usan para verificación facial, la cara

representa únicamente una pequeña parte de toda la imagen. Por ello, lo primero que hay que hacer es detectar la cara. En una imagen se pueden encontrar más de una cara, por ejemplo, si es una cámara en un aeropuerto o situada en la calle. Todas las caras detectadas en una imagen habrá que verificarlas para ver si coinciden con la persona objetivo. Para realizar la detección facial, en imágenes muy lejanas o de baja resolución, donde la cara es muy pequeña en pixeles, a veces es conveniente usar previamente un algoritmo de detección de personas, y a continuación detectar la cara usando solo el recorte de la persona. Con ello se puede aumentar notablemente el número de caras localizadas en una imagen. En el siguiente capítulo se analizarán los algoritmos más usados para realizar detecciones faciales.

***Fig. 3**. Ejemplo de proceso de verificación facial en cuatro etapas*

- **Recorte y alineación facial:** La siguiente etapa será procesar la detección facial anterior. Para ello, lo primero que se debe hacer es realizar un recorte de la cara. Se cogen las coordenadas obtenidas por el algoritmo de detección facial que dice donde se encuentra la cara y se recortan esas coordenadas de la imagen completa. Con ello, se consigue tener únicamente una imagen de la cara. A continuación, para mejorar la precisión de la verificación facial se puede realizar un alineamiento facial. Si la cara está torcida o girada se pueden usar algoritmos para colocar correctamente la cara, y que sea más similar a la que queremos comparar. En la **Fig. 4** se puede ver un ejemplo de alineamiento facial, en el que una cara torcida se transforma en la misma pero bien posicionada y recta para lograr una mejor verificación facial. Se puede ver como primero se gira la imagen para ponerla recta y después se realiza el recorte de la cara únicamente. Estos dos pasos

se pueden realizar también de forma inversa, realizando primero el recorte y después el alineamiento.

Fig. 4. *Ejemplo de recorte y alineación facial. [5]*

- **Técnicas de super resolución:** Una vez tenemos la cara recortada y alineada se puede añadir una etapa intermedia opcional antes de la verificación facial para aumentar la calidad de la imagen. Si la cara que hemos detectado estaba a bastante distancia o la cámara no tenía mucha resolución, la imagen recortada estará muy pixelada y será muy difícil realizar verificación facial con ella. Por ello, se pueden aplicar técnicas de super resolución para aumentar la calidad de la imagen y mejorar el número de pixeles de la cara para poder realizar una verificación más precisa. Un ejemplo de técnica que se puede aplicar es una red GAN (*Generative Adversarial Network*) llamada SRGAN (*Super Resolution* GAN) [6] o el mejorado ESRGAN (*Enhanced Super Resolution* GAN) [7]. Partiendo de una imagen con baja resolución, usando esta red se puede conseguir la misma imagen, pero con una resolución mayor, como se puede ver en el ejemplo de la **Fig. 5**. [8] El problema de utilizar este tipo de técnicas es que suelen tener un tiempo de inferencia muy alto, por lo que el tiempo de procesamiento de nuestro *pipeline* aumentaría notablemente. Por ello, habrá que tener un compromiso entre la velocidad y la precisión que queremos que tenga nuestro algoritmo.
- **Verificación facial:** La última etapa del proceso será realizar finalmente la verificación facial. Como entrada a esta etapa tendremos el rostro de una persona recortada, alineada y con una resolución adecuada gracias a las etapas anteriores. Dentro de esta etapa se realizará el proceso de verificación facial que consistirá en comparar esta cara que se ha preprocesado junto con otra previamente almacenada y como resultado obtendremos si ambas corresponden a la misma persona o no. En la **Fig. 6** se puede ver en qué consistiría la etapa de verificación de forma simplificada. Cada una de las caras se pasará por una red neuronal convolucional (CNN) para obtener sus características (a partir de ahora *features*). Esto nos dará como resultado un vector para cada una de ellas. Después se calcula la distancia entre ambos vectores usando una métrica, algunas de las cuales se

desarrollarán en el próximo capítulo. Después el valor de distancia se comparará con un umbral predefinido. Si la distancia es menor que el umbral se concluirá que son la misma persona, si la distancia es mayor, serán dos personas diferentes.

Fig. 5. *Ejemplo de uso de ESRGAN en dos imágenes de baja resolución. [8]*

Cabe destacar, además, que el conjunto de las dos CNN se le conoce como red siamesa o *siamese network* [9]. Esto se debe a que son dos CNN exactamente iguales, utilizan el mismo *backbone* y los mismos pesos. Por lo que, si introducimos la misma imagen de entrada para las dos, la distancia de los vectores resultantes será nula. Esto es así para poder obtener un valor de cero al comparar la misma imagen. Además, las dos imágenes que vamos a comparar deberán haber pasado el mismo preprocesamiento para poder obtener un resultado correcto, tanto la cara a verificar como la que usamos como base.

Con ello, se ha conseguido explicar el proceso completo que se sigue para poder verificar una cara. Si en una imagen hay más de una cara, se deberá realizar el preprocesamiento para cada una de ellas.

Normalmente, como se ha comentado previamente, la etapa de super resolución se suele omitir del proceso ya que tiene un coste computacional y tiempo de inferencia alto. Por lo que, si se quiere priorizar tener una verificación facial rápida, se omitirá esta etapa a costa de empeorar la precisión de nuestro algoritmo.

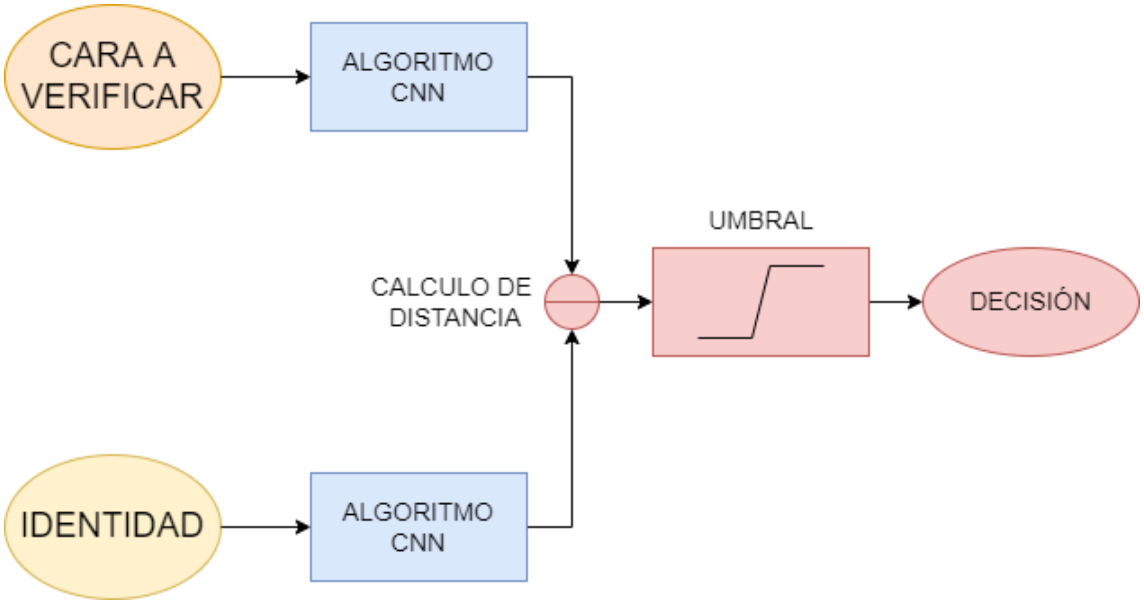


Fig. 6. *Últimas etapas simplificas del proceso de verificación facial*

CAPÍTULO III

ESTADO DEL ARTE - COMPARATIVA

En este tercer capítulo se analizará el estado del arte actual de la verificación facial. Para ello primero se tratarán los *dataset*s disponibles. A continuación, se hablará sobre los algoritmos de detección facial y algoritmos de extracción de *features* más utilizados. Además, se analizarán las métricas más usadas para el cálculo de distancias.

Datasets disponibles

Para poder entrenar correctamente una CNN es necesario tener un *dataset* completo y suficientemente grande. Normalmente deberán contener imágenes de personas que abarquen estadísticamente todas las etnias para entrenar al algoritmo para verificar y detectar a todas las personas. Además, se pueden añadir imágenes en situaciones complejas como por ejemplo baja luminosidad, bajas resolución o poses faciales raras para poder entrenar en más situaciones a los algoritmos.

Los *dataset*s para realizar verificación facial deberán estar compuestos de imágenes personas donde se les vea el rostro y que estén correctamente etiquetados con un ID para poder identificar a cada una de ellas. Es decir, para etiquetar estos *dataset*s solo hará falta etiquetar cada una de las imágenes con un ID para identificar a cada persona. Para los *dataset*s de detección facial harán falta imágenes donde aparezcan caras y que estén etiquetadas mediante las coordenadas en la imagen donde está la cara. Cada una de las imágenes tendrá unas coordenadas asociadas que corresponderán a cada una de las caras dentro de cada imagen.

A continuación, se van a explicar algunos de los *dataset*s más usados para entrenar o testear algoritmos de detección y verificación facial.

a) Labelled Faces in the Wild (LFW) [9]

Es un *dataset* público para verificación facial. Está compuesto de 13.233 imágenes de caras realizadas a una distancia cercana. 1.680 de las personas en el *dataset* tienen más de una imagen en el *dataset* y hay un total de 5.749 personas en él. Existen tres variantes de este *dataset* aparte del original, con diferentes métodos de alineamiento de las imágenes para ayudar en la verificación facial. Es uno de los *dataset*s más usados para testear los algoritmos de verificación y reconocimiento facial. Siempre que se diseña un nuevo algoritmo se compara con el resto usando este *dataset*. En la **Fig. 7** se pueden ver algunos ejemplos de las imágenes que contiene.

Como se puede ver en la figura, las imágenes en este *dataset* son desde una distancia bastante cercana, abarcando la cara la mayoría de la imagen. Por ello, este *dataset* no es útil para el objetivo de poder verificar a personas desde varias distancias. Pero servirá para comparar nuestro sistema con otros a distancias cercanas.

Fig. 7. *Ejemplos de imágenes en el dataset LFW [9]*

b) YouTube Faces (YTF) [10]

Este *dataset* junto con LFW es también muy usado para el testeo y comparativa de los algoritmos de verificación y reconocimiento facial. Consiste en videos descargados de la plataforma YouTube. Contiene 3.425 videos con 1.595 personas diferentes en ellos. El video más corto del *dataset* tiene una duración de únicamente 48 *frames*, mientras que el más largo es de 6.070 *frames*. La duración media de los videos es de 181,3 *frames*. Los videos están etiquetados para identificar que personas salen en cada uno de ellos. También se pueden encontrar variantes de este *dataset*, incluyendo uno con la detección facial ya realizada y con las caras alineadas, para facilitar el proceso de verificación facial y podernos concentrar únicamente en esta última etapa.

En **Fig. 8** se pueden ver algunos ejemplos de *frames* de este *dataset*. La fila inferior representa algunas de las imágenes más complicadas, incluyendo baja resolución, baja luminosidad o posiciones complejas de la cara para la verificación facial. Como se puede ver, este *dataset* también se compone principalmente de imágenes a cortas distancias, por

lo que la cara ocupa la mayoría del *frame*. Por ello, tampoco nos será muy útil para el objetivo de verificar a varias distancias.

Fig. 8. *Ejemplos de frames en los videos de YTF dataset [10]*

c) DroneSURF [11]

Este es el primer *dataset* para verificación facial que contiene imágenes grabadas desde un dron. Se compone de 200 videos de 58 personas diferentes, capturadas en 411.000 *frames*. Contiene más de 786.000 anotaciones faciales. Al ser grabado desde un dron contiene imágenes a diferentes distancias lo que le hace útil para nuestro estudio. El problema con este *dataset* es que es privado por lo que no se ha podido acceder a él. Se divide en dos partes diferentes. La primera consiste en vigilancia activa, y se basa en cuando se quiere seguir a un sujeto concreto usando el dron. Por ello los videos son siguiendo a unos metros de distancia a una persona concreta que se le pidió que caminara de un punto a otro. Los videos se graban de frente para poder coger correctamente la cara de la persona. El otro consiste en vigilancia pasiva. Usando el dron se graba un área sin seguir a una persona específicamente. El movimiento del dron es independiente del de las personas y su único objetivo es grabar lo que sucede en una zona concreta y no seguir a determinadas personas. En el *dataset* también se incluyen situaciones complejas para la verificación facial como por ejemplo baja o muy alta iluminación, obstáculos que impiden ver correctamente la cara de una persona o posiciones de la cara complejas.

Todos los sujetos en este *dataset* tenían entre 18 y 40 años mezclando hombres y mujeres. Por ello, un problema que tiene es que no se ha abarcado un rango más grande de edades

por lo que el *dataset* puede quedar incompleto. Por ello, y por ser un *dataset* privado, no se ha podido usar para el proyecto.

d) VGG-Face 2

Este *dataset* desarrollado por un grupo de la universidad de Oxford contiene 3,31 millones de imágenes de 9131 personas diferentes. Teniendo una media de 362,6 imágenes de cada persona. El *dataset* se obtuvo descargando las imágenes mediante búsquedas en Google Imágenes. Además, contiene grandes variaciones en posiciones, edades, luminosidad y etnias por lo que lo hace un *dataset* muy completo para realizar verificación y reconocimiento facial. Está completamente etiquetado cada una de las identidades de forma manual. Es una mejora del *dataset* creado por el mismo grupo en 2015, pero que tenía cinco veces menos de personas.

El problema de este *dataset* es que las imágenes han sido tomadas a cortas distancias como se puede ver en **Fig. 9**. Por lo que no es un *dataset* completo para poder evaluar los algoritmos a diferentes distancias de una persona.

Fig. 9. *Ejemplos de imágenes contenidas en el dataset VGG-Face 2 [12]*

e) DroneFace [13]

Este *dataset* se basa en imágenes que simulan ser grabadas desde un dron. Para ello se usó una cámara GoPro Hero3+ y un móvil HTC One M8 para las fotos cercanas. Se compone de personas (7 hombres y 4 mujeres) de las que se han tomado en total 2.057 imágenes. El objetivo de la creación del *dataset* es poder comprobar como las diferencias distancias y ángulos afectan a la hora de realizar reconocimiento o detección facial, además de cómo de buenas son estas técnicas en videos grabados desde drones.

Este *dataset* es útil para poder ver el efecto a diferentes distancias de los algoritmos y con diferentes ángulos de incidencia. El problema que tiene el *dataset* es que todos los sujetos eran personas entre 23 y 36 años, lo que es un rango muy acotado, además, de no incluir personas de diferentes etnias. Por todo ello, este *dataset* tampoco ha sido usado para este proyecto.

f) WiderFace [14]

Este es uno de los *datasets* más completos para entrenar algoritmos de detección facial. Se compone de 32.203 imágenes con un total de 393.703 caras etiquetadas con una gran variedad de posiciones, obstáculos y escalas. Incluye numerosas imágenes grabadas a largas distancias y con muchas personas en ellas por lo que es un *dataset* muy útil para poder detectar caras a diferentes distancias, como por ejemplo mediante videos grabados con un dron. Usando este *dataset* se han entrenado muchos algoritmos de detección facial como por ejemplo RetinaFace, que se desarrollará más adelante.

El *dataset* está dividido en 61 clases diferentes en función de los eventos o situaciones que muestran cada una de ellas. Está dividido en tres partes, el 40% para entrenamiento, un 10% para validación y un 50% para *testing*. En la **Fig. 10** se pueden ver algunos ejemplos de imágenes del *dataset* con las caras etiquetadas mostrando diferentes situaciones complejas que lo hacen un *dataset* muy completo para la detección facial.

Fig. 10. *Ejemplo de imágenes del dataset WiderFace mostrando diferentes situaciones complejas que pueden dificultar la detección facial [14]*

Tabla 1. *Comparativa de los diferentes datasets existentes*

Dataset	N.º Imágenes	Identidades	Distancia a la cara	Problemas
LFW	13.233	5.749	Cercana	Solo imágenes cercanas
YTF	621.000	1.595	Cercana	Solo imágenes cercanas
DroneSurf	411.451	58	Lejana	No disponible
VGG-Face2	3.31M	9.131	Cercana	Solo imágenes cercanas
DroneFace	2.057	58	Cercana y lejana	Poca variedad de identidades
WiderFace	32.203	393.703	Cercana y lejana	*Dataset* para detección facial

En la **Tabla 1** se puede ver un breve resumen de los *datasets* comparados con sus características principales.

Algoritmos de detección facial

Uno de los pasos más importantes de la verificación facial es la detección de las caras. Si este paso no se realiza correctamente la precisión del *pipeline* bajará notablemente. Si se detecta solo la mitad de la cara o si se coge muchos más pixeles de los que corresponden a la cara, las *features* de la cara que extraerán no se corresponderán totalmente, por lo que la verificación seguramente falle, bajando la *accuracy*. Por ello, es muy importante elegir un algoritmo de detección facial adecuado. Pero, como siempre, hay un compromiso entre la velocidad del algoritmo y su precisión. Por ello, si se quiere un algoritmo muy rápido, la precisión de este bajará. A continuación, se va a realizar un análisis de los mejores algoritmos actuales de detección facial para nuestro caso de uso.

a) Viola Jones [15]

Es uno de los algoritmos de detección facial más famosos. Fue desarrollado en el año 2001 por Paul Viola y Michael Jones, de ahí su nombre. El objetivo del algoritmo era detectar objetos en imágenes rápidamente y de forma precisa, pero funciona también perfectamente para detectar caras. A pesar de la antigüedad del algoritmo, sigue siendo usado ampliamente para realizar detecciones faciales rápidas, aunque en precisión ya haya sido superado por muchos algoritmos más modernos. Para realizar las detecciones combina cuatro conceptos diferentes: *Haar-like features*, *Integral Images*, el algoritmo *AdaBoost* y clasificación en cascada.

b) RetinaFace [16]

Este es uno de los algoritmos con mejores resultados de todos los existentes. Está entrenado usando el *dataset* WiderFace. Además de la detección facial, este algoritmo también es capaz de encontrar cinco puntos clave dentro de la cara: los dos ojos, la nariz y los dos extremos de la cara. Esto lo hace útil para aplicarlos en otras técnicas como por ejemplo para detectores de estado de ánimo, pudiendo reconocer si una persona está sonriendo o triste. También permite añadir de forma artificial máscaras a una persona, cambiar la cara de una persona y muchas más técnicas de tratamiento facial.

El algoritmo se puede aplicar habitualmente en dos *backbones* diferentes. El primero de ellos es ResNet-50. Es bastante complejo, su fichero de pesos tiene un gran tamaño, además de que el tiempo de inferencia es alto, pero consigue una precisión muy alta. El otro *backbone* es MobileNet0.5, que es muy ligero, teniendo un tiempo de inferencia bajo, pero baja con ello también la precisión del algoritmo. Por ello, en función de la aplicación que se quiera, si velocidad o precisión, se puede elegir un *backbone* u otro.

c) MTCNN [17]

Es otro de los algoritmos más usados actualmente por sus buenos resultados. Su nombre viene de Multi Task Cascaded Convolucional Network, es decir Red Convolucional en cascada multitarea. Este nombre se debe a que su estructura se divide en tres etapas diferentes que permiten predecir no solo la cara, sino también los puntos clave de la cara, al igual que hacía RetinaFace. Esta entrenado también con el *dataset* WiderFace. A mayores tiene un sistema de alineamiento facial, ahorrando así una etapa en el *pipeline* de verificación facial al hacerlo a la vez que la detección. Además, es un algoritmo más rápido que RetinaFace, aunque tiene una precisión más baja.

d) Dlib [18]

Es una librería de código abierto que tiene como objetivo desarrollar *software* de *machine learning*. Está programado originalmente en C++, pero hay implementaciones en Python. Al ser una librería, para usarlo únicamente hay que realizar una llamada a la función correspondiente y se obtiene el resultado. Hay opciones para poder reentrenar la librería, pero fue entrenado con un *dataset* lo suficientemente grande como para obtener buenos resultados. El problema que tiene es que el tiempo de inferencia es alto, pero consigue muy buena precisión en las detecciones.

e) SSD [19]

Es un método que permite detectar objetos en imágenes usando una única red neuronal. No está creado expresamente para detectar caras, pero se puede entrenar para ello. El entrenamiento del algoritmo se suele realizar con el *dataset* WiderFace, como la mayoría de los algoritmos de detección facial, ya que es el más completo existente actualmente.

Al no ser un algoritmo creado expresamente para la detección facial no consigue los buenos resultados que tienen otros algoritmos, pero lo compensa con la velocidad con la que realiza las detecciones. Tiene un tiempo de inferencia muy bajo en comparación con otros algoritmos al usar una red neuronal no muy compleja. Es un modelo de detección *single-shot* al igual que YOLO [20], otro algoritmo de detección de objetos muy usado en la actualidad.

Fig. 11. *Comparativa de los algoritmos de detección facial [21] [22]*

En la **Fig. 11** se puede ver una comparativa de cómo hace la detección facial cada uno de los algoritmos que se han visto. Además, se ha incluido el algoritmo que viene incorporado en la librería OpenCV de Python. Como se puede ver, tanto Dlib como OpenCV introducen parte del pelo de la persona y la oreja y no únicamente la cara. También, se puede apreciar que RetinaFace no realiza un alineamiento de la cara, mientras sí que lo hacen el resto de los algoritmos. Por último, hay que destacar que como se puede apreciar, MTCNN y RetinaFace son los que realizan una mejor detección, únicamente de la cara, sin incluir otras partes, como oreja o pelo.

Algoritmos de extracción de características

La parte más importante de todo el proceso es la extracción de las características de la cara. Esto se realiza mediante una red neuronal que tiene como entrada la imagen de la cara y se obtiene como resultado un vector correspondiente a las características. La elección del algoritmo adecuado es clave para tener una alta precisión a la hora de realizar la verificación facial. A continuación, se presentan los mejores algoritmos actuales para la extracción de características.

a) FaceNet [23]

Es un modelo de reconocimiento facial creado por Google. La red neuronal tiene como entrada una imagen de 160 pixeles y tiene como salida un vector de 128 dimensiones. Usa para el entrenamiento un *dataset* propio con 200 millones de imágenes y más de 8 millones de identidades. Tiene una precisión alta, pero también tiene un tiempo de inferencia alto.

b) OpenFace [24]

Es un algoritmo de código abierto muy ligero que consigue un tiempo de inferencia muy pequeño a costa de una precisión baja en comparación con otros algoritmos. Está entrenado con 500.000 imágenes combinando dos *datasets* diferentes: CASIA-WebFace [25] y FaceScrub [26]. Usa como entrada a la red una imagen cuadrada con un tamaño de 96 pixeles y tiene como salida un vector de 128 dimensiones. El modelo usa como *backbone Inception Resnet-V1*. El algoritmo solo se ha probado con el *dataset* LFW para obtener resultados, por lo que no se puede comparar adecuadamente con los otros algoritmos.

c) ArcFace [27]

Es un algoritmo desarrollado por el *Imperial College London*. Su implementación está realizada en Python y MXNet [30] y se puede encontrar en el siguiente enlace: *https://github.com/deepinsight/insightface*. El modelo usa un *backbone* ResNet-34. Como entrada a la red neuronal se usa una imagen cuadrada con un tamaño de 112 pixeles y la salida es un vector de 512 dimensiones. El algoritmo se ha entrenado con el *dataset* MS1MV2 que es una modificación de MS-Celeb-1M [28], no se ha hablado de él en apartados anteriores, pero tiene un total de 5,8 millones de imágenes. Además, es el algoritmo que consigue la mayor precisión y no tiene un tiempo de inferencia muy alto en comparación con otros.

d) VGG-Face [29]

Algoritmo desarrollado por la Universidad de Oxford junto con el *dataset* con el mismo nombre. Su arquitectura tiene 22 capas y 37 *deep units.* La entrada de la red es una imagen cuadrada con un tamaño de 224 pixeles. La salida de la red es un vector de 2622 dimensiones, siendo el mayor de los cuatro algoritmos. Por ello, el algoritmo consigue una gran precisión, pero al ser también muy pesado, el tiempo de inferencia es muy alto. Este algoritmo será muy útil para aplicaciones en las que se quiera muy buenos resultados sin importar el tiempo que tarde. Por ejemplo, en verificaciones faciales de seguridad en las que sea muy importante no tener falsos positivos.

Tabla 2. *Comparativa de los diferentes algoritmos de extracción de características faciales*

Algoritmo	Tamaño entrada	Tamaño salida	Precisión en LFW	Precisión en YTF	Tamaño pesos preentrenados	Imágenes de entrenamiento
FaceNet	160x160	128	98,87%	95,12%	88 MB	200 M
OpenFace	96x96	128	92,92%	-	14 MB	500k
ArcFace	112x112	512	99,83%	98,02%	131 MB	5,8 M
VGG-Face	224x224	2622	98,95%	97,30%	554 MB	2,6 M

Finalmente, en la **Tabla 2** se puede ver un resumen de la comparativa entre los cuatro algoritmos previamente comentados. Se puede ver el tamaño de la imagen que se usa como entrada a la red neuronal y el tamaño del vector de salida. También se compara la precisión en dos *datasets*: LFW e YTF, el tamaño del fichero de pesos preentrenados y el número de imágenes que se han usado para este entrenamiento.

Métricas de cálculo de distancias

Por último, se van a analizar diferentes métricas que se pueden usar para calcular la distancia entre los dos vectores de características de una cara. Cada una de estas métricas son más útiles para algunos algoritmos concretos, en función de cómo sea el método que usan para extraer las características. Las tres métricas más usadas para calcular la distancia en verificación facial son las siguientes.

a) Coseno [30]

La distancia de coseno se calcula como el coseno del ángulo que hay entre los dos vectores. Es decir, dos vectores serán parecidos no porque se encuentren cerca en el espacio, sino porque el ángulo entre ambos es pequeño. Suponiendo dos vectores que representan las características de una cara, es decir, la salida de una red neuronal, con *n* dimensiones y definidos como $a = (x_1, x_2, \dots, x_n)$ y $b = (y_1, y_2, \dots, y_n)$, la distancia de coseno se define como:

$$CD(a,b) = 1 - \frac{\sum_1^n x_i\, y_i}{\sqrt{\sum_1^n x_i^2}\sqrt{\sum_1^n y_i^2}}$$

A mayores valores de distancia significa que los vectores son más diferentes, mientras que, en valores pequeños, los vectores son más parecidos. Si se obtiene un valor de cero significa que los dos vectores son iguales.

b) Euclídea [31]

La distancia euclídea nos da la distancia entre dos puntos en el espacio, es decir, la longitud de la línea recta que los conecta. Suponiendo los mismos vectores que en el caso

anterior ($a = (x_1, x_2, \dots, x_n)$ y $b = (y_1, y_2, \dots, y_n)$) la distancia euclídea de ambos se calcularía con la siguiente fórmula:

$$ED(a, b) = \sqrt{\sum_i^n (x_i - y_i)^2}$$

Al igual que con el coseno, a mayores valores de distancia los vectores serán más diferentes, al estar más lejos en el espacio, mientras que cuanto más cerca estén más parecidos serán entre ellos.

c) Manhattan [31]

Esta distancia se define como la suma de la diferencia de las proyecciones de los dos vectores. Es decir, suponiendo los mismos vectores que en el caso anterior ($a = (x_1, x_2, \dots, x_n)$ y $b = (y_1, y_2, \dots, y_n)$) la distancia Manhattan de ambos se calcularía con la siguiente fórmula:

$$MD(a, b) = \sum_i^n |x_i - y_i|$$

Al igual que las dos distancias anteriores, a mayor valor de distancia, los vectores serán menos similares. Por lo que para verificar a una persona necesitaríamos un valor bajo de distancia.

CAPÍTULO IV

NUEVO DISEÑO PROPUESTO

En este capítulo se va a desarrollar el diseño propuesto para mejorar la precisión de diferentes algoritmos de verificación facial. Para ello, primero se explicará el nuevo *pipeline* usado y cuál es la nueva contribución en él. Después se explicará cómo se ha diseñado el umbral dinámico y como se han calculado los umbrales nuevos que se van a usar.

Nuevo *pipeline*

Como *pipeline* se va a usar una modificación del visto en el capítulo 2, pero basado plenamente en él. El *pipeline* diseñado se puede ver en la **Fig. 12**. Como se ha querido realizar todo el procesamiento de forma rápida, para acercarse a procesamiento en tiempo real (24 fps), se han omitido las etapas que más tiempo consumían, como por ejemplo el alineamiento o las técnicas de super resolución. Como se ha venido comentando, hay que encontrar un compromiso entre tiempo de inferencia y precisión, y se ha decidido que las mejoras de precisión que aportan estas etapas no compensan el tiempo que lleva realizarlas.

Por eso, se ha reducido el *pipeline* a únicamente cinco etapas. La primera es la detección de caras, a continuación, tenemos el preprocesamiento. Justo después tenemos ya la red siamesa y a sus salidas se realiza el cálculo de distancias y en función de eso se toma la decisión. El *pipeline* tiene dos parámetros de entrada, el video en el cual se quiere verificar la persona y la cara de la persona que se quiere verificar. La salida de la red es si esa persona ha sido verificada o no. A continuación, se va a explicar cada una de las etapas de forma más detallada incluyendo la principal contribución de este proyecto.

a) Detección de caras

La primera etapa es la detección facial. Para ello, introducimos las dos imágenes de entrada del *pipeline* en una red neuronal para detectar las caras. El algoritmo que se va a usar para la detección facial es RetinaFace, del que se ha hablado en el capítulo 3. Se usará este ya que consigue muy buenos resultados tanto a largas distancias como a cortas, por lo que lo hace ideal para nuestro caso en el que tendremos que detectar a varias distancias. La salida de esta red neuronal será el número de caras que se han detectado en esa imagen y las coordenadas de cada una de ellas para poder preprocesarlas.

Además, se ha modificado el umbral predefinido de este algoritmo para poder reconocer caras a distancias más lejanas de hasta 30 metros. A esta distancia, una cara tiene un

tamaño aproximado de 14x19 pixeles. Al modificar el umbral, lograremos detectar más caras, pero también tendremos más falsos positivos. Esto no nos supone un problema ya que posteriormente, en la red siamesa y el cálculo de distancias, estas falsas caras se descartarán al no superar el umbral con la persona a verificar. En este caso de uso se ha optado por tener el mayor número de positivos verdaderos, aunque sea a costa de tener más falsos positivos. Para este proyecto se ha supuesto además que solo habrá una persona por cada imagen, por lo que el algoritmo solo debería detectar una cara. Si más de una cara es detectada se asumirá que en esa imagen se han detectado falsos positivos.

Esta etapa es la más lenta de nuestro *pipeline*. Detectar caras en cada una de las imágenes tarda aproximadamente una media de 170 milisegundos, por lo que la máxima velocidad de nuestro *pipeline* estará siempre limitada a como mucho 6 fps. A pesar de ello, este algoritmo es muy útil ya que permite detectar caras a muy baja resolución, como son a más de 15 metros de distancia. Como para este proyecto solo se van a usar videos pregrabados, no es tan importante conseguir el procesamiento en tiempo real. Como lo que nos interesa es conseguir detectar todas las caras en los videos, se ha decidido usar este algoritmo más lento. Para aplicaciones reales en las que se quiera usar este *pipeline*, se podrá cambiar este algoritmo por uno más rápido, aunque tenga una precisión menor y detecte menos caras o tengo más falsos positivos.

b) Preprocesamiento

La segunda etapa de nuestro diseño es el preprocesamiento de la imagen. Está dividida en dos componentes diferentes. Primero, la cara detectada se recorta de la imagen completa en la que se ha detectado. Para ello, el algoritmo de detección manda a la siguiente etapa las coordenadas de la cara en la imagen junto con la imagen en la que ha sido detectada. Las coordenadas las provee RetinaFace y el recorte se realiza con la librería OpenCV de Python, la cual tiene muchas funcionalidades para tratamiento de imágenes. Las coordenadas de una cara se pueden dar de diversas maneras. Como lo que se está detectando es en realidad un rectángulo en una imagen, necesitaremos cuatro valores para poder ubicarlo correctamente. RetinaFace nos da las coordenadas de la imagen como *(x, y, w, h)*. Siendo *x* e *y* la ubicación de la esquina superior izquierda del rectángulo, y luego *w* es el ancho (*width)* de la imagen y *h* la altura (*height)*.

En la siguiente etapa se redimensiona la imagen de la cara recortada al tamaño requerido por la entrada de la red neuronal siamesa. Por ejemplo, si se está usando ArcFace, habrá que redimensionar la imagen hasta 112x112 pixeles, que es la entrada requerida por este algoritmo. Para hacer este redimensionamiento se usa también OpenCV ya que tiene una función explicita para hacerlo. Esta etapa de nuestro diseño es una de las más rápidas ya que solo se tarda aproximadamente 0,3 milisegundos en realizarla por completo. Todos los algoritmos requieren como entrada una imagen cuadrada. Prácticamente nunca las caras detectadas serán cuadradas por lo que para no distorsionar la cara modificando la ratio de la imagen se añadirán pixeles negros a los lados para mantener la ratio de la imagen de la cara y obtener una imagen rectangular para las redes neuronales.

En este punto es donde se realiza la mayor contribución del nuevo diseño: se guardará el valor del tamaño de la cara detectada en pixeles ya que se usará más adelante en la última etapa del *pipeline*.

c) Red siamesa

La tercera etapa es de las más importantes del *pipeline* ya que es donde se extraen las características faciales de la cara. Se compone de una red siamesa, es decir, dos redes neuronales convolucionales exactamente iguales. No hay ninguna diferencia entre las dos redes, tienen la misma arquitectura, *backbone* y los mismos pesos. Por lo que, si a las dos redes les introducimos las mismas imágenes nos darán como resultado dos vectores exactamente iguales. La red siamesa se puede usar con cualquier algoritmo de verificación facial, pero en este estudio solo se va a trabajar con los cuatro algoritmos que se han mencionado en el capítulo anterior: FaceNet, OpenFace, VGG-Face y ArcFace.

La entrada a la red siamesa son las dos imágenes recortadas y redimensionadas al tamaño de entrada determinado por cada uno de los algoritmos. La primera imagen será la imagen que se quiere verificar, mientras que la segunda es la identidad confirmada de la persona que se quiere verificar. La salida de la red siamesa serán dos vectores, uno para cada imagen, que contienen las características de esa cara. La longitud de cada vector será diferente en función del algoritmo que se está usando como se ha podido ver en el capítulo anterior. El tiempo que consuma esta etapa depende del algoritmo que se está usando. Más adelante se analizarán los tiempos de inferencia de cada uno de los algoritmos para poderlos comparar y se podrá ver el tiempo real que lleva realizar esta tercera etapa de nuestro *pipeline*.

d) Cálculo de distancias

La cuarta etapa del *pipeline* es el cálculo de la distancia entre los dos vectores de características faciales. Para ello se van a usar dos métricas diferentes de las tres que se han explicado en el capítulo anterior: Distancia de coseno y distancia euclídea. La distancia Manhattan no se va a usar ya que según el estado del arte no es una distancia muy utilizada para la verificación facial ya que la distancia euclídea suele obtener mejores resultados en la mayoría de los casos. Por eso, nos centraremos solamente en dos tipos de distancias, una de ángulos que es la del coseno y otra de longitudes que es la euclídea.

Para el análisis de los diferentes algoritmos se usarán las dos métricas, pero como se comprobará en el último capítulo, al final cada una de las métricas funciona mejor para algoritmos diferentes, en función de cómo obtengan estos las características faciales. Esta es una etapa también muy rápida ya que solo hay que realizar un cálculo sencillo entre dos vectores. Aun así, hay pequeñas diferencias de tiempos entre los diferentes algoritmos al tener cada uno de ellos una longitud de vector diferente. Pero estos tiempos son muy pequeños comparados con los de otras etapas del algoritmo como la detección facial o la red siamesa.

e) Toma de decisiones

Finalmente, se tiene la última etapa de nuestro *pipeline* que es donde se toma la decisión de si las dos personas que se están comparando son la misma persona o diferentes. Esta decisión se toma en base a un umbral previamente definido. Si la distancia entre los dos vectores faciales es inferior al umbral, se concluye que son la misma persona, si la distancia es superior, serán dos personas diferentes. Por ello, es muy importante elegir un umbral adecuado para lograr minimizar los falsos positivos y maximizar los verdaderos. Se suele usar un umbral estático previamente definido que no varía.

La principal contribución de este proyecto es la introducción de un umbral dinámico que varía en función de la distancia de la persona al dron. A medida que la distancia del dron a la cara varía, también lo hace el vector de características faciales, por lo que la distancia entre los vectores variará también. Si esto cambia, es lógico pensar que, variando el umbral acorde con esto, se conseguirán unos mejores resultados a la hora de verificar. Si el dron está muy cerca de la persona, se apreciarán perfectamente todas las características faciales de la persona, por lo que la distancia será menor entre los vectores, por ello habrá que imponer un umbral estricto. Mientras que si el dron está lejos de la persona la cara tendrá muy poca resolución y las características serán difíciles de sacar, por lo que habrá que poner un umbral más laxo para no bajar la precisión de nuestro algoritmo. Además, el umbral usado también variará en función del algoritmo y la distancia que se estén usando, ya que cada uno de ellos da unos resultados diferentes.

Finalmente, la salida de nuestro *pipeline* será si las dos imágenes introducidas corresponden a la misma persona dando únicamente como salida un booleano: *true* (si son la misma persona) o *false* (si son personas diferentes). Todo el *pipeline* con las etapas que se acaban de explicar se puede ver en **Fig. 12**.

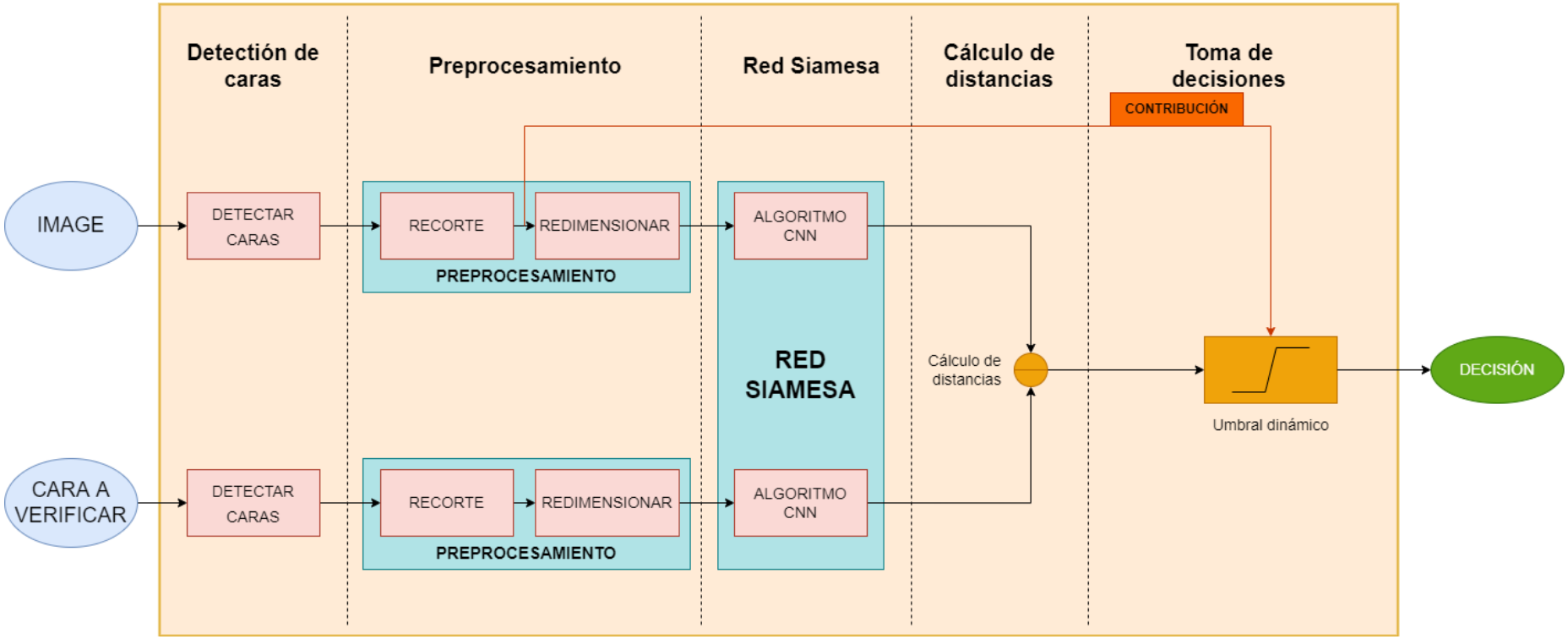


***Fig. 12**. Diseño propuesto para realizar verificación facial usando umbrales dinámicos en función de la distancia*

Diseño del umbral dinámico

Una vez explicado el nuevo *pipeline*, se va a explicar cómo se ha diseñado este nuevo umbral dinámico y cómo se han obtenido los valores de los umbrales que se van a usar. Para ello, primero se va a explicar el *dataset* que se ha usado, después se explicará en sí cómo se obtienen los umbrales dinámicos y finalmente se mostrarán los valores de los umbrales que se han calculado.

a) *Dataset*

Para este proyecto se ha obtenido un nuevo *dataset* compuesto de imágenes grabadas desde un dron, para poder tener imágenes de rostros desde una variedad de distancias y desde un ángulo concreto, no en horizontal como se suelen sacar desde el suelo. Se ha decidido crear un nuevo *dataset* debido a la ausencia de ellos con imágenes grabadas desde drones para verificación facial, como se ha discutido en el capítulo 3. La mayoría de *datasets* existentes ahora mismo en la literatura son incompletos, o no tienen las imágenes que se buscan en este proyecto.

Debido a esto se ha creado este nuevo *dataset*. El dron usado para grabar las imágenes es un DJI Mini 2. Las imágenes que se obtienen con la cámara de este tienen una resolución de 4K (3840x2160 px) y los videos son a 30 fps. El terreno que se ha usado de fondo en todos los videos ha sido césped, por lo que el fondo es verde, salvo en algunos vídeos donde se puede apreciar también el cielo, asfalto o edificios. El fondo de nuestro *dataset* es muy importante ya que el algoritmo de detección facial va a tener que reconocer caras en él, por lo que se ha elegido un fondo no muy complicado para que se puedan detectar de forma fácil. Ya que el objetivo de nuestro proyecto es mejorar la precisión de los algoritmos de verificación facial, no de detección facial, por lo que se quiere facilitar el trabajo de estos para poder centrarnos en la verificación.

Los videos se han grabado desde ocho distancias diferentes desde la cara a la cámara del dron: 2, 5, 7, 10, 15, 20, 25, 30 metros. En total se ha grabado a veinticinco personas diferentes de diferentes edades, géneros y etnias para poder tener un *dataset* completo, obteniendo unos resultados variados y estadísticamente apropiados. Cada uno de los videos dura 30 segundos, durante los cuales se le pidió al sujeto que hiciera varios movimientos con la cabeza para poder obtener varias perspectivas de la cara y tener aún más datos. Los movimientos que se solicitaron fue girar la cabeza hacia la izquierda, después hacia la derecha, para después hacer un giro completo con la cabeza, y finalmente quedarse mirando a la cámara del dron el resto del tiempo. Los videos también fueron grabados a horas diferentes del día, para tener más datos con diferente iluminación, teniendo el sol de frente, por los lados y por detrás, además de tener videos con el cielo nublado, es decir, iluminación no muy alta.

Tabla 3. *Características del dataset creado*

N.º Personas	Géneros	Etnias	Rango de edad
25	Masculino Femenino	Caucásica Africana Asiática Persa India	16-55

En los videos que se han grabado solo se encuentra una persona en cada uno de ellos, por lo que si se detecta más de una cara se asumirá que hay falsos positivos en él. Las imágenes obtenidas se han grabado con una inclinación de 30 grados del dron sobre la cara del usuario. Por lo que las distancias de la cara al dron representan la diagonal que los une. Pero la altura máxima que alcanza el dron es de 15 metros sobre la cara de la persona. Las distancias que se han usado para grabar los videos se pueden ver en **Fig. 13**, incluyendo la altura del dron sobre la persona y la distancia horizontal al usuario.

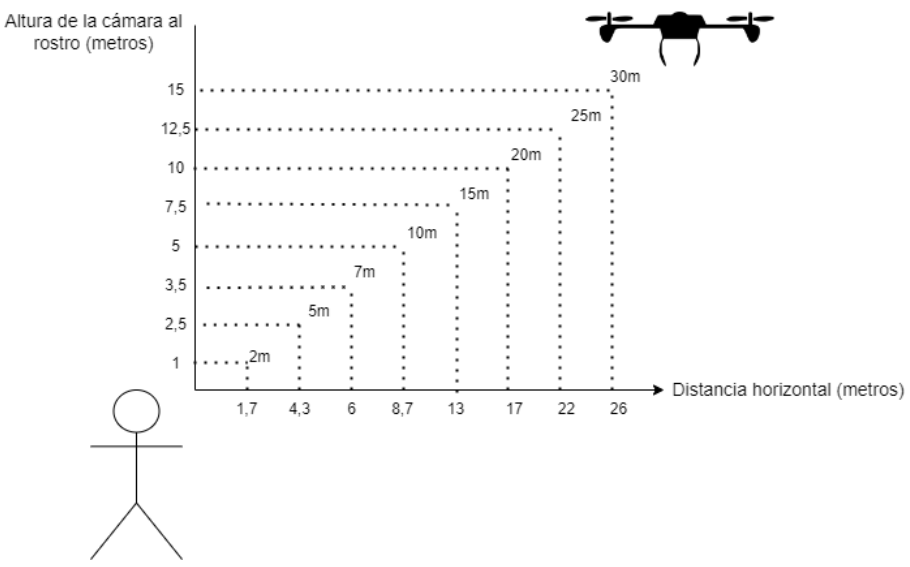
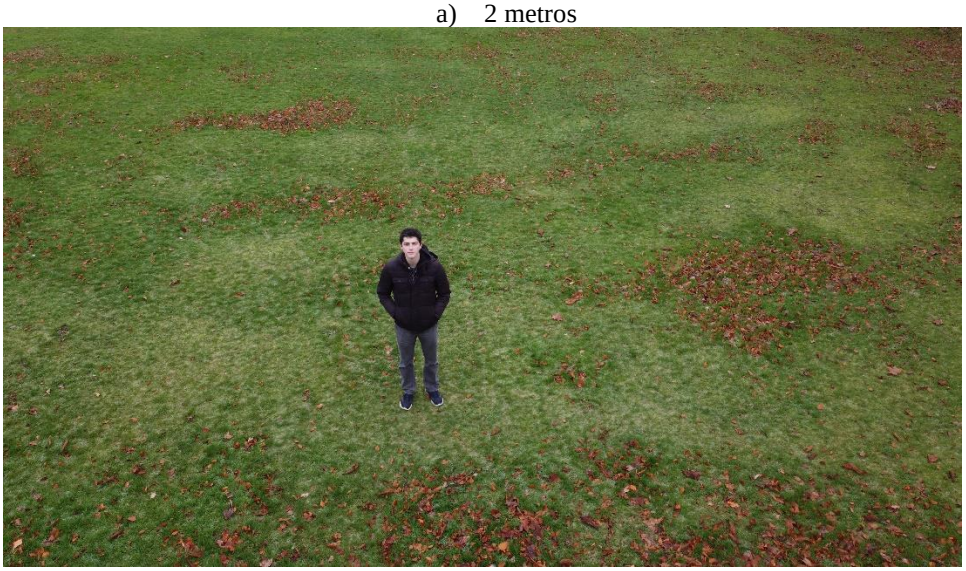


Fig. 13. *Distancias de la cámara del dron a la cara del usuario de los videos contenidos en el dataset creado*

También en la **Fig. 14** se pueden ver varias imágenes correspondientes al *dataset*, como ejemplo de cómo son los videos que contiene y los rasgos faciales que se pueden obtener de él.

a) 2 metros



b) 5 metros

c) 7 metros

d) 10 metros

e) 15 metros

f) 20 metros

g) 25 metros

h) 30 metros

Fig. 14. *Frames obtenidos de videos del dataset creado a las 8 distancias (2, 5, 7, 10, 15, 20, 25 y 30 metros)*

b) Definición de una escala

Como se ha explicado, nuestro umbral dinámico se basa en modificar los umbrales de verificación en función de la distancia a la que esté el dron de la cara del usuario. Si está muy lejos se pondrá un umbral más laxo, mientras que si está muy cerca se pondrá uno más restrictivo. Para poder implementar el umbral dinámico lo primero que se tiene que hacer es definir unos rangos dentro de una escala para cambiar de un umbral dinámico. Es decir, si estamos a una distancia del usuario usamos un umbral, pero si nos alejamos

de objetivo, cambiaremos de rango dentro de la escala y se usará un umbral diferente. En este apartado se va a elegir de forma empírica esa escala.

Lo primero es averiguar cómo saber la distancia de la persona al dron. A partir de una imagen no se puede saber precisamente la distancia a un objeto. Con el dron se podría usar un telémetro que nos dice la distancia a un objetivo. Pero como el dron usado no lo tiene y no se puede presuponer que todos los drones que se usen lo tienen hay que buscar otra técnica para averiguar la distancia.

La solución a la que se ha llegado es establecer una relación entre el tamaño de la cara detectada y la distancia a la que está a la persona. una resolución de la cámara fija, se puede saber de forma aproximada la distancia a la que está una persona en función del tamaño de la cara. Es verdad que hay cabezas más grandes y pequeñas pero las diferencias no son muy notables y más a largas distancias. No se va a obtener una distancia muy precisa pero sí que servirá para el umbral dinámico. Si se cambiara la resolución de las imágenes, también habría que cambiar la relación distancia-tamaño de la cara, porque el tamaño de la cara en pixeles variará de la misma manera que la resolución de la imagen.

En la **Tabla 4** se puede ver la relación entre la distancia del dron a la cara y el tamaño de esta. Para obtener estos valores se han usado las imágenes contenidas en nuestro *dataset* y se ha hecho una media de los tamaños de las caras obtenidos para cada distancia. Como se puede ver, el tamaño de la cara no disminuye linealmente con la distancia. A partir de 10 metros, el tamaño de la cara en pixeles disminuye poco debido a que tiene ya un tamaño muy reducido de píxeles. Si las imágenes en vez de tener una resolución 4K, como los grabados, tuviera una menor, el tamaño de la cara en pixeles sería también menor proporcionalmente. A partir de estos valores se sacará la escala de la que se hablará a continuación.

Tabla 4. *Distancia de la cámara del dron a la cara frente al tamaño del rostro detectado en pixeles*

Distancia	Tamaño de la cara
2 m	125x170 px
5 m	80x100 px
7 m	50x60 px
10 m	40x45 px
15 m	25x30 px
20 m	20x25 px
25 m	17x20 px
30 m	14x18 px

En la **Fig 15** se pueden ver las caras recortadas a cada una de las 8 distancias de nuestro *dataset*. En la primera imagen a 2 metros se puede apreciar perfectamente todos los rasgos

de la persona y se la puede identificar perfectamente. Pero a medida que nos alejamos la resolución va disminuyendo hasta los 30 metros cuando prácticamente no se reconoce ninguna característica del rostro, únicamente se aprecia que es una cara con dos ojos, una nariz y una boca, pero sería muy difícil identificar a esa persona. Estas imágenes se han obtenido ejecutando el algoritmo ArcFace usando los umbrales predefinidos antiguos y distancia euclídea como métrica. Además, la persona a verificar es la misma que la de las imágenes. Si se observa el borde de las ocho imágenes se puede distinguir un reborde rojo y verde, será del primer color si no se ha reconocido y del segundo si sí lo ha hecho. Como se puede ver a 2 metros y 7 metros se reconoce a la persona, pero a partir de 10 metros ya no se reconoce. 5 metros es una excepción, ya que la persona no se ha reconocido a pesar de la corta distancia, seguramente porque la persona está mirando hacia abajo en vez de directamente a la cámara. Con nuestro nuevo umbral dinámico se espera poder aumentar la precisión del algoritmo y poder obtener verificaciones correctas incluso a partir de 10 metros de distancia.

Fig 15. *Rostros recortados a las ocho diferentes distancias grabadas en el dataset*

Como se ha mencionado, la escala que se desarrollará se compondrá de diferentes rangos en función del tamaño de la cara. Por simplificar el método se ha decidido no utilizar el ancho y el alto de la cara para los rangos, sino que se utilizará únicamente el ancho de la cara en pixeles para definir la escala. Se hace así ya que los rostros varían más en altura que en ancho entre diferentes personas, por lo que al usar el ancho tenemos una medida más estable e independiente de la persona.

Después, a partir de la tabla anterior se ha decidido definir cuatro rangos diferentes dentro de nuestra escala en función del tamaño de la cara: Muy lejos, lejos, medio y cerca. Para cambiar de un rango a otro se han definido tres valores de forma empírica. De *Cerca* a *Medio* se cambia cuando el ancho de la cara disminuye de 75 pixeles, de *Medio* a *Lejos* cuando disminuye de 35 pixeles y finalmente se pasará al último rango si es menor a 20 pixeles. Acorde con la **Tabla 4** y las imágenes de nuestro *dataset*, el rango *muy lejos* se corresponde a las distancias de 25 y 30 metros de nuestro *dataset*. El rango *lejos* a las distancias 15 y 20 metros, el *medio* a 7 y 10 metros y finalmente el rango *Cerca* a 2 y 5 metros. En función de en qué rango de la escala se está se usará un umbral u otro, por lo que irá variando dinámicamente. Por ello se necesitará tener cuatro umbrales diferentes.

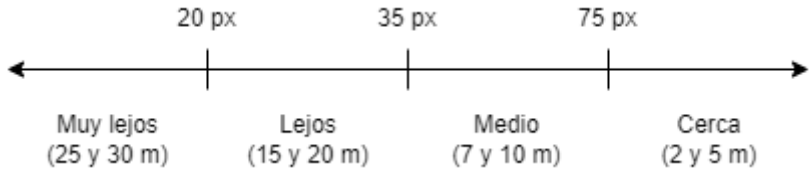


Fig 16. *Escala para el umbral dinámico en función de la anchura de la cara recortada en pixeles*

Usando esta escala recién diseñada y cambiando los umbrales en función del ancho de la cara se espera conseguir una mejora en la precisión de los algoritmos de verificación facial. También, al realizar la escala en función del tamaño de la cara en vez de la distancia real de la cámara a la persona se ha conseguido hacer el sistema independiente a la resolución de la cámara, por lo que se podrá usar no solo para drones sino para cualquier sistema con cámara y seguirá siendo totalmente funcional, logrando alcanzar los mismos buenos resultados en cualquier caso.

c) Algoritmo desarrollado

En la siguiente página se puede ver el algoritmo desarrollado para calcular los umbrales dinámicos para cada una de las distancias, métricas y algoritmos de verificación facial (**Algoritmo 1)**.

Lo primero que se hace es definir el *set* de verificación, que está compuesto por una imagen tomada de cerca de cada una de las personas contenidas en el *dataset*, el cual también se usará. Después se definen los pares positivos como el par de la imagen de una persona del *set* de verificación con cada uno de los *frames* del *dataset* en el que sale esa persona. Luego se definen los pares negativos como el par de la imagen de una persona del *set* de verificación con cada uno de los *frames* del *dataset* en los que no sale esa persona. Es decir, en los pares positivos la verificación debería ser siempre positivo ya que sabemos que la persona está contenida en el *frame*, mientras que en los pares negativos la verificación debería ser negativa ya que la persona no está en el *frame*. En cada *frame* del *dataset* solo hay contenida una persona.

Cada par positivo y negativo tendrá asociado un índice de similaridad que será la distancia calculada entre las dos caras, la del *set* de verificación y la del *dataset*. También se define un hiperparámetro que será el que se use para iterar en el bucle del algoritmo. Este valor dependerá de la métrica o el algoritmo de verificación facial usado.

En el algoritmo lo primero que se hace es calcular dos ratios, uno para los pares positivos y otro para los negativos. Esto se debe a que el número de muestras en cada uno de los pares no es el mismo, por lo que, si no se aplicará una ratio, el peso que tendría uno de los pares sería mayor al otro. En este caso, al tener un mayor número de pares negativos, el peso de estos a la hora de calcular la precisión y por ello los umbrales sería mayor al de los pares positivos.

A continuación, se define el bucle desde el índice de similaridad de los pares positivos mínimo al máximo usando como salto el hiperparámetro. El parámetro que se irá iterando será el candidato a ser umbral. Lo primero que se calcula son los *True Positives* (TP) y los *True Negatives* (TN). TP se calcula como el número de pares positivos cuyo índice de similaridad es inferior al candidato a umbral. TN se calcula igual, pero usando los pares negativos. Después se calcula la precisión usando estos valores calculados junto con las ratios.

Por último, si la precisión calculada es la máxima calculada hasta ahora en el bucle, el candidato será el umbral escogido, de momento. El acabar el bucle, el candidato a umbral que consiga la precisión más alta será el umbral escogido. **Algoritmo 1 – Cálculo del Umbral Óptimo**

- Se define el *set* de verificación (*V*) como una muestra (V_i) de cada una de las *y* identidades faciales a ser verificadas ($\boldsymbol{V_i}$) y un *dataset* (*D*), compuesto por *x* muestras (S_j) de las mismas identidades faciales (F_i).

$$V = \sum_{i=o}^{y} V(i)\,, D = \sum_{i=o}^{y}\sum_{j=o}^{x} F_i(S_j)$$

- Creemos ahora los **Pares Positivos de la identidad** *I*, definidos como:

$$P_I^+ = (Vi, \sum_{j=o}^{x} F_I(S_j))$$

- Y los **Pares Negativos de la identidad** *I*, definidos como:

$$P_I^- = \left(V_i, \sum_{i=0}^{y}\sum_{=0}^{x} if(i \neq I)\left[F_I\left(S_j\right)\right],\ otherwise\ 0 \right)$$

- Definamos la **distancia de un par** como: $E(x, y)$.

- Y definamos la **mínima distancia de todos los pares positivos** como:

$$E^{+}_{min} = \min\left(\sum_{i=o}^{y} E(P_i^{+})\right)$$

- Análogamente, definamos la **máxima distancia de todos los pares positivos** como:

$$E^{+}_{max} = \max\left(\sum_{i=o}^{y} E(P_i^{+})\right)$$

- Definamos ***hyperparameter*** como el salto usado para iterar los umbrales candidatos dentro del bucle *for*. El valor depende del algoritmo y de la métrica usada.

Y aplicamos el cálculo del umbral dinámico propuesto de la siguiente manera:

- **Entradas:** $P_I^{-}, P_I^{+}, E^{+}_{max}, E^{+}_{min}$, *hyperparameter*
- **Salidas:** *Threshold, MaxAccuracy*

1. Inicializamos *threshold = 0, MaxAccuracy=0*

2. Calculamos la ratio de todos los pares positivos:

$$R(P^{+}) = \frac{\sum_{i=o}^{y} Size(P_i^{+})}{\sum_{i=o}^{y} Size(P_i^{+}) + \sum_{i=o}^{y} Size(P_i^{-})}$$

3. Calcular la ratio de todos los pares negativos:

$$R(P^{-}) = \frac{\sum_{i=o}^{y} Size(P_i^{-})}{\sum_{i=o}^{y} Size(P_i^{+}) + \sum_{i=o}^{y} Size(P_i^{-})}$$

4. for$(candidato = E^{+}_{min}; candidato < E^{+}_{max}; candidato = candidato + hyperparameter)$

4.1. $TP(TruePositives) = count\left(\sum_{i=0}^{y} if(E(P_i^{+}) < candidate)\right) 1,\ otherwise\ 0$

4.2. $TN(TrueNegatives) = count\left(\sum_{i=0}^{y} if(E(P_i^{-}) < candidate)\right) 1,\ otherwise\ 0$

4.3. $A = 100 * \frac{TP*R(P^+)+TN*R(P^-)}{\sum_{i=0}^{y} Size(P_i^+)*R(P^+)+\sum_{i=0}^{y} Size(P_i^-)*R(P^-)}$

4.4. if (A > *MaxAccuracy)*

4.4.1. *MaxAccuracy = A*

4.4.2. *Threshold = candidate*

CAPÍTULO V

CÁLCULO DE LOS UMBRALES

En este capítulo se van a calcular los nuevos umbrales para nuestro diseño dinámico en función de la distancia a la cara. Para ello, se va a usar el algoritmo desarrollado en el apartado anterior y se calculará usando el programa Octave. A continuación, se mostrarán los valores de umbrales recomendados según el cálculo realizado con nuestro *dataset*.

Aplicación del nuevo *pipeline*

Para el cálculo de los umbrales dinámicos se ha implementado el algoritmo explicado en el capítulo anterior usando Matlab. Además, para obtener los índices de similaridad de los pares positivos y negativos se ha usado el *framework* DeepFace junto con el *dataset* y el *set* de verificación obtenidos. Como se ha mencionado previamente, los umbrales son calculados para lograr maximizar la precisión de los algoritmos en cada rango de la escala que se ha definido en el capítulo anterior. A continuación, se va a explicar de forma gráfica como el algoritmo calcula los umbrales óptimos.

a) Distancia de coseno

Primero se van a analizar las gráficas obtenidas usando como métrica la distancia de coseno. Para explicar de forma gráfica el algoritmo se han obtenido dos gráficas para cada uno de los algoritmos, la primera es la distribución de los índices de similaridad. Y muestra dos curvas diferentes: una para los pares positivos y otra para los pares negativos. La segunda gráfica representa la precisión obtenida en función del umbral seleccionado.

Estas gráficas se han obtenido para dos distancias diferentes, 5 y 15 metros, que corresponden a distancias cercanas y lejanas, respectivamente para poder apreciar como varía la distribución de los índices de similaridad normalizados para cada algoritmo en función de la distancia.

En las siguientes páginas se muestran cinco figuras que corresponden con cada uno de los algoritmos de verificación facial a 5 metros de distancia y usando la distancia de coseno: FaceNet (**Fig. 17**), FaceNet512 (**Fig. 18**), OpenFace (**Fig. 19**), ArcFace (**Fig 20**), VGG-Face (**Fig 21**).

Al haber usado nuestro propio *dataset* incluyendo diferentes etnias y rangos de edad se ha conseguido obtener unos índices de similaridad bien distribuidos abarcando unas muestras estadísticamente apropiadas.

Respecto a las gráficas, los índices de similaridad de los pares positivos se han obtenido ejecutando el *pipeline* diseñado y comparando la imagen del *set* de verificación de una

persona con los *frames* de esa misma persona en el *dataset*. Los índices de los pares negativos se han obtenido comparando la imagen del *set* de verificación de una persona con los *frames* del *dataset* en los que no sale esa persona. Cuanto más separados estén las distribuciones de los pares positivos y negativos, se conseguirá una mejor precisión y será más fácil obtener el umbral óptimo. Mientras que, si las distribuciones están muy juntas o mezclándose la precisión será muy baja al ser muy parecidos los índices de similaridad de los pares positivos y negativos, obteniendo numerosos falsos positivos y falsos negativos.

Centrémonos, por ejemplo, en la **Fig. 17** que muestra las gráficas para FaceNet a 5 metros usando distancia de coseno como métrica. Se puede observar que los índices de similaridad están separados uno del otro, y es fácil separarlos, hasta manualmente se podría definir un umbral en el punto de cruce de las dos curvas. En la siguiente gráfica se puede ver la distribución de la precisión en función del umbral. Se puede ver que cercano al punto de cruce de las curvas es donde se maximiza, por ello, ese será el umbral escogido. Al escoger este umbral habrá algunos falsos positivos como se puede ver, ya que parte de la curva de los pares negativos cae en el lado izquierdo del umbral. Pero como el objetivo es maximizar la precisión, esto no supone un problema. Si se hubiera escogido el umbral predefinido no habría ningún falso positivo, pero la precisión del algoritmo sería muy baja, por lo que tendríamos numerosos falsos negativos.

Si se observan el resto de las figuras, se puede apreciar que todos los algoritmos consiguen curvas bien separadas excepto OpenFace. En esta gráfica se solapan ligeramente las dos curvas, pero se sigue pudiendo establecer una división entre ambas, aunque el número de falsos positivos será mayor al resto de algoritmos. Por ello, su precisión es la más baja de todos los algoritmos comparados.

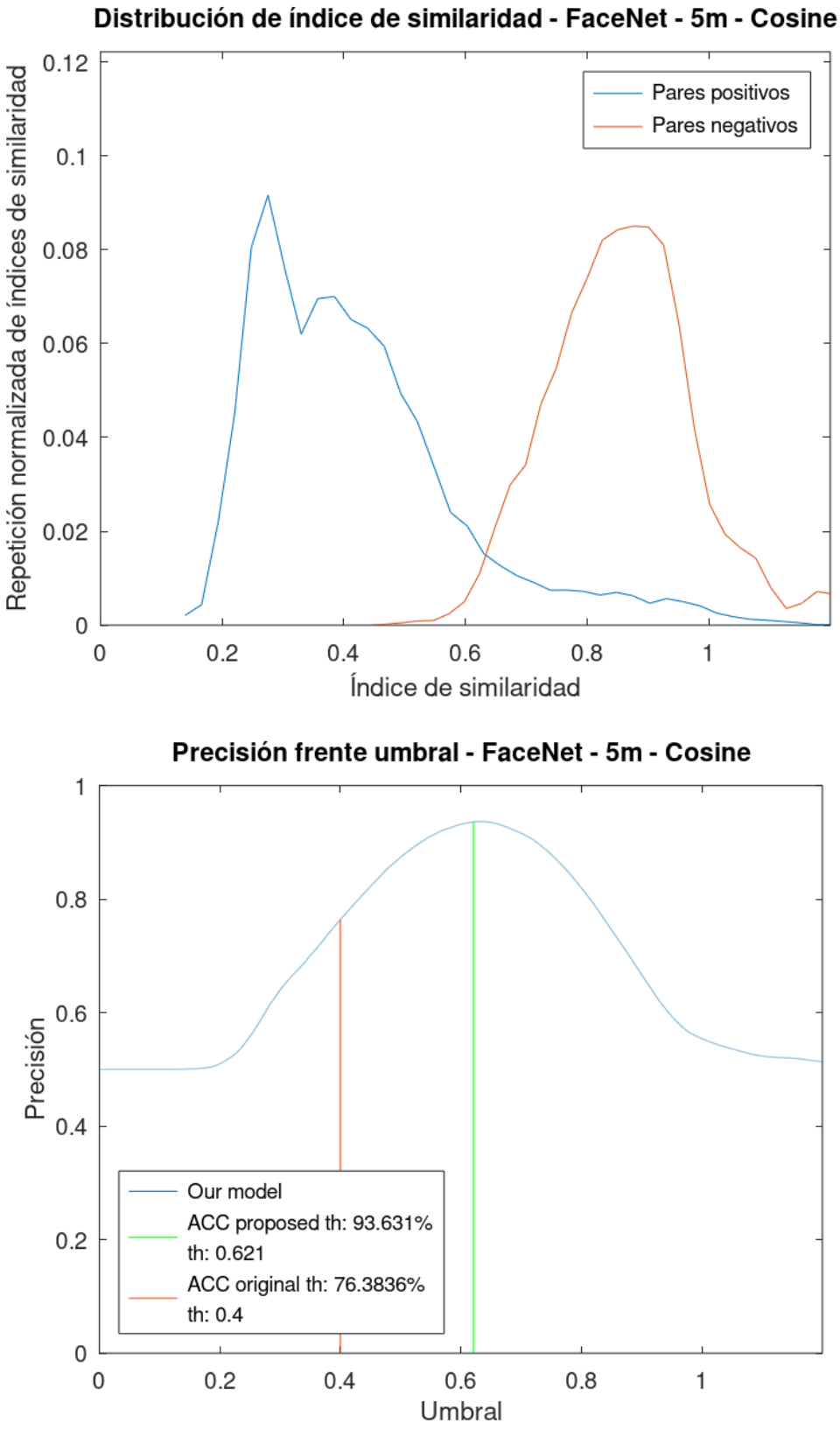


Fig. 17. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de FaceNet. La métrica usada es* ***distancia de coseno***

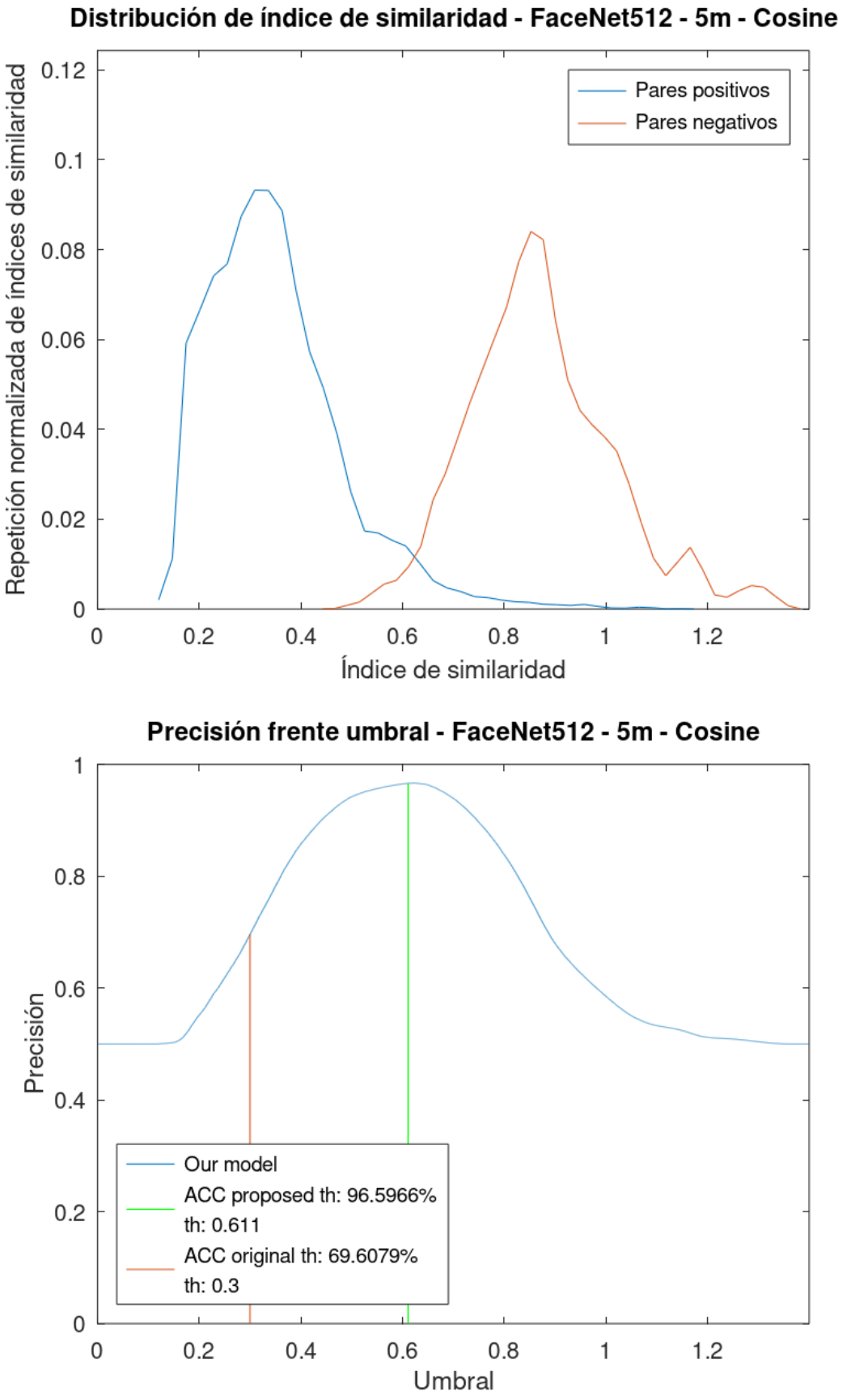


Fig. 18. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de FaceNet512. La métrica usada es* ***distancia de coseno***

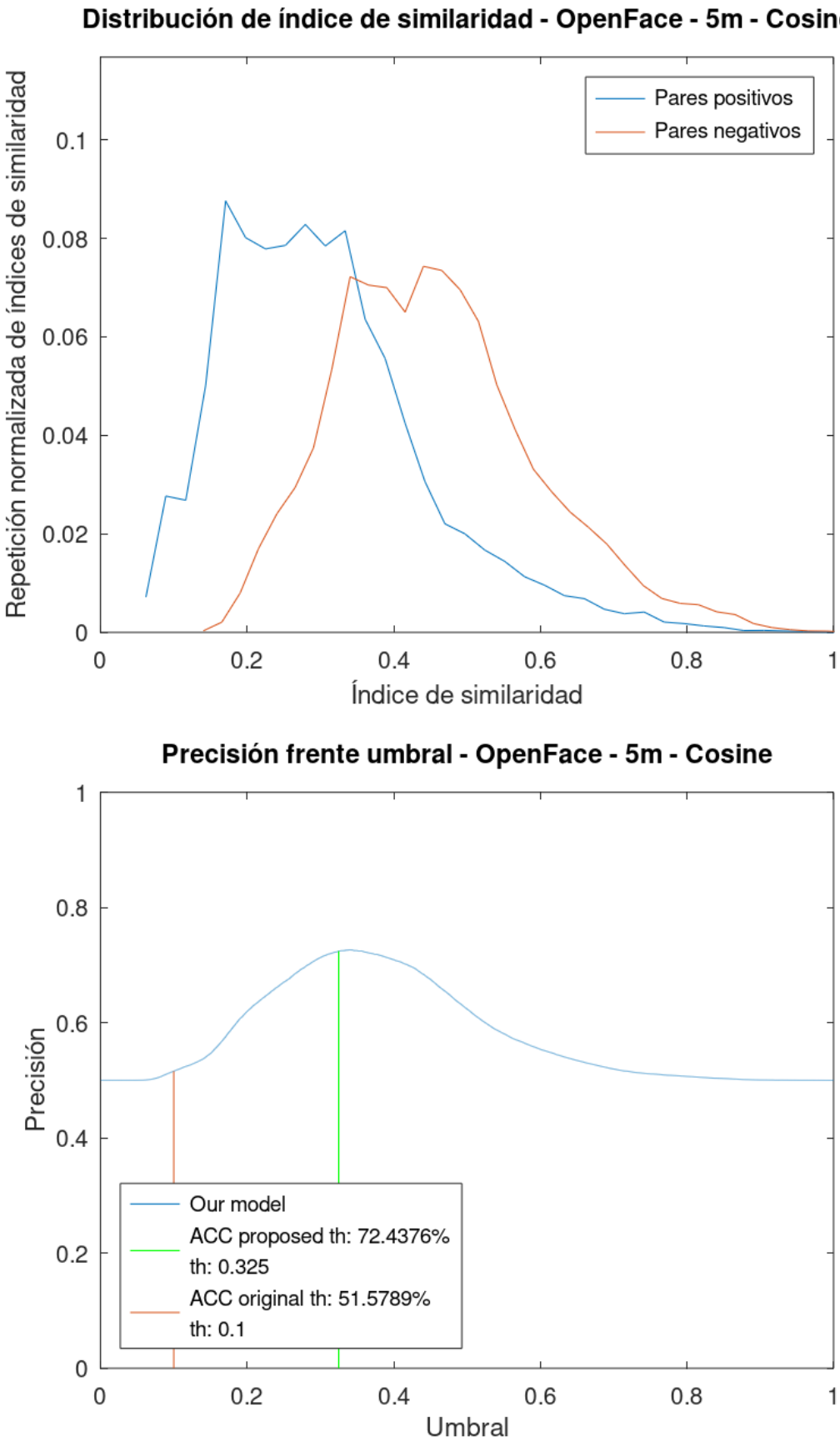


Fig. 19**. Distribución de distancias y la precisión en función del umbral usado a **5 metros** de distancia de OpenFace. La métrica usada es **distancia de coseno

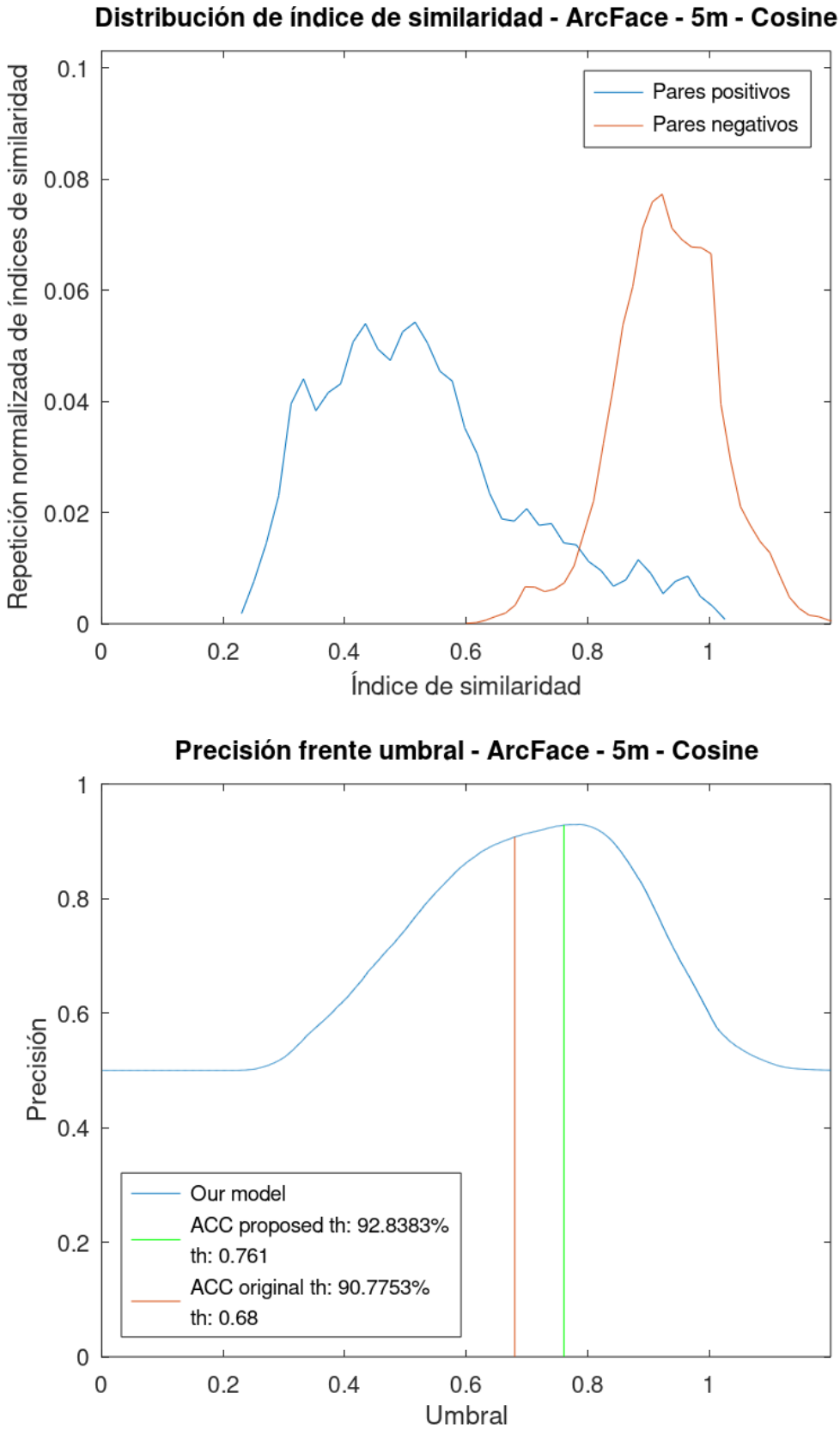


Fig 20. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de ArcFace. La métrica usada es* ***distancia de coseno***

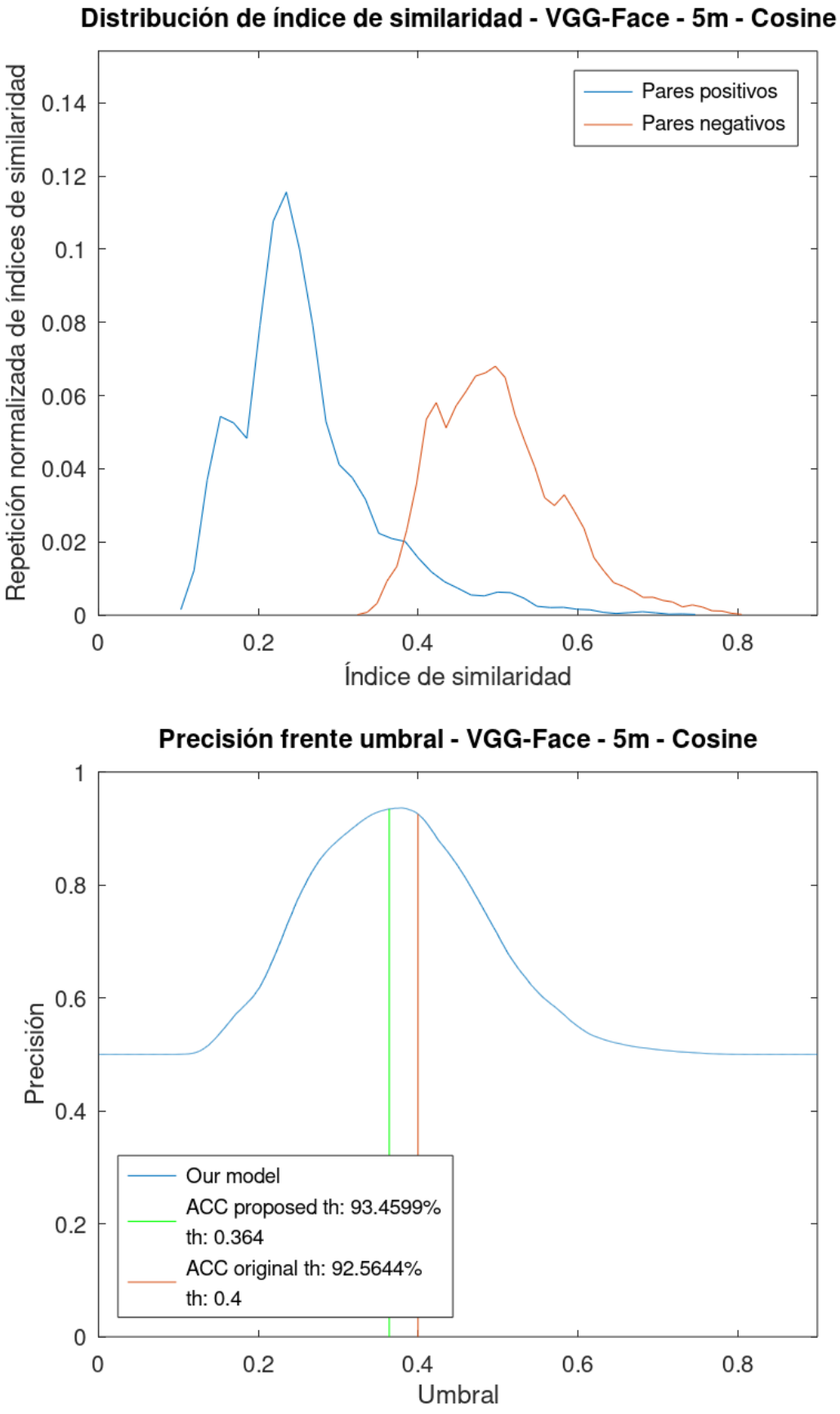


Fig 21. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de VGG-Face. La métrica usada es* ***distancia de coseno***

A continuación, se pueden ver las mismas gráficas, pero a una distancia de 15 metros para los cinco algoritmos: FaceNet (**Fig 22**), FaceNet512 (**Fig 23**), OpenFace (**Fig 24**), ArcFace (**Fig 25**), VGG-Face (**Fig 26**). Como se puede observar y cabía esperar, todas las curvas están más solapadas que en las gráficas anteriores. Esto se debe a que a medida que nos alejamos, las caras tienen peor resolución, por lo que el índice de similaridad de los pares positivos será cada vez más alto, lo que significa que son más diferentes, y por lo tanto los índices de similaridad de los pares positivos serán cada vez más parecidos a los negativos.

Aun así, en todos los algoritmos se pueden seguir separando ambas curvas o al menos una parte de ellas, por lo que la precisión obtenida por cada uno de los algoritmos sigue teniendo valores altos. Siendo la más baja FaceNet con un 65%.

Es importante destacar que, como se puede ver en algunas gráficas, el umbral escogido no es el mismo que maximiza la precisión. Por ejemplo, fijémonos en la gráfica de VGG-Face (**Fig 26**). El umbral escogido es 4,628, mientras que el que maximizaría la precisión sería 4,3. Esta diferencia se debe a que no maximiza la precisión de una sola distancia, sino de un rango de distancias. Es decir, en este caso que nos situamos a 15 metros, estamos maximizando las distancias lejanas, que abarcan 15 y 20 metros. Por lo que, lo que se está maximizando es la precisión en este rango de distancias, no únicamente a 15 metros. Por lo que, a esta distancia no se consigue la máxima precisión, como tampoco sucederá a 20 metros, pero se consigue la máxima precisión para el conjunto de ambas distancias.

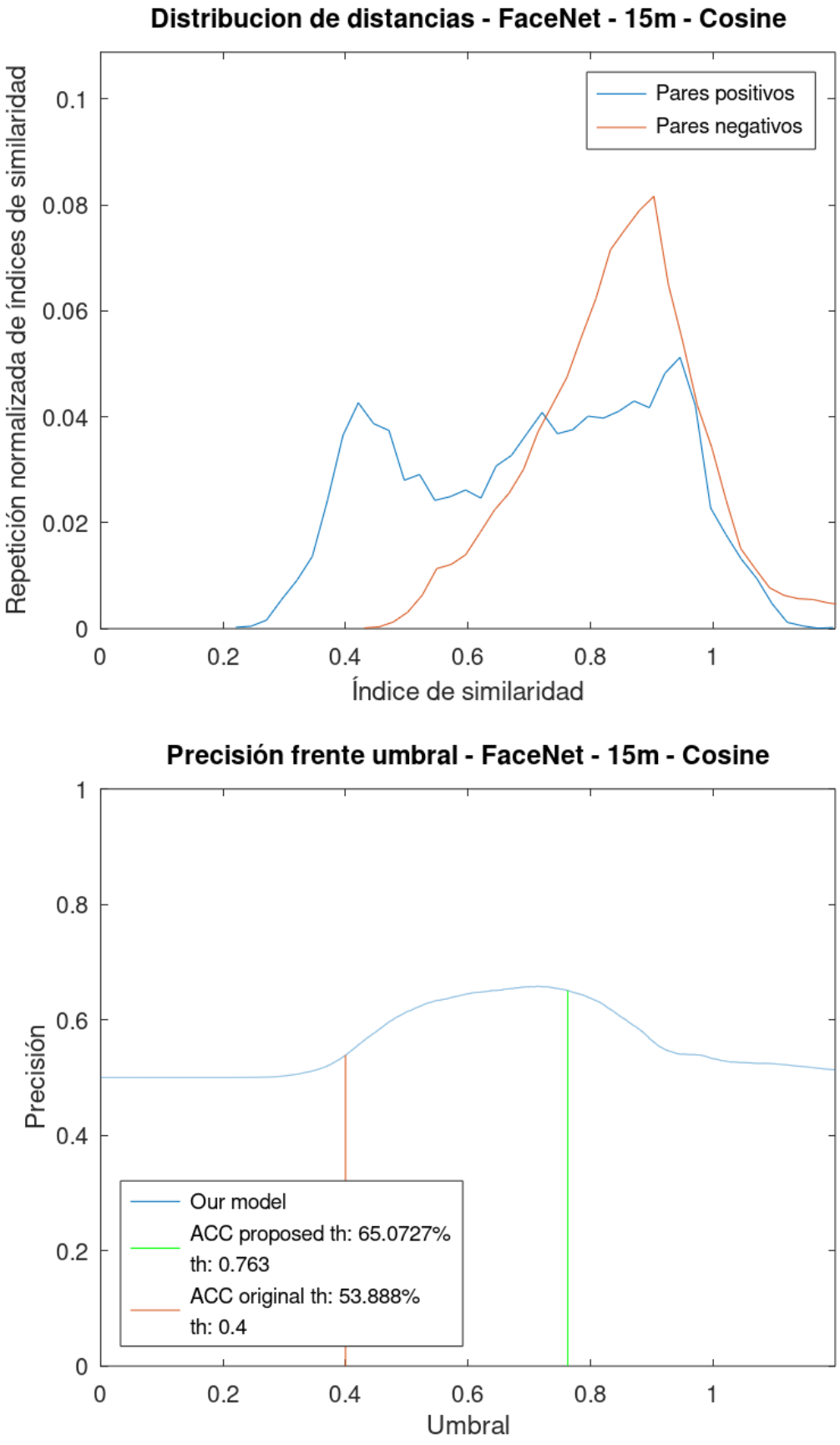


Fig 22. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de FaceNet. La métrica usada es* ***distancia de coseno***

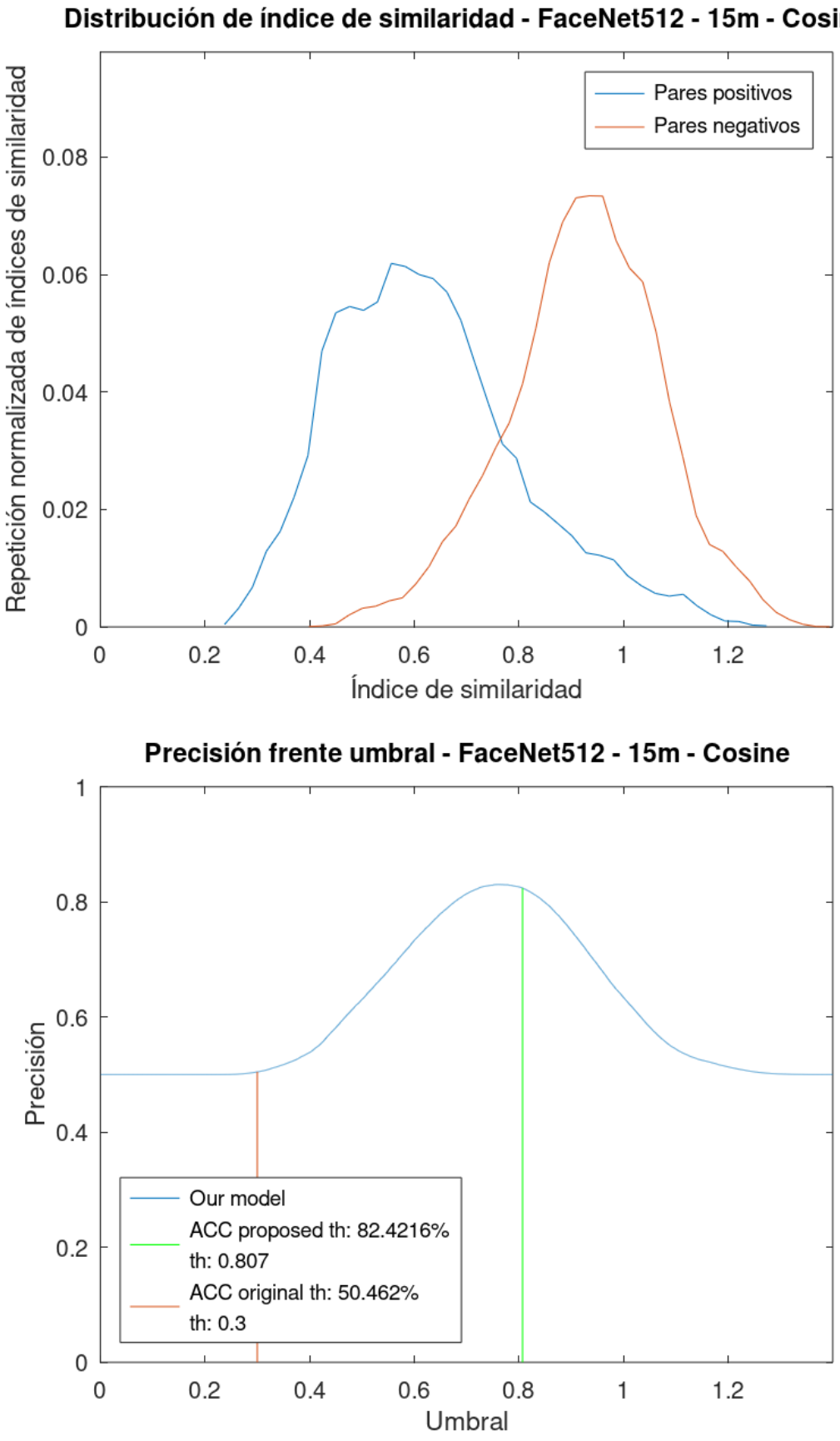


Fig 23. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de FaceNet512. La métrica usada es* ***distancia de coseno***

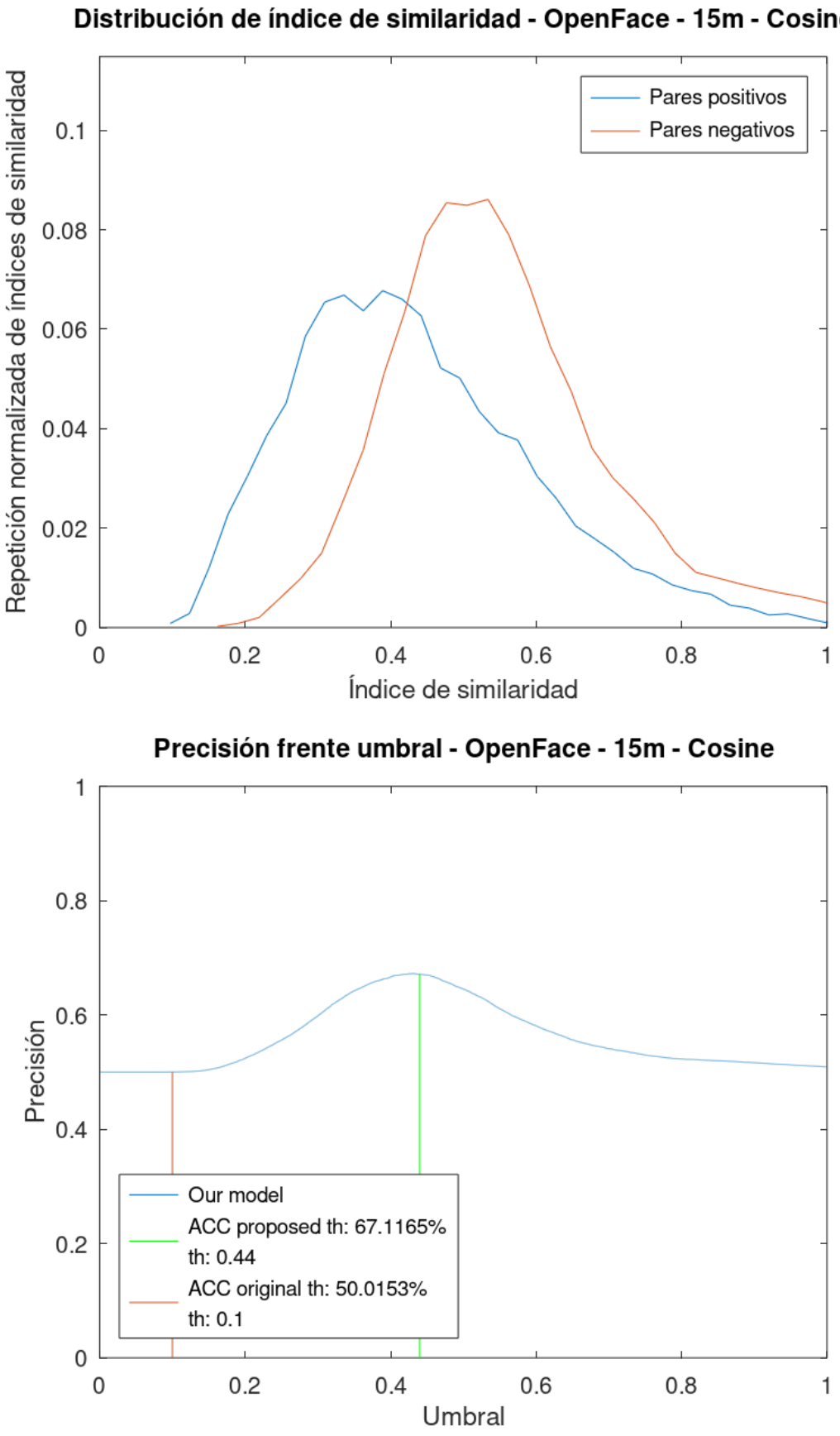


Fig 24. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de OpenFace. La métrica usada es* ***distancia de coseno***

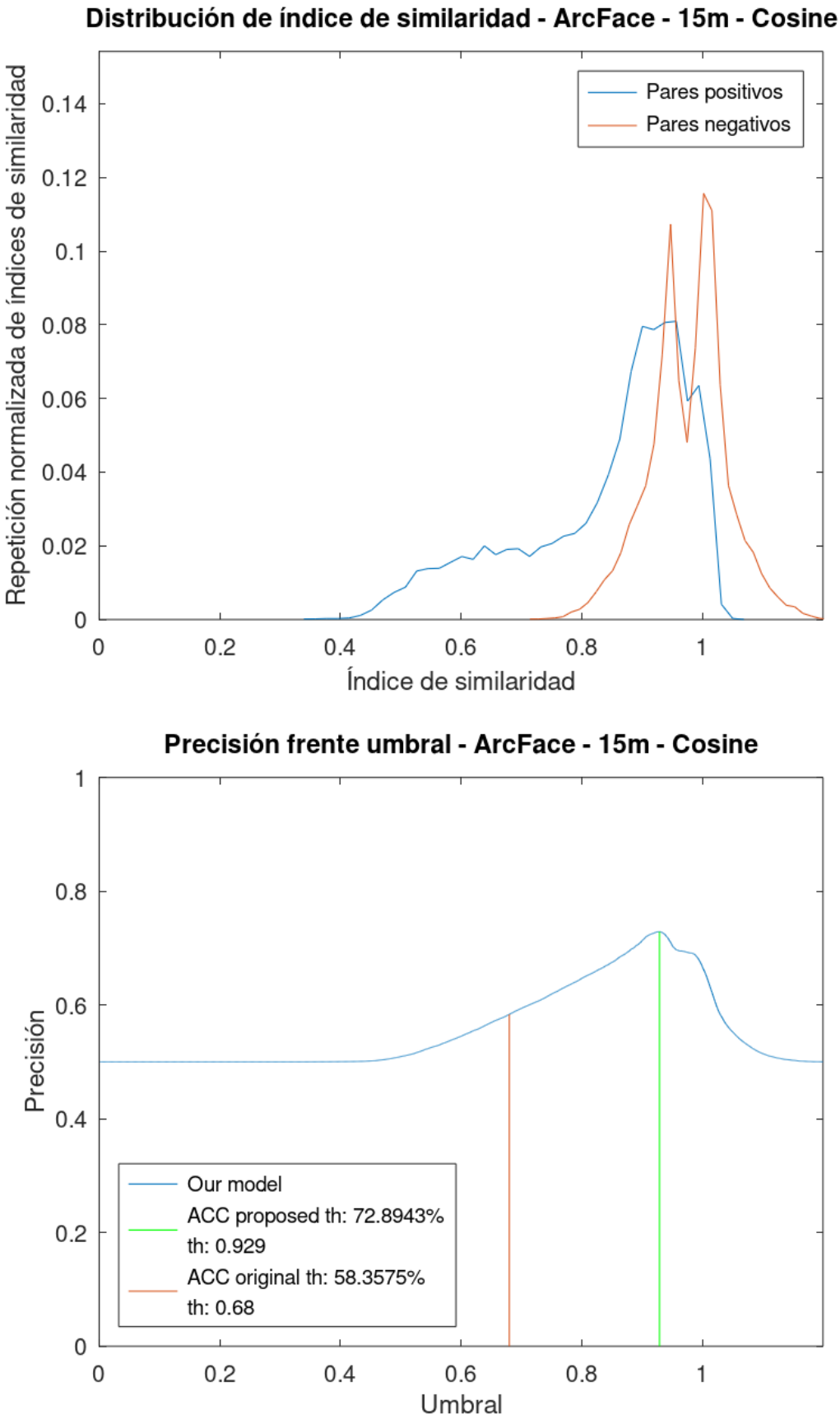


Fig 25. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de ArcFace. La métrica usada es* ***distancia de coseno***

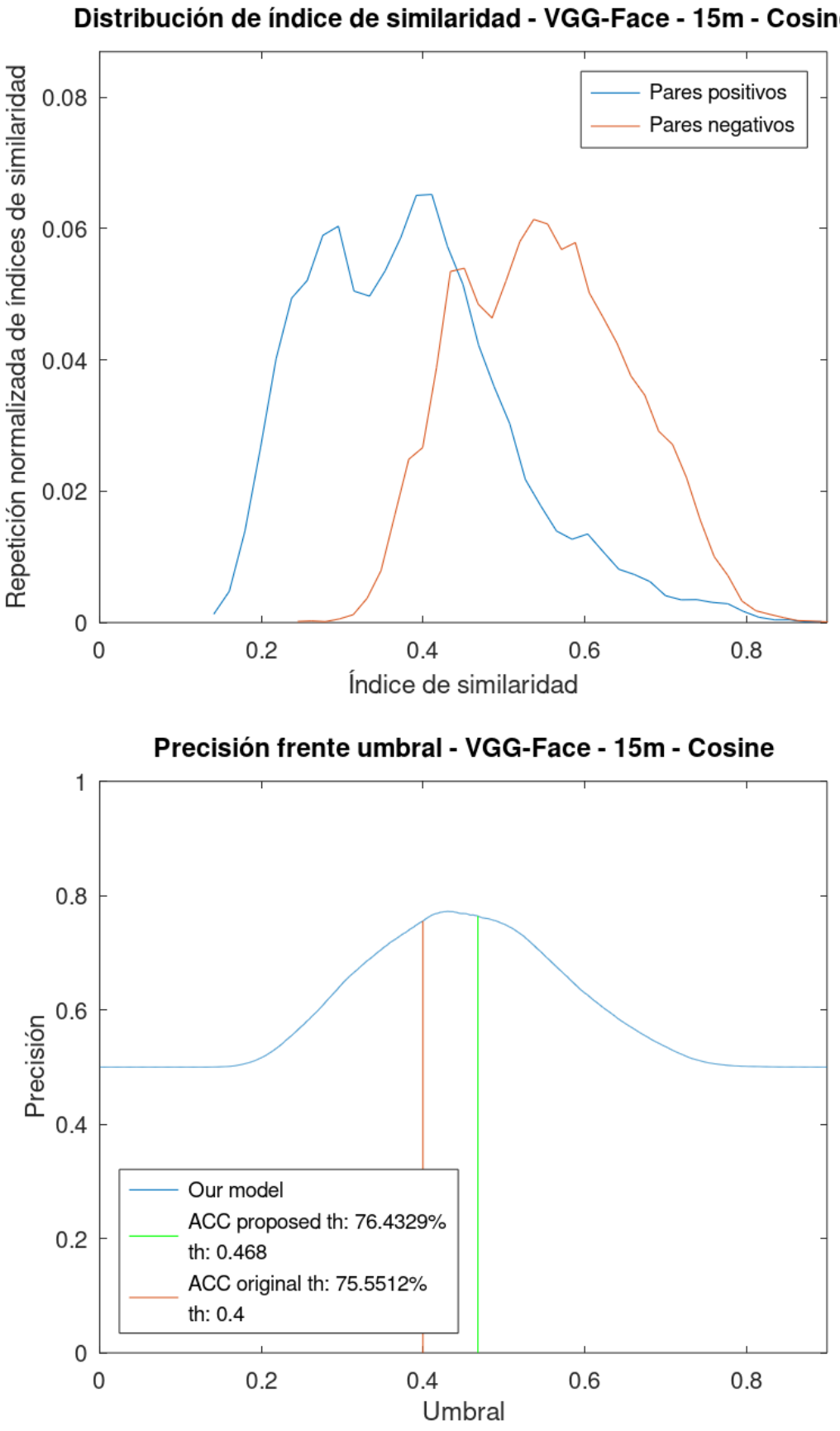


Fig 26. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de VGG-Face. La métrica usada es* ***distancia de coseno***

b) Distancia euclídea

A continuación, se pueden ver los mismos gráficos obtenidos para la distancia de coseno, pero para la distancia euclídea. La principal diferencia es el rango de valores de índices de similaridad que se obtienen. Como la distancia de coseno se obtiene a partir restando 1 menos el coseno calculado su rango esta entre 0 y 2 mientras que la distancia euclídea no tiene un rango de valores definido y puede tomar cualquiera desde 0 hasta infinito. Por ello, se puede observar que para FaceNet512 se obtienen valores hasta 40, mientras que para OpenFace solo se consigue hasta 1,5.

Al igual que en el caso anterior se han obtenido dos gráficas para cada uno de los algoritmos de verificación usados a 5 metros de distancia: FaceNet (**Fig 27**), FaceNet512 (**Fig 28**), OpenFace (**Fig 29**), ArcFace (**Fig 30**), VGG-Face (**Fig 31**).

Las curvas de las gráficas de FaceNet, FaceNet512 y ArcFace se diferencian claramente, mientras que para OpenFace y VGG-Face se puede apreciar que hay un ligero solapamiento entre ellas, por lo que no se logran valores muy altos de precisión en estos casos.

Es muy significativa la mejora que se consigue en FaceNet ya que se pasa de una precisión del 68% a casi 94% únicamente ajustando correctamente el umbral como se ha hecho en este proyecto.

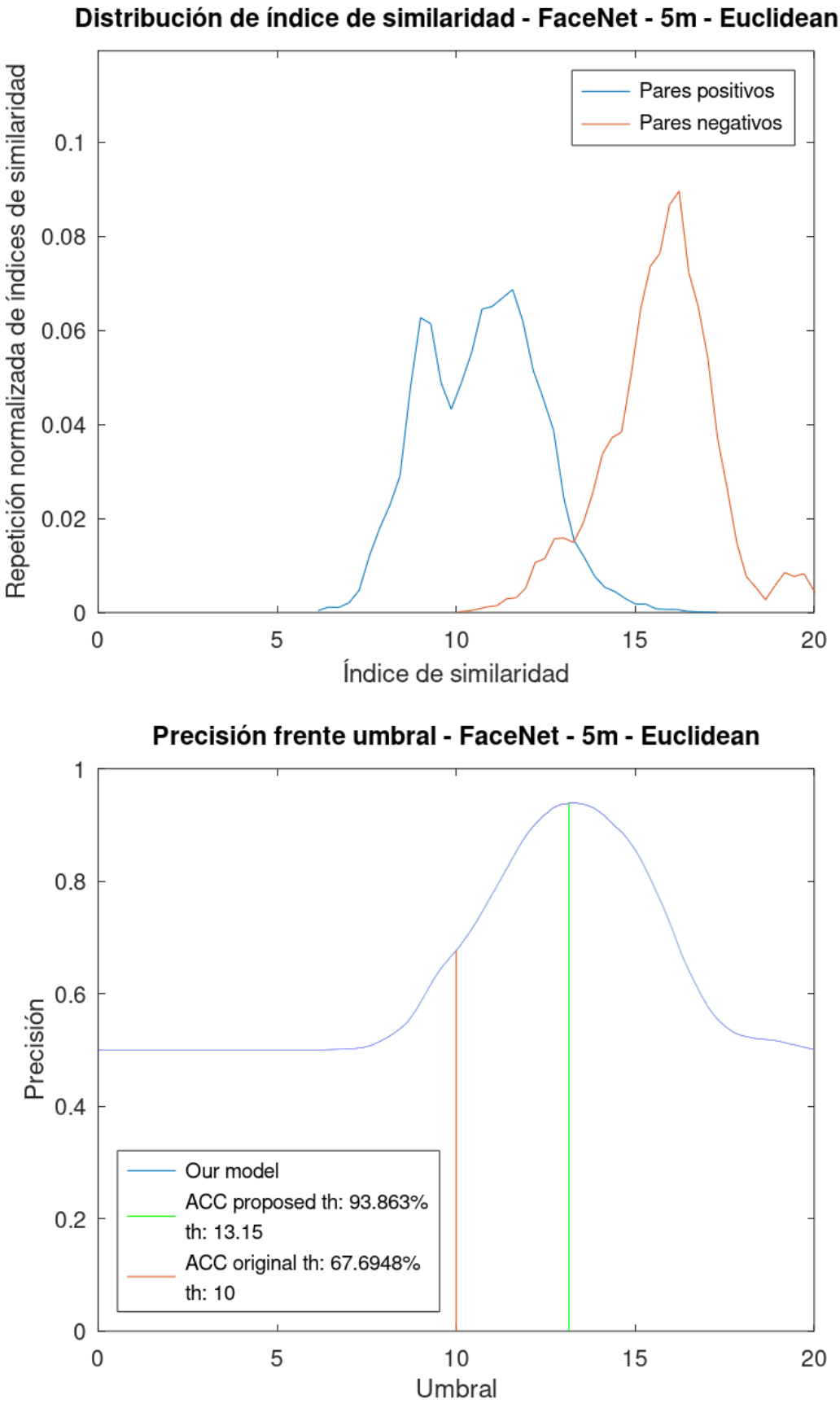


Fig 27. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de FaceNet. La métrica usada es* ***distancia euclídea***

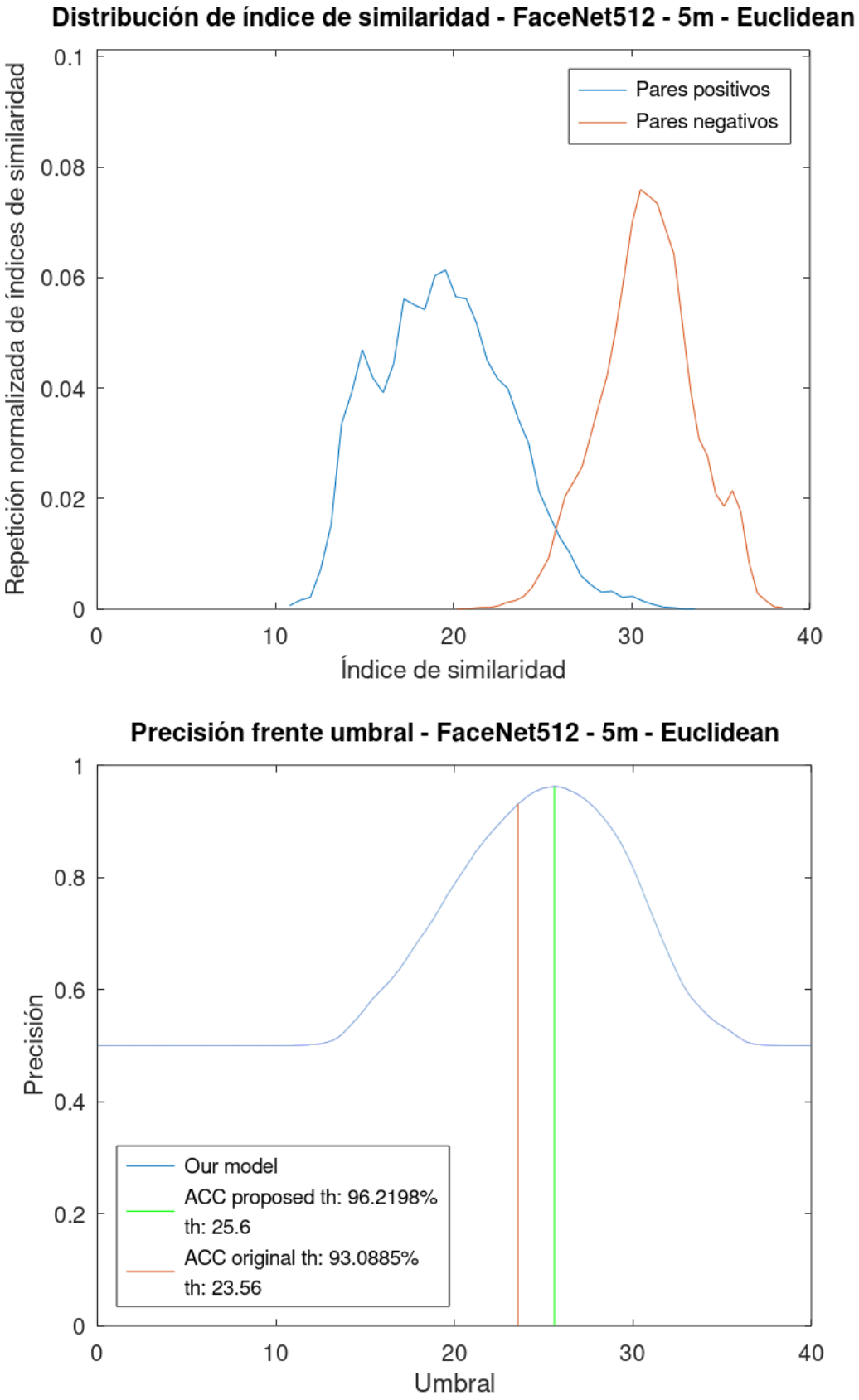


Fig 28. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de FaceNet512 evaluados. La métrica usada es* ***distancia euclídea***

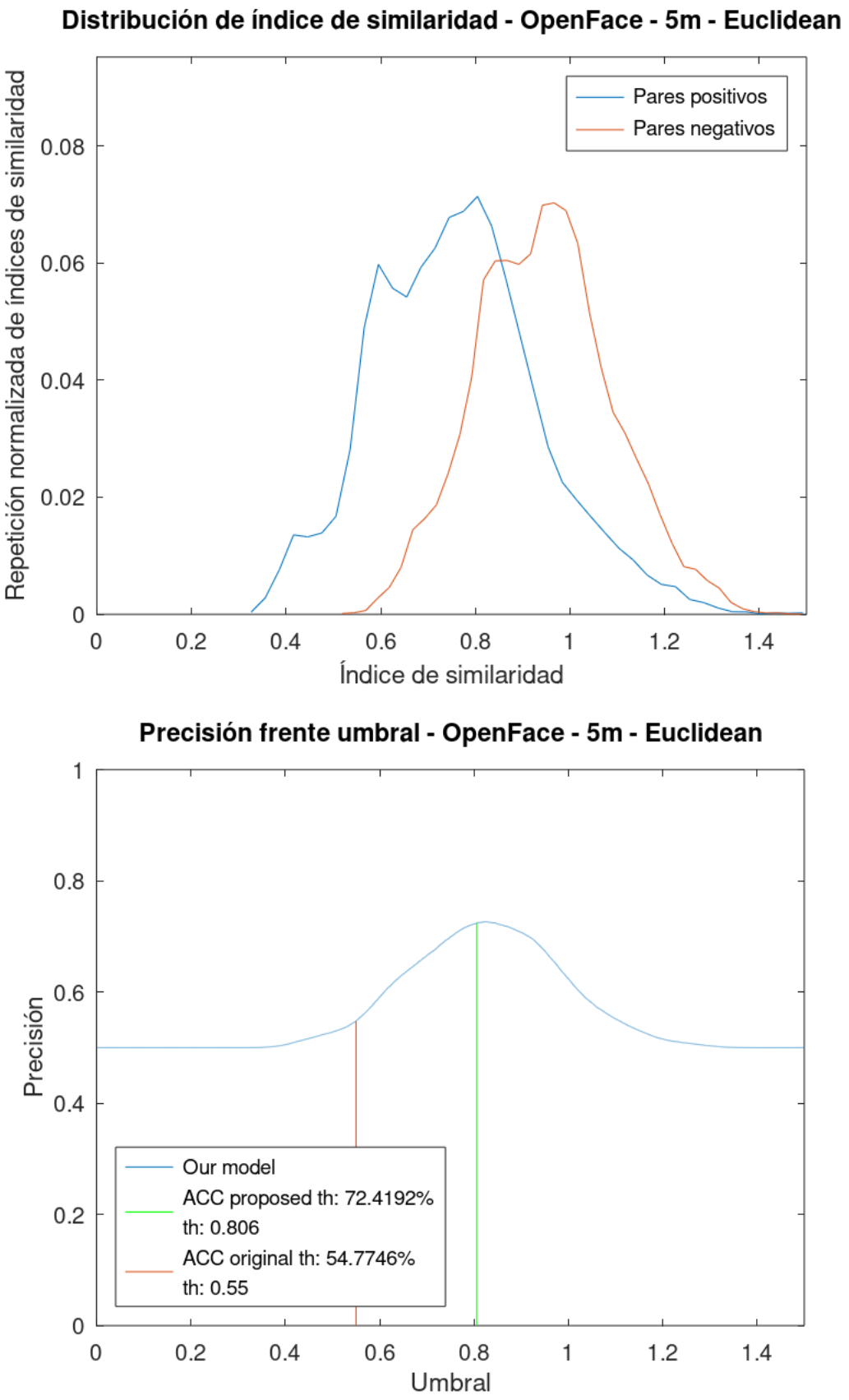


Fig 29. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de OpenFace evaluados. La métrica usada es* ***distancia euclídea***

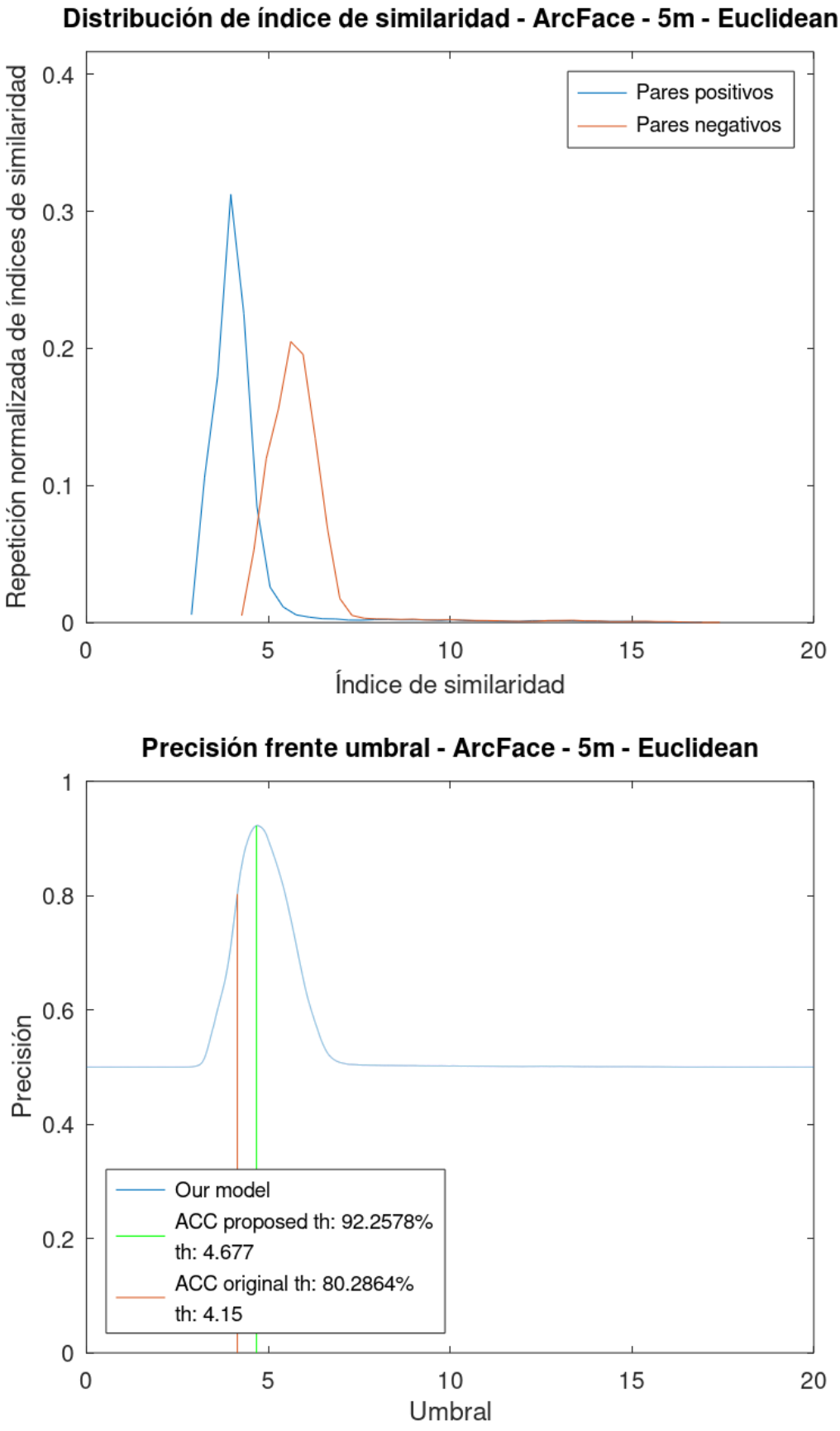


Fig 30. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de ArcFace evaluados. La métrica usada es* ***distancia euclídea***

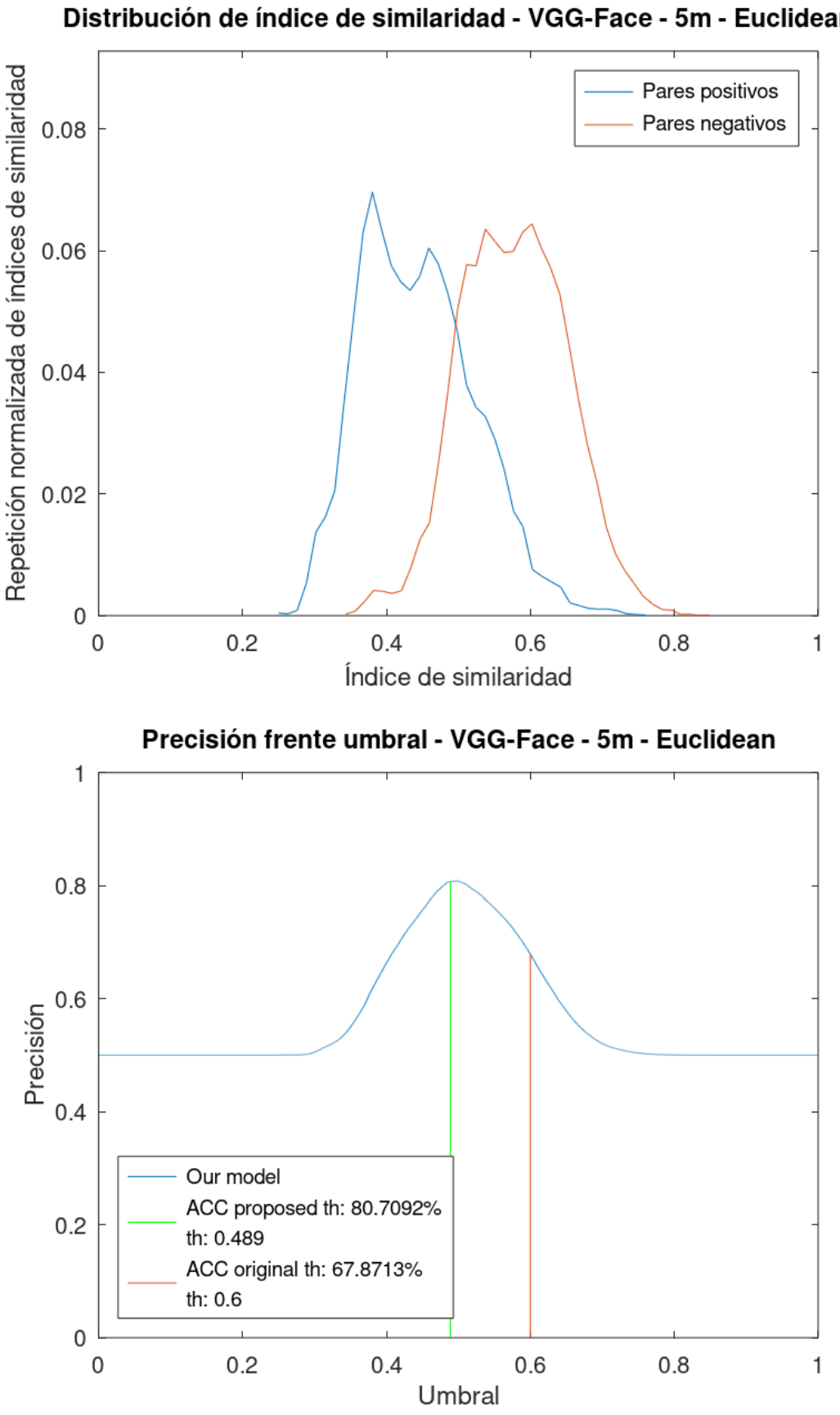


Fig 31. *Distribución de distancias y la precisión en función del umbral usado a* ***5 metros*** *de distancia de VGG-Face evaluados. La métrica usada es* ***distancia euclídea***

Por último, se han obtenido las mismas gráficas, pero usando una distancia de 15 metros para los cinco algoritmos de verificación facial: FaceNet (**Fig 32**), FaceNet512 (**Fig 33**), OpenFace (**Fig 34**), ArcFace (**Fig 35**), VGG-Face (**Fig 36**).

Al igual que ocurría con la distancia de coseno, las curvas a esta distancia están mucho más solapadas, ya que los índices de similaridad de los pares positivos y negativos son más parecidos al bajar la resolución de las caras. Esto se puede apreciar especialmente en la gráfica de FaceNet, donde ambas curvas están totalmente solapadas, por lo que es difícil obtener una alta precisión debido al gran número de falsos positivos que se obtendrán.

El resto de los algoritmos, aunque las curvas estén algo más solapadas, se pueden seguir diferenciando claramente, por lo que su precisión no baja notablemente en comparación con las distancias medias y cercanas.

Es curiosa además la gráfica de ArcFace, porque tanto la curva de pares positivos como la de negativos tienen un pico que representa el mayor número de índices de similaridad, pero después presentan unas colas que llegan a valores mayores a 20.

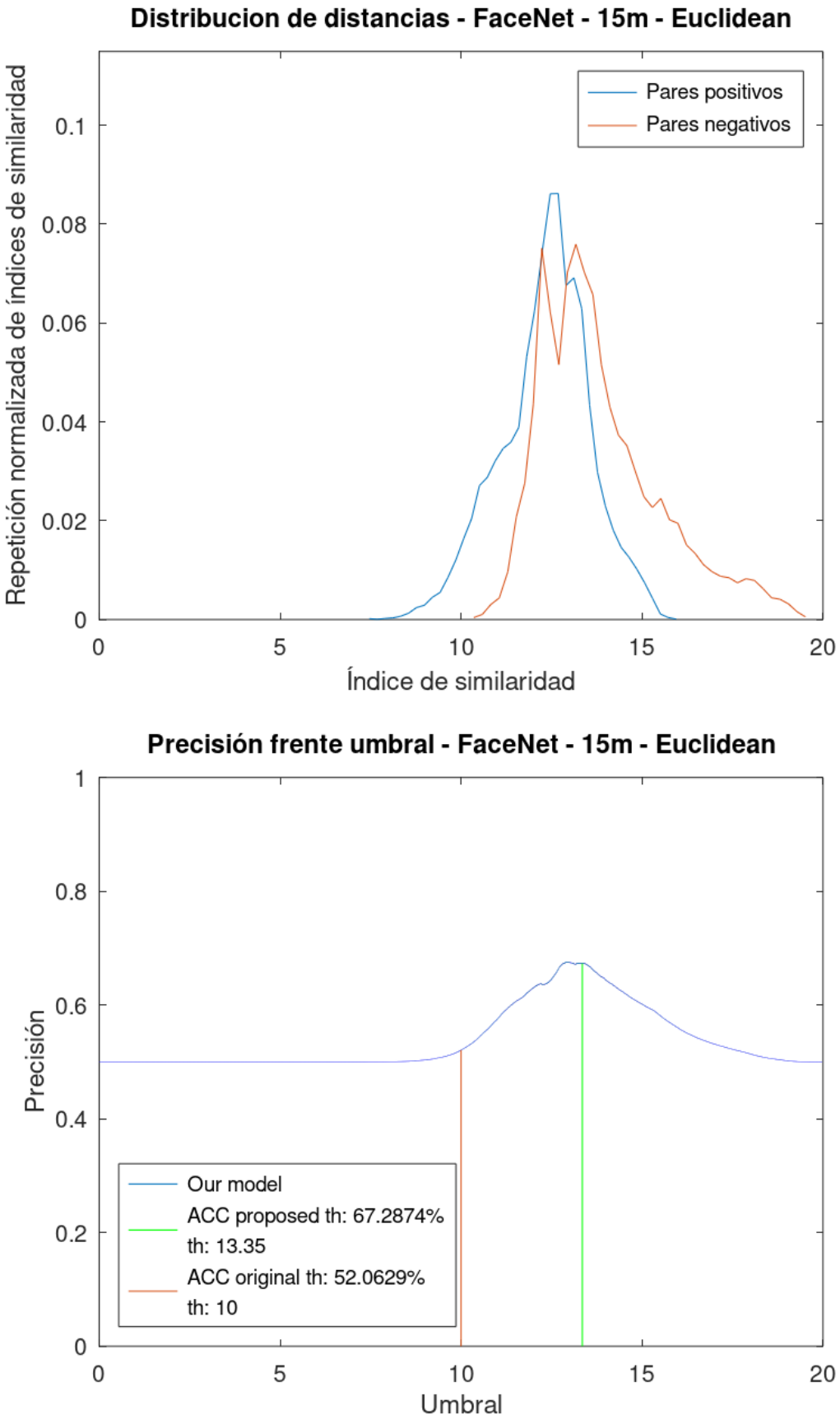


Fig 32. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de* ***FaceNet*** *evaluados. La métrica usada es* ***distancia euclídea***

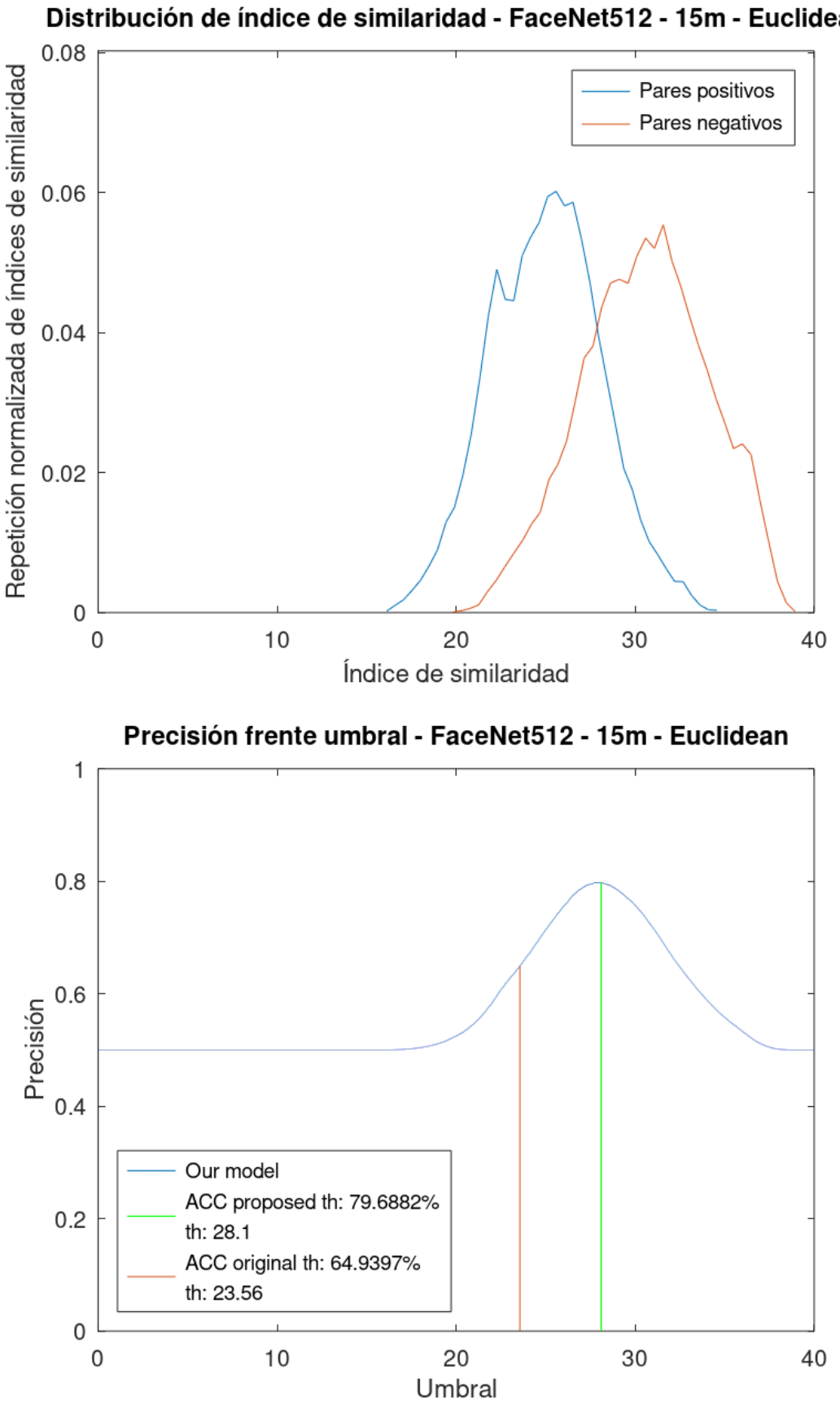


Fig 33. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de* ***FaceNet512*** *evaluados. La métrica usada* ***es distancia euclídea***

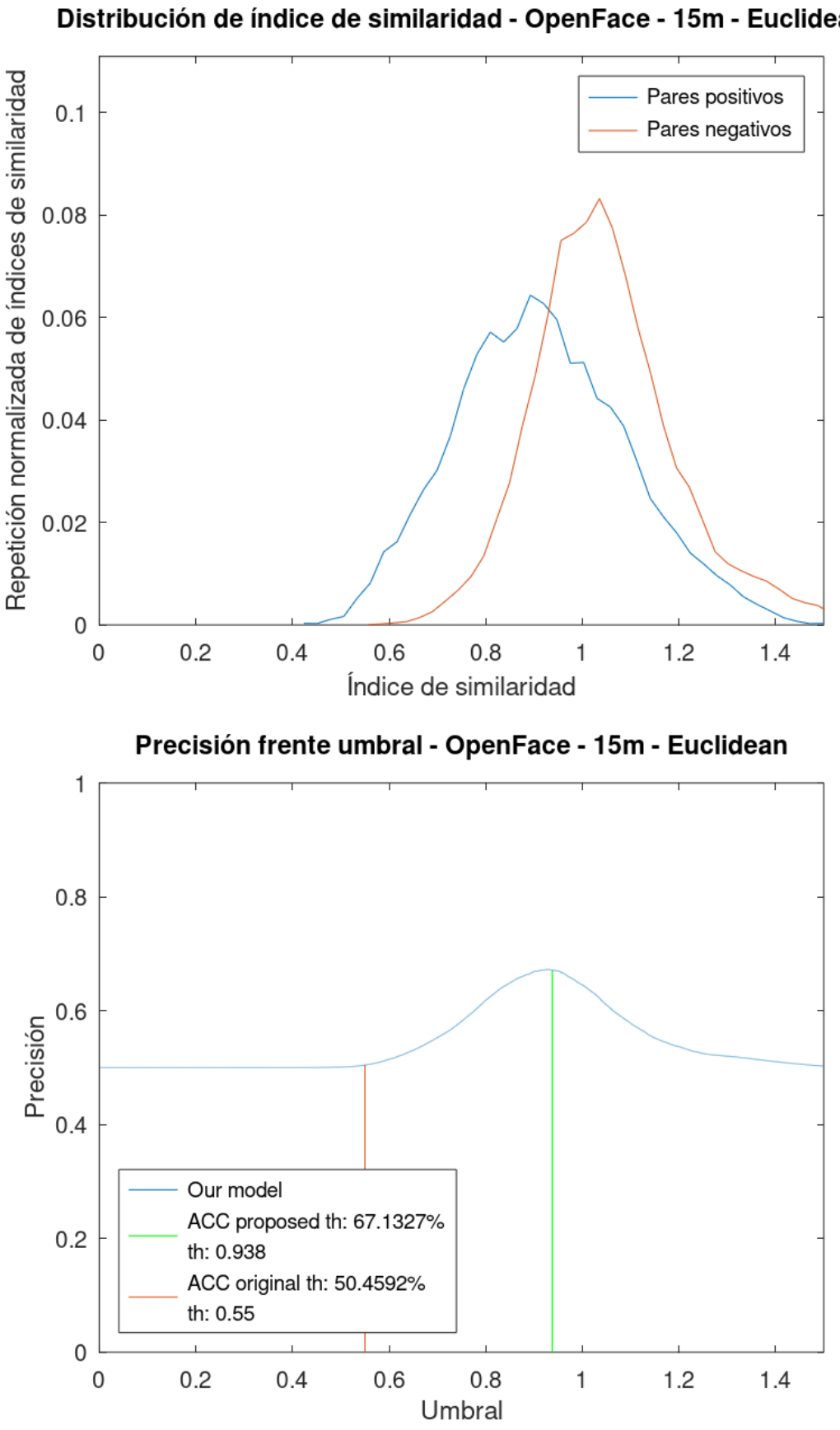


Fig 34. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de* ***OpenFace*** *evaluados. La métrica usada* ***es distancia euclídea***

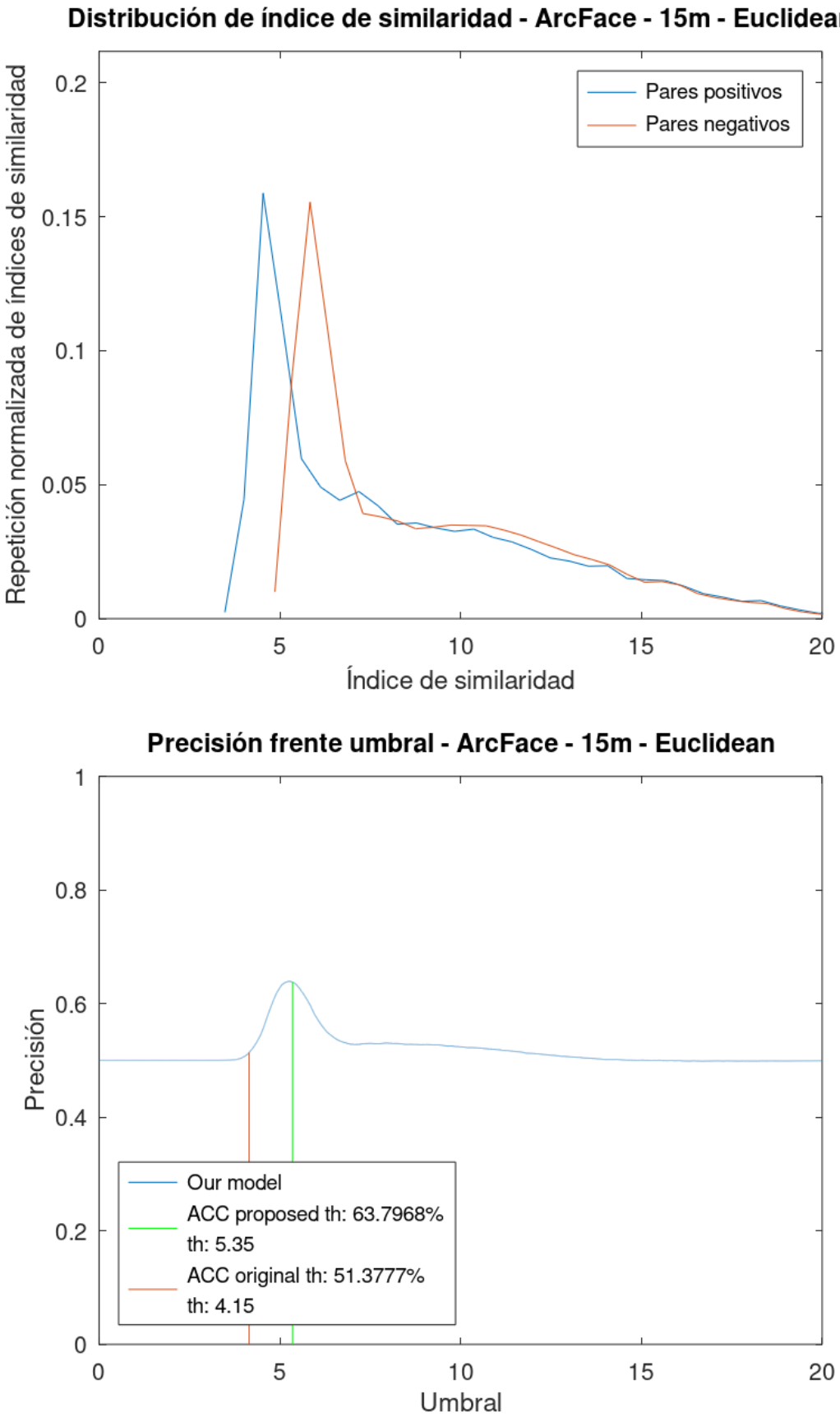


Fig 35. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de* ***ArcFace*** *evaluados. La métrica usada es* ***distancia euclídea***

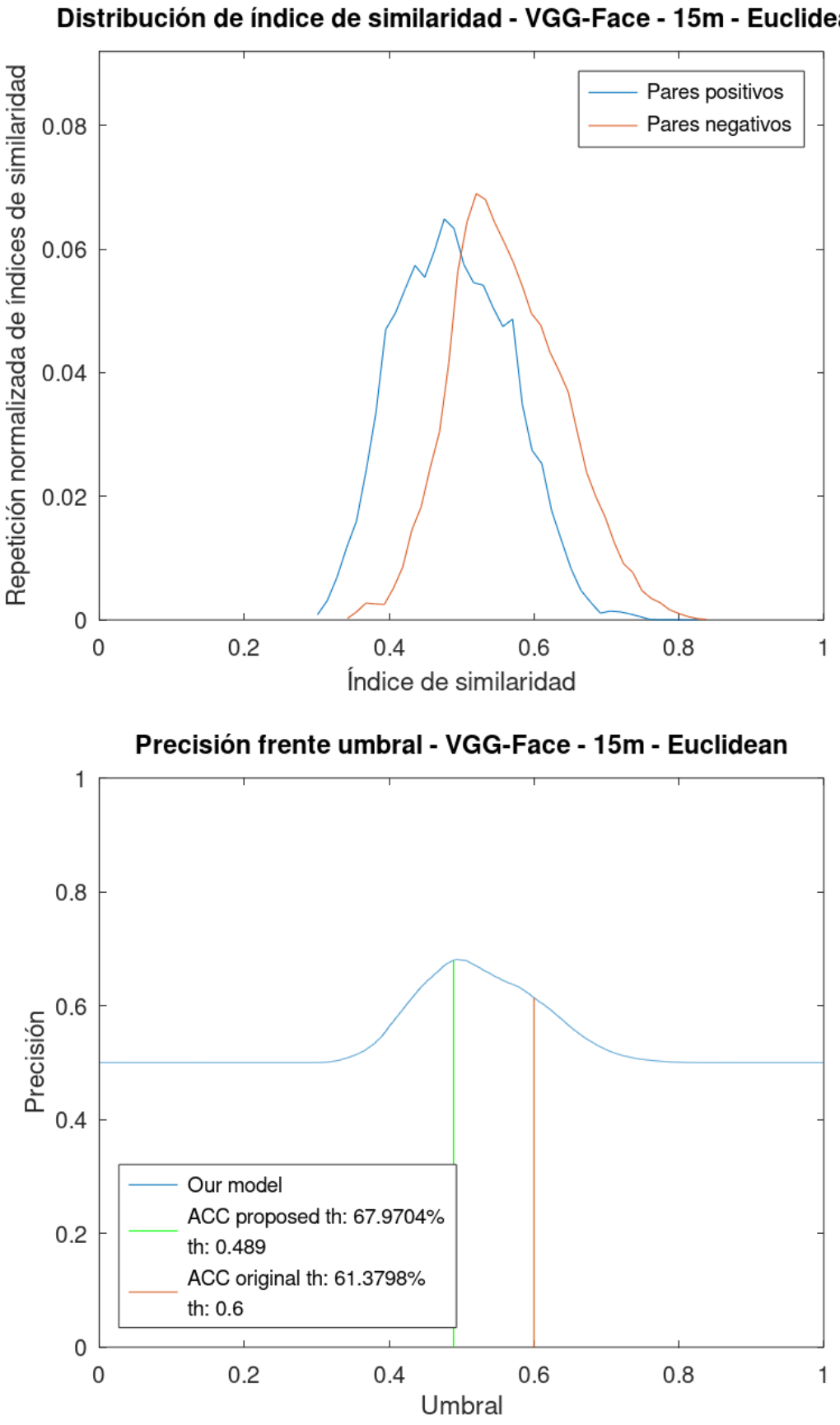


Fig 36. *Distribución de distancias y la precisión en función del umbral usado a* ***15 metros*** *de distancia de* ***VGG-Face*** *evaluados. La métrica usada es* ***distancia euclídea***

Umbrales recomendados

En la **Tabla 5** y **Tabla 6** se pueden ver los umbrales predefinidos recomendados por el *framework* DeepFace [21] usado para la realización del proyecto usando como métricas la distancia de coseno y la distancia euclídea, respectivamente. Mientras que en la **Tabla 7** y **Tabla 8** se pueden observar los umbrales dinámicos recomendados calculados previamente para la distancia de coseno y euclídea respectivamente. Para todos los algoritmos se pueden apreciar diferencias notables entre el umbral predefinido y los dinámicos calculados. Esto se debe a que en nuestro caso se ha querido maximizar la precisión de los algoritmos mientras que en este *framework* se maximizó la ganancia de información usando un algoritmo de decisión en árbol.

Además, los umbrales predefinidos están solo calculados para distancias muy cercanas donde se puede apreciar la cara de la persona con alta calidad. Mientras que, en nuestro caso de uso al usar distancias lejanas, los umbrales son más altos. Hasta la distancia más cercana usada en este proyecto (2 metros) es más alta que la distancia habitual de reconocimiento facial, que suele ser de centímetros, por ejemplo, al reconocer una cara en un teléfono móvil para desbloquearlo, pero por razones de seguridad no se puede grabar a menos de 2 metros de una persona con un dron, por lo que es lógico que los umbrales predefinidos y los calculados en este proyecto difieran en sus valores.

Tabla 5. *Umbrales predefinidos para los algoritmos seleccionados usando la* ***distancia de Coseno***

Algoritmos	Umbral Predefinido
FaceNet	0.4
FaceNet512	0.3
OpenFace	0.1
ArcFace	0.68
VGG-Face	0.4

Tabla 6. *Umbrales predefinidos para los algoritmos seleccionados usando la* ***distancia Euclídea***

Algoritmos	Umbral Predefinido
FaceNet	10
FaceNet512	23.56
OpenFace	0.55
ArcFace	4.15
VGG-Face	0.6

Tabla 7. *Umbrales recomendados para los algoritmos seleccionados usando la distancia de* ***Coseno***

Algoritmos	Cerca	Medio	Lejos	Muy lejos
FaceNet	0.621	0.634	0.763	0.889
FaceNet512	0.611	0.632	0.807	0.905
OpenFace	0.325	0.345	0.44	0.436
ArcFace	0.761	0.798	0.929	1.002
VGG-Face	0.364	0.382	0.468	0.645

Tabla 8. *Umbrales recomendados para los algoritmos seleccionados usando la distancia* ***Euclídea***

Algoritmos	Cerca	Medio	Lejos	Muy lejos
FaceNet	13.15	13.24	13.35	12.98
FaceNet512	25.6	26.4	28.1	27.5
OpenFace	0.806	0.83	0.938	0.934
ArcFace	4.68	4.99	5.35	8.624
VGG-Face	0.489	0.507	0.526	0.559

CAPÍTULO VI

ANÁLISIS DE LOS RESULTADOS OBTENIDOS

En este penúltimo capítulo se van a analizar los resultados obtenidos empíricamente usando los nuevos umbrales dinámicos que se han calculado en el anterior capítulo, que se corresponden con la principal contribución de este proyecto. Se va a analizar la nueva precisión obtenida por los algoritmos y también el tiempo de inferencia de cada uno de ellos, para poder realizar una comparativa empírica tiempo-precisión.

Evaluación de la precisión

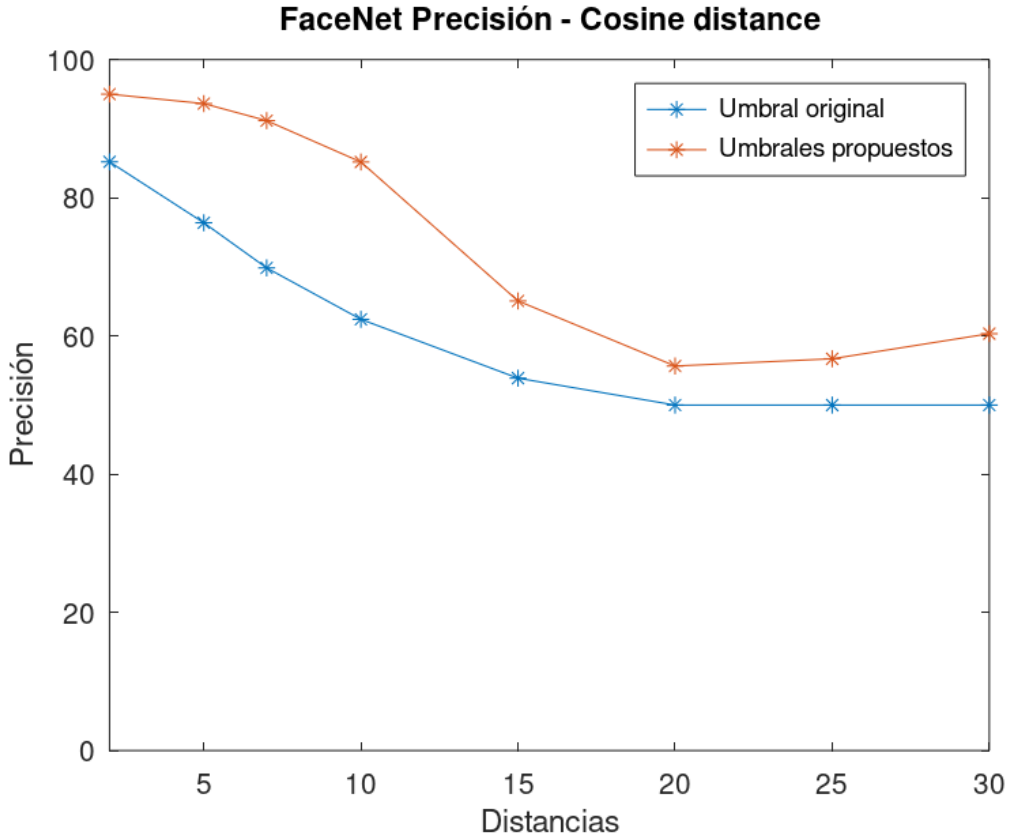
Primero, se va a analizar la precisión obtenida por los algoritmos de verificación facial usando los nuevos umbrales dinámicos calculados. Para ello, se van a coger los valores de umbrales calculados en el anterior capítulo y se va a ejecutar el diseño propuesto con el dataset creado. Se va a analizar la precisión usando las dos métricos usadas a lo largo de todo el proyecto: distancia de coseno y distancia euclídea.

a) Distancia de coseno

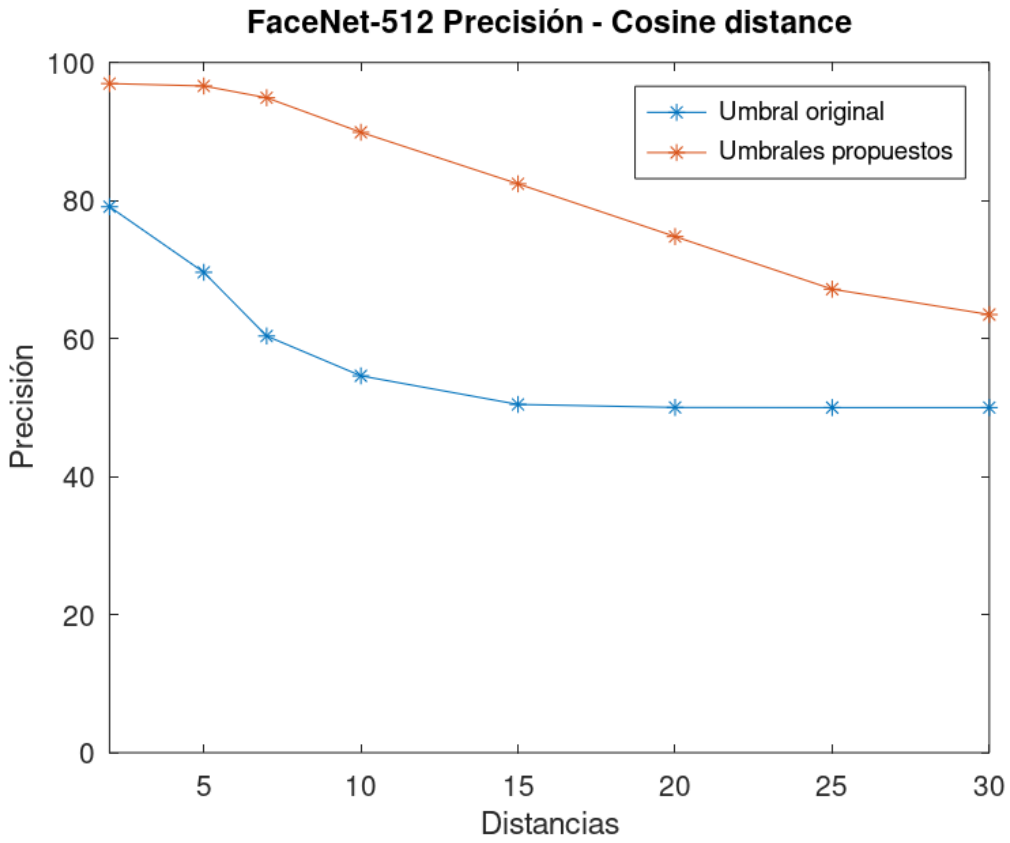
Primero se va a analizar la precisión usando la distancia de coseno como métrica. La **Fig. 37** muestra cinco gráficas con la precisión de los algoritmos, una para cada uno de los algoritmos de verificación facial. Cada gráfica muestra la precisión del algoritmo a lo largo de todas las distancias contenidas en el *dataset* usando el umbral original y los nuevos umbrales dinámicos propuestos. Como se puede ver, FaceNet y FaceNet512 mejoran a todas las distancias respecto al umbral original, consiguiendo las mayores mejoras a distancias medias (entre 7 y 10 metros). OpenFace se ha conseguido mejorar notablemente, de tener una media de 50% de precisión a obtener más de un 60% en prácticamente todas las distancias, por lo que supone una gran mejora, aunque no se consiga una precisión muy alta. Usando ArcFace no se consiguen grandes mejoras a distancias cortas ya que se partía de una precisión muy alta, pero a largas y medias distancias se consigue una mejora notable de la precisión.

Por último, VGG-Face consigue una ligera mejora en distancias muy cortas, mientras que la mayor mejora la consigue a largas y muy largas distancias. Por lo que este algoritmo, con esta métrica será muy útil de usar si se requiere verificar a largas distancias, como es nuestro caso de uso.

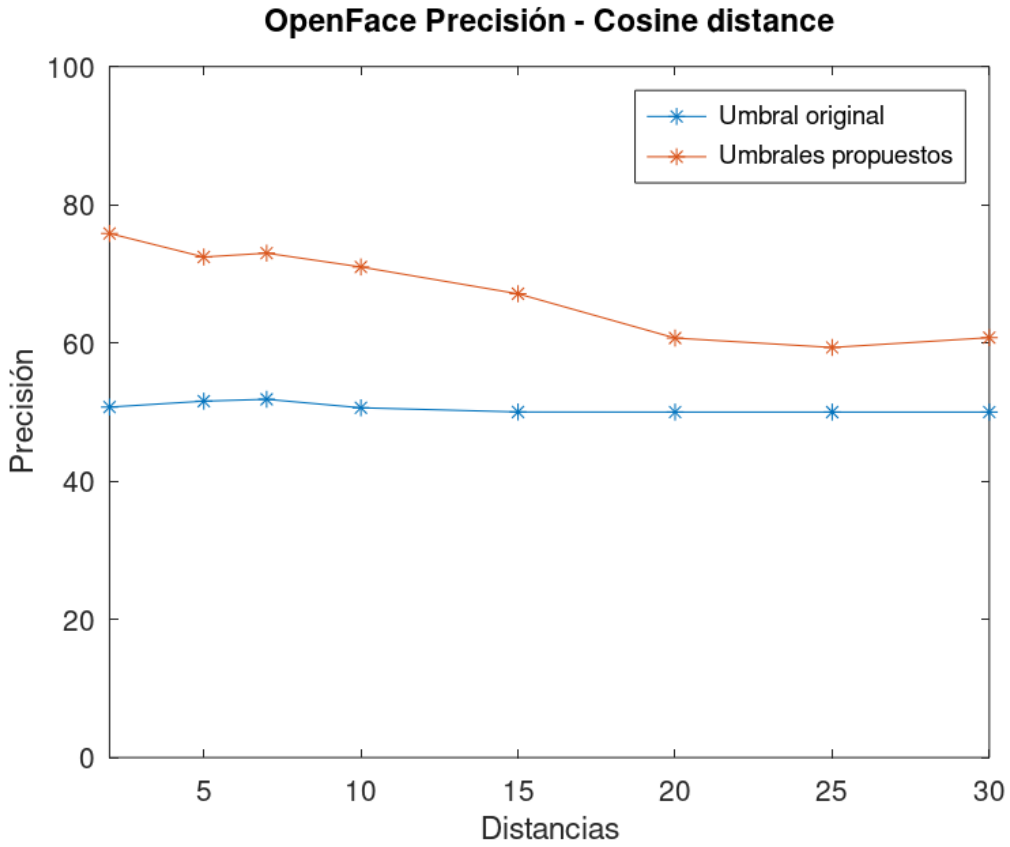
Después de haber analizado todas las gráficas, se puede concluir que el algoritmo que contiene las mayores mejoras es FaceNet512, obteniendo una precisión notablemente alta, aunque a muy largas distancias sea superado por VGG-Face. Si se quisiera usar esta métrica, el algoritmo recomendado sin duda sería FaceNet512 ya que obtiene buenos resultados a todas las distancias analizadas. Si solo se quisiera verificar a largas distancias, el elegido sería VGG-Face.



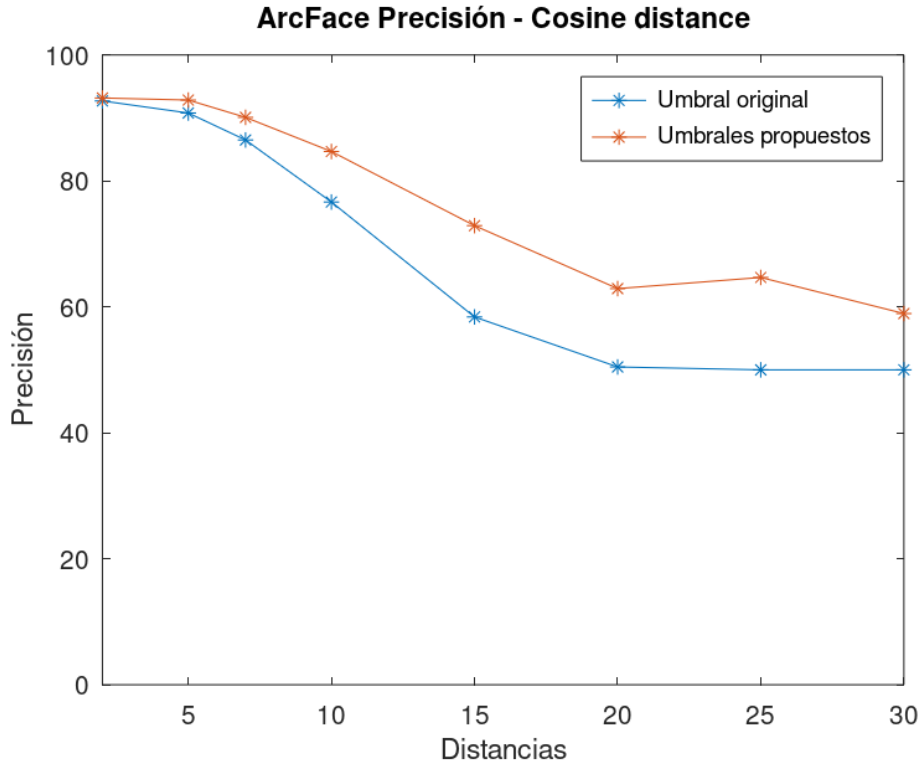
a) FaceNet



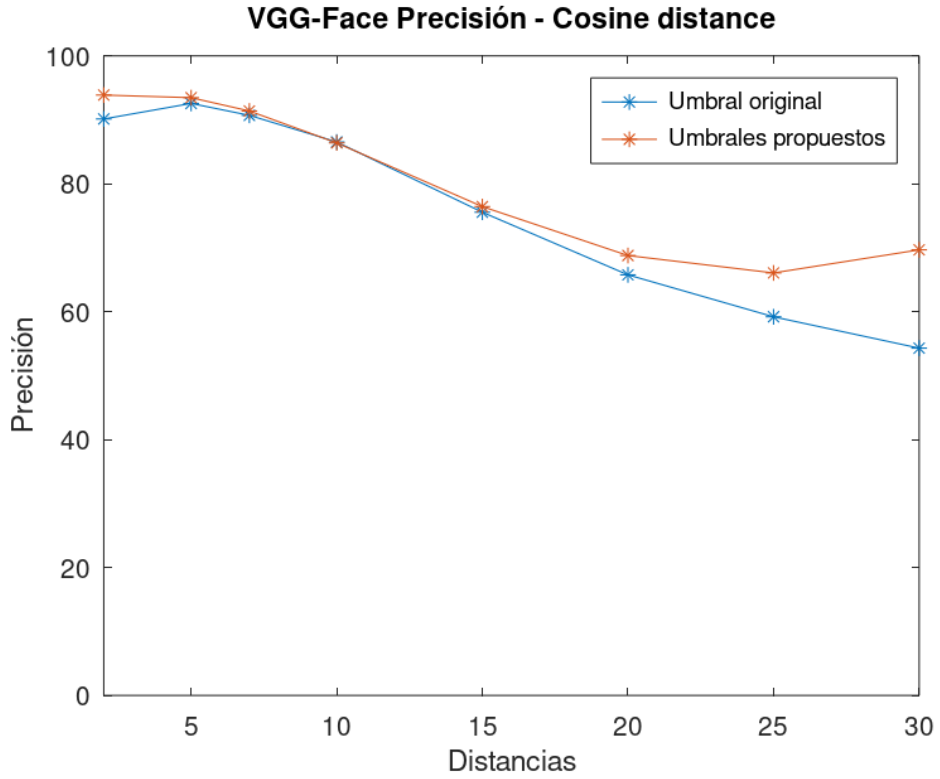
b) FaceNet512



c) OpenFace



d) ArcFace



e) VGG-Face

Fig. 37. *Precisión de los algoritmos a diferentes distancias usando los umbrales predefinidos y los dinámicos usando* ***distancia de coseno*** *como métrica*

En la **Tabla 9** se puede conseguir la precisión obtenida usando los umbrales dinámicos calculados para cada uno de los algoritmos de verificación facial. La métrica usada ha sido distancia de coseno. Como se puede observar, a cortas distancias los mejores resultados los consiguen los dos FaceNet y ArcFace, mientras que a largas distancias el mejor resultado lo tiene VGG-Face.

Tabla 9**. Precisión de los diferentes algoritmos usando los umbrales dinámicos y **distancia de coseno

Algoritmos	2 m	5 m	7 m	10 m	15 m	20 m	25 m	30 m
FaceNet	95	93.63	91.16	85.18	65.07	55.65	56.7	60.33
FaceNet512	96.95	96.6	94.89	89.89	82.42	74.78	67.16	63.48
OpenFace	75.85	72.44	73	71	67.133	60.7	59.36	60.76
ArcFace	96.17	92.84	90.09	84.67	72.9	62.9	64.67	58.94
VGG-Face	93.87	93.46	91.38	86.39	76.43	68.78	66.06	69.67

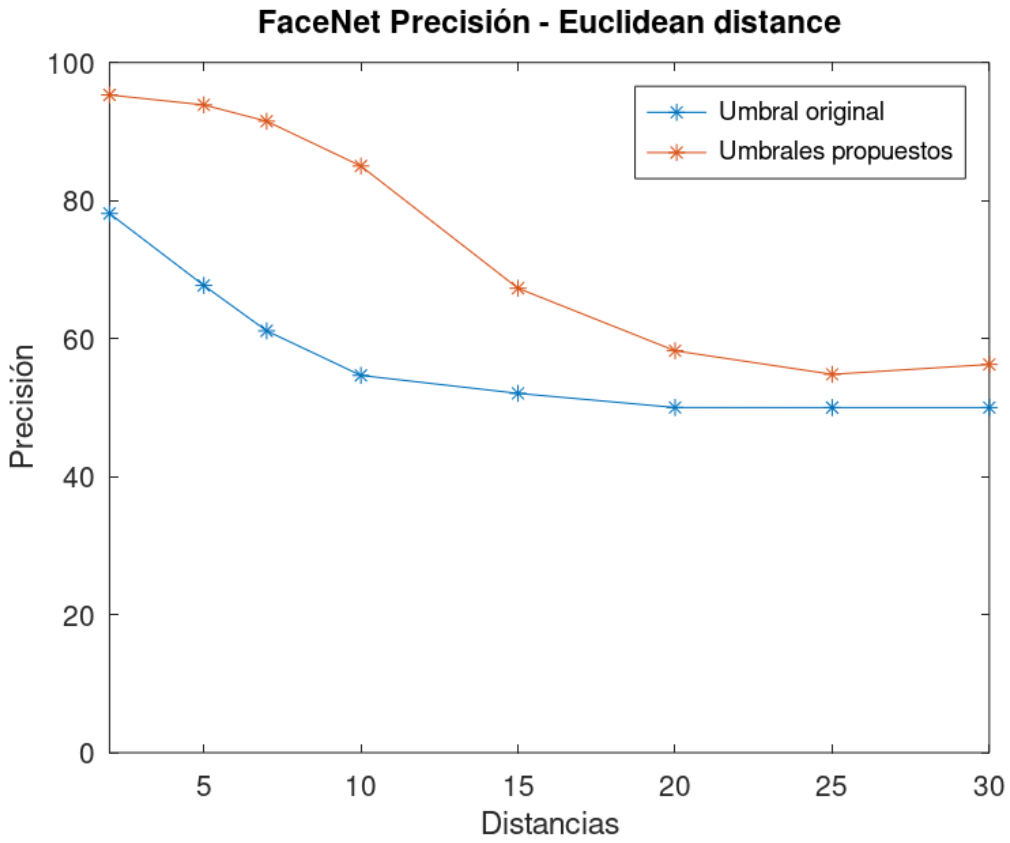
b) Distancia euclídea

La **Fig. 38** muestra las mismas gráficas que el anterior apartado, pero usando la distancia euclídea como métrica. Se pueden ver cambios significativos en las gráficas. Por ejemplo, FaceNet mejora notablemente a cortas distancias, pero a medida que nos alejamos la mejora es menor. Mientras que FaceNet512 tiene poca mejora a cortas distancias, pero alta a largas distancias. Esto se debe a que en distancias cercanas ya tenía una precisión muy alta, por lo que es difícil lograr aumentarla más.

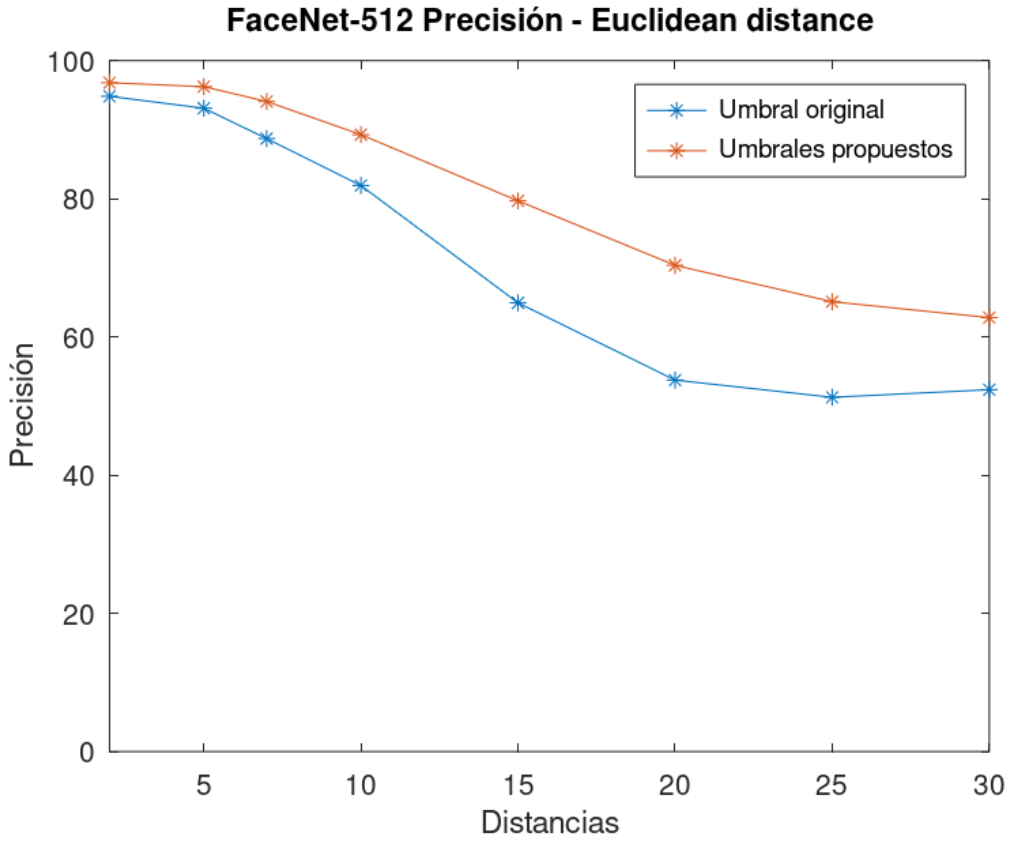
OpenFace consigue prácticamente los mismos resultados que usando la distancia de coseno. Se pasa de tener una precisión cercana a 50% para todas las distancias a superar prácticamente el 60% en todas. Lo que no es una precisión muy alta, pero pasa a convertirse en un algoritmo aceptable si se prefiere alta velocidad a precisión.

ArcFace mejora su precisión a cortas distancias, mientras que a largas distancias (más de 20 metros) no se consigue prácticamente ninguna mejora. Por lo que este algoritmo con esta métrica no será útil para nuestro caso de uso, al ser necesario poder verificar una persona a una distancia lejana.

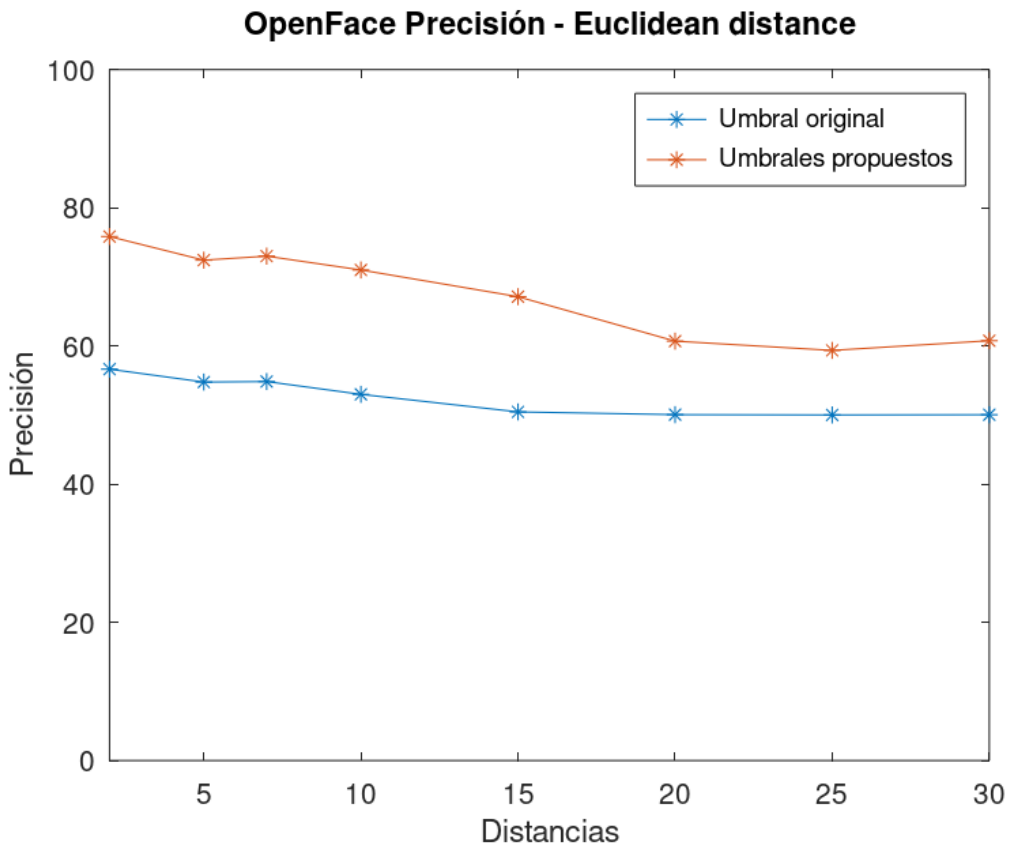
Por último, VGG-Face mejora en cortas distancias, mientras que a largas distancia no mejora prácticamente y hasta se consigue un resultado peor a 20 metros de distancia. Por lo que este algoritmo con esta métrica tampoco nos será útil para verificar personas a largas distancias.



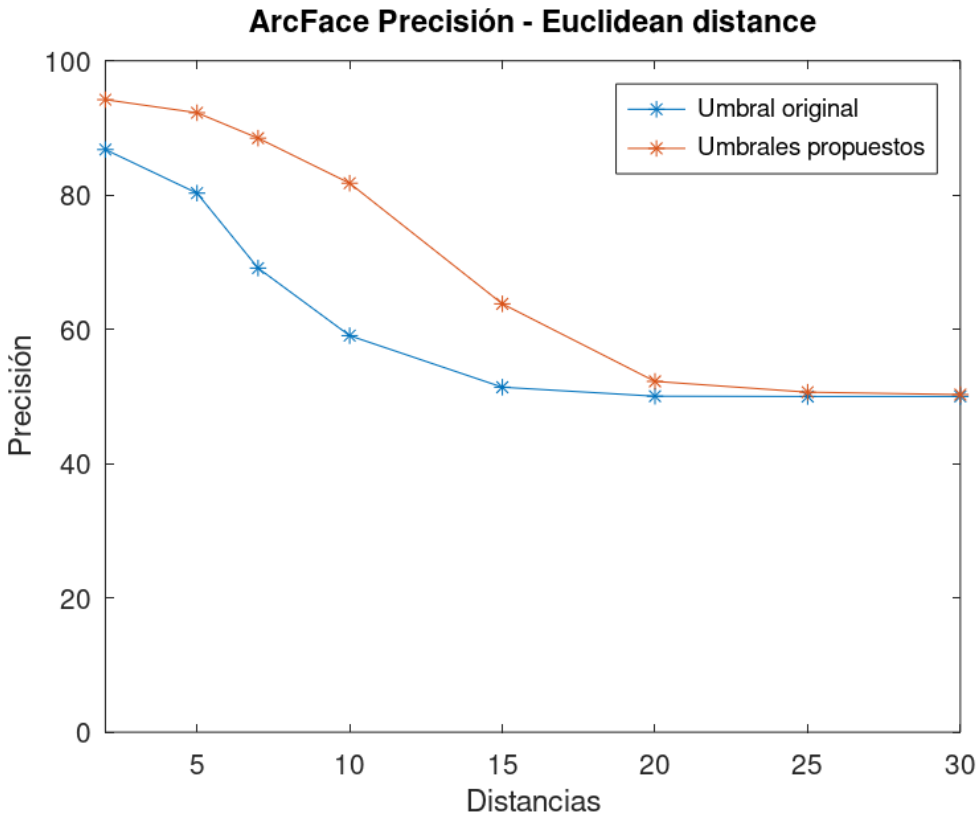
a) FaceNet



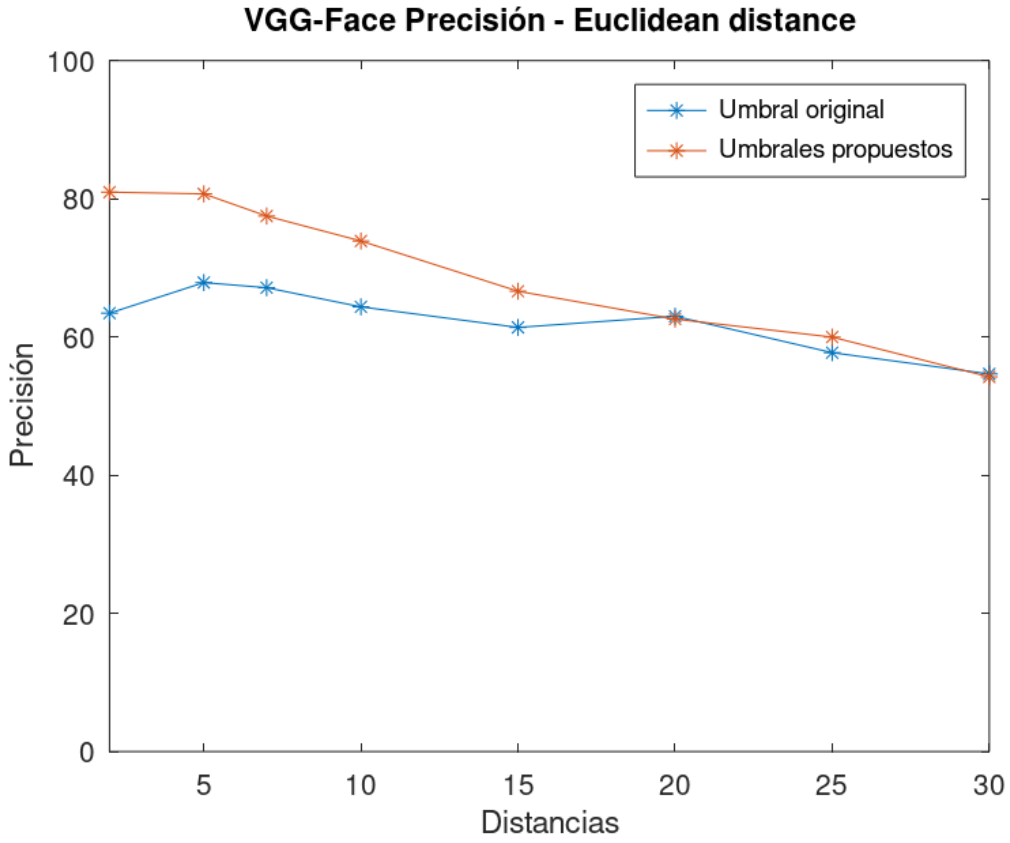
b) FaceNet512



c) OpenFace



d) ArcFace



e) VGG-Face

Fig. 38. *Precisión de los algoritmos a diferentes distancias usando los umbrales predefinidos y los dinámicos usando* ***distancia de euclídea*** *como métrica*

Finalmente, **Tabla 10** muestra la precisión de cada uno de los algoritmos de verificación facial usando los umbrales dinámicos calculados y siendo la métrica la distancia euclídea. Como se puede ver a cortas distancias los que mejores resultados consiguen son los dos FaceNet y ArcFace, mientras que a largas distancias

Tabla 10. *Precisión de los diferentes algoritmos usando los umbrales dinámicos y* ***distancia euclídea***

Algoritmos	2 m	5 m	7 m	10 m	15 m	20 m	25 m	30 m
FaceNet	95.3	93.86	91.49	85	67.29	58.23	54.82	56.25
FaceNet512	96.8	96.22	94.08	89.28	79.7	70.37	65.1	62.8
OpenFace	75.84	72.42	73	71	67.133	60.7	59.36	60.76
ArcFace	94.211	92.258	88.477	81.766	63.8	52.25	50.64	50.28
VGG-Face	80.96	80.71	77.5	73.89	66.62	62.588	59.98	54.22

Evaluación empírica de la mejor métrica para cada algoritmo

En este apartado se va a analizar cuál es la mejor métrica que se puede aplicar para cada uno de los algoritmos de verificación facial. En la **Tabla 11** se puede ver la métrica recomendada para cada uno de ellos y los rangos de mejora que hay para cada métrica. Como se puede apreciar, FaceNet, FaceNet512 y OpenFace mejoran en todas las distancias usando cualquiera de las dos métricas. No obstante, para los dos FaceNet se recomienda usar la distancia euclídea ya que las mejoras que se consiguen son mayores. Para OpenFace no se ha apreciado gran diferencia entre ambas métricas por lo que usando cualquiera de las dos se consiguen mejoras parecidas.

Tabla 11. *Rangos de la escala donde se consiguen las mejores mejoras con el umbral dinámico para cada algoritmo y cuál es la métrica recomendada para cada uno de ellos*

Algoritmo	Rangos de mejora		Métrica recomendada
	Distancia Coseno	Distancia Euclídea	
FaceNet	Todas	Todas	Euclídea
FaceNet512	Todas	Todas	Euclídea
OpenFace	Todas	Todas	Ambas
ArcFace	Cerca, Medio y Lejos	Todas	Coseno
VGG-Face	Cerca y Medio	Lejos y Muy lejos	Coseno – Cercano Euclídea - Lejano

Por otro lado, ArcFace mejora en todas las distancias con la distancia euclídea. Con la distancia de coseno no se logra mejorar a largas distancias (más de 25 metros). Aun así, la métrica recomendada es la distancia de coseno al conseguirse una mejora de la precisión para el resto de las distancias.

Finalmente, con VGG-Face se consigue una mejora en distancias cercanas y medianas (menos de 10 metros) usando distancia de coseno. Mientras que para largas distancias (más de 10 metros) se consiguen mejores resultados usando la distancia euclídea. Por lo que la recomendación es si se intenta verificar una persona a cortas distancias se deberá usar distancia de coseno, mientras que si la persona está situada lejos se usará distancia euclídea.

Evaluación del tiempo de inferencia

Finalmente, se van a comparar los tiempos de inferencia de los algoritmos seleccionados. Para obtener los tiempos de inferencia se ha ejecutado el *pipeline* sobre un video compuesto de aproximadamente 900 *frames* y se ha obtenido el tiempo de procesamiento

de todo el *pipeline* para cada uno de ellos. Las *features* de la cara a comparar se han extraído previamente de la ejecución por lo que el tiempo obtenido consiste en una única ejecución del algoritmo de extracción de características y del detector facial.

En la **Fig. 39** se puede ver la comparativa de los tiempos de inferencia. Para obtener el resultado se ha realizado el promedio acumulado. Es decir, el valor del tiempo del último *frame* corresponde a la media de todos los tiempos de inferencia. En la gráfica vienen representados los cuatro algoritmos utilizados, incluyendo los dos diferentes *backbones* del algoritmo FaceNet. Los valores obtenidos corresponden al tiempo total de inferencia del *pipeline*, por lo que se incluyen las cinco etapas: detección facial, preprocesamiento, extracción de características, cálculo de distancias y toma de decisión.

Como se puede apreciar en la figura, el algoritmo más lento con mucha diferencia es VGG-Face, teniendo una media de 275 ms de tiempo de ejecución. Esto significa que usando este algoritmo en nuestro *pipeline* podríamos obtener como máximo una velocidad de menos de 4 fps. Teniendo en cuenta que la detección facial con RetinaFace tarda aproximadamente 170 ms y que el tiempo del resto de etapas es cercano a los 5 ms, se puede deducir que VGG-Face extrae las características de una cara en 100 ms. Por lo que, si omitiéramos el resto de las etapas, obtendríamos igualmente una velocidad de 10 fps, muy alejado de los 25 fps de tiempo real.

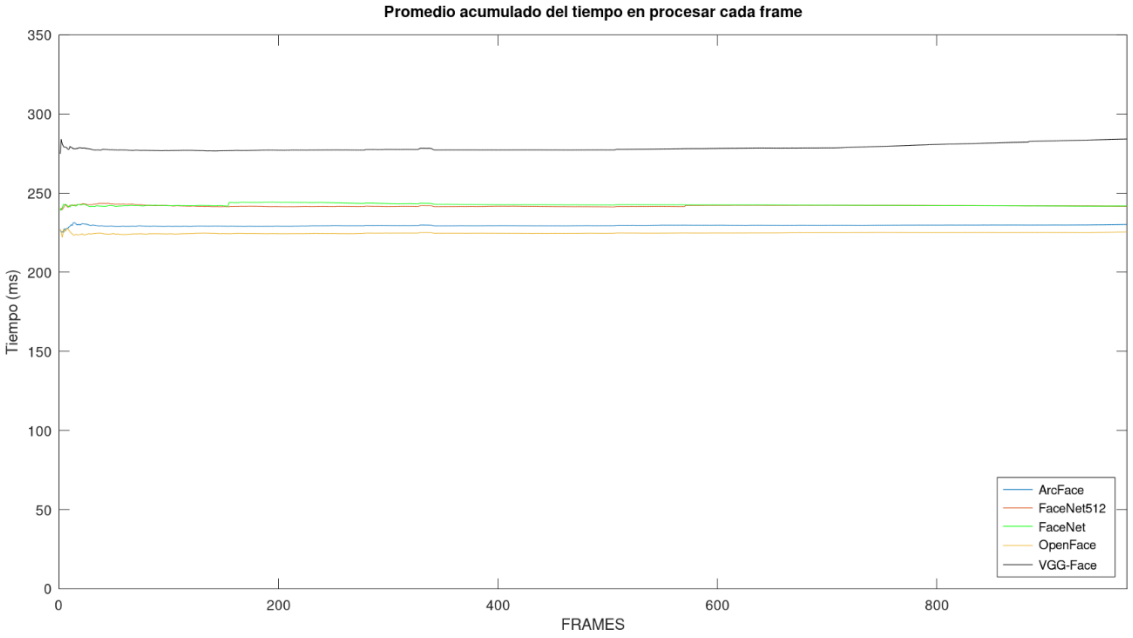


Fig. 39. *Tiempo de inferencia de los algoritmos comparados*

El siguiente algoritmo más lento es FaceNet. Como se puede apreciar, no hay prácticamente diferencia de tiempos entre los dos *backbones*, por lo que usar uno u otro no supone una mejora en el tiempo de inferencia. Una ejecución con este algoritmo tarda aproximadamente 245 ms. Suponiendo lo mismo que con VGG-Face, la extracción de características supone 70 ms. Por lo que, ignorando el resto de las etapas, se podría

obtener como máximo una velocidad de 14 fps, todavía lejos del procesamiento en tiempo real.

El siguiente algoritmo es ArcFace, que tiene un tiempo de inferencia total de 230 ms. Únicamente la extracción de características tarda unos 55 ms, lo que supone una velocidad máxima de 18 fps. Por lo que en relación precisión y velocidad, este algoritmo es uno de los mejores.

Finalmente, el algoritmo más rápido de todos los evaluados es OpenFace, con un tiempo de inferencia de 220 ms. Lo que supone que el algoritmo tarda en extraer las *features* de una cara 45 ms. Con esto se conseguiría una velocidad de 22 fps como máximo. Muy cercano al procesamiento en tiempo real. Pero esta velocidad no es real ya que se han omitido el resto de las etapas del *pipeline*.

Por todo ello se puede concluir que ArcFace es el algoritmo que tiene un mejor compromiso entre tiempo de inferencia y la precisión que consigue. OpenFace es el algoritmo más rápido, pero a la hora de obtener resultados no consigue una precisión muy alta, por lo que, si se usara este algoritmo, a pesar de ser muy rápido, se tendría muchos fallos. Por otro lado, VGG-Face es un algoritmo muy lento pero que consigue muy buenos resultados, por lo que si no nos importara la velocidad el algoritmo a usar seguramente fuera este.

Como se ha podido comprobar, con ninguno de estos algoritmos se consigue procesamiento en tiempo real aun eliminando el resto de las etapas del *pipeline*. En consecuencia, para conseguir procesamiento en tiempo real hará falta realizar modificaciones a las redes neuronales de estos algoritmos para conseguir mejorar los tiempos y reducir el tiempo de inferencia, aunque con ello se reduzca también la precisión, ya que esta se puede aumentar de otras maneras, como, por ejemplo, el sistema que hemos introducido en este proyecto o con reentrenamientos de los algoritmos con *datasets* más completos y adecuados para nuestros casos de usos.

CAPÍTULO VII

CONCLUSIONES Y LÍNEAS FUTURAS

Como se ha comentado en el primer capítulo, el principal objetivo de este proyecto era mejorar la precisión de los algoritmos de verificación facial usando umbrales dinámicos que varían en función de la distancia a la que se encuentre la persona de la cámara. Para ello, se creó un *dataset* conteniendo imágenes de personas de diferentes etnias y edades para que fuera estadísticamente correcto. Como se ha podido ver en las curvas de los índices de similaridad se ha conseguido un *dataset* adecuado al ser las distribuciones óptimas para nuestro proyecto.

En este proyecto se ha conseguido mejorar la precisión de los cinco algoritmos de verificación facial usados. Esta mejora es aún más importante ya que se ha conseguido en distancias de hasta 30 metros, en los que la resolución de la cara de la persona tiene de media 15 pixeles.

Por ejemplo, el algoritmo FaceNet512 se ha logrado mejorar su precisión usando la distancia de coseno más de un 30% en algunas distancias. Y en todos los algoritmos se ha conseguido aumentar su precisión a 25 y 30 metros de distancia, considerado en este proyecto como muy lejano.

También se ha dado una recomendación de cuáles son las mejores métricas de cálculo de distancias para cada uno de los algoritmos de verificación facial, así como los umbrales dinámicos recomendados para nuestro caso de uso.

Habiendo analizado todos los resultados obtenidos, se puede llegar a la conclusión de que los dos algoritmos con la mejor relación precisión-tiempo de inferencia son ArcFace y FaceNet512, ya que ambos consiguen una gran precisión con un tiempo de inferencia no muy alto en comparación con los otros algoritmos. Entre ellos dos, FaceNet512 consigue una precisión algo mayor pero también su tiempo es menor, mientras que ArcFace es un algoritmo más rápido.

El proyecto se podría continuar por varias vías que tienen opción de mejora. Por ejemplo, se podría crear un *dataset* más completo en el que se incluyan un mayor número de personas para tener una muestra mayor. También se podrían obtener más videos de cada persona, variando el ángulo de incidencia del dron con respecto a la persona. Además, se podrían conseguir imágenes en situaciones de baja visibilidad como por ejemplo de noche, con niebla o lloviendo. Otra opción interesante por explorar, debido a la reciente pandemia, es incluir en el *dataset* personas llevando mascarilla. En el futuro podría ser necesario verificar a personas llevando mascarillas por razones de seguridad sanitaria.

Otra opción por explorar es añadir en el *pipeline* una técnica de super resolución como las vistas en el capítulo de estado del arte. Con ello se conseguiría mejorar notablemente

la precisión de los algoritmos, pero a costa de aumentar notablemente el tiempo de ejecución.

En este proyecto se han calculado los umbrales dinámicos para maximizar la precisión de los algoritmos. Pero también se pueden calcular los umbrales maximizando otras métricas como por ejemplo *F1-Score*. También se podría definir una probabilidad máxima de obtener falsos positivos y calcular el umbral para nunca superar este valor, por lo que se podría minimizar su número hasta el valor que se quiera. La métrica a maximizar variará en función del caso de uso.

Finalmente, se podría también buscar otros algoritmos de detección facial que consigan un menor tiempo de inferencia, aunque eso suponga perder precisión. El algoritmo de detección facial elegido variará también en función del caso de uso, si se prefiere alcanzar detección en tiempo real bajando el número de detecciones o si, por otro lado, se quiere detectar todas las personas sin importar el tiempo de ejecución.

BIBLIOGRAFÍA

[1] M. Police, «Facial Recognition,» 2022. [En línea]. Available: https://www.met.police.uk/advice/advice-and-information/fr/facial-recognition.

[2] Wired, «London is buying heaps of facial recognition tech,» 27 09 2021. [En línea]. Available: https://www.wired.co.uk/article/met-police-facial-recognition-new. [Último acceso: 23 06 2022].

[3] F. Chollet y others, «Keras,» GitHub, 2015. [En línea]. Available: https://github.com/fchollet/keras.

[4] N. K. A. Wirdiani, P. Hidrayami, N. P. A. Widiari, K. D. Rismawan, P. B. Candradinatha y I. P. D. Jayantha, «Face Identification Based on K-Nearest Neighbor,» *Scientific Journal of Informatics,* vol. 6, nº 2, pp. 150-159, 2019.

[5] I. Masi, Y. Wu, T. Hassner y P. Natarajan, «Deep Face Recognition: a Survey,» de *31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, 2018.

[6] S. I. Serengil, «Face Alignment for Face Recognition in Python within OpenCV,» 23 Febrero 2020. [En línea]. Available: https://sefiks.com/2020/02/23/face-alignment-for-face-recognition-in-python-within-opencv/. [Último acceso: 26 Junio 2022].

[7] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang y W. Shi, «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,» *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 4681-4690, 2017.

[8] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, C. C. Loy, Y. Qiao y X. Tang, «ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,» *Proceedings of the European conference on computer vision (ECCV) workshops,* 2018.

[9] eriklindernoren, «PyTorch-GAN,» 6 Enero 2021. [En línea]. Available: https://github.com/eriklindernoren/PyTorch-GAN#enhanced-super-resolution-gan. [Último acceso: 27 Junio 2022].

[10] D. Chicco, «Siamese Neural Networks: An overview,» de *Artificial Neural Networks: An Overview*, 2020, pp. 73-94.

[11] G. B. Huang, M. Ramesh, T. Berg y E. Learned-Miller, «Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.,» University of Massachusetts, Amherst, October, 2007.

[12] L. Wolf, T. Hassner y I. Maoz, «Face Recognition in Unconstrained Videos with Matched Background Similarity.,» de *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[13] I. Kalra, M. Singh, S. Nagpal, R. Singh, M. Vatsa y P. B. Sujit, «DroneSURF: Benchmark Dataset for Drone-based Face Recognition,» de *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, 2019.

[14] Q. Cao, L. Shen, W. Xie, O. M. Parkhi y A. Zisserman, «VGGFace2: A dataset for recognising faces across pose and age,» de *13th IEEE International Conference on Automatic Face & Gesture Recognition*, 2018.

[15] H.-J. Hsu y K.-T. Chen, «DroneFace: An Open Dataset for Drone Research,» de *8th ACM on Multimedia Systems Conference*, 2017.

[16] Y. Shuo, P. Luo, C. C. Loy y T. Xiaoou, «WIDER FACE: A Face Detection Benchmark,» de *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] P. Viola y M. Jones, «Rapid Object Detection using a Boosted Cascade of Simple Features,» de *Computer Vision and Pattern Recognition Conference (CVPR)*, 2001.

[18] J. Deng, J. Guo, E. Ververas, I. Kotsia y S. Zafeiriou, «RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild,» de *Computer Vision and Pattern Recognition Conference (CVPR)*, 2020.

[19] K. Zhang, Z. Zhang, Z. Li y Q. Yu, «Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks,» *IEEE Signal Processing Letters*, vol. 23, nº 10, pp. 1499-1503, 2016.

[20] D. E. King, «Dlib-ml: A Machine Learning Toolkit,» *Journal of Machine Learning Research*, vol. 10, pp. 1755-1758, 2009.

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu y A. C. Berg, «SSD: Single Shot MultiBox Detector,» de *Computer Vision - ECCV*, 2016.

[22] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection,» de *CVPR*, 2016.

[23] S. I. Serengil y A. Ozpinar, «LightFace: A Hybrid Deep Face Recognition Framework,» de *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2020, pp. 23-27.

[24] S. I. Serengil y A. Ozpinar, «HyperExtended LightFace: A Facial Attribute Analysis Framework,» de *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, 2021, pp. 1-4.

[25] F. Schroff, D. Kalenichenko y J. Philbin, «FaceNet: A unified embedding for face recognition and clustering,» de *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[26] T. Baltrušaitis, A. Zadeh, Y. C. Lim y L.-P. Morency, «OpenFace 2.0: Facial Behavior Analysis Toolkit,» de *IEEE International Conference on Automatic Face and Gesture Recognition*, 2018.

[27] D. Yi, Z. Lei, S. Liao y S. Z. Li, «Learning Face Representation from Scratch,» 2014.

[28] H.-W. Ng y S. Winkler, «A DATA-DRIVEN APPROACH TO CLEANING LARGE FACE DATASETS,» de *IEEE International Conference on Image Processing (ICIP)*, Paris, France, 2014.

[29] J. Deng, J. Guo, X. Nianman y S. Zafeiriou, «ArcFace: Additive Angular Margin Loss for Deep Face Recognition,» de *Computer Vision and Pattern Recognition Conference (CVPR)*, 2019.

[30] T. Chen, M. Li, Y. Li, M. Lin, M. Wang, T. Xiao, B. Xu, C. Zhang y Z. Zhang, «MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems,» *arXiv*, 2015.

[31] Y. Guo, L. Zhang, Y. Hu, X. He y J. Gao, «MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition,» de *Computer Vision - ECCV 2016*, 2016.

[32] O. M. Parkhi, A. Vedaldi y A. Zisserman, «Deep Face Recognition,» de *British Machine Vision Conference*, 2015.

[33] H. V. Nguyen y L. Bai, «Cosine Similarity Metric Learning for Face Verification,» de *Computer Vision - ACCV 2010*, 2010.

[34] M. D. Malkauthekar, «Analysis of euclidean distance and Manhattan Distance measure in face recognition,» de *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, 2013.

[35] Y. Sun, D. Liang, X. Wang y X. Tang, «DeepID3: Face Recognition with Very Deep Neural Networks,» 2015.