Escuela Técnica Superior de Ingeniería de Telecomunicación

Universidad Politécnica de Cartagena

RECONOCIMIENTO FACIAL MEDIANTE TÉCNICAS DE CLASIFICACIÓN DE TIPO FINO

AUTOR: ÁNGEL SILVENTE TERUEL

TUTOR: JAVIER VALES ALONSO

SEPTIEMBRE DE 2022

Universidad Politécnica de Cartagena

Índice

1. Introducción y objetivos **3**
- 1.1. Resumen . . . 3
- 1.2. Machine learning y tipos . . . 3
  - 1.2.1. Aprendizaje supervisado . . . 4
  - 1.2.2. Aprendizaje no supervisado . . . 4
  - 1.2.3. Aprendizaje semisupervisado . . . 4
  - 1.2.4. Aprendizaje por refuerzo . . . 4
- 1.3. Objetivos . . . 5
- 1.4. El problema de la distinción fina de caras . . . 5
- 1.5. Redes neuronales . . . 6
  - 1.5.1. Introducción . . . 6
  - 1.5.2. Entrenamiento . . . 7
  - 1.5.3. Organización . . . 7
- 1.6. Redes siamesas y su aplicación a la distinción fina de caras . . . 8
  - 1.6.1. Arquitectura . . . 8
  - 1.6.2. Redes siamesas . . . 8
- 1.7. Entorno de desarrollo y herramientas . . . 9

2. Descripción de los datos **10**
- 2.1. Dataset de entrenamiento . . . 10
- 2.2. Datasets de prueba . . . 11

3. Desarrollo **12**
- 3.1. Obtención del dataset . . . 12
  - 3.1.1. Obtención de los datos . . . 12
  - 3.1.2. Preparación de los datos . . . 15
  - 3.1.3. Construcción del dataset . . . 16
- 3.2. Implementación de la red siamesa . . . 31
  - 3.2.1. Configuración del modelo generador de incrustaciones . . . 31
  - 3.2.2. Configuración del modelo de la red siamesa . . . 31
  - 3.2.3. Entrenamiento y carga de la red . . . 34
- 3.3. Pruebas de la red siamesa . . . 35

4. Presentación de los resultados **36**

4.1. Primera evaluación 52
4.1.1. Primera prueba 54
4.1.2. Segunda prueba 60
4.2. Segunda evaluación 62
4.3. Tercera evaluación 67
4.4. Cuarta evaluación 69
4.4.1. Primera prueba 69
4.4.2. Segunda prueba 72

5. Conclusiones y líneas futuras **82**
5.1. Conclusiones por evaluación 82
5.1.1. Primera evaluación 82
5.1.2. Segunda evaluación 82
5.1.3. Tercera evaluación 82
5.1.4. Cuarta evaluación 83
5.2. Conclusiones generales 83
5.3. Líneas futuras 84

Bibliografía **85**

Introducción y objetivos

1.1. Resumen

El proyecto consiste en la evaluación de técnicas de clasificación de tipo fino para su uso en aplicaciones de reconocimiento facial. Para ello se ha trabajado sobre una red con un modelo RestNet50 preentrenada con ImageNet, para posteriormente ampliarla. Esta ampliación consistirá en la fijación de los pesos asociados al preentreno, la creación de nuevas capas de entrenamiento sin pesos fijados y el uso de un dataset nuevo y de elaboración propia el cual estará formado únicamente por caras de diferentes personalidades. De esta manera tendremos una red previa con capacidad para identificar imágenes, la cual especializaremos en el reconocimiento facial a través de un nuevo entrenamiento y de los datos proporcionados.

1.2. Machine learning y tipos

Podemos entender el marchine learning o aprendizaje automático como un campo dentro de la inteligencia artificial que se centra en otorgarle la capacidad a un ordenador de aprender a realizar una tarea, sin necesidad de haberlo programado de forma explícita para ello. Una definición más técnica del machine learning sería la de que un programa aprende de una experiencia E con respecto a una tarea T y con alguna medición de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E (Tom Mitchell, 1997).

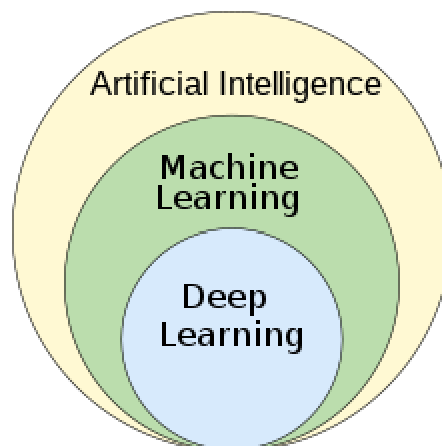


Figura 1.1: Machine learning como subcampo de inteligencia artificial [1] .

Debido a la importancia de los datos que se usan para el aprendizaje, podemos dividirlos en diferentes tipos según la naturaleza de los mismos. Las principales categorías son:

- Aprendizaje supervisado.
- Aprendizaje no supervisado.

- Aprendizaje semisupervisado.
- Aprendizaje por refuerzo.

1.2.1. Aprendizaje supervisado

Los algoritmos de este tipo consisten en un modelo matemático construido a partir de dos conjuntos, uno con los datos que usará como entrada y otro con los datos que espera obtener a la salida. A estos conjuntos se los conoce como datos de entrenamiento y funcionan a modo de ejemplos de entrenamiento. Cada ejemplo puede estar formado por uno o más valores de entrada así como del resultado deseado a la salida del sistema. Matemáticamente, este ejemplo de entrenamiento estará representado por un vector, y el conjunto de los datos de entrenamiento por una matriz. De esta forma y mediante optimización iterativa de una función objetivo, el aprendizaje supervisado consigue aprender una función capaz de predecir una salida asociada a nuevos datos de entrada. De esta manera consideraremos que una función es óptima cuando es capaz de producir salidas válidas para datos de entrada que no formaban parte del conjunto de datos de entrada inicial. Y si es capaz de mejorar la precisión con la que obtiene salidas a lo largo del tiempo, se dirá que ha aprendido a realizar dicha tarea.

1.2.2. Aprendizaje no supervisado

Los algoritmos de aprendizaje no supervisado toman un conjunto de datos que sólo contiene entradas y se dedican a encontrar un patrón en los datos, como puede ser mediante agrupación o clustering de puntos de datos. Estos algoritmos, por tanto, aprenden a partir de datos de prueba que no han sido etiquetados, clasificados o categorizados. Y en lugar de responder a la retroalimentación, los algoritmos de aprendizaje no supervisado identifican puntos comunes en los datos y reaccionan basándose en la presencia o ausencia de dichos puntos comunes en cada nuevo dato.

1.2.3. Aprendizaje semisupervisado

El aprendizaje semisupervisado se sitúa entre el aprendizaje no supervisado (sin datos de entrenamiento etiquetados) y el supervisado (con datos de entrenamiento completamente etiquetados). Algunos de los ejemplos de entrenamiento carecen de etiquetas de entrenamiento, pero muchos investigadores de machine learning han descubierto que los datos no etiquetados, cuando se utilizan junto con una pequeña cantidad de datos etiquetados, pueden producir una mejora considerable en la precisión del aprendizaje.

En este tipo de aprendizaje, las etiquetas de entrenamiento pueden contener ruido, estar limitadas o ser imprecisas; sin embargo, estas etiquetas suelen ser más baratas de obtener, lo que da lugar a conjuntos de entrenamiento efectivos más grandes.

1.2.4. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un área del aprendizaje automático que se ocupa de cómo los agentes de software deben realizar acciones en un entorno para maximizar alguna noción de recompensa acumulada. Debido a su generalidad, este campo se estudia en muchas otras disciplinas, como la teoría de los juegos, la del control, la investigación operativa, la teoría de la información,

la optimización basada en la simulación, los sistemas multiagente, la inteligencia de enjambre, la estadística y los algoritmos genéticos. En el aprendizaje automático, el entorno suele representarse como un proceso de decisión de Markov. Muchos algoritmos de aprendizaje por refuerzo utilizan técnicas de programación dinámica. Los algoritmos de aprendizaje por refuerzo no asumen el conocimiento de un modelo matemático exacto, y se utilizan cuando los modelos exactos son inviables. También se utilizan en vehículos autónomos o en el aprendizaje de un juego contra un oponente humano.

1.3. Objetivos

Son varias las técnicas que existen para el reconocimiento facial. Por ello, el objetivo de este proyecto será el de evaluar una de las muchas técnicas existentes, utilizando una red neuronal siamesa y aplicando técnicas de reconocimiento tipo fino. Para ello, se utilizará un dataset personalizado, lo que nos podrá a su vez dar una idea de los puntos débiles y los fuertes de nuestro sistema. Para ello, hemos dividido en diferentes etapas el desarrollo de este proyecto:

- Obtención de los datos: para realizar un algoritmo de reconocimiento facil, necesitaremos entrenarlo con unos datos, en este caso caras. Para ello, elaboraremos una aplicación sencilla que nos permita automatizar las tareas que hacen falta para la creación de los conjuntos de datos que utilizaremos, ya que debido a su gran volumen, hacerlo a mano sería una tarea imposible.
- Preparación de los datos: no solo vale con obtenerlos, ya que según la técnica utilizada, la disposición de los datos varía. Para ello también realizaremos pequeñas aplicaciones que sirvan para automatizar el proceso y los organizaremos teniendo en cuenta el tipo de algoritmo que vamos a implementar.
- Desarrollo del algoritmo: una vez tengamos los datos que usaremos listos, podremos desarrollar la aplicación principal del proyecto según los criterios establecidos.
- Pruebas del algoritmo: con todo listo, quedará poner a prueba el algoritmo realizado para poder analizar los resultados que produzca. Para ello se realizará una serie de pruebas a fin de testearlo en diferentes condiciones y que los resultados sean lo más objetivos posible.
- Análisis de los resultados: finalmente, podremos pasar a evaluar la calidad de los resultados en función de las premisas establecidas, así como establecer unas líneas futuras.

1.4. El problema de la distinción fina de caras

Vamos a pasar a centrarnos en el problema que nos ocupa, el reconocimiento facial. Donde podemos deducir fácilmente que se trata de un problema de clasificación. Donde el objetivo es que el algoritmo sea capaz de clasificar una imagen dados unos criterios; en nuestro caso, si se trata de una persona u otra.

Para este tipo de problemas existen diferentes enfoques, nosotros nos hemos decantado por utilizar técnicas de clasificación de tipo fino, donde el objetivo es ser capaz de clasificar imágenes en

subcategorías más pequeñas dentro de una categoría que las contendría a todas. Esto es, podríamos tener una categoría general que fuera "perros", y querer diferenciar un conjunto de imágenes de entrada en función de la raza del perro; es decir, subcategorías hijas del conjunto inicial. En nuestro caso, tendremos un conjunto inicial que será "personas", del que esperamos conseguir subcategorías que coincidan correctamente con las diferentes personas que compongan ese conjunto inicial donde el elemento a analizar de cada individuo será la cara. Para ello, el entrenamiento requiere de una disposición concreta de los datos de entrada, ya que necesitará de tres separaciones previas de los datos:

- Imágenes ancla: serán la imagen que compararemos con los otros dos tipos para evaluar la distancia entre las mismas, diferenciando en cada iteración del algoritmo las imágenes que pertenecen a la misma subcategoría.
- Imágenes de refuerzo positivo: serán aquellas que deben parecerse a la imagen de ancla, de manera que cuando calculemos su distancia, podamos deducir que efectivamente son imágenes similares y el algoritmo pueda almacenar ese resultado aprendido.
- Imágenes de refuerzo negativo: serán aquellas que deben ser distintas a la imagen de ancla, de manera que ucando calculemos su distancia, podamos deducir que efectivamente no pertenecen a la misma subcategoría.

Una vez claro esto, encontramos dos grandes obstáculos asociados a esta técnica. En primer lugar, las diferencias visuales entre las distintas clases son tan pequeñas que pueden verse influidas por factores como la pose, la ubicación del objeto, etc. De hecho, incluso una pequeña sección del objeto puede tener un gran impacto en la clasificación. En segundo lugar, es muy complejo identificar las diminutas partes discriminantes dentro de las imágenes. Para solventar este problema, se han explorado diferentes metodologías con tal de identificar y localizar esas partes dentro de las imágenes utilizando representaciones sin orden como los vectores de Fisher, VLAD, y modelos neuronales convolucionales (CNN por sus siglas en inglés). En nuestro caso, en vista del éxito que han probado tener en este campo, hemos optado por el uso de redes CNN siamesas. Pero antes de explicar más en detalle la técnica utilizada, hay que entender en qué consiste una red neuronal.

1.5. Redes neuronales

1.5.1. Introducción

Las redes neuronales artificiales (RNA), normalmente llamadas simplemente redes neuronales (NN por sus siglas en inglés), son sistemas informáticos inspirados en las redes neuronales biológicas que constituyen los cerebros de los animales. Una RNA se basa en un conjunto de unidades o nodos conectados, denominados neuronas artificiales, que modelan vagamente las neuronas de un cerebro biológico. Cada conexión, como las sinapsis de un cerebro biológico, puede transmitir una señal a otras neuronas. Una neurona artificial recibe señales, las procesa y puede enviar señales a las neuronas conectadas a ella. La señal en una conexión es un número real, y la salida de cada neurona se calcula mediante una función no lineal de la suma de sus entradas, donde las conexiones se denominan aristas. Las neuronas y los bordes suelen tener un peso que se ajusta a medida que avanza el aprendizaje y donde el peso aumenta o disminuye la fuerza de la señal en una conexión. Las neuronas pueden tener un umbral, de manera que sólo se envía una señal

si la señal agregada cruza ese umbral. Normalmente, las neuronas se agrupan en capas, donde diferentes capas pueden realizar diferentes transformaciones en sus entradas. De esta manera, las señales viajarán desde la primera capa (la de entrada) hasta la última (la de salida), posiblemente después de atravesar las capas varias veces.

1.5.2. Entrenamiento

Las redes neuronales aprenden (o se entrenan) procesando ejemplos, cada uno de los cuales contiene una entrada y un resultado conocidos, formando asociaciones de probabilidad ponderada entre ambos, que se almacenan dentro de la estructura de datos de la propia red. El entrenamiento de una red neuronal a partir de un ejemplo dado suele realizarse determinando la diferencia entre la salida procesada de la red (a menudo una predicción de la misma) y una salida objetivo. Esta diferencia es el error. A continuación, la red ajustará sus asociaciones ponderadas según una regla de aprendizaje y utilizando este valor de error. Los ajustes sucesivos harán que la red neuronal produzca una salida cada vez más parecida a la salida objetivo. Después de un número suficiente de estos ajustes, el entrenamiento puede terminar basándose en ciertos criterios.

En este proceso, podemos identificar diferentes conceptos que son clave. Uno de ellos es la tasa de aprendizaje, la cual define el tamaño de los pasos correctivos que el modelo tomará para ajustar los errores en cada observación. Donde una tasa de aprendizaje alta acortará el tiempo de entrenamiento, pero con una menor precisión final, mientras que una tasa de aprendizaje más baja llevará más tiempo, pero con el potencial de una mayor precisión. El concepto de momemtum permite ponderar el equilibrio entre el gradiente y el cambio anterior, de forma que el ajuste del peso depende en cierta medida del cambio anterior. Un impulso cercano a 0 enfatizará el gradiente, mientras que un valor cercano a 1 enfatizará el último cambio.
Otro elemento a tener en cuenta son los hiperparámetros, los cuáles son constantes cuyo valor se establece antes de que comience el proceso de aprendizaje. Algunos ejemplos de hiperparámetros son la tasa de aprendizaje, el número de capas ocultas y el tamaño del lote. Los valores de algunos hiperparámetros pueden depender de los de otros hiperparámetros, como por ejemplo, el tamaño de algunas capas puede depender del número total de capas de la red.

1.5.3. Organización

Las neuronas suelen estar organizadas en múltiples capas, especialmente en el aprendizaje profundo. Las neuronas de una capa sólo se conectan con las neuronas de las capas inmediatamente anterior y posterior. La capa que recibe los datos externos es la capa de entrada. La que produce el resultado final es la de salida. Entre ellas hay cero o más capas ocultas, aunque también se pueden utilizar redes de una sola capa o sin capas. Entre dos capas, son posibles múltiples patrones de conexión. Pueden ser "totalmente conectadas", con cada neurona de una capa conectada a cada neurona de la siguiente. Pueden ser "pooling", en las que un grupo de neuronas de una capa se conectan a una sola neurona de la capa siguiente, reduciendo así el número de neuronas de esa capa. Las neuronas con sólo este tipo de conexiones forman un grafo acíclico dirigido y se conocen como redes feedforward. Alternativamente, las redes que permiten conexiones entre neuronas de la misma capa o de capas anteriores se conocen como redes recurrentes.

1.6. Redes siamesas y su aplicación a la distinción fina de caras

Una red neuronal convolucional, como las que vamos a implementar, es una variación del modelo general que acabamos de resumir. En este caso, se trata de un modelo donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas de la corteza visual primaria de un cerebro biológico, tratándose de una variación de un perceptron multicapa. No obstante, debido a que su aplicación es realizada en matrices bidimensionales, resultan muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes entre otras aplicaciones.

1.6.1. Arquitectura

Una red neuronal convolucional consta de una capa de entrada, capas ocultas y una capa de salida. En cualquier red neuronal feed-forward, las capas intermedias se llaman ocultas porque sus entradas y salidas están enmascaradas por la función de activación y la convolución final. Ahora bien, en una red neuronal convolucional, las capas ocultas incluyen capas que realizan convoluciones. Normalmente, esto incluye una capa que realiza un producto punto del núcleo de convolución con la matriz de entrada de la capa. Este producto suele ser el producto interno de Frobenius, y su función de activación suele ser ReLU. A pedida que el núcleo de convolución genera un mapa de características, que a su vez contribuye a la entrada de la siguiente capa. A ésta le siguen otras capas, como las capas de agrupación, las capas totalmente conectadas y las capas de normalización.

Esta arquitectura es especialmente útil en el procesamiento de imágenes, ya que resulta mucho más óptima que los modelos tradicionales. Esto es debido a que gracias a las capas convolucionales, se reduce drásticamente el número de parámetros que se utilizan durante el entrenamiento de la red, ya que se tienen en cuenta las relaciones espaciales entre las distintas características durante la convolución y/o agrupación, permitiendo redes más profundas capaces de procesar datos de mayor tamaño dispuestos en cuadrículas (imágenes).

1.6.2. Redes siamesas

La red final que vamos a construir y entrenar consistirá en un conjunto de redes convolucionales como las descritas, formando lo que se conoce como una red neuronal siamesa. Estas tienen dos o más redes convolucionales idénticas, las cuáles trabajarán con los mismos pesos pero a los que se le pasarán diferentes vectores de entrada. Donde el objetivo suele ser comparar las salidas producidas por ambas redes, de ahí su gran importancia en el procesamiento de imágenes.

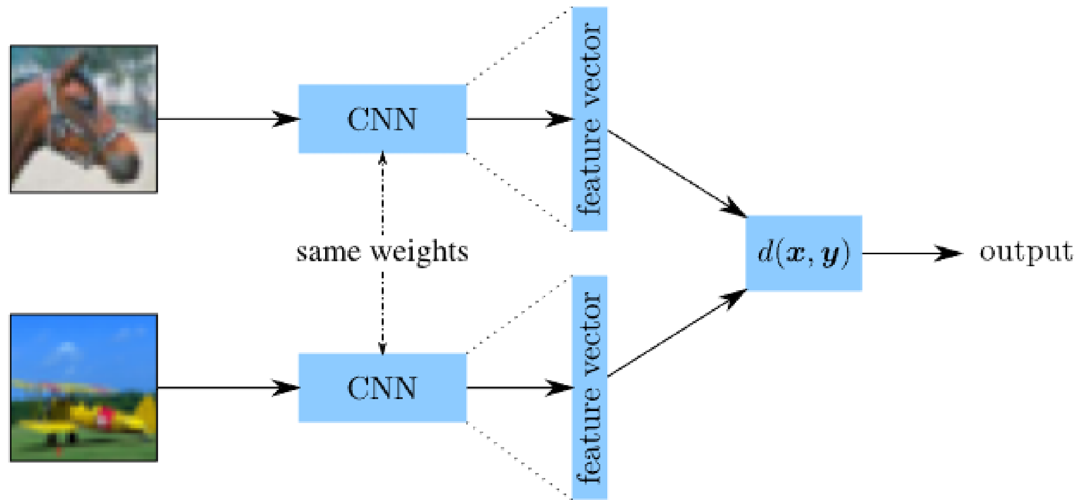


Figura 1.2: Ejemplo de arquitectura de una red neuronal siamesa [3].

1.7. Entorno de desarrollo y herramientas

Una vez claros el contexto teórico del proyecto así como los objetivos del mismo, vamos a hablar de las principales herramientas que vamos a utilizar para su desarrollo:

- Python: para todo el proceso, haremos uso de este lenguaje de programación, ya que es ampliamente utilizado en el sector del aprendizaje automático, lo que se traduce en una gran cantidad de librerías y de documentación que permiten una curva de dificultad y de aprendizaje más suavizada.
- Notebooks de Jupyter: son un entorno web para programar en python que resulta especialmente cómodo debido a su sencillez para visualizar y ejecutar código. Además, también permiten compartir los proyectos en forma de documentos haciendo que sea especialmente sencillo implementarlo en esta memoria. A fin de evitar complejidad, nos hemos valido del entorno de Anaconda para su instalación y uso.
- OpenCV: se trata de una librería de código abierto hecha en python, la cual cuenta con una gran cantidad de algoritmos de visión computerizada.
- TensorFlow: se trata de una plataforma de código abierto dedicada a computación numérica y modelos a gran escala de aprendizaje automático. Debido a su popularidad, cuenta con una gran cantidad de documentación, lo que la hace ideal para labores de investigación.
- Keras: se trata de una API de código abierto escrita en Python, para generar redes neuronales. Es capaz de ejecutarse sobre TensorFlow, y también cuenta con una gran cantidad de documentación.
- Overleaf: herramienta web para la elaboración de documentos en latex, utilizada para la construcción de esta memoria.

Descripción de los datos

Previo a comentar el desarrollo del algoritmo, vamos a explicar las decisiones de diseño de los conjuntos de datos (datasets) que utilizaremos a lo largo del proyecto. Distinguiendo aquí dos categorías, el dataset de entrenamiento de la red y los datasets de prueba.

2.1. Dataset de entrenamiento

El dataset de entrenamiento es clave para el correcto funcionamiento de la red. Para ello, y siempre dentro de las dimensiones correspondientes al proyecto, hemos tratado de mantener un equilibrio en cuanto a magnitud, diversidad y emociones a fin de evitar sesgos de entrenamiento dentro de lo posible. A pesar de ello, será inevitable la aparición de estos en mayor o menor medida, en parte por la dificultad inherente a encontrar datos válidos que cumplan todos los requisitos deseados.

El dataset no ha sido extraído de ninguna plataforma, este ha sido elaborado íntegramente para este proyecto, y es por ello que sus dimensiones y características pueden resultar limitadas. Para generarlo, se ha tomado la decisión de recurrir a un conjunto de actores y actrices lo más variados posibles, debido a una mayor facilidad para acceder no solo a imágenes de los mismos, también a diferentes caracterizaciones de la misma persona y a un mayor abanico de expresiones. La elaboración del dataset y el código encargado de ello están explicados en capítulos posteriores.

Figura 2.1: Imágenes representativas de cada persona correspondientes al dataset de entrenamiento.

En cuanto a dimensiones y estructura, tenemos tres grandes bloques a diferenciar:

- Imágenes ancla: se trata de una carpeta formada por 1000 imágenes, 100 por cada una de las 20 personas seleccionadas. Este subconjunto será el grupo de imágenes que se utilizará para comparar su distancia con los dos siguientes subconjuntos, estableciendo así cuáles son similares y cuáles no.
- Imágenes de refuerzo positivo: se trata de una carpeta formada por 1000 imágenes, 100 por cada una de las 20 personas seleccionadas. Este subconjunto de imágenes se utilizará para asociar las imágenes ancla a una imagen similar a ellas; es decir, siempre serán imágenes correspondientes a la misma persona que la ancla.

- Imágenes de refuerzo negativo: este conjunto es una mezcla de las dos carpetas anteriores, y el criterio para su elaboración ha sido simplemente el de garantizar que para una imagen ancla, la imagen de refuerzo negativo asociada, sea siempre la de una persona diferente.

Figura 2.2: Ejemplo de asociación entre una imagen ancla y sus refuerzos positivos y negativos.

2.2. Datasets de prueba

Para estos datasets tendremos diferentes condiciones en función de la prueba a realizar, esto podrá ir desde pequeñas variaciones en el dataset de entrenamiento original, modificaciones considerables, o directamente un dataset con las mismas premisas de diseño pero de personas totalmente diferentes. Al final, se tratará de un dataset que utilizaremos como datos de entrada para la red ya entrenada, a fin de obtener predicciones de la misma y ver si es capaz de identificar correctamente a las personas pasadas como ancla. Se dará una explicación más extensa cuando se desarrollen las pruebas planteadas para evaluar el algoritmo.

Desarrollo

3.1. Obtención del dataset

Hay que tener en cuenta que el dataset utilizado ha sido creado desde cero, con fines puramente académicos y sin otro objetivo que el de probar el funcionamiento de esta red en las condiciones descritas en este proyecto. Así pues, para su elaboración hemos seguido la siguiente línea de trabajo:

1. Descargar vídeos de youtube tipo tráilers de películas con el fin de tener un amplio repertorio de personajes a identificar.
2. Uso de la librería OpenCV para identificar y recortar caras de dichos vídeos, almacenándolas en una carpeta general. [13]
3. Separación de dichas caras en función de la persona.
4. Limpieza de los datos.
5. Predisposición de los datos para la elaboración del dataset.

3.1.1. Obtención de los datos

Código utilizado para descargar un vídeo de youtube

```
[9]: import os
video_url =#Video_Url
path = #Video_Path
video_index=1
ytdlp = "D:\ytdlp\yt-dlp.exe -o video%(video_index+1)s.mp4 -P " +path+ ' ' +␣
 ↪video_url
os.system('cmd /k ' + ytdlp)
```

Con esto simplemente podremos obtener un amplio repertorio de escenas y caracterizaciones de las diferentes personas elegidas. Esto nos permitirá disponer de un abanico de imágenes más variado aun dentro de la misma persona, pudiendo tener imágenes donde varíe su edad, la iluminación, las expresiones, etc.

Código para obtener las caras de un vídeo local

```
[11]: from __future__ import print_function
import numpy as np
import cv2 as cv
import argparse
from os import listdir
```

```
from os.path import isfile, join

use_path =#Use_Path
cap_path =#Output_Path

n_face = 0

parser = argparse.ArgumentParser(description='Code for Cascade Classifier␣
↪tutorial.')
parser.add_argument('--face_cascade', help='Path to face cascade.', default='D:
↪\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade_frontalface_alt.
↪xml')

parser.add_argument('--eyes_cascade', help='Path to eyes cascade.', default='D:
↪\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade_eye_tree_eyeglasses.
↪xml')
parser.add_argument('--camera', help='Camera divide number.', type=int,␣
↪default=0)
args = parser.parse_args(args=[])
face_cascade_name = args.face_cascade
eyes_cascade_name = args.eyes_cascade
face_cascade = cv.CascadeClassifier()
eyes_cascade = cv.CascadeClassifier()
#-- 1. Load the cascades
if not face_cascade.load('D:
↪\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade_frontalface_alt.
↪xml'):
    print('--(!)Error loading face cascade')
    exit(0)
if not eyes_cascade.load('D:
↪\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade_eye_tree_eyeglasses.
↪xml'):
    print('--(!)Error loading eyes cascade')
    exit(0)
camera_device = args.camera

cap =#Video

if not cap.isOpened:
    print('--(!)Error opening video capture')
    exit(0)
while True:
    ret, frame = cap.read()
    if frame is None:
```

```
        print('--(!) No captured frame -- Break!')
        break

    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    frame_gray = cv.equalizeHist(frame_gray)
    #-- Detect faces
    faces = face_cascade.detectMultiScale(frame_gray)

    #Recortamos las caras del frame para guardarlas en escala de grises y␣
 ↪realizar el autoencoder a partir de ellas
    for (x, y, w, h) in faces:
        crop_faces = frame[y:y+h, x:x+w]
        gray_faces = cv.cvtColor(crop_faces, cv.COLOR_BGR2GRAY)

        #Establecemos la altura y el ancho de la foto para compararlo con la␣
 ↪plantilla de reescalado y elegir la técnica
        if w >= 200:
        #Si la imagen a reescalar es superior al tamaño que queremos utilizar␣
 ↪como plantilla, usamos reescalado INTER_CUBIC
            res_gray_faces = cv.resize(gray_faces,(200,200),fx=1, fy=1,␣
 ↪interpolation = cv.INTER_AREA)
        elif w < 200:
        #Si la imagen a reescalar es inferior al tamaño que queremos utilizar␣
 ↪como plantilla, usamos reescalado INTER_AREA
            res_gray_faces = cv.resize(gray_faces,(200,200),fx=1, fy=1,␣
 ↪interpolation = cv.INTER_CUBIC)
        status = cv.imwrite('D:/Angel/Proyectos/OpenCV/Output/Faces/
 ↪f_det_'+str(n_face)+'.png', res_gray_faces)
        faces_xs.append(res_gray_faces)
        n_face += 1

    # Separamos el guardado de caras del dibujo del rectángulo para evitar los␣
 ↪fallos que se producían en el guardado
    # ya que a veces aparecían marcos que no debían.
    for (x, y, w, h) in faces:
        cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv.imshow('Capture - Face detection', frame)
    if cv.waitKey(10) == 27:
        break

cap.release()
cv.destroyAllWindows()
```

Este código se encargará de identificar con una precisión decente dado el caso, las caras de aquellos

vídeos que le pasemos como entrada (los descargados en el apartado anterior). Para ello, utilizará una técnica llamada clasificador en cascada implementada gracias a las librerías de OpenCV.

3.1.2. Preparación de los datos

El camino elegido para preparar los datos ha sido ir persona a persona, consiguiendo datos suficientes de diferentes vídeos hasta almacenar un repertorio lo suficientemente amplio. Una vez listo, hemos pasado a limpiar dicho conjunto, eligiendo un total de 100 imágenes por individuo y apartándolas en carpetas individuales.

Código auxiliar para renombrar los archivos de una carpeta

A fin de estandarizar el nombre de los datos y poder manejarlos con comodidad, se ha optado por renombrar el total mediante el siguiente código:

[26]:
```
import os
from os import listdir
import cv2 as cv
from os.path import isfile, join

path_uso   =#Use_Path

#- El valor de n establecerá el número de cara que le pondremos de nombre al␣
 ↪primer archivo.
n = 1

caras = sorted(os.listdir(path_uso),key=len)
print(len(caras))
for cara in caras:
    #print(cara)
    #print(n)
    os.system('cmd /k rename "' +path_uso+'"\\"'+cara+ '" f_det_"'+str(n)+'".
 ↪png')
    n= n+1
```

Separación de los datos

Debido a la disposición necesaria para la elaboración del dataset en forma de tripletes, se ha optado por separar los datos de la carpeta principal (formada por un total de 2000 imágenes, 100 por cada individuo), en dos carpetas: izquierda y derecha. Donde cada una tendrá 1000 imágenes con 50 imágenes por individuo. Cada carpeta servirá como base para formar los datos de ancla y de refuerzo positivo respectivamente (el negativo será una mezcla randomizada de ambos). El código para realizar esta separación ha sido:
A modo de aclaración: la idea es separar en dos carpetas con el mismo número de caras por cada actor, y para mantener el parecido lo más cercano posible (debido a la naturaleza del Dataset generado), separaré en pares e impares las imágenes obtenidas.

[27]:
```
import os
from os import listdir
import cv2 as cv

path  = #Path
left  = #Left_Path
right = #Right_Path

person = sorted(os.listdir(path),key=len)
n = 0
for cara in person:
    #print(cara)
    if n % 2 == 0:
        #print("El archivo correspondiente a la cara: " + cara + "  --> Es par.")
        os.system('cmd /k copy "' +path+'"\\"'+cara+ '" ' + left)
    else:
        #print("El archivo correspondiente a la cara: " + cara + "  --> Es impar.
 ↪")
        os.system('cmd /k copy "' +path+'"\\"'+cara+ '" ' + right)
    n = n+1
```

Una vez tenemos las imágenes preparadas, solo falta el código para construir el dataset.

3.1.3. Construcción del dataset

Lo primero que debemos hacer es cargar los datos, para ello simplemente le pasamos el directorio donde hemos preparado las fotos previamente.

[4]:
```
cache_dir = Path(Path.home()) / ".keras"
anchor_images_path = cache_dir / "left"
positive_images_path = cache_dir / "right"
```

Una vez cargados, vamos a utilizar un pipeline tf.data para generar los tripletes que necesitamos para entrenar la red siamesa. Para ello, lo configuraremos usando una lista comprimida con nombres de archivos ancla, positivos y negativos como fuente donde la pipeline cargará y preprocesará las imágenes correspondientes.

[4]:
```
def preprocess_image(filename):

    #Cargamos el archivo especificado en formato JPEG, lo preprocesamos y
    #lo reescalamos a un tamaño preestablecido.
    image_string = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(image_string, channels=3)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, target_shape)
    return image
```

```
def preprocess_triplets(anchor, positive, negative):

    return (
        preprocess_image(anchor),
        preprocess_image(positive),
        preprocess_image(negative),
    )
```

Pasamos a generar el dataset de anclas y el de positivos. Para ello, la estructura que hemos planteado es la siguiente. A fin de maximizar la distinción de características entre las 20 personas seleccionadas, y teniendo 50 caras por persona en cada dataset, vamos a comparar cada una de las caras con las otras 49 de esa persona para asociar el refuerzo positivo. Y cada cara con las 950 restantes de otras personas para asociar el negativo. De esta manera, estaremos ampliando el número de relaciones entre las imágenes disponibles al máximo.

[5]:

```
anchor_images=[]
left_data = sorted(os.listdir(anchor_images_path),key=len)
for file in left_data:
    l_file = str(anchor_images_path)+"/"+str(file)
    anchor_images.append(l_file)

positive_images=[]
right_data = sorted(os.listdir(positive_images_path),key=len)
for file in right_data:
    r_file = str(positive_images_path)+"/"+str(file)
    positive_images.append(r_file)

#-------------------------------------------
new_anchor_images = []
new_positive_images = []

#Creamos arrays temporales para cada persona
bloque_pos_1=[]
bloque_pos_2=[]
bloque_pos_3=[]
bloque_pos_4=[]
bloque_pos_5=[]
bloque_pos_6=[]
bloque_pos_7=[]
bloque_pos_8=[]
bloque_pos_9=[]
bloque_pos_10=[]
bloque_pos_11=[]
```

```
bloque_pos_12=[]
bloque_pos_13=[]
bloque_pos_14=[]
bloque_pos_15=[]
bloque_pos_16=[]
bloque_pos_17=[]
bloque_pos_18=[]
bloque_pos_19=[]
bloque_pos_20=[]

#print(right_data)
# Vamos a dividir las imáneges positivas en bloques de 50 para tener a cada␣
↪actor/actriz por separado
n = 0
for file in right_data:
    if (0 <= n < 50):
        #print("Las fotos son de Samuel L.Jackson:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_1.append(r_file)
        #print(r_file)
        n += 1
    elif(50 <= n < 100):
        #print("Las fotos son de Cate Blanchett:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_2.append(r_file)
        #print(r_file)
        n += 1
    elif(100 <= n < 150):
        #print("Las fotos son de Benedict Cumberbatch:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_3.append(r_file)
        #print(r_file)
        n += 1
    elif(150 <= n < 200):
        #print("Las fotos son de Hale Berry:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_4.append(r_file)
        #print(r_file)
        n += 1
    elif(200 <= n < 250):
        #print("Las fotos son de Gary Oldman:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_5.append(r_file)
        #print(r_file)
        n += 1
```

```
    elif(250 <= n < 300):
        #print("Las fotos son de Viola Davis:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_6.append(r_file)
        #print(r_file)
        n += 1
    elif(300 <= n < 350):
        #print("Las fotos son de Denzel Washington:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_7.append(r_file)
        #print(r_file)
        n += 1
    elif(350 <= n < 400):
        #print("Las fotos son de Octavia Spencer:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_8.append(r_file)
        #print(r_file)
        n += 1
    elif(400 <= n < 450):
        #print("Las fotos son de Tom Hanks:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_9.append(r_file)
        #print(r_file)
        n += 1
    elif(450 <= n < 500):
        #print("Las fotos son de Sandra Oh:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_10.append(r_file)
        #print(r_file)
        n += 1
    elif(500 <= n < 550):
        #print("Las fotos son de Jet Li:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_11.append(r_file)
        #print(r_file)
        n += 1
    elif(550 <= n < 600):
        #print("Las fotos son de Lucy Liu:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_12.append(r_file)
        #print(r_file)
        n += 1
    elif(600 <= n < 650):
        #print("Las fotos son de Jackie Chan:")
        r_file = str(positive_images_path)+"/"+str(file)
```

```
        bloque_pos_13.append(r_file)
        #print(r_file)
        n += 1
    elif(650 <= n < 700):
        #print("Las fotos son de Meryl Streep:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_14.append(r_file)
        #print(r_file)
        n += 1
    elif(700 <= n < 750):
        #print("Las fotos son de Song Knag Ho:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_15.append(r_file)
        #print(r_file)
        n += 1
    elif(750 <= n < 800):
        #print("Las fotos son de Lena Headey:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_16.append(r_file)
        #print(r_file)
        n += 1
    elif(800 <= n < 850):
        #print("Las fotos son de Morgan Freeman:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_17.append(r_file)
        #print(r_file)
        n += 1
    elif(850 <= n < 900):
        #print("Las fotos son de Jessica Chastein:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_18.append(r_file)
        #print(r_file)
        n += 1
    elif(900 <= n < 950):
        #print("Las fotos son de Ian McKellen:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_19.append(r_file)
        #print(r_file)
        n += 1
    else:
        #print("Las fotos son de Maggie Smith:")
        r_file = str(positive_images_path)+"/"+str(file)
        bloque_pos_20.append(r_file)
        #print(r_file)
        n += 1
```

[6]:
```
new_anchor_images_test = []

n = 0
for file in left_data:
    if (0 <= n < 50):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(50 <= n < 100):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(100 <= n < 150):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(150 <= n < 200):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(200 <= n < 250):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(250 <= n < 300):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
```

```
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(300 <= n < 350):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(350 <= n < 400):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(400 <= n < 450):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(450 <= n < 500):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(500 <= n < 550):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(550 <= n < 600):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
```

```
    elif(600 <= n < 650):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(650 <= n < 700):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(700 <= n < 750):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(750 <= n < 800):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(800 <= n < 850):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(850 <= n < 900):
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    elif(900 <= n < 950):
        j = 0
        while (j < 50):
```

```
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
    else:
        j = 0
        while (j < 50):
            l_file = str(anchor_images_path)+"/"+str(file)
            new_anchor_images_test.append(l_file)
            j += 1
        n += 1
```

[7]:
```
new_positive_images_test = []

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_1)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_2)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_3)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_4)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_5)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_6)
    m +=1

m = 0
```

```
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_7)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_8)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_9)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_10)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_11)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_12)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_13)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_14)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_15)
    m +=1

m = 0
```

```
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_16)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_17)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_18)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_19)
    m +=1

m = 0
while m < 50:  #Cada persona la recorremos 50 veces
    new_positive_images_test.extend(bloque_pos_20)
    m +=1
```

Una vez listo el dataset de anclas y de refuerzos positivos, falta el de refuerzo negativo. Este, como dijimos anteriormente, será una mezcla randomizada de los dos anteriores, donde el requisito principal será que la imagen negativa nunca coincida con la persona que se utiliza de ancla/positiva. Para la elaboración de este dataset se ha elaborado el siguiente código, en el que primero sumamos los dos primeros datasets, luego los randomizamos y pasamos a comparar uno a uno cada triplete resultante, modificando la imagen negativa resultante en caso de que coincida con la persona ancla/positiva (para ello nos hemos valido del nombre del archivo, ya que al agruparlos por números resultan fácilmente identificables).

[8]:
```
anchor_images_temp=[]
positive_images_temp=[]

anchor_images_temp = new_anchor_images_test.copy()
positive_images_temp = new_positive_images_test.copy()

rng = np.random.RandomState(seed=42)
rng.shuffle(anchor_images_temp)
rng.shuffle(positive_images_temp)
```

```
negative_images = anchor_images_temp + positive_images_temp
np.random.RandomState(seed=32).shuffle(negative_images)
#print(new_anchor_images_test)
```

[9]:
```
#print("La longitud de los dataset positivos y de ancla son:"+␣
 ↪str(len(positive_images)) +", "+str(len(anchor_images)))
#print("Y la longitud del dataset negativo es: "+ str(len(negative_images)) +␣
 ↪"\n")
# Ahora pasamos a recorrer todo el conjunto de parejas ancla-positiva para␣
 ↪comprobar que no haya solapamientos con
# las fotos negativas
new_negative_images=[]
is_left=0
is_right=0
j = 0
for i in new_anchor_images_test:
    #Cogemos la imagen ancla:
    num_ancla=i[33:]
    num_ancla=num_ancla.rpartition('.')[0]
    #print("El número de la foto de ancla es:  "+str(num_ancla))

    #Cogemos la imagen positiva:
    num_positiva=new_positive_images_test[j]
    num_positiva=num_positiva[34:]
    num_positiva=num_positiva.rpartition('.')[0]
    #print("El número de la foto positiva es:  "+str(num_positiva))

    #Cogemos la negativa asociada a esta pareja para comprobar si es válida,␣
 ↪como estas están randomizadas, tendremos
    #tanto de la carpeta de left como de right por lo que debemos identificarlas:
    neg_img=negative_images[j]
    str_neg=neg_img[22:]
    str_neg=str_neg.rpartition('/')[0]

    if str_neg=="left":
        #print("Estamos en la carpeta: left.") #Valores impares
        str_f_neg=neg_img[33:]
        neg_im=str_f_neg.rpartition('.')[0]
        #print("El valor de la foto negativa es:   " + neg_im + "   (left,␣
 ↪impares)")
        neg_im=int(neg_im)
        is_left = 1

    elif str_neg=="right":
```

```
        #print("Estamos en la carpeta: right.") #Valores pares
        str_f_neg=neg_img[34:]
        neg_im=str_f_neg.rpartition('.')[0]
        #print("El valor de la foto negativa es:   " + neg_im + "   (right,␣
↪pares)")
        neg_im=int(neg_im)
        is_right = 1
    #print("Imagen negativa: " +str(neg_im))
    #print("\nEstablecemos el intervalo en el que consideramos que no debe haber␣
↪coincidencias
    #(100 fotos en cada dirección):")
    max_up  = neg_im+100
    max_down=neg_im-100
    if(max_down< 0):
        max_down=0
    if(max_up >2000):
        max_up = 2000
    n = range(max_down, max_up)
    #print("Rango para esta foto:  "+str(n))

    #Vemos si el valor de la imagen negativa está lo suficientemente lejos de la␣
↪pareja:
    if(max_down<= int(num_ancla) <= max_up or max_down<= int(num_positiva) <=␣
↪max_up):
        #print("¡¡¡ CONFLICTO !!!")

        #Lo que queremos es alejarnos lo suficiente de la pareja para que deje␣
↪de haber conflicto, para ello
        #podemos simplemente sumarle 110 posiciones a la foto negativa que está␣
↪cogiendo, pero llevando cuidado
        #porque en los extremos, tendremos situaciones algo diferentes.

        if(neg_im >= 1700):
            new_neg_im = neg_im - 300 #Le restamos algo más de 100 fotos ya que␣
↪por persona hay 100 exactas
        else:
            new_neg_im = neg_im + 300 #Le sumamos algo más de 100 fotos ya que␣
↪por persona hay 100 exactas

        #print("\nEl nuevo valor de la negativa es:  " + str(new_neg_im))

        #Comprobamos que efectivamente el conflicto ha dejado de existir.
        new_max_up = new_neg_im+100
        new_max_down = new_neg_im-100
```

```
        if(new_max_down<0):max_down=0
        if(max_up>2000):max_up=2000
        new_n=range(new_max_down, new_max_up)
        #print("El nuevo rango de seguridad sería:  "+str(new_n))
        if(new_max_down<= int(num_ancla) <= new_max_up or new_max_down<=␣
↪int(num_positiva) <= new_max_up):
            #print("¡¡¡ SIGUE HABIENDO CONFLICTO !!!")
            pass
        else:
            #print("Ahora todo OK!")

            #Como aquí hemos tenido que corregir la imagen, el método de␣
↪guardado será algo distinto:

            #Primero determinamos si era una foto de left o right:
            #print("El valor de is_left es: " +str(is_left))
            #print("El valor de is_right es: " +str(is_right))
            if(is_left==1):
                new_im_path=i[0:33]
                #print("El path nuevo será "+ new_im_path)
                #print("La foto irá en left.")
            elif(is_right==1):
                new_im_path=new_positive_images_test[j]
                new_im_path=new_im_path[0:34]
                #print("El path nuevo será "+ new_im_path)
                #print("La foto irá en right.")

            nueva_img_final=new_im_path+str(new_neg_im)+".png"
            #print(nueva_img_final)
            new_negative_images.append(nueva_img_final)
            #print("Se ha guardado: " +new_negative_images[j])

    else:
        #print("Todo OK!")
        #Aquí el método de guardado consistirá simplemente en añadir las␣
↪imágenes negativas que no han necesitado modificación
        #print(negative_images[j])
        new_negative_images.append(negative_images[j])
        #print("Se ha guardado: " +new_negative_images[j])

    is_left=0
    is_right=0
    j = j+1
```

Finalmente, juntamos los tres conjuntos de imágenes en lo que resulta el dataset de entrenamiento.

[10]:
```
image_count = len(new_anchor_images_test)

anchor_dataset = tf.data.Dataset.from_tensor_slices(new_anchor_images_test)
positive_dataset = tf.data.Dataset.from_tensor_slices(new_positive_images_test)

negative_dataset = tf.data.Dataset.from_tensor_slices(new_negative_images)

dataset = tf.data.Dataset.zip((anchor_dataset, positive_dataset,␣
 ↪negative_dataset))
dataset = dataset.shuffle(buffer_size=50000)
dataset = dataset.map(preprocess_triplets)

# Let's now split our dataset in train and validation.
train_dataset = dataset.take(round(image_count * 1))
val_dataset = dataset.skip(round(image_count * 1))

train_dataset = train_dataset.batch(20, drop_remainder=False)
train_dataset = train_dataset.prefetch(5)

val_dataset = val_dataset.batch(20, drop_remainder=False)
val_dataset = val_dataset.prefetch(5)
```

Una vez preparado el dataset de entrenamiento, lo que resta es la elaboración de la red siamesa que utilizará dichos datos.

3.2. Implementación de la red siamesa

Para desarrollar nuestro algoritmo, vamos a implementar una red siamesa con tres subredes idénticas, como hemos indicado antes, una para cada tipo de imagen dada (ancla, positiva y negativa). Y donde nuestro modelo aprenderá a estimar la longitud entre dichas imágenes, pudiendo así establecer el parecido entre las mismas.

Para que la red aprenda, utilizaremos una función de pérdidas en tripletes, donde la expresión utilizada será algo así:

$$L(A, P, N) = max(f(A) - f(P) - f(A) - f(N) + margin, 0) \tag{3.1}$$

3.2.1. Configuración del modelo generador de incrustaciones

La red siamesa generará incrustaciones para cada una de las imágenes del triplete. Para ello, vamos a utilizar un modelo ResNet50 preentrenado en ImageNet y le conectaremos nuevas capas. De esta manera lo que haremos será utilizar una red que ya sabe diferenciar imágenes y a la que ampliaremos para que dicho aprendizaje se especialice en el reconocimiento facial. Para ello, congelaremos los pesos correspondientes al modelo preentrenado, esto es, hasta la capa conv5_block1_out. Las capas siguientes serán las que entrenemos nosotros.

[12]:
```
base_cnn = resnet.ResNet50(
    weights="imagenet", input_shape=target_shape + (3,), include_top=False
)

flatten = layers.Flatten()(base_cnn.output)
dense1 = layers.Dense(512, activation="relu")(flatten)
dense1 = layers.BatchNormalization()(dense1)
dense2 = layers.Dense(256, activation="relu")(dense1)
dense2 = layers.BatchNormalization()(dense2)
output = layers.Dense(256)(dense2)

embedding = Model(base_cnn.input, output, name="Embedding")

trainable = False
for layer in base_cnn.layers:
    if layer.name == "conv5_block1_out":
        trainable = True
    layer.trainable = trainable
```

3.2.2. Configuración del modelo de la red siamesa

La red siamesa recibirá cada una de las imágenes del triplete como entrada, generará las incrustaciones, y emitirá la distancia entre el ancla y la incrustación positiva, así como entre el ancla y la incrustación negativa.

Para calcular la distancia, podemos utilizar una capa personalizada llamada DistanceLayer que

devuelve ambos valores en forma de tupla.

[13]:
```
class DistanceLayer(layers.Layer):
       def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def call(self, anchor, positive, negative):
        ap_distance = tf.reduce_sum(tf.square(anchor - positive), -1)
        an_distance = tf.reduce_sum(tf.square(anchor - negative), -1)
        return (ap_distance, an_distance)

anchor_input   = layers.Input(name="anchor",   shape=target_shape + (3,))
positive_input = layers.Input(name="positive", shape=target_shape + (3,))
negative_input = layers.Input(name="negative", shape=target_shape + (3,))

distances = DistanceLayer()(
    embedding(resnet.preprocess_input(anchor_input)),
    embedding(resnet.preprocess_input(positive_input)),
    embedding(resnet.preprocess_input(negative_input)),
)

siamese_network = Model(
    inputs=[anchor_input, positive_input, negative_input], outputs=distances
)
```

Ahora necesitamos implementar un modelo con un bucle de entrenamiento personalizado para poder calcular la pérdida del triplete utilizando las tres incrustaciones (ancla, positiva y negativa) producidas por la red siamesa. Para ello, vamos a crear una instancia de la métrica de la media para seguir la pérdida del proceso de entrenamiento.

[14]:
```
class SiameseModel(Model):
    def __init__(self, siamese_network, margin=0.5, name=None):
        super(SiameseModel, self).__init__(name=name)
        self.siamese_network = siamese_network
        self.margin = margin
        self.loss_tracker = metrics.Mean(name="loss")

    def call(self, inputs):
        return self.siamese_network(inputs)

    def train_step(self, data):
        # GradientTape es un gestor de contexto que almacena cualquier operación
        # que se haga dentro. Nosotros lo usamos para calcular la perdida de tal
        # forma que podamos obtener los gradientes y aplicarlos usando el
        #  optimizador especificado en `compile()`.
        with tf.GradientTape() as tape:
            loss = self._compute_loss(data)

        # Almacenamos los gradientes de la función de pérdidas con respecto
        # a los pesos/parámetros.
        gradients = tape.gradient(loss, self.siamese_network.trainable_weights)

        # Aplicamos los gradientes al modelo usando el optimizador seleccionado.
        self.optimizer.apply_gradients(
            zip(gradients, self.siamese_network.trainable_weights)
        )

        #Actualizamos y devolvemos el valor de las pérdidas.
        self.loss_tracker.update_state(loss)
        return {"loss": self.loss_tracker.result()}

    def test_step(self, data):
        loss = self._compute_loss(data)

        #Actualizamos y devolvemos el valor de las pérdidas.
        self.loss_tracker.update_state(loss)
        return {"loss": self.loss_tracker.result()}

    def _compute_loss(self, data):
        # La salida de la red es una tupla que contiene las distancias
        # entre el ancla y el refuerzo positivo, así como del ancla y
        # el refuerzo negativo.
        ap_distance, an_distance = self.siamese_network(data)

        # Calculamos el valor de pérdida final restando ambas distancias
```

```
        # y asegurándonos de no obtener un valor negativo.
        loss = ap_distance - an_distance
        loss = tf.maximum(loss + self.margin, 0.0)
        return loss

    @property
    def metrics(self):
        # Necesitamos listar las métricas para que la función `reset_states()`
        # pueda ser llamada automáticamente.
        return [self.loss_tracker]
```

3.2.3. Entrenamiento y carga de la red

El siguiente paso sería el entrenamiento de nuestra red, para lo cual hemos utilizado el siguiente código:

```
[ ]: siamese_model = SiameseModel(siamese_network)
     siamese_model.compile(optimizer=optimizers.Adam(0.0001))
     history = siamese_model.fit(train_dataset, epochs=4, validation_data=val_dataset)
     pathSavedModel = #Model_Path
     siamese_model.save_weights(pathSavedModel, save_format='tf')
```

Donde Model_Path corresponde al directorio en el que queramos guardar nuestro modelo una vez entrenado.
Una vez entrenado, podremos hacer uso de la siguiente celda de código para cargar el modelo directamente:

```
[15]: # RECREAMOS EL MODELO:
      new_siamese_model = SiameseModel(siamese_network)
      new_siamese_model.compile(optimizer=optimizers.Adam(0.0001))
      new_siamese_model.fit(train_dataset.take(1), validation_data=val_dataset.take(1))

      # AHORA CARGAMOS LOS PESOS DEL MODEL INICIAL EN EL NUEVO MODELO:
      pathSavedModel = #Model_Path

      new_siamese_model.load_weights(pathSavedModel)
      print("Modelo cargado correctamente.")
```

```
1/1 [==============================] - 12s 12s/step - loss: 0.5800
Modelo cargado correctamente.
```

3.3. Pruebas de la red siamesa

Aquí tendremos ya nuestra red completamente entrenada. Es hora de ponerla a prueba. Para ello, vamos a definir la serie de pruebas a realizar y el objetivo de las mismas para comprobar el funcionamiento de la red y así poder evaluarla:

- Primera evaluación: pasarle a la red fotos pertenecientes al dataset de entrenamiento. De esta manera comprobaremos si efectivamente ha aprendido a diferenciar las diferentes personas que conforman nuestro dataset.
- Segunda evaluación: pasar a la red fotos que no estén en el dataset, pero que sean de las personas que conforman el dataset de entrenamiento. De esta manera podremos comprobar si la red ha aprendido únicamente a diferenciar las fotos de entrenamiento o si realmente ha sido capaz de extraer las características de cada individuo para diferenciarlos.
- Tercera evaluación: pasar a la red fotos de personas que no estén en el dataset, esto debería de arrojar resultados de aquellas personas que físicamente se parezcan más a la persona de entrada. En esta prueba será interesante ver si es capaz de arrojar resultados con rasgos faciales similares dentro del dataset que teníamos.
- Cuarta evaluación: pasar a la red fotos de personas que no estén en el dataset de entrenamiento pero sí añadidas a posteriori. Esto es, ver si una vez la red ha sido entrenada con un dataset inicial, se puede ampliar dicho dataset y seguir obteniendo resultados válidos sin necesidad de volver a entrenarla para los nuevos datos. Esto sería especialmente interesante por la escalabilidad de la misma y la gran carga inherente al proceso de entrenamiento.

Presentación de los resultados

Antes de pasar a presentar los resultados, vamos a exponer el código utilizado para la representación visual de los mismos.

[28]:
```
def visualize(anchor, positive, negative):
    #Visualize a few triplets from the supplied batches.

    def show(ax, image):
        ax.imshow(image)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

    fig = plt.figure(figsize=(10, 3))
    axs = fig.subplots(1, 3)
    for i in range(3):
        show(axs[i], anchor[i])
        show(axs[i+1], positive[i])
        show(axs[i+2], negative[i])

def visualize_3(anchor, positive, negative, dist_pos, min_i):

        for i in range(1000):
            #if(i == min_i):
            if(dist_pos[i] < 1):
                fig = plt.figure()
                ax = fig.add_subplot(1,2,1)
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)
                ax.set_title("Ancla nº: " + str(i))
                imgplot = plt.imshow(anchor[i])
                ax = fig.add_subplot(1,2,2)
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)
                ax.set_title("Foto similar ("+str(dist_pos[i])+")")
                imgplot = plt.imshow(positive[i])

def visualize_4(anchor, positive, negative, dist_pos):

        k = 5
        min_index_5 = np.argpartition(dist_pos, k)
        min_index = dist_pos[min_index_5[:k]]
```

```
        min_sorted = sorted(min_index)

        for j in min_sorted:
            for i in range(1000):
                if(dist_pos[i] == j):
                #if(dist_pos[i] < 0.5):
                    fig = plt.figure()
                    ax = fig.add_subplot(1,2,1)
                    ax.get_xaxis().set_visible(False)
                    ax.get_yaxis().set_visible(False)
                    ax.set_title("Ancla nº: " + str(i))
                    imgplot = plt.imshow(anchor[i])
                    ax = fig.add_subplot(1,2,2)
                    ax.get_xaxis().set_visible(False)
                    ax.get_yaxis().set_visible(False)
                    ax.set_title("Foto similar ("+str(dist_pos[i])+")")
                    imgplot = plt.imshow(positive[i])

def visualize_5(anchor, positive, negative, dist_pos):

        k = 5
        min_index_5 = np.argpartition(dist_pos, k)
        min_index = dist_pos[min_index_5[:k]]
        min_sorted = sorted(min_index)

        for j in min_sorted:
            for i in range(len(dist_pos)):
                if(dist_pos[i] == j):
                #if(dist_pos[i] < 0.5):
                    fig = plt.figure()
                    ax = fig.add_subplot(1,2,1)
                    ax.get_xaxis().set_visible(False)
                    ax.get_yaxis().set_visible(False)
                    ax.set_title("Ancla nº: " + str(i))
                    imgplot = plt.imshow(anchor[i])
                    ax = fig.add_subplot(1,2,2)
                    ax.get_xaxis().set_visible(False)
                    ax.get_yaxis().set_visible(False)
                    ax.set_title("Foto similar ("+str(dist_pos[i])+")")
                    imgplot = plt.imshow(positive[i])

def visualize_6(anchor, positive, negative, dist_pos, min_i):

        for i in range(1000):
            if(i == min_i):
```

```
            #if(dist_pos[i] < 0.3):
                fig = plt.figure()
                ax = fig.add_subplot(1,2,1)
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)
                #ax.set_title("Ancla nº: " + str(i))
                ax.set_title("Input")
                imgplot = plt.imshow(anchor[i])
                ax = fig.add_subplot(1,2,2)
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)
                ax.set_title("Foto más similar ("+str(dist_pos[i])+")")
                imgplot = plt.imshow(positive[i])

def visualize_7(anchor, positive, negative):

    for i in range(20):
        fig = plt.figure()
        ax = fig.add_subplot(1,3,1)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        imgplot = plt.imshow(anchor[i])
        ax = fig.add_subplot(1,3,2)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        imgplot = plt.imshow(positive[i])
        ax = fig.add_subplot(1,3,3)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        imgplot = plt.imshow(negative[i])

def visualize_8(anchor, positive, negative):

    for i in range(1050):
        fig = plt.figure()
        ax = fig.add_subplot(1,3,1)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        imgplot = plt.imshow(anchor[i])
        ax = fig.add_subplot(1,3,2)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        imgplot = plt.imshow(positive[i])
        ax = fig.add_subplot(1,3,3)
        ax.get_xaxis().set_visible(False)
```

```
        ax.get_yaxis().set_visible(False)
        imgplot = plt.imshow(negative[i])

def visualize_10(anchor, positive, negative, dist_pos):

    k = 5
    min_index_5 = np.argpartition(dist_pos, k)
    min_index = dist_pos[min_index_5[:k]]
    min_sorted = sorted(min_index)

    for j in min_sorted:
        for i in range(1050):
            if(dist_pos[i] == j):
            #if(dist_pos[i] < 0.5):
                fig = plt.figure()
                ax = fig.add_subplot(1,2,1)
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)
                ax.set_title("Ancla nº: " + str(i))
                imgplot = plt.imshow(anchor[i])
                ax = fig.add_subplot(1,2,2)
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)
                ax.set_title("Foto similar ("+str(dist_pos[i])+")")
                imgplot = plt.imshow(positive[i])

def visualize_test1_images():

    def show(ax, image):
        ax.imshow(image)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

    fig = plt.figure(figsize=(20, 4))
    axs = fig.subplots(2, 10)
    for j in range(10):
        processed_image=preprocess_image(first_half[j])
        show(axs[0, j], processed_image)
    for j in range(10):
        processed_image=preprocess_image(second_half[j])
        show(axs[1, j], processed_image)
```

También nos harán falta algunas variaciones del dataset de entrada para las pruebas próximamente detalladas. El código elaborado para dichas modificaciones es el siguiente:

[16]:
```
anchor_images=[]
left_data = sorted(os.listdir(anchor_images_path),key=len)
for file in left_data:
    l_file = str(anchor_images_path)+"/"+str(file)
    anchor_images.append(l_file)

positive_images=[]
right_data = sorted(os.listdir(positive_images_path),key=len)
for file in right_data:
    r_file = str(positive_images_path)+"/"+str(file)
    positive_images.append(r_file)

image_count = len(anchor_images)

anchor_images_temp=[]
positive_images_temp=[]
anchor_images_temp=anchor_images.copy()
positive_images_temp=positive_images.copy()

rng = np.random.RandomState(seed=42)
rng.shuffle(anchor_images_temp)
rng.shuffle(positive_images_temp)


negative_images = anchor_images_temp + positive_images_temp
np.random.RandomState(seed=32).shuffle(negative_images)
#print("La longitud de los dataset positivos y de ancla son:"+␣
 ↪str(len(positive_images)) +", "+str(len(anchor_images)))
#print("Y la longitud del dataset negativo es: "+ str(len(negative_images)) +␣
 ↪"\n")
# Ahora pasamos a recorrer todo el conjunto de parejas ancla-positiva para␣
 ↪comprobar que no haya solapamientos con
# las fotos negativas
test_negative_images=[]
is_left=0
is_right=0
j = 0
for i in anchor_images:
    #Cogemos la imagen ancla:
    num_ancla=i[33:]
    num_ancla=num_ancla.rpartition('.')[0]
    #print("El número de la foto de ancla es:  "+str(num_ancla))

    #Cogemos la imagen positiva:
    num_positiva=positive_images[j]
```

```
    num_positiva=num_positiva[34:]
    num_positiva=num_positiva.rpartition('.')[0]
    #print("El número de la foto positiva es:  "+str(num_positiva))

    #Cogemos la negativa asociada a esta pareja para comprobar si es válida,␣
↪como estas están randomizadas, tendremos
    #tanto de la cacrpeta de left como de right por lo que debemos␣
↪identificarlas:
    neg_img=negative_images[j]
    str_neg=neg_img[22:]
    str_neg=str_neg.rpartition('/')[0]

    if str_neg=="left":
        #print("Estamos en la carpeta: left.") #Valores impares
        str_f_neg=neg_img[33:]
        neg_im=str_f_neg.rpartition('.')[0]
        #print("El valor de la foto negativa es:   " + neg_im + "   (left,␣
↪impares)")
        neg_im=int(neg_im)
        is_left = 1

    elif str_neg=="right":
        #print("Estamos en la carpeta: right.") #Valores pares
        str_f_neg=neg_img[34:]
        neg_im=str_f_neg.rpartition('.')[0]
        #print("El valor de la foto negativa es:   " + neg_im + "   (right,␣
↪pares)")
        neg_im=int(neg_im)
        is_right = 1

    #print("\nEstablecemos el intervalo en el que consideramos que no debe haber␣
↪coincidencias (100 fotos en cada dirección):")
    max_up  = neg_im+100
    max_down=neg_im-100
    if(max_down< 0):
        max_down=0
    if(max_up >2000):
        max_up = 2000
    n = range(max_down, max_up)
    #print("Rango para esta foto:  "+str(n))

    #Vemos si el valor de la imagen negativa está lo suficientemente lejos de la␣
↪pareja:
```

```
    if(max_down<= int(num_ancla) <= max_up or max_down<= int(num_positiva) <=␣
↪max_up):
        #print("¡¡¡ CONFLICTO !!!")

        #Lo que queremos es alejarnos lo suficiente de la pareja para que deje␣
↪de haber conflicto, para ello
        #podemos simplemente sumarle 110 posiciones a la foto negativa que está␣
↪cogiendo, pero llevando cuidado
        #porque en los extremos, tendremos situaciones algo diferentes.

        if(neg_im >= 1700):
            new_neg_im = neg_im - 300 #Le restamos algo más de 100 fotos ya que␣
↪por persona hay 100 exactas
        else:
            new_neg_im = neg_im + 300 #Le sumamos algo más de 100 fotos ya que␣
↪por persona hay 100 exactas

        #print("\nEl nuevo valor de la negativa es:  " + str(new_neg_im))

        #Comprobamos que efectivamente el conflicto ha dejado de existir.
        new_max_up = new_neg_im+100
        new_max_down = new_neg_im-100
        if(new_max_down<0):max_down=0
        if(max_up>2000):max_up=2000
        new_n=range(new_max_down, new_max_up)
        #print("El nuevo rango de seguridad sería:  "+str(new_n))
        if(new_max_down<= int(num_ancla) <= new_max_up or new_max_down<=␣
↪int(num_positiva) <= new_max_up):
            #print("¡¡¡ CONFLICTO !!!")
            pass
        else:
            #print("Ahora todo OK!")

            #Como aquí hemos tenido que corregir la imagen, el método de␣
↪guardado será algo distinto:

            #Primero determinamos si era una foto de left o right:
            #print("El valor de is_left es: " +str(is_left))
            #print("El valor de is_right es: " +str(is_right))
            if(is_left==1):
                new_im_path=i[0:33]
                #print("La foto irá en left.")
            elif(is_right==1):
                new_im_path=positive_images[j]
```

```
                new_im_path=new_im_path[0:34]
                #print("El path nuevo será "+ new_im_path)
                #print("La foto irá en right.")

            nueva_img_final=new_im_path+str(new_neg_im)+".png"
            #print(nueva_img_final)
            test_negative_images.append(nueva_img_final)
            #print("Se ha guardado: " +new_negative_images[j])

    else:
        #print("Todo OK!")
        #Aquí el método de guardado consistirá simplemente en añadir las␣
↪imágenes negativas que no han necesitado modificación
        #print(negative_images[j])
        test_negative_images.append(negative_images[j])
        #print("Se ha guardado: " +new_negative_images[j])

    is_left=0
    is_right=0
    j = j+1

anchor_dataset = tf.data.Dataset.from_tensor_slices(anchor_images)
positive_dataset = tf.data.Dataset.from_tensor_slices(positive_images)
#---
negative_dataset = tf.data.Dataset.from_tensor_slices(test_negative_images)


#---------------------------------------------
def create_test_dataset(image_1):
    dataset_de_prueba_1=[]

    for i in anchor_images:
        dataset_de_prueba_1.append(image_1)

    dataset_de_prueba_1 = tf.data.Dataset.from_tensor_slices(dataset_de_prueba_1)

    dataset_test = tf.data.Dataset.zip((dataset_de_prueba_1, positive_dataset,␣
↪negative_dataset))
    dataset_test = dataset_test.shuffle(buffer_size=1000)
    dataset_test = dataset_test.map(preprocess_triplets)
    dataset_test = dataset_test.batch(1000, drop_remainder=False)
    dataset_test = dataset_test.prefetch(5)

    return dataset_test
```

Para las penúltimas pruebas:

[19]:
```
anchor_images_path   = 'C:\\Users\\angel\\Resultados_TFG\\Prueba_4\\left'
positive_images_path = 'C:\\Users\\angel\\Resultados_TFG\\Prueba_4\\right'

anchor_images=[]
left_data = sorted(os.listdir(anchor_images_path),key=len)
for file in left_data:
    l_file = str(anchor_images_path)+"/"+str(file)
    anchor_images.append(l_file)

positive_images=[]
right_data = sorted(os.listdir(positive_images_path),key=len)
for file in right_data:
    r_file = str(positive_images_path)+"/"+str(file)
    positive_images.append(r_file)


image_count = len(anchor_images)

anchor_images_temp=[]
positive_images_temp=[]
anchor_images_temp=anchor_images.copy()
positive_images_temp=positive_images.copy()

rng = np.random.RandomState(seed=42)
rng.shuffle(anchor_images_temp)
rng.shuffle(positive_images_temp)

negative_images = anchor_images_temp + positive_images_temp
np.random.RandomState(seed=32).shuffle(negative_images)
#print("La longitud de los dataset positivos y de ancla son:"+␣
 ↪str(len(positive_images)) +", "+str(len(anchor_images)))
#print("Y la longitud del dataset negativo es: "+ str(len(negative_images)) +␣
 ↪"\n")
# Ahora pasamos a recorrer todo el conjunto de parejas ancla-positiva para␣
 ↪comprobar que no haya solapamientos con
# las fotos negativas
test_negative_images=[]
is_left=0
is_right=0
j = 0
#print(len(anchor_images))
for i in anchor_images:
    #Cogemos la imagen ancla:
```

```
    #num_ancla=i[33:]
    num_ancla=i[50:]
    #print(num_ancla)
    num_ancla=num_ancla.rpartition('.')[0]
    #print("El número de la foto de ancla es:  "+str(num_ancla))
    #print(num_ancla)

    #Cogemos la imagen positiva:
    num_positiva=positive_images[j]
    #num_positiva=num_positiva[34:]
    num_positiva=num_positiva[51:]
    #print(num_positiva)
    num_positiva=num_positiva.rpartition('.')[0]
    #print("El número de la foto positiva es:  "+str(num_positiva))
    #print(num_positiva)

    #Cogemos la negativa asociada a esta pareja para comprobar si es válida,␣
↪como estas están randomizadas, tendremos
    #tanto de la cacrpeta de left como de right por lo que debemos␣
↪identificarlas:
    neg_img=negative_images[j]
    str_neg=neg_img[39:]
    #print(str_neg)
    str_neg=str_neg.rpartition('/')[0]
    #print("El string negativo es: "+ str(str_neg))

    if str_neg=="left":
        #print("Estamos en la carpeta: left.") #Valores impares
        str_f_neg=neg_img[50:]
        neg_im=str_f_neg.rpartition('.')[0]
        #print("El valor de la foto negativa es:   " + neg_im + "   (left,␣
↪impares)")
        neg_im=int(neg_im)
        is_left = 1

    elif str_neg=="right":
        #print("Estamos en la carpeta: right.") #Valores pares
        str_f_neg=neg_img[51:]
        neg_im=str_f_neg.rpartition('.')[0]
        #print("El valor de la foto negativa es:   " + neg_im + "   (right,␣
↪pares)")
        neg_im=int(neg_im)
        is_right = 1
```

```
    #print("\nEstablecemos el intervalo en el que consideramos que no debe haber␣
↪coincidencias (100 fotos en cada dirección):")
    max_up  = neg_im+100
    max_down=neg_im-100
    if(max_down< 0):
        max_down=0
    if(max_up >2100):
        max_up = 2100
    n = range(max_down, max_up)
    #print("Rango para esta foto:  "+str(n))

    #Vemos si el valor de la imagen negativa está lo suficientemente lejos de la␣
↪pareja:
    if(max_down<= int(num_ancla) <= max_up or max_down<= int(num_positiva) <=␣
↪max_up):
        #print("¡¡¡ CONFLICTO !!!")

        #Lo que queremos es alejarnos lo suficiente de la pareja para que deje␣
↪de haber conflicto, para ello
        #podemos simplemente sumarle 110 posiciones a la foto negativa que está␣
↪cogiendo, pero llevando cuidado
        #porque en los extremos, tendremos situaciones algo diferentes.

        if(neg_im >= 1800):
            new_neg_im = neg_im - 300 #Le restamos algo más de 100 fotos ya que␣
↪por persona hay 100 exactas
        else:
            new_neg_im = neg_im + 300 #Le sumamos algo más de 100 fotos ya que␣
↪por persona hay 100 exactas

        #print("\nEl nuevo valor de la negativa es:  " + str(new_neg_im))

        #Comprobamos que efectivamente el conflicto ha dejado de existir.
        new_max_up = new_neg_im+100
        new_max_down = new_neg_im-100
        if(new_max_down<0):max_down=0
        if(max_up>2100):max_up=2100
        new_n=range(new_max_down, new_max_up)
        #print("El nuevo rango de seguridad sería:  "+str(new_n))
        if(new_max_down<= int(num_ancla) <= new_max_up or new_max_down<=␣
↪int(num_positiva) <= new_max_up):
            #print("¡¡¡ CONFLICTO !!!")
            pass
        else:
```

```
            #print("Ahora todo OK!")

            #Como aquí hemos tenido que corregir la imagen, el método de␣
↪guardado será algo distinto:

            #Primero determinamos si era una foto de left o right:
            #print("El valor de is_left es: " +str(is_left))
            #print("El valor de is_right es: " +str(is_right))
            if(is_left==1):
                new_im_path=i[0:50]
                #print("El path nuevo será "+ new_im_path)
                #print("La foto irá en left.")
            elif(is_right==1):
                new_im_path=positive_images[j]
                new_im_path=new_im_path[0:51]
                #print("El path nuevo será "+ new_im_path)
                #print("La foto irá en right.")

            nueva_img_final=new_im_path+str(new_neg_im)+".png"
            #print(nueva_img_final)
            test_negative_images.append(nueva_img_final)
            #print("Se ha guardado: " +test_negative_images[j])

    else:
        #print("Todo OK!")
        #Aquí el método de guardado consistirá simplemente en añadir las␣
↪imágenes negativas que no han necesitado modificación
        #print(negative_images[j])
        test_negative_images.append(negative_images[j])
        #print("Se ha guardado: " +test_negative_images[j])

    is_left=0
    is_right=0
    j = j+1

anchor_dataset = tf.data.Dataset.from_tensor_slices(anchor_images)
positive_dataset = tf.data.Dataset.from_tensor_slices(positive_images)
negative_dataset = tf.data.Dataset.from_tensor_slices(test_negative_images)
```

[20]:
```
def create_test_dataset_2(image_1):
    dataset_de_prueba_1=[]

    for i in anchor_images:
        dataset_de_prueba_1.append(image_1)
```

```
    dataset_de_prueba_1 = tf.data.Dataset.from_tensor_slices(dataset_de_prueba_1)

    dataset_test = tf.data.Dataset.zip((dataset_de_prueba_1, positive_dataset,␣
↪negative_dataset))
    dataset_test = dataset_test.shuffle(buffer_size=1050)
    dataset_test = dataset_test.map(preprocess_triplets)
    dataset_test = dataset_test.batch(1050, drop_remainder=False)
    dataset_test = dataset_test.prefetch(5)

    return dataset_test
```

Y para las pruebas finales:

[26]:
```
anchor_images_path   = 'C:\\Users\\angel\\Resultados_TFG\\Prueba_5\\left'
positive_images_path = 'C:\\Users\\angel\\Resultados_TFG\\Prueba_5\\right'

anchor_images=[]
left_data = sorted(os.listdir(anchor_images_path),key=len)
for file in left_data:
    l_file = str(anchor_images_path)+"/"+str(file)
    anchor_images.append(l_file)

positive_images=[]
right_data = sorted(os.listdir(positive_images_path),key=len)
for file in right_data:
    r_file = str(positive_images_path)+"/"+str(file)
    positive_images.append(r_file)

image_count = len(anchor_images)

anchor_images_temp=[]
positive_images_temp=[]
anchor_images_temp=anchor_images.copy()
positive_images_temp=positive_images.copy()

rng = np.random.RandomState(seed=42)
rng.shuffle(anchor_images_temp)
rng.shuffle(positive_images_temp)

negative_images = anchor_images_temp + positive_images_temp
np.random.RandomState(seed=32).shuffle(negative_images)
#print("La longitud de los dataset positivos y de ancla son:"+␣
↪str(len(positive_images)) +", "+str(len(anchor_images)))
#print("Y la longitud del dataset negativo es: "+ str(len(negative_images)) +␣
↪"\n")
```

```
# Ahora pasamos a recorrer todo el conjunto de parejas ancla-positiva para␣
↪comprobar que no haya solapamientos con
# las fotos negativas
test_negative_images=[]
is_left=0
is_right=0
j = 0
#print(len(anchor_images))
for i in anchor_images:
    #Cogemos la imagen ancla:
    #num_ancla=i[33:]
    num_ancla=i[50:]
    #print(num_ancla)
    num_ancla=num_ancla.rpartition('.')[0]
    #print("El número de la foto de ancla es:  "+str(num_ancla))
    #print(num_ancla)

    #Cogemos la imagen positiva:
    num_positiva=positive_images[j]
    #num_positiva=num_positiva[34:]
    num_positiva=num_positiva[51:]
    #print(num_positiva)
    num_positiva=num_positiva.rpartition('.')[0]
    #print("El número de la foto positiva es:  "+str(num_positiva))
    #print(num_positiva)

    #Cogemos la negativa asociada a esta pareja para comprobar si es válida,␣
↪como estas están randomizadas, tendremos
    #tanto de la cacrpeta de left como de right por lo que debemos␣
↪identificarlas:
    neg_img=negative_images[j]
    str_neg=neg_img[39:]
    #print(str_neg)
    str_neg=str_neg.rpartition('/')[0]
    #print("El string negativo es: "+ str(str_neg))

    if str_neg=="left":
        #print("Estamos en la carpeta: left.") #Valores impares
        str_f_neg=neg_img[50:]
        neg_im=str_f_neg.rpartition('.')[0]
        #print("El valor de la foto negativa es:   " + neg_im + "   (left,␣
↪impares)")
        neg_im=int(neg_im)
        is_left = 1
```

```
    elif str_neg=="right":
        #print("Estamos en la carpeta: right.") #Valores pares
        str_f_neg=neg_img[51:]
        neg_im=str_f_neg.rpartition('.')[0]
        #print("El valor de la foto negativa es:   " + neg_im + "   (right,␣
↪pares)")
        neg_im=int(neg_im)
        is_right = 1

    #print("\nEstablecemos el intervalo en el que consideramos que no debe haber␣
↪coincidencias (100 fotos en cada dirección):")
    max_up  = neg_im+30
    max_down=neg_im-30
    if(max_down< 0):
        max_down=0
    if(max_up >240):
        max_up = 240
    n = range(max_down, max_up)
    #print("Rango para esta foto:  "+str(n))

    #Vemos si el valor de la imagen negativa está lo suficientemente lejos de la␣
↪pareja:
    if(max_down<= int(num_ancla) <= max_up or max_down<= int(num_positiva) <=␣
↪max_up):
        #print("¡¡¡ CONFLICTO !!!")

        #Lo que queremos es alejarnos lo suficiente de la pareja para que deje␣
↪de haber conflicto, para ello
        #podemos simplemente sumarle 110 posiciones a la foto negativa que está␣
↪cogiendo, pero llevando cuidado
        #porque en los extremos, tendremos situaciones algo diferentes.

        if(neg_im >= 180):
            new_neg_im = neg_im - 60 #Le restamos algo más de 100 fotos ya que␣
↪por persona hay 100 exactas
        else:
            new_neg_im = neg_im + 60 #Le sumamos algo más de 100 fotos ya que␣
↪por persona hay 100 exactas

        #print("\nEl nuevo valor de la negativa es:  " + str(new_neg_im))

        #Comprobamos que efectivamente el conflicto ha dejado de existir.
        new_max_up = new_neg_im+30
```

```
        new_max_down = new_neg_im-30
        if(new_max_down<0):max_down=0
        if(max_up>240):max_up=240
        new_n=range(new_max_down, new_max_up)
        #print("El nuevo rango de seguridad sería:  "+str(new_n))
        if(new_max_down<= int(num_ancla) <= new_max_up or new_max_down<=␣
↪int(num_positiva) <= new_max_up):
            #print("¡¡¡ CONFLICTO !!!")
            pass
        else:
            #print("Ahora todo OK!")

            #Como aquí hemos tenido que corregir la imagen, el método de␣
↪guardado será algo distinto:

            #Primero determinamos si era una foto de left o right:
            #print("El valor de is_left es: " +str(is_left))
            #print("El valor de is_right es: " +str(is_right))
            if(is_left==1):
                new_im_path=i[0:50]
                #print("El path nuevo será "+ new_im_path)
                #print("La foto irá en left.")
            elif(is_right==1):
                new_im_path=positive_images[j]
                new_im_path=new_im_path[0:51]
                #print("El path nuevo será "+ new_im_path)
                #print("La foto irá en right.")

            nueva_img_final=new_im_path+str(new_neg_im)+".png"
            #print(nueva_img_final)
            test_negative_images.append(nueva_img_final)
            #print("Se ha guardado: " +test_negative_images[j])

    else:
        #print("Todo OK!")
        #Aquí el método de guardado consistirá simplemente en añadir las␣
↪imágenes negativas que no han necesitado modificación
        #print(negative_images[j])
        test_negative_images.append(negative_images[j])
        #print("Se ha guardado: " +test_negative_images[j])

    is_left=0
    is_right=0
    j = j+1
```

```
anchor_dataset = tf.data.Dataset.from_tensor_slices(anchor_images)
positive_dataset = tf.data.Dataset.from_tensor_slices(positive_images)
negative_dataset = tf.data.Dataset.from_tensor_slices(test_negative_images)
```

[29]:
```
def create_test_dataset_3(image_1):
    dataset_de_prueba_1=[]

    for i in anchor_images:
        dataset_de_prueba_1.append(image_1)

    dataset_de_prueba_1 = tf.data.Dataset.from_tensor_slices(dataset_de_prueba_1)

    dataset_test = tf.data.Dataset.zip((dataset_de_prueba_1, positive_dataset,␣
 ↪negative_dataset))
    dataset_test = dataset_test.shuffle(buffer_size=120)
    dataset_test = dataset_test.map(preprocess_triplets)
    dataset_test = dataset_test.batch(120, drop_remainder=False)
    dataset_test = dataset_test.prefetch(5)

    return dataset_test
```

Ahora sí, podemos empezar con las pruebas para testear el funcionamiento de la red.

4.1. Primera evaluación

Utilizaremos fotos de prueba de cada una de las personas del dataset y las pasaremos a la red para ver si los identifica correctamente. Haremos dos pruebas:

- Una para ver cuál es la imagen más similar a la de entrada. Esta deberá coincidir con la de entrada y por tanto la distancia entre ambas será de 0.0.
- Cuáles son las 5 fotos más similares a la de entrada, donde debería mostrar diferentes fotos de la misma persona.

Para esta primera evaluación, vamos a hacer una selección aleatoria de fotos para cada persona del dataset:

[18]:
```
n = random.randint(0,49)
test1_images=[]
test1_images.append(bloque_pos_1[n])
test1_images.append(bloque_pos_2[n])
test1_images.append(bloque_pos_3[n])
test1_images.append(bloque_pos_4[n])
test1_images.append(bloque_pos_5[n])
```

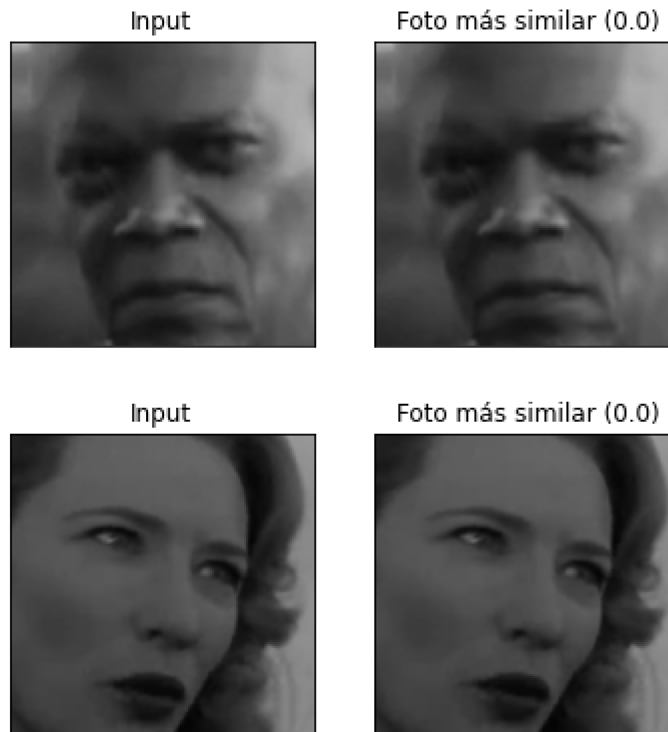
```
test1_images.append(bloque_pos_6[n])
test1_images.append(bloque_pos_7[n])
test1_images.append(bloque_pos_8[n])
test1_images.append(bloque_pos_9[n])
test1_images.append(bloque_pos_10[n])
test1_images.append(bloque_pos_11[n])
test1_images.append(bloque_pos_12[n])
test1_images.append(bloque_pos_13[n])
test1_images.append(bloque_pos_14[n])
test1_images.append(bloque_pos_15[n])
test1_images.append(bloque_pos_16[n])
test1_images.append(bloque_pos_17[n])
test1_images.append(bloque_pos_18[n])
test1_images.append(bloque_pos_19[n])
test1_images.append(bloque_pos_20[n])

first_half = test1_images[:10]
second_half = test1_images[10:]
visualize_test1_images()
```

Figura 4.1: Imágenes representativas de cada persona correspondientes al dataset de entrenamiento.

4.1.1. Primera prueba

Una vez tenemos muestras de cada persona del dataset, las pasamos por la red para buscar cuál es la imagen más similar a ellas.

[19]:
```
for j in test1_images:

    dataset_test = create_test_dataset(j)

    iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
    predict_test = new_siamese_model.predict(iteration_test)
    predict_0 = predict_test[0]
    min_index = np.argmin(predict_0, 0)

    visualize_6(*list(iteration_test), predict_0, min_index)
    del iteration_test
    del predict_test
    del predict_0
    del min_index
    del dataset_test
    gc.collect()
```



Input

Foto más similar (0.0)

Input

Foto más similar (0.0)

Input

Foto más similar (0.0)

Input

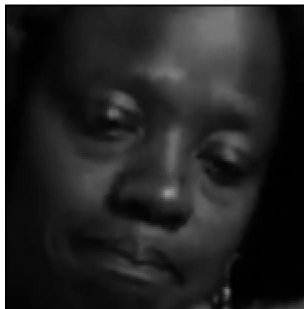
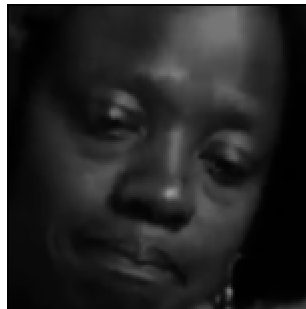
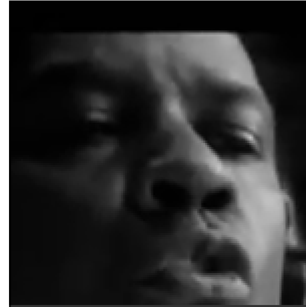


Foto más similar (0.0)



Input

Foto más similar (0.0)



Input

Foto más similar (0.0)

Input

Foto más similar (0.0)

Input

Foto más similar (0.0)

Input

Foto más similar (0.0)

Input

Foto más similar (0.0)

Input

Foto más similar (0.0)

Input

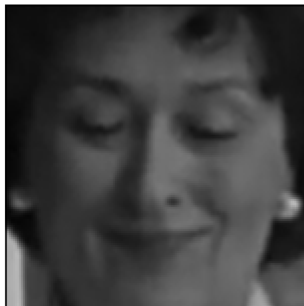
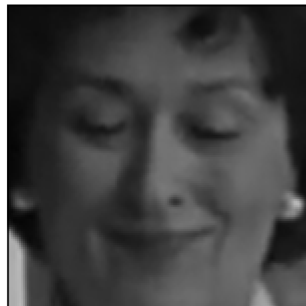
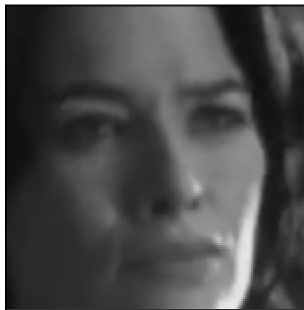


Foto más similar (0.0)



Input

Foto más similar (0.0)


Input

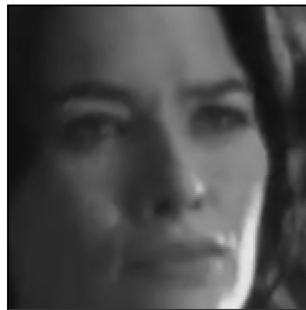


Foto más similar (0.0)

Input

Foto más similar (0.0)

Input

Foto más similar (0.0)

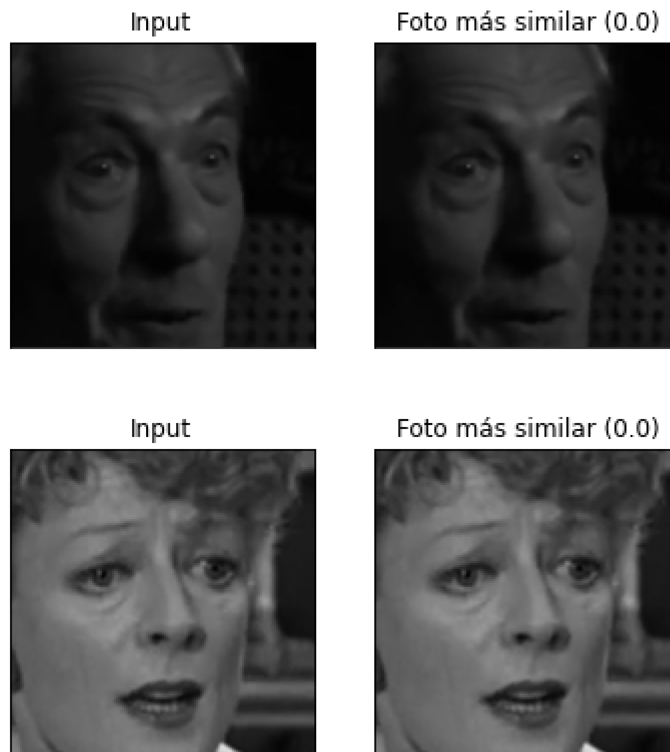


Figura 4.2: Resultados primera evaluación, primera prueba.

4.1.2. Segunda prueba

Comprobamos si la red nos muestra imágenes similares a la de entrada, esto es, fotos de la misma persona.
El código utilizado es el siguiente, donde para simplificar la prueba y reducir la carga sobre el pc, hemos reducido la prueba a una persona aleatoria de las 20 obtenidas anteriormente:

[20]:
```
n = random.randint(0,19)
input_image = test1_images[n]
dataset_de_prueba_1=[]

dataset_test = create_test_dataset(input_image)

iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_4(*list(iteration_test), predict_0)
del iteration_test
del predict_test
del predict_0
del dataset_test
del dataset_de_prueba_1
gc.collect()
```

Por tratar de reutilizar la mayor cantidad de código posible, se ha reutilizado la función visualize_4, la cual muestra un valor para el número de ancla que en este caso no se corresponde con el real, ya que todas las imágenes de ancla son la misma. Esto no tiene mayor efecto y por eso no se ha modificado.

Los resultados obtenidos son los siguientes:

[20]: 98519

Ancla nº: 803

Foto similar (0.0)

Figura 4.3: Resultados primera evaluación, segunda prueba.

4.2. Segunda evaluación

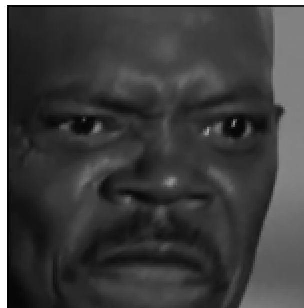
Pasar a la red fotos que no estén en el dataset pero que sean de personas que sí estén en el dataset con el que se ha entrenado la red. Para esta evaluación solo tendremos una prueba que consistirá en ver cuáles son las fotos más similares dentro de todo el dataset a la de entrada.

[21]:
```
input_image =#Image_Path

dataset_test = create_test_dataset(input_image)

iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_4(*list(iteration_test), predict_0)

del iteration_test
del predict_test
del predict_0
del dataset_test
gc.collect()
```

Donde #Image_Path se corresponde con el directorio en el que tengamos la foto a utilizar de prueba. Para esta prueba hemos utilizado tres imágenes diferentes. Los resultados obtenidos son los siguientes:

Para la primera imagen:

[21]: 24961



Ancla nº: 548

Foto similar (0.40797782)

Ancla nº: 991

Foto similar (0.47318903)

Ancla nº: 467

Foto similar (0.5984458)

Ancla nº: 500

Foto similar (0.6676588)

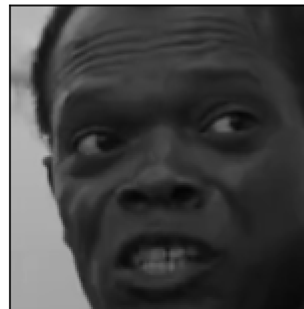
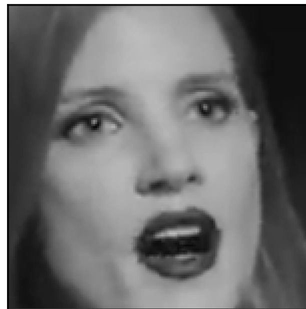
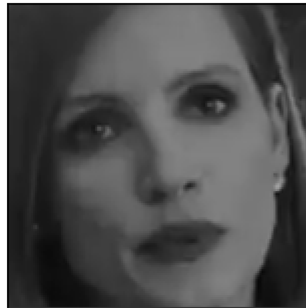


Figura 4.4: Resultados segunda evaluación, primera foto.

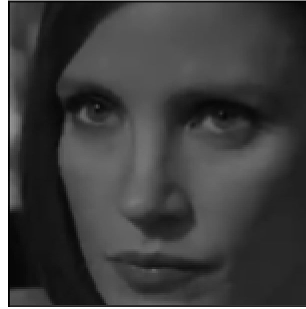
Para la segunda imagen:





Ancla nº: 286

Foto similar (0.31277567)



Ancla nº: 582

Foto similar (0.36663425)

Ancla nº: 789

Foto similar (0.36671585)

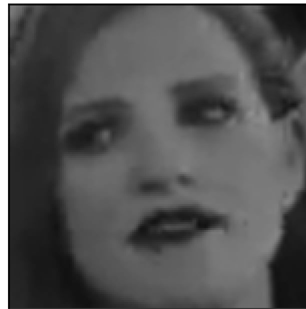
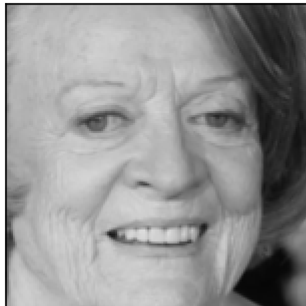
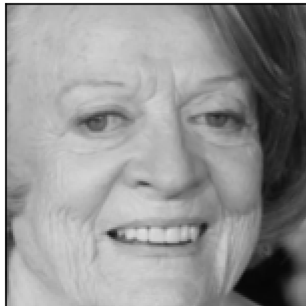


Figura 4.5: Resultados segunda evaluación, segunda foto.

Para la tercera imagen:





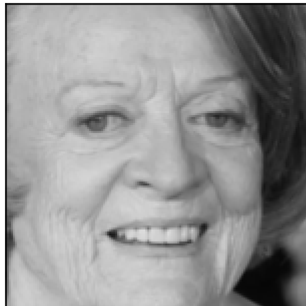


Figura 4.6: Resultados segunda evaluación, tercera foto.

4.3. Tercera evaluación

[23]:
```
input_image =#Image_Path

dataset_test = create_test_dataset(input_image)

iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_4(*list(iteration_test), predict_0)

del iteration_test
del predict_test
del predict_0
del dataset_test
gc.collect()
```

[23]: 24966

Figura 4.7: Resultados tercera evaluación.

4.4. Cuarta evaluación

Para esta evaluación, vamos a realizar dos pruebas diferentes:

- Una prueba en la que vamos a añadir al dataset fotos pertenecientes a una persona nueva, que no estuviera durante la fase de entrenamiento. El objetivo será ver si es capaz de identificarla sobre las caras con las que se ha entrenado la red.
- Otra prueba en la que utilizaremos un dataset completamente nuevo, en el que trataremos de ver si el entrenamiento de la red sirve para identificar caras de personas ajenas al dataset de entrenamiento, sobre un dataset en el que tampoco se ha entrenado.

4.4.1. Primera prueba

[25]:
```
input_image = create_test_dataset(input_image)

dataset_test = create_test_dataset_2(input_image)
iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_10(*list(iteration_test), predict_0)

del iteration_test
del predict_test
del predict_0
del dataset_test
gc.collect()
```

[25]:
```
49491
```

Ancla nº: 318

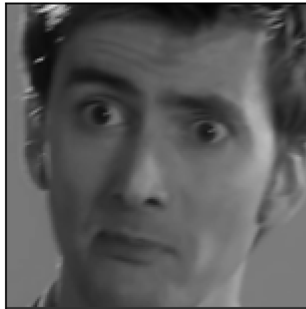


Foto similar (0.11606812)

Ancla nº: 505

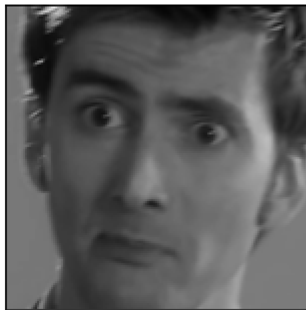
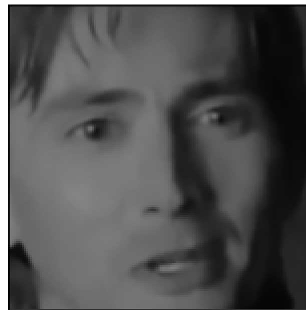


Foto similar (0.13768315)



Ancla nº: 731

Foto similar (0.14571053)

Ancla nº: 70

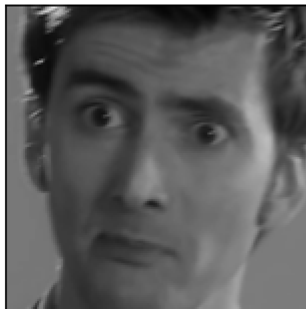
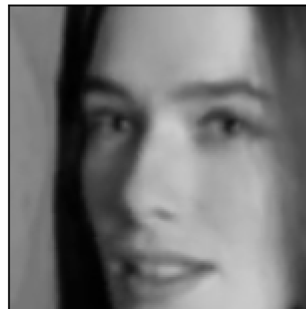


Foto similar (0.15013671)



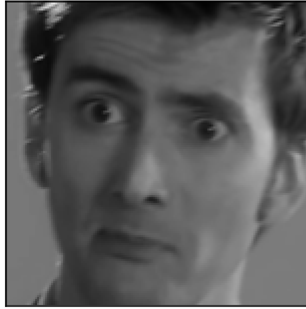


Figura 4.8: Resultados cuarta evaluación, primera prueba.

4.4.2. Segunda prueba

Para la primera persona:

[20]:
```
n = random.randint(0,119)

input_image = positive_images[n]

dataset_test = create_test_dataset_3(input_image)
iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_5(*list(iteration_test), predict_0)
del iteration_test
del predict_test
del predict_0
del dataset_test
del input_image
del n
gc.collect()
```

Ancla nº: 42

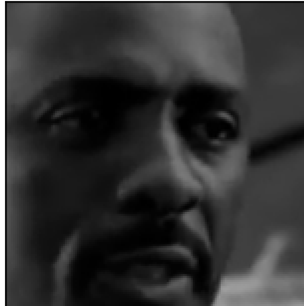
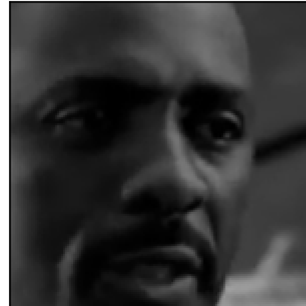


Foto similar (0.0)



Ancla nº: 5

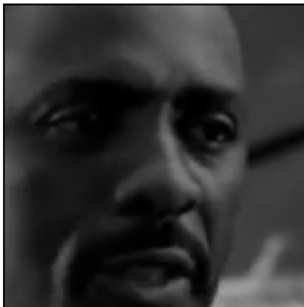
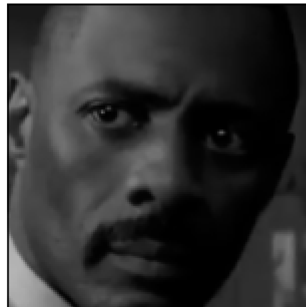


Foto similar (0.25855038)



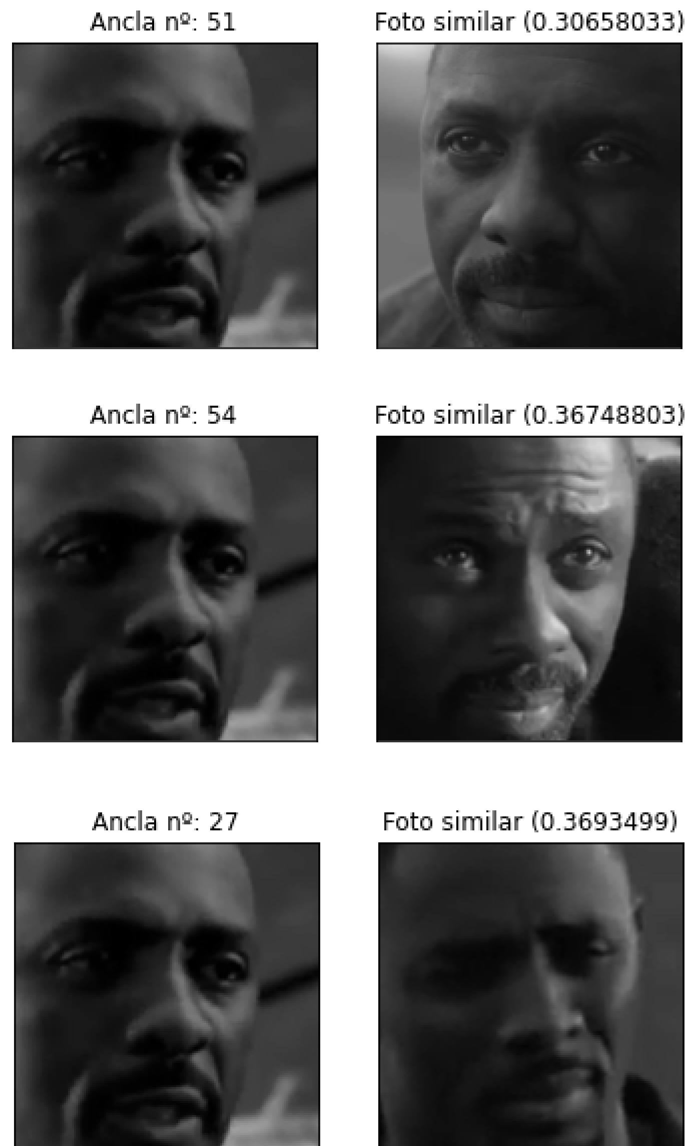


Figura 4.9: Resultados cuarta evaluación, segunda prueba, primera persona.

Para la segunda persona:

[27]:
```
n = random.randint(0,119)

input_image = positive_images[n]

dataset_test = create_test_dataset_3(input_image)
iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_5(*list(iteration_test), predict_0)
del iteration_test
del predict_test
del predict_0
del dataset_test
del input_image
del n
gc.collect()
```

[27]: 24971

Ancla nº: 25

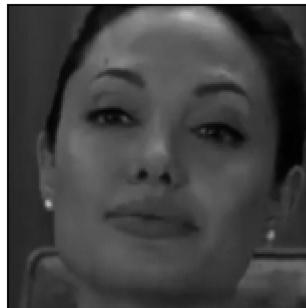
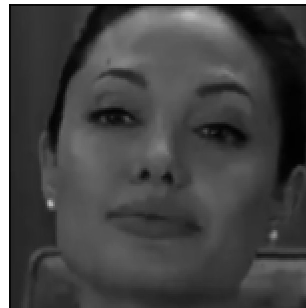


Foto similar (0.0)



Ancla nº: 9

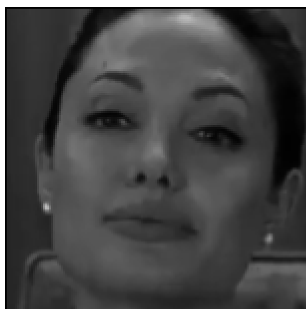


Foto similar (0.34577906)

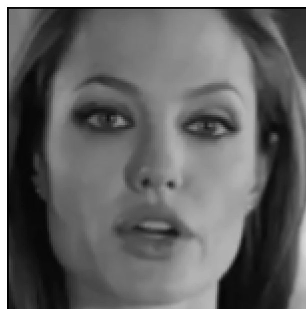


Figura 4.10: Resultados cuarta evaluación, segunda prueba, segunda persona.

Para la tercera persona:

[39]:
```
n = random.randint(0,119)

input_image = positive_images[n]

dataset_test = create_test_dataset_3(input_image)
iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_5(*list(iteration_test), predict_0)
del iteration_test
del predict_test
del predict_0
del dataset_test
del input_image
del n
gc.collect()
```

[39]: 24961

Ancla nº: 67

Foto similar (0.0)

Ancla nº: 100

Foto similar (0.415385)

Figura 4.11: Resultados cuarta evaluación, segunda prueba, tercera persona.

Para la cuarta persona:

[37]:
```
n = random.randint(0,119)

input_image = positive_images[n]

dataset_test = create_test_dataset_3(input_image)
iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_5(*list(iteration_test), predict_0)
del iteration_test
del predict_test
del predict_0
del dataset_test
del input_image
del n
gc.collect()
```

[37]: 24966

Ancla nº: 33

Foto similar (0.0)

Ancla nº: 26

Foto similar (0.044310298)

Figura 4.12: Resultados cuarta evaluación, segunda prueba, cuarta persona.

Para la quinta persona:

```
[32]: n = random.randint(0,119)

input_image = positive_images[n]

dataset_test = create_test_dataset_3(input_image)
iteration_test = *list(dataset_test.take(1).as_numpy_iterator())[0],
predict_test = new_siamese_model.predict(iteration_test)
predict_0 = predict_test[0]

visualize_5(*list(iteration_test), predict_0)
del iteration_test
del predict_test
del predict_0
del dataset_test
del input_image
del n
gc.collect()
```

```
[32]: 24956
```

Ancla nº: 54

Foto similar (0.0)

Ancla nº: 83

Foto similar (0.4557965)

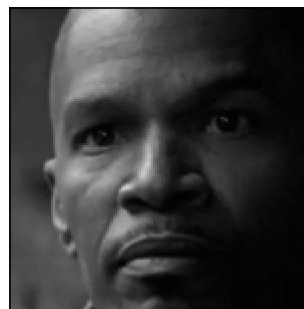


Figura 4.13: Resultados cuarta evaluación, segunda prueba, quinta persona.

Conclusiones y líneas futuras

5.1. Conclusiones por evaluación

5.1.1. Primera evaluación

En esta primera fase, hemos podido observar resultados muy positivos, siendo certera en la totalidad de los casos para la primera prueba, y un índice de acierto en la segunda prueba muy elevado. Si bien por la extensión del proyecto no se han manifestado la totalidad de las pruebas debido a su larga extensión, se han proporcionado resultados en los que se demuestra su correcto funcionamiento.

5.1.2. Segunda evaluación

En esta segunda fase, hemos obtenido nuevamente unos resultados bastante prometedores. Obteniendo un índice de acierto superior al 60 % en la mayoría de los casos. En esta prueba no obstante, se ha podido observar un índice de acierto algo más disparejo en función de la persona elegida, encontrando casos en los que el índice de acierto rara vez no era del 100 %, a casos en los que rondaba entre el 40 % y el 60 %. Lo positivo del resultado, es que si bien hay algún caso en el que la identificación no es lo precisa que se desearía, los individuos que muestra sí que muestran rasgos muy similares a los de la imagen de entrada.
Esto podría deberse a la calidad de los datos utilizados durante el entrenamiento y podría ser síntoma de que los datos para estos individuos son de calidad reducida. Otro planteamiento podría ser simplemente el de modificar el proceso de entrenamiento, esperando conseguir una mayor diferenciación entre las características de los individuos.

5.1.3. Tercera evaluación

En esta tercera fase, si bien los resultados pueden parecer menos evidentes, nos mostramos optimistas. Evidentemente debido a la naturaleza de la prueba, no nos muestra individuos iguales al de entrada, pero sí que nos muestra imágenes donde algunos rasgos faciales cuadran bastante con los del ancla.
Sí que podemos observar que hay algunos rasgos que parecen no tener tanto peso en la identificación, como pueden ser la longitud del pelo o los ojos (al menos en las pruebas obtenidas). Pero otros como la nariz, boca, cejas o disposición del pelo en la cara, sí que parecen tener un gran peso en la identificación.

5.1.4. Cuarta evaluación

Para esta última evaluación, vamos a diferenciar en función de la prueba:

- Primera prueba: hemos logrado un índice de acierto aceptable, obteniendo un buen índice de acierto. Este optimismo se ve acentuado cuando observamos que los resultados ofrecidos que no coinciden, si bien por definición son técnicamente un fracaso, sí que poseen rasgos que guardan una similitud considerable con la imagen de entrada.
- Segunda prueba: en esta última prueba los resultados han sido nuevamente muy prometedores. Ofreciendo mejoras con respecto al caso anterior, incrementándose el índice de aciertos de la identificación. Además, ha mantenido una similitud bastante decente en aquellos casos donde ha errado al reconocer a los individuos.

5.2. Conclusiones generales

En vista de las pruebas realizadas y los resultados obtenidos, podemos obervar una tendencia hacia un mejor funcionamiento de la red para aquellos datos en los que ha sido entrenada. No obstante, sí que muestra resultados prometedores aun cuando se le pasan imágenes ajenas al dataset de entrenamiento, logrando un índice de acierto aceptable. Ahora bien, esta tendencia podría ser corregida hasta cierto punto, mediante una mejora de los datos, tanto los utilizados para el entrenamiento como para los utilizados en las pruebas, remarcando una vez más la importancia de estos en el funcionamiento de este tipo de redes.

5.3. Líneas futuras

Una vez dado por terminado el proyecto. Son varias las líneas futuras que se abren para su ampliación:

- Ampliación del dataset: teniendo en cuenta las conclusiones obtenidas, no es ninguna sorpresa que una de las mejoras posibles sea ampliar el dataset utilizado, intentando mantener siempre un equilibrio en la variedad de las muestras con el objetivo de evitar sesgos en el entrenamiento de la red.
- Mejora del dataset: es importante resaltar que no solo sería interesante ampliar el dataset para esperar mejores resultados. Pues otro de los factores que hemos encontrado claves es la calidad de los datos. En nuestro caso se planteó la opción de utilizar técnicas para mejorar la extracción de los rasgos para el reconocimiento facial, pero se dejaron de lado debido a la ya extensa amplitud del proyecto. Sería por tanto bastante interesante la implementación de este tipo de técncias y la nueva evaluación de la red, comprobando si mejoran sus estimaciones o no.
- Generar estadísticas asociadas a los resultados: juntando ambos puntos anteriores y usando como base lo hecho en este trabajo, sería interesante realizar un estadístico en profundidad de cada una de las pruebas propuestas. Indicando con una precisión mucho mayor el índice de acierto de la red para cada una de las condiciones de funcionamiento.
- Posibilidad de etiquetamiento de los datos: una vez se dispone de una red capaz de identificar correctamente a diferentes individuos. Podría ser interesante otorgarle la capacidad para identificarles dada una etiqueta como podría ser el nombre. Usos como este podemos verlos continuamente en aplicaciones de películas y series vía streaming, donde dada una escena, podemos ver en pantalla los actores que la conforman. Esto puede resultar útil para expandirlo conectándolo a una base de datos donde se nos podría indicar la filmografía de un actor o actriz seleccionados, premios obtenidos, próximos estrenos, por poner algunos ejemplos.

Las posibilidades a partir de aquí son muchas y variadas, simplemente se han expuesto algunas ideas que han surgido como ideas a realizar durante el proyecto pero que finalmente no se han incluido por exceder las dimensiones del proyecto. Aunque seguro que hay muchas otras vías que no se han planteado en este documento.

Bibliografía

[1] Aurélien Géron. HandsOn Machine Learning with ScikitLearn, Keras & TensorFlow (Second Edition). O'Reilly Media, 2019.

[2] Sindhu V,Nivedha S,Prakash M, "AN EMPIRICAL SCIENCE RESEARCH ON BIOINFORMATICS IN MACHINE LEARNING – Journal" pp.3

[3] Roberto Castro Sundin, Tony Rönnqvist, Alejandro Sarmiento González & Simon Westberg "Siamesifying the COVID-Net" (disponible en https://people.kth.se/~rosun/deeplearning/index.html).

[4] Hazem Essam & Santiago L. Valdarrama, Image similarity estimation using a Siamese Network with a triplet loss. <https://keras.io/examples/vision/siamese_network/>

[5] François Chollet, Introduction to Keras for Researchers (2020/10/02). <https://keras.io/getting_started/intro_to_keras_for_researchers/>

[6] Kathy Wu & François Chollet, Serialization and saving (2020/04/28). <https://keras.io/guides/serialization_and_saving/>

[7] Rick Chao & François Chollet, Writing your own callbacks (2020/07/12). <https://keras.io/guides/writing_your_own_callbacks/>

[8] TensorFlow. <https://www.tensorflow.org/guide?hl=es-419>

[9] Numpy. <https://numpy.org/install/>

[10] Numpy quickstart. <https://numpy.org/doc/stable/user/quickstart.html>

[11] OpenCV. <https://opencv.org/>

[12] OpenCV Tutorials. <https://docs.opencv.org/4.x/d9/df8/tutorial_root.html>

[13] OpenCV Tutorials. Cascade Classifier. <https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html>

[14] Inteligencia artificial. <https://en.wikipedia.org/wiki/Artificial_intelligence>

[15] Machine Learning. <https://en.wikipedia.org/wiki/Machine_learning>

[16] matplotlib Tutorials. <https://matplotlib.org/stable/tutorials/index.html>

[17] Overleaf How-to Guides. <https://es.overleaf.com/learn/how-to>