eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Grado en Ingeniería Informática

Computación

Proyecto de Fin de Grado

Reconocimiento facial mediante Matconvnet

Autor

Gontzal Ruiz Sandoval

informatika fakultatea
facultad de informática

2017

Resumen

Este proyecto se centra en el uso de las redes neuronales convolutivas, mas específicamente, el uso de la herramienta Matconvnet para la extracción de vectores de características de una imagen facial y su posterior uso para la clasificación. A lo largo de la memoria se explicarán los métodos utilizados para analizar estos vectores, como pueden ser Piramid Multi-Level Features, el análisis de clusters, y el escalamiento multidimensional, y los resultados obtenidos.

Índice general

Índice de figuras

Indice de tablas

1. CAPÍTULO

Introducción

Ya hace bastantes décadas, si no siglos, que se han estado utilizando modelos similares a las redes neuronales simples. Sin embargo, modelos más cercanos al aprendizaje profundo, como son la sucesión de muchas capas de neuronas no lineales empezaron a llevarse a cabo a partir de 1960. Tras sufrir algunos baches debido al coste computacional que estas conllevan, y el hardware del que se disponía entonces, que hicieron que el avance de estas tecnologías fuera lento, las redes neuronales profundas empezaron a llamar la atención de cada vez más gente a partir del año 2000. Esto sucedió sobre todo debido a que estas redes demostraban dejar atras en rendimiento a otros métodos de aprendizaje automático como por ejemplo lós métodos basados en kernel (p. ej. Maquinas de Vectores de Soporte). Desde 2009, las redes neuronales profundas han ganado muchos concursos internacionales en el ámbito del reconocimiento de patrones. [Schmidhuber, 2014]

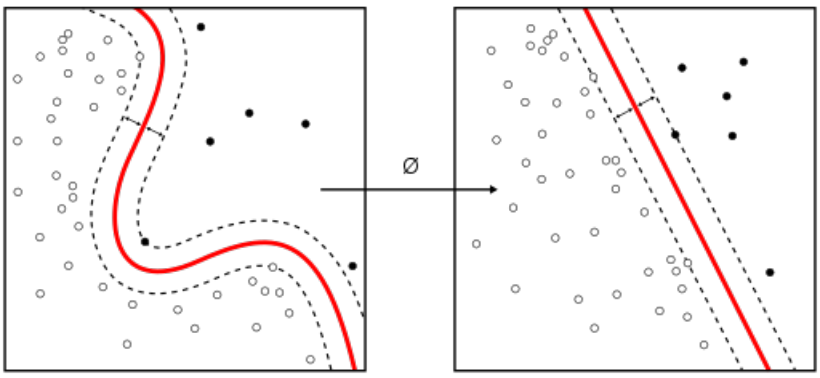

Figura 1.1: Izquierda: clasificador profundo. Derecha: clasificador lineal.

Entre estas redes de aprendizaje profundo[1], a la hora de tratar con imágenes destacan las redes convolucionales. En estas redes a cada capa se le asigna una operación. Por lo tanto, el ingeniero que diseñe la red es quién decide en que orden se van a combinar estas capas, anidando así las operaciones. Esta decisión determinará en parte la eficiencia final de la red, y es por lo tanto un paso importante a la hora de construirla.

Entre las capas que se utilizan destacan las de convolución, que podría verse con un filtro o máscara, donde cada pixel de salida es una combinación lineal de los píxeles de entrada; y las de pooling, que se usa normalmente para submuestrear el resultado de una capa de convolución 1.2. Además, las capas de convolución van acompañadas de una función de activación (1.1), normalmente *rectified linear unit* (ReLU). Esta función hará que las neuronas se activen o no, y por lo tanto es la función que decide que neuronas tendrán influencia en el resultado.

$$f(x) = max(0,x), \textit{ donde } x \textit{ es el valor de entrada a la neurona.} \tag{1.1}$$

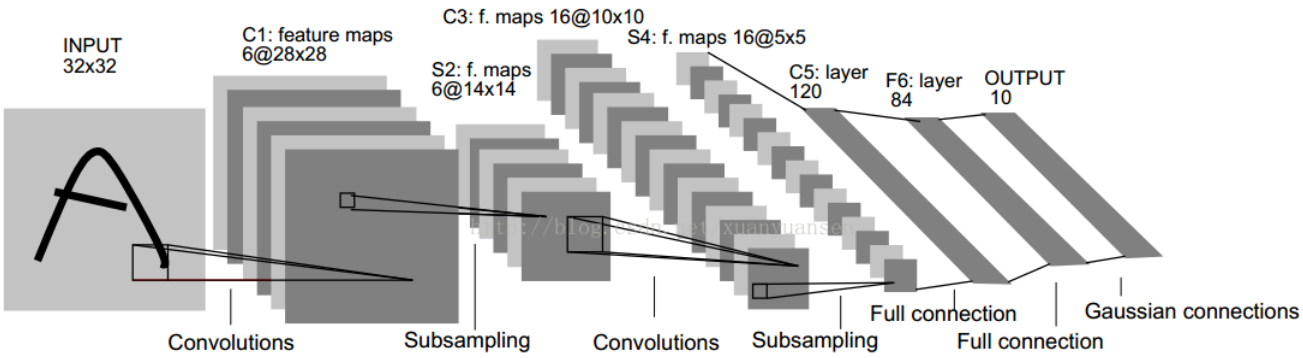

Figura 1.2: Esquema de una red convolutiva.

Tal como se ve en la figura 1.2, las ultimas capas de la red suelen estar totalmente conectadas a las neuronas activadas de la capa anterior. Estas se utilizan como método de clasificación. En el caso de este proyecto, aunque la red cuenta con capas de clasificación, como se explicará más adelante no se va a hacer uso de ello, y nos quedaremos con los datos obtenidos en la capa previa a esta clasificación. Esta capa es un vector, que en

[1]Cuando hablamos de aprendizaje profundo nos referimos a la topología de redes y algoritmos de aprendizaje que permiten a la red discriminar conjuntos de puntos más allá de lo que puede hacer un clasificador lineal. En la figura 1.1, el clasificador de la izquierda es lo que consideramos un clasificador profundo, mientras que el de la derecha es lineal.

principio, contiene las características de la imagen de entrada. Es aquí cuando se plantea la pregunta, ¿Son estas características obtenidas un buen discriminante de individuos? Si es así, ¿discriminarán mejor a las personas por el color de su piel, por su edad, por su sexo...? ¿O quizá estos atributos no tengan importancia?

Durante el proyecto se intentará dar respuesta a estas preguntas. Para ello se utilizarán diferentes métodos con la intención de entender el por qué de los buenos resultados que se obtienen con esta red. Se intentará deducir si esta red funciona bien en cualquier caso, o si funciona mejor al trabajar con conjuntos de imágenes con ciertos atributos y peor con otros.

2. CAPÍTULO

Documento de objetivos del proyecto

2.1. Objetivos

La motivación del proyecto es utilizar una red convolutiva para extraer características de una imagen y experimentar con ellas para determinar su utilidad y validez a la hora de resolver problemas de clasificación. Los objetivos del proyecto son (i) hablar de los métodos que se utilizarán para después realizar las pruebas, haciendo hincapié en cuanto a su aplicación sobre imágenes faciales; y (ii) utilizar el clasificador para después analizar los resultados, siendo el objetivo de esta clasificación reconocer un rostro en una imagen, y utilizar otros métodos para comparar y dar más veracidad a las conclusiones. Para este proyecto el software utilizado principalmente será Matlab y Matconvnet (Toolbox de Matlab).

2.2. Planificación

El proyecto comienza el 7 de enero del 2017, y según la planificación realizada terminará el 1 de junio. Sin embargo, dado que la entrega de la memoria tiene como limite el 15 del mismo mes, se contempla la posibilidad de utilizar esos días de diferencia en caso de necesitarlo por haberse alargado el tiempo de ejecución del proyecto.

2.2.1. Etapa de prueba y recogida de herramientas:

1. Instalar software a utilizar: Matlab, Toolbox Matconvnet. Además hay que probar el software instalado para comprobar que funcione de forma correcta.

2. Lectura y resumen de artículos. En este caso se destaca el articulo "Pyramid Multi-Level Features for Facial Demographic Estimation", ya que se utilizará un método basado en el método de procesamiento de la imagen definido en este.

3. Selección y obtención de la base de datos.

2.2.2. Etapa de desarrollo de la aplicación:

1. Realizar el pre-procesado de las imágenes.

2. Extraer el vector de características de cada una de las imágenes de la base de datos para cada construcción de la pirámide.

3. Experimentación: se probará lo hecho anteriormente para el reconocimiento de uno o varios individuos de la base de datos.

4. Analizar los datos obtenidos en el punto anterior.

2.2.3. Etapa de mejora del análisis:

1. Se hará mediante varios métodos un análisis de los vectores de características obtenidos por la red. Este punto dependerá del tiempo disponible tras terminar la segunda etapa, y de la consideración que se haga de lo que es viable llevar a cabo.

2. Comparar los resultados de la segunda etapa con los de esta.

Para planificar los tiempos de trabajo, y a que se dedicarán esos tiempos se utilizara el gráfico gant. Se ha realizado un gráfico por cada una de las etapas.

Figura 2.1: Gráfico Gant de la primera etapa

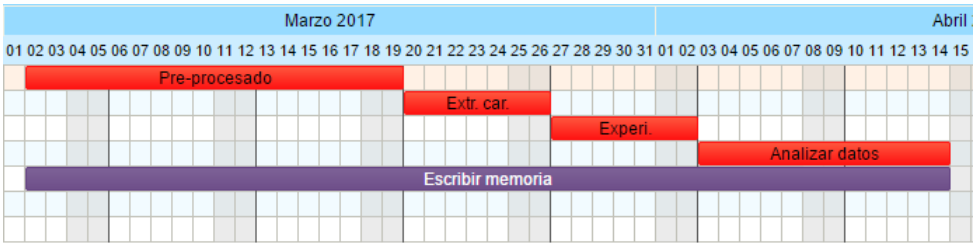

Figura 2.2: Gráfico Gant de la segunda etapa

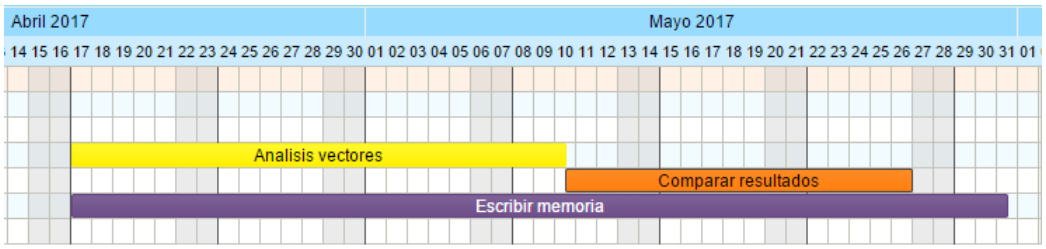

Figura 2.3: Gráfico Gant de la tercera etapa

Estos gráficos se han realizado con ayuda de la herramienta Tom's planner [N.V., 2017]. Esta es una aplicación online. Se ha elegido ya que es muy sencilla. Sin embargo, dado que se ha utilizado la versión gratuita de la herramienta, algunas funcionalidades no han podido utilizarse. Es por ello que las dependencias no se han indicado mediante una flecha (que es como se hace normalmente). Sin embargo, en las etapas segunda y tercera todas las tareas del mismo color dependen de la inmediatamente anterior en el tiempo, por lo que es fácil ver las dependencias aún sin indicarlo mediante flechas.

Para llevar esta planificación a cabo, se ha utilizado la herramienta online Kanbantool [KT, 2017]. Esta es gratuita, y presenta una interfaz que simula un tablero donde las tareas a realizar se escriben en papeletas tipo post it y pueden moverse entre diferentes columnas. En este caso, las columnas son: Por hacer, En progreso, Terminado.

2.3. Riesgos del proyecto

A continuación se listarán los riesgos que se han considerado destacables y las medidas que se tomarán en caso de que sucedan, ya que aunque haya otros como la indisponibilidad de algún participante del proyecto, no se considera que estos tengan un impacto suficientemente alto como para buscarles una solución de forma preventiva.

- **Pérdida de información:** en cualquier momento se podrían perder, por motivos inesperados, datos necesarios para el desarrollo del proyecto, bien sean ficheros de código, figuras a utilizar en la memoria, u otros ficheros donde se haya guardado información relevante y/o necesaria. Como medida preventiva se realizarán copias de seguridad de todos los archivos de forma periódica. En caso de perder algún dato y no tener copia del mismo, se volverá a obtener mediante el mismo método por el que se consiguió inicialmente, aplazando el proyecto el tiempo necesario para hacerlo.

- **Estimación de tiempo inviable:** podría darse el caso en el que la estimación de trabajo a realizar para el plazo planteado no se haya calculado correctamente, imposibilitando la finalización del proyecto, al menos dentro del plazo esperado. En caso de darse esta situación se reestimará el tiempo necesario para realizar los objetivos planificados, haciendo uso de el tiempo que se puso como colchón al principio del proyecto (15 dias). Si se considera que estos objetivos no son alcanzables en ese plazo de tiempo, se modificarán estos objetivos para que se ajusten al tiempo disponible.

- **Problemas de desarrollo:** aunque la estimación de tiempo se haya hecho correctamente, podrían darse situaciones en las que una tarea específica no pueda terminarse en el intervalo de tiempo que se le asignó en la planificación. Este problema causará un retraso en la planificación del proyecto. Para lidiar con ello se utilizará el tiempo que se puso como colchón entre la fecha límite de entrega y la fecha de finalización planificada. Si este problema se agravara, haciendo imposible la finalización del proyecto aun usando los días de colchón, se modificarán los objetivos para que se ajusten al tiempo disponible.

3. CAPÍTULO

Desarrollo del proyecto

3.1. Descripción del problema

Mediante el uso de la red convolutiva VGG-Face (sección 3.4) y el método PMLF (sección 3.3), se pretende reconocer un individuo de entre un grupo grande de personas en una base de datos. Para ello se utilizará una imagen como dato de entrada y se mostrarán las imágenes y el identificador de ese mismo individuo que haya en la base de datos.

Los resultados publicados con esta red neuronal son ya muy buenos, con una tasa de acierto entre 92,83 % y 99,13 %, dependiendo de los métodos utilizados [Parkhi et al., 2015]. En este caso se utilizará la versión no comercial de la base de datos MORPH [MOR, 2017], de la cual se hace una breve descripción en la sección 3.2. Esta no es la misma base de datos que se utilizó para las pruebas del artículo, así que se esperan resultados algo distintos.

Debido a los buenos resultados que se han obtenido con el método PMLF en combinación con otros descriptores de imagen como *Local Phase Quantization* y *Binarized Statistical Image Features* [Bekhouche et al., 2016], se intentará utilizar en combinación con la red neuronal, para ver si esto puede ayudar también a mejorar los resultados.

3.2. Base de datos MORPH

La base de datos MORPH contiene 55134 imágenes de más de 13000 individuos, fotografiados entre los años 2003 y 2007. El rango de edad está entre 16 y 77 años, con una media de 33. La media de imágenes por individuo es 4. La tabla 3.1 muestra el número de imágenes por origen y género. La tabla 3.2 muestra el número de imágenes adicionales que existen ademas de la imagen facial inicial. La tabla 3.3 muestra la distribución de imágenes en cuanto a la década de vida de la persona a la que se le saco la foto.

	Africano	**Europeo**	**Asiático**	**Otros**	**Africano**	**Total**
Hombre	36832	7961	141	1667	44	46645
Mujer	5757	2598	13	102	19	8489
Total	42589	10559	154	1769	63	55134

Tabla 3.1: Número de imágenes por género y origen.

	1	**2**	**3**	**4**	**5+**	**Total**
Hombre	373	2350	3606	1975	3155	11459
Mujer	85	478	712	352	532	2159
Total	458	2828	4318	1308	3687	13618

Tabla 3.2: Número de imágenes adicionales por sujeto.

	<20	**20-29**	**30-39**	**40-49**	**50+**	**Total**
Hombre	6638	14016	12447	10062	3482	46645
Mujer	831	2309	2910	1988	451	8489
Total	7469	16325	15357	12050	3933	55134

Tabla 3.3: Número de imágenes por género y década de vida.

En la figura 3.1 se puede ver cómo está organizada la base de datos. Para este proyecto se destacan los siguientes datos.

- Subject Identifier: identificador de cada sujeto. Todas las fotos de un mismo individuo tienen el mismo identificador.
- Race: indica el origen del sujeto. Puede ser B (Africano), W (Europeo), A (Asiático), H (Hispano) u O (otro).

- Gender: indica el género del sujeto. Puede ser M (hombre) o F (mujer).
- Age: indica la edad del sujeto en el momento de tomar la foto.
- Photo: indica la ruta en la que esta la foto.

id_num	picture_num	dob	doa	race	gender	facial_hair	age	age_diff	glasses	photo
9055	0	06/13/1949	09/08/2003	W	M	NULL	54	NULL	NULL	Album2/009055_0M54.JPG
9055	1	06/13/1949	10/23/2003	W	M	NULL	54	45	NULL	Album2/009055_1M54.JPG
19066	0	12/14/1935	12/08/2003	B	M	NULL	67	NULL	NULL	Album2/019066_0M67.JPG
19066	1	12/14/1935	01/23/2004	B	M	NULL	68	46	NULL	Album2/019066_1M68.JPG
19066	2	12/14/1935	03/21/2005	B	M	NULL	69	423	NULL	Album2/019066_2M69.JPG
19066	3	12/14/1935	04/04/2005	B	M	NULL	69	14	NULL	Album2/019066_3M69.JPG
19066	4	12/14/1935	04/07/2005	B	M	NULL	69	3	NULL	Album2/019066_4M69.JPG
19066	5	12/14/1935	05/26/2005	B	M	NULL	69	49	NULL	Album2/019066_5M69.JPG
19066	6	12/14/1935	05/28/2005	B	M	NULL	69	2	NULL	Album2/019066_6M69.JPG
19066	7	12/14/1935	12/22/2005	B	M	NULL	70	208	NULL	Album2/019066_7M70.JPG
19066	8	12/14/1935	07/01/2006	B	M	NULL	70	191	NULL	Album2/019066_08M70.JPG
20490	0	08/14/1937	09/26/2004	W	M	NULL	67	NULL	NULL	Album2/020490_0M67.JPG
20490	1	08/14/1937	12/23/2004	W	M	NULL	67	88	NULL	Album2/020490_1M67.JPG

Figura 3.1: Ejemplo de formato de los metadatos de la base de datos.

En este proyecto no se utilizarán todas las imágenes disponibles, ya que supondría un coste de tiempo demasiado grande, y no aportaría ningún beneficio, ya que se pueden hacer las mismas pruebas con menos imágenes. Por ello, se utilizarán normalmente 25000 imágenes.

3.3. Pyramid Multi-Level Features

En este trabajo se utilizarán las "*Pyramid Multi-Level Features*"[Bekhouche et al., 2016]. Para hacer esto, el primer paso es alinear la cara, lo cual se hará en tres pasos: (i) detección de los puntos faciales y de la propia cara, (ii) corrección de pose, y (iii) recorte de la región facial. En la figura 3.2 se pueden ver estos pasos.

Tras convertir la imagen a escala de grises, utilizando el algoritmo Viola-Jones [Wang, 2014] se detectará la cara de la persona en la imagen. Para realizar la correción de pose, se utiliza el algoritmo *Ensemble of Regression Trees* (ERT), tomando como referencia los puntos cogidos por este al localizar las marcas de los ojos. Estos puntos se rotarán hasta hacer que estén alineados horizontalmente, y después se escala la imagen, para normalizar la distancia entre estas marcas a un valor predeterminado d. El recorte de la imagen se hace estirando las marcas de los ojos a la izquierda y derecha por un valor $d.k_{side} = 0{,}5$, hacia arriba con $d.k_{top} = 0{,}75$, y hacia abajo con $d.k_{bottom} = 1{,}5$.

Figura 3.2: Detección de ojos, correción de pose y recorte [Bekhouche et al., 2016].

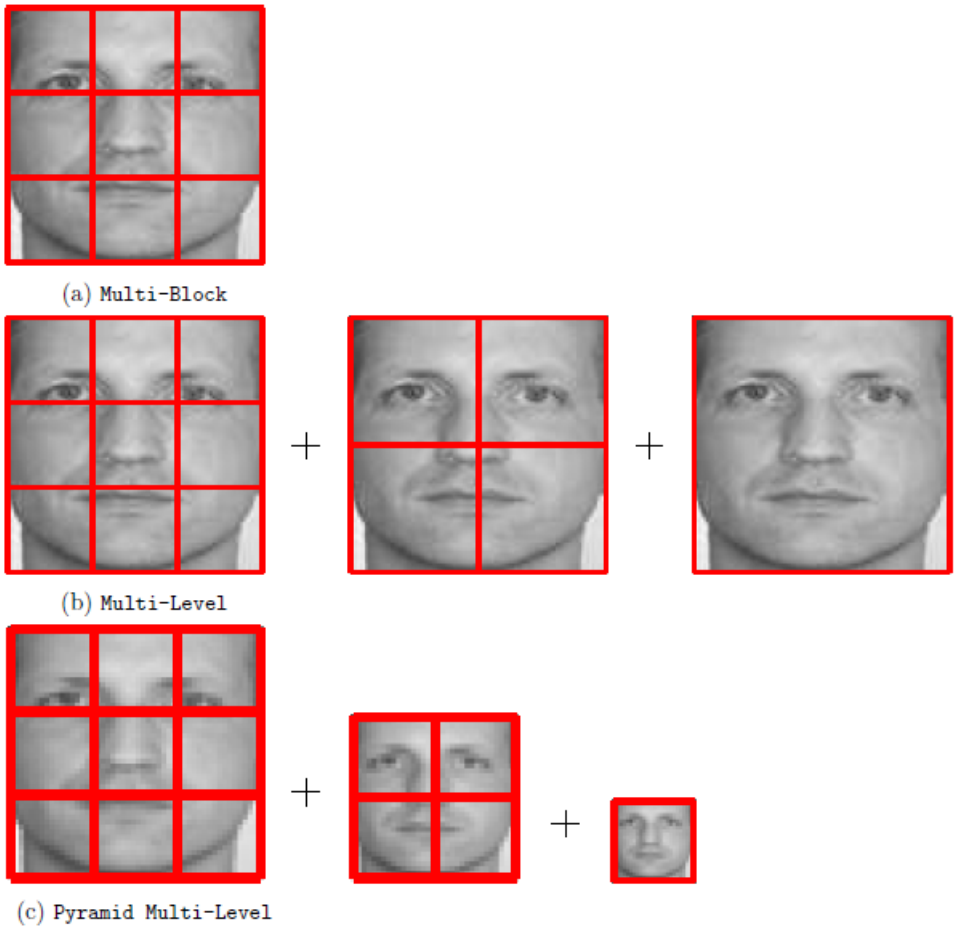

Figura 3.3: Representaciones multi-bloque, multi-nivel y multi-nivel piramidal de una misma imagen [Bekhouche et al., 2016].

Para realizar la piramide, se toma como inspiración la representación *Multi-Level*, que se basa en construir diferentes niveles ordenados de representación *Multi-Block* (MB). Esta ultima, consiste en dividir la imagen en n^2 bloques, siendo n la variable que determinará el nivel de MB que se busca. De esta forma, mientras que en la representacion Multi-nivel tendríamos varios niveles, cada uno de ellos con n niveles diferentes, en en el enfoque piramidal estos niveles se escalan para obtener una representación estrictamente piramidal. Esto se puede ver más claramente en la figura 3.3.

Los niveles de la piramide $(n, n-1, n-2, ..., 1)$ se construyen de la siguiente manera. La imagen del un nivel se obtiene utilizando como referencia la imagen del nivel anterior (función 3.1). Para cada nivel n, la imagen se dividirá en n^2 bloques.

$$w' = w \times \frac{(n-1)}{n} \quad , \quad h' = h \times \frac{(n-1)}{n} \tag{3.1}$$

w' *y* h' son el ancho y la altura de la imagen en el nivel n, mientras que w *y* h son el ancho y la altura en el nivel $n-1$.

3.4. MatConvNet

Matlab ofrece diferentes herramientas relacionadas con las redes neuronales, pero como se ha mencionado en los objetivos del proyecto, se utilizará MatConvNet [MCN, 2017] [Vedaldi and Lenc, 2015], que trabaja con redes neuronales convolucionales, ya que estas son más eficientes a la hora de trabajar con imagenes. La herramienta además esta adaptada para ser usada también mediante la GPU, pero es una característica que solo se tendrá en cuenta en este proyecto si se dispone de tiempo suficiente.

Aunque en este proyecto se utiliza una red previamente entrenada, la herramienta proporciona utilidades que permiten construir una red y trabajar con ella, como por ejemplo las funciones de activación ReLU y Sigmoide. MatConvNet esta preparado para trabajar, además de en Matlab, con los lenguajes C++ y CUDA.

La red que se va a utilizar en el proyecto es VGG-Face [Parkhi et al., 2015]. Esta ha sido entrenada usando fotos de actores y actrices famosos, una base de datos no pública. Esta red se puede descargar fácilmente de Internet, desde la página de MatConvNet. Utilizando la función *vl_simplenn* con esta red y una imagen previamente procesada (en el anexo C se puede ver su uso, donde se reescala la imagen y se modifican sus colores según los parámetros de la red), de la penúltima capa se puede obtener un vector de 4096 posiciones,

es decir, el vector de características de la imagen. El procesamiento de la imagen se ha hecho tal como se explica en la web de la herramienta *mathworks.com/products/computer-vision.html*, es decir, reescalándola y normalizando sus colores.

La red VGG-Face ha sido entrenada considerando que se trata un problema de clasificación de N posibles resultados [Parkhi et al., 2015]. Para hacerlo se tomaron $N = 2622$ individuos y se hizo que la ultima capa de la red funcionara como resultado del problema, dando la probabilidad de la imagen de entrada de pertenecer a cada una de las 2622 posibles clases. Es por eso que, si se quiere utilizar la red, se sugiere que tras utilizarla, se obvie la ultima capa, y se utilice la anterior, de 4096 posiciones, que es el vector de puntuaciones. Esto se ve en la figura 3.4, donde la ultima capa, que hace softmax se puede eliminar para quedarse solamente con 36 capas.

layer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
type	input	conv	relu	conv	relu	mpool	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	relu	mpool	conv
name	–	conv1_1	relu1_1	conv1_2	relu1_2	pool1	conv2_1	relu2_1	conv2_2	relu2_2	pool2	conv3_1	relu3_1	conv3_2	relu3_2	conv3_3	relu3_3	pool3	conv4_1
support	–	3	1	3	1	2	3	1	3	1	2	3	1	3	1	3	1	2	3
filt dim	–	3	–	64	–	–	64	–	128	–	–	128	–	256	–	256	–	–	256
num filts	–	64	–	64	–	–	128	–	128	–	–	256	–	256	–	256	–	–	512
stride	–	1	1	1	1	2	1	1	1	1	2	1	1	1	1	1	1	2	1
pad	–	1	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	1

layer	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
type	relu	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	softmx
name	relu4_1	conv4_2	relu4_2	conv4_3	relu4_3	pool4	conv5_1	relu5_1	conv5_2	relu5_2	conv5_3	relu5_3	pool5	fc6	relu6	fc7	relu7	fc8	prob
support	1	3	1	3	1	2	3	1	3	1	3	1	2	7	1	1	1	1	1
filt dim	–	512	–	512	–	–	512	–	512	–	512	–	–	512	–	4096	–	4096	–
num filts	–	512	–	512	–	–	512	–	512	–	512	–	–	4096	–	4096	–	2622	–
stride	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1
pad	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0

Figura 3.4: Configuración de la red VGG-Face [Parkhi et al., 2015].

Este vector se puede tratar como un vector de características de la imagen. Este representa la misma información de la imagen, pero lo hace de forma que la información que contiene es discriminante. Esto hace que sean una estructura de datos óptima para un problema de clasificación, ya que realza las diferencias entre las imágenes.

3.5. Procesamiento de las imágenes faciales

Para realizar el reconocimiento, primero se harán pruebas para determinar qué clasificador de los que se prueben es el que mejores resultados da. Para ello, tomando como referencia el articulo [Bekhouche et al., 2016] hay que preprocesar las imágenes, tal como se explica en la sección 3.5.1, y después extraer los vectores de características de las imágenes

preprocesadas. Estos vectores se extraerán para 3 niveles de piramide distintos, como se explica en la sección 3.5.2.

Tras esto, estos vectores se probarán y se analizarán los resultados (sección 3.6). En base a estos resultados, se desarrollará una pequeña aplicación que sirva como demostración de la funcionalidad o el uso que estos métodos pueden tener. En otras palabras, se hará un programa que reconozca un individuo en una base de datos dándole una foto del mismo, utilizando los datos obtenidos en los pasos anteriores (sección 3.7).

3.5.1. Preprocesamiento

Esto incluye la detección de la cara, determinar la posición de los ojos, rotar y recortar la imagen.

En cuanto a la detección de la cara, el propio Matlab proporciona una herramienta que facilita esta labor. Esta se llama *Computer Vision System Toolbox*, que incluye una función que utiliza el algoritmo Viola-Jones para detectar objetos, entre ellos los rostros de una imagen. Esta función devuelve como salida un cuadro delimitador (*bounding box*) que indica la posición del rostro.

Determinar la posición de los ojos (preferiblemente de la pupila) es la tarea que más tiempo ha requerido y que más problemas ha causado. En un principio se pretendió escribir una función propia que realizara la tarea. Para ello se intentó utilizar la misma herramienta que para la detección de caras, pero la eficiencia de la misma para los ojos no era suficiente, ya que tras realizar unas pocas pruebas con distintos conjuntos de imágenes se determinó que la posición de ambos ojos no era correcta en más del 50 % de los casos. Al ver que la herramienta no podía usarse, se optó por investigar acerca de otros algoritmos, como por ejemplo la identificación de patrones circulares en una imagen para detectar el iris del ojo, pero todos los métodos que se encontraron sobrepasaban la cantidad de trabajo que se esperaba dedicar a la tarea. Al final, no se ha realizado la función, y se ha utilizado una función ya hecha (proporcionada por Ignacio Arganda-Carreras y Fadi Dornaika, del grupo Ikerbasque) que valiéndose de la herramienta empleada para detectar la cara, aproxima la posición de ambos ojos.

La función tiene el inconveniente de que puede no funcionar para un tamaño, pero si para otro en una misma imagen, por lo que a la hora de preprocesar las imagenes se itera en un bucle que las reescala varias veces hasta que se detectan los ojos (o hasta que se ha probado un número máximo de veces).

Estos puntos se utilizan para rotar la imagen utilizando funciones que proporciona Matlab. Por último, se recorta la imagen tomando como referencia las proporciones indicadas en el articulo. En el caso de $d.k_{bottom}$ se ha tomado un valor menor del indicado (1,3), ya que tomando el valor del articulo algunas imágenes mantienen un trozo blanco causado por la rotación.

Una vez preprocesada cada una de las imágenes, se guardará en un directorio con nombre *AlbumPreprocess*, para realizar el preprocesado una sola vez en vez de hacerlo cada vez que sea necesario usar una imagen preprocesada.

En la tabla 3.4 se puede ver la comparación de varios scripts de preprocesado distintos (se ha probado con 400 imágenes), haciendo cada uno diferente número de iteraciones al realizar el reescalado de las imagenes y utilizando cada uno diferentes funciones con el mismo propósito (por ejemplo, el algoritmo Viola-Jones puede trabajar tanto con el parámetro *face* como con *faceCART* para detectar la región de la cara). El tiempo medio indica el tiempo medio que tarda cada imagen en preprocesarse, y la eficacia indica el porcentaje de imagenes que se han preprocesado correctamente y pueden por lo tanto utilizarse en las fases siguientes.

Script	**Tiempo medio (s)**	**Eficacia**
Preprocess1	2.2566	97.25 %
Preprocess2	2.2057	93.16 %
Preprocess3	1.4472	89.41 %

Tabla 3.4: Tiempo medio y eficacia de diferentes códigos de preprocesado.

Preprocess1 realiza hasta 6 iteraciones de reescalado de la imagen, y utiliza primero el parámetro *face* al usar Viola-Jones, y el parámetro *faceCART* en caso de que el primero no haya funcionado. **Preprocess2** hace lo mismo, pero sin probar con *faceCART*. **Preprocess3**, en cambio, realiza menos iteraciones de reescalado.

La tabla 3.6 muestra el tiempo medio y la eficacia del script a la hora de ejecutarlo para todas las imágenes de la base de datos.

Tiempo medio	**Eficacia**
1.9059	96.563 %

Tabla 3.5: Tiempo medio y eficacia de la ejecución del preprocesado.

Además de esto, se hizo también otro código de preprocesado, esta vez evitando reescalar las imágenes. En la sección 3.6 se muestran las diferencias entre ambos métodos de preprocesado a la hora de clasificar las imágenes.

Tiempo medio	Eficacia
1.1566	99.1886 %

Tabla 3.6: Tiempo medio y eficacia de la ejecución del preprocesado sin reescalar las imágenes.

3.5.2. Construcción de la pirámide y extracción del vector de características

Se han programado en Matlab los scripts para el procesado piramidal, el código se encuentra en el apéndice D.

Para construir la pirámide, se hace empezando por la base de la misma. Para ello, se ajusta ligeramente el tamaño de la imagen para que la longitud de sus lados sean múltiplos de la cantidad de los n niveles que tendrá la pirámide. Es decir, si por ejemplo, la pirámide se va a construir con $n = 3$ niveles, reescalaremos la imagen para que la cantidad de pixeles de cada lado sea múltiplo de 3. Así, cuando se divida la imagen en 3^2 bloques, todos ellos tendrán las mismas dimensiones y no se producirán irregularidades.

Tras esto, se determina el tamaño de los bloques teniendo en cuenta la altura y el ancho de la imagen. El ancho de cada bloque será el ancho total de la imagen dividido por el nivel, y de la misma forma, la altura de cada bloque será la altura total de la imagen dividida por el nivel. Teniendo las dimensiones calculadas, uno por uno, se hace el recorte del bloque, y se usa este recorte de la imagen como entrada para la red neuronal, que devolverá un vector de 4096 posiciones.

Una vez hecho esto con cada bloque del primer nivel, la imagen se reescala para que tome el tamaño que permita dividirlo en $2^{(n-1)}$ bloques de igual tamaño al calculado para el primer nivel, y se repite el proceso, hasta llegar al nivel 1, en el que solamente habrá un bloque. De esta forma, se consigue un vector de características por cada bloque. Estos vectores se concatenan, formando un solo vector que será el que se utilice para realizar la clasificación del sujeto posteriormente. Este vectores tendrán una longitud de $4096 * \sum_{i=1}^{n} i^2$, donde n es el nivel de la pirámide. Por lo tanto, se consigue un vector por cada imagen de la base de datos. Estos vectores se guardan en una matriz, donde cada

fila corresponde a una imagen, y el primer elemento de cada fila indica el número de identificación que tiene el individuo en la base de datos.

Nivel	Tiempo medio por individuo (s)
1	0.6805
2	2.2858
3	6.1844

Tabla 3.7: Tiempo medio de extracción de características por imagen, para los 3 niveles.

Se ha escrito un script adicional que da formato a esta matriz de vectores de características. La matriz tiene por defecto lineas sin contenido (todas sus columnas son 0), consecuencia de las imágenes que no se han preprocesado bien. Por esto, el script *organizeVectors* quita estas lineas innecesarias, y separa la matriz en dos: *trainMatrix* y *testMatrix*. La primera es la matriz que se utilizará para entrenar los clasificadores que lo necesiten, y la segunda es el conjunto de 400 vectores que se utilizarán para probar la eficiencia de estos clasificadores.

No se han utilizado técnicas como la validación cruzada debido a que el problema cuenta con muchas clases y muy pocas muestras por clase. Utilizar validación cruzada o una muestra completamente aleatoria para probar la eficiencia hubiera causado un porcentaje de error superior al real, ya que hay individuos que solo tienen una imagen, lo que hace que si esa imagen no se utiliza en el entrenamiento la clase correspondiente al individuo ni siquiera existiría en el modelo entrenado. Por esto se ha seleccionado un conjunto de elementos de la base de datos que, aunque aleatorios, se ha comprobado que exista al menos otra imagen del mismo individuo.

3.6. Clasificación

Tras realizar la extracción de los vectores de características para diferentes niveles de la pirámide, el objetivo es probar estos vectores con diferentes clasificadores, para así determinar su eficiencia. Matlab ofrece una herramienta para ayudarnos a ello: *Statistics and Machine Learning Toolbox*. La herramienta permite trabajar con muchos clasificadores distintos, incluyendo aprendizaje supervisado y no supervisado, tal como se ve en la figura 3.5.

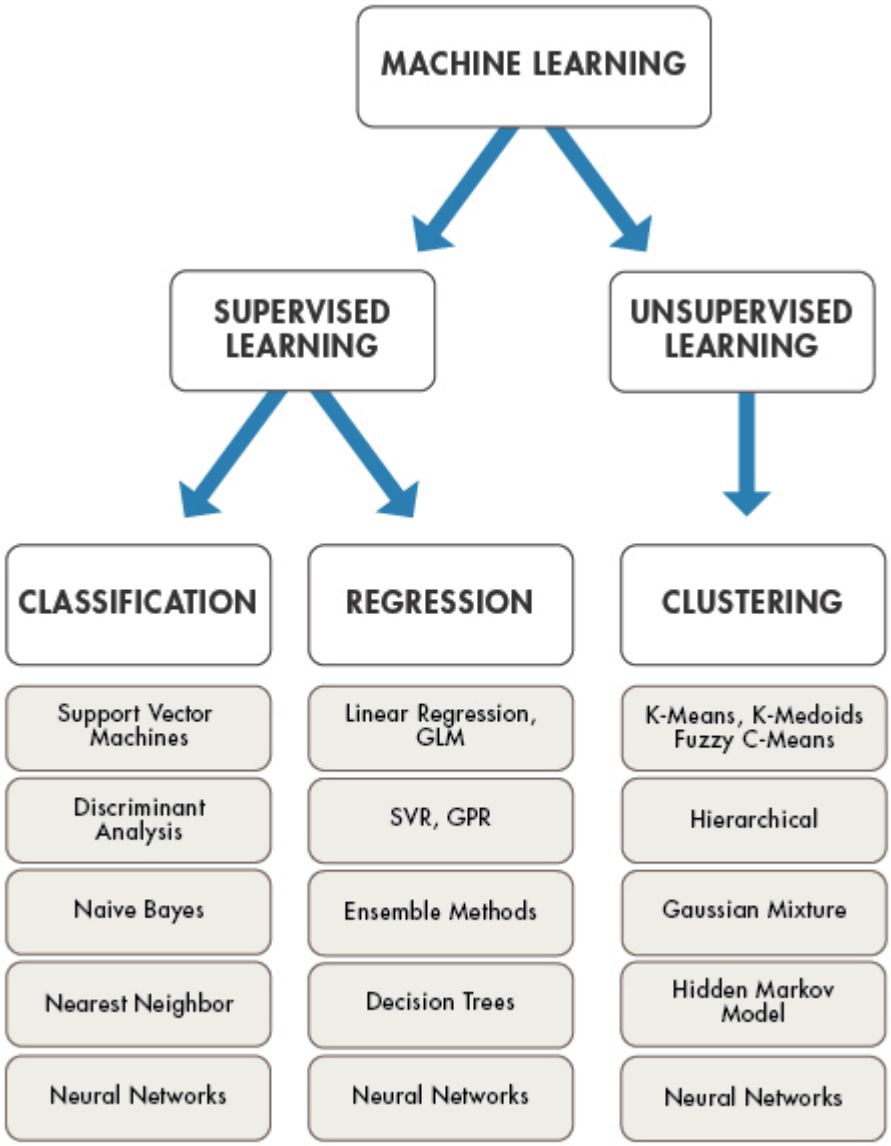

Figura 3.5: https://se.mathworks.com/help/stats/machine-learning-in-matlab.html

Los clasificadores se pueden usar de dos formas. La primera de ellas, es una interfaz gráfica que proporciona la herramienta. En esta se pueden añadir variables de entrenamiento, se pueden entrenar modelos utilizando esas variables, y se pueden ver los resultados de manera sencilla (la interfaz proporciona gráficos y tablas que representan estos resultados). La segunda es mediante funciones. Esto es menos cómodo que la herramienta, pero se puede hacer lo mismo que con la interfaz gráfica. Debido a que la interfaz trabaja de forma más lenta, y a que la alta cantidad de clases a clasificar hacen que los gráficos y tablas no sean legibles, en el proyecto se utilizan las funciones.

Dado que la red fue pensada para clasificar utilizando las distancias entre los vectores de características, el clasificador a tener en cuenta será KNN (K nearest neighbours), ya que este tiene en cuenta precisamente eso mismo, las distancias entre elementos. Matlab permite modificar, por ejemplo, el método mediante el que se calcula la distancia (euclidea, chebychev, minkowsky...), el peso de las distancias (sin pesos, inverso o inverso cuadrado) y el número de vecinos a tener en cuenta, es decir, el valor de K.

Las tablas que se muestran a continuación (3.8, 3.9 y 3.10) contienen los resultados obtenidos al probar a realizar la clasificación con los parámetros indicados en la columna *Método*. Para estas pruebas se utilizó un conjunto de entrenamiento de 24600 observaciones[1] y una muestra aleatoria de 400 como conjunto de test, tal como se ha explicado en la sección 3.5.2. En ellas se puede ver cómo la eficiencia es mayor cuanto menor es el nivel de la pirámide.

Método	**Eficiencia**	**Tiempo medio por individuo (s)**
1NN	83.25 %	0.1033
3NN pesos (inverso)	82.50 %	0.1038
3NN pesos (inverso cuadrado)	82.50 %	0.1029
2NN	74.75 %	0.1014
4NN	74.75 %	0.1015

Tabla 3.8: Tiempo medio y eficacia de las clasificaciones mediante KNN para la piramide de nivel 1.

Método	**Eficiencia**	**Tiempo medio por individuo (s)**
1NN	72.75 %	14.3290
3NN pesos (inverso)	72.50 %	13.6587
3NN pesos (inverso cuadrado)	72.50 %	14.3853
2NN	62.25 %	14.2108
4NN	62.50 %	14.1555

Tabla 3.9: Tiempo medio y eficacia de las clasificaciones mediante KNN para la piramide de nivel 2.

[1] Cada observación es una imagen. Si la base de datos cuenta con 55135 imagenes diremos que la base de datos contiene 55135 observaciones.

Método	Eficiencia	Tiempo medio por individuo (s)
1NN	42 %	40.6997
3NN pesos (inverso)	42.25 %	6.1331
3NN pesos (inverso)	42.25 %	7.5385
2NN	37.25 %	10.0304
4NN	35.50 %	9.1806

Tabla 3.10: Tiempo medio y eficacia de las clasificaciones mediante KNN para la pirámide de nivel 3.

Estos resultados niegan la hipótesis planteada al anteriormente en el proyecto. Es decir, combinar PMLF con la red VGG-Face no da mejores resultados, al contrario, hace que la tasa de acierto sea menor. La razón más plausible para esto es que la red está preparada solo para reconocer características en fotos completas de una cara. Es decir, no es capaz de identificar las características de por ejemplo el ojo izquierdo solo con la foto del ojo, sino que se basa en la posición que este ocupa en la foto para extraer la información.

Cabe destacar que el conjunto de elementos que se ha utilizado para estas pruebas contenía individuos con una sola imagen, y por lo tanto era esperable conseguir un mejor resultado para $K = 1$. Es por eso que se hicieron pruebas también con un conjunto reducido de datos, donde todos los individuos tenían entre 3 y 5 fotos. Estas se hicieron sin aplicar PMLF, ya que simplemente se pretende observar la diferencia de eficiencia cuando el modelo de clasificación tiene mas observaciones de una misma clase al entrenar. Los resultados de esto se pueden ver en la tabla 3.11.

Método	Eficiencia	Tiempo medio por individuo (s)
1NN	84.50 %	0.0639
3NN pesos (inverso)	84.50 %	0.0645
3NN pesos (inverso)	84.50 %	0.0637
2NN	81.50 %	0.0625
4NN	80.50 %	0.0679

Tabla 3.11: Resultados de la clasificación con 3 o mas observaciones por clase, sin utilizar PMLF.

Hay una pequeña mejora en cuanto a eficiencia con respecto a las pruebas anteriores. Sin embargo sorprende ver que valores mayores de K no obtienen mejores resultados.

Se esperaba que la eficiencia fuera mayor, ya que la presencia de más fotos del mismo individuo debería reducir el error que puede ocurrir en caso de que una observación de otra clase se encuentre cerca.

Además, en la sección 3.5.1 se ha explicado que se ha realizado el preprocesamiento también evitando el reescalado. La tabla 3.12 muestra los resultados obtenidos al entrenar y clasificar utilizando este preprocesamiento, sin utilizar PMLF. En ella se puede ver que los resultados son similares a los obtenidos con imagenes de diferentes tamaños, por lo que se puede deducir que la red funciona bien incluso cuando las escalas de las imágenes son diferentes.

Método	**Eficiencia**	**Tiempo medio por individuo (s)**
1NN	84.25 %	0.1921
3NN pesos (inverso)	83.25 %	0.1581
3NN pesos (inverso cuadrado)	84.25 %	0.1477
2NN	76.25 %	0.1503
4NN	73.00 %	0.1707

Tabla 3.12: Resultados de la clasificación con imágenes originales.

3.7. Aplicación

Se ha desarrollado una aplicación muy sencilla. Esta, utilizando una imagen como dato de entrada, devuelve como salida el identificador que tiene el individuo al que pertenece la imagen en la base de datos. Además muestra en pantalla todas las imágenes disponibles del mismo individuo.

Para hacer esto, primero se cargan la red neuronal y el resto de variables en el programa (las matrices referentes a la base de datos y el modelo de clasificación que se va a utilizar, que en este caso es KNN con $K = 1$). La imagen se preprocesa, y el resultado de este preprocesamiento se utiliza como entrada de la red. Esta red devuelve un vector, que es el que se usa como entrada después para el clasificador. Este clasificador devuelve el identificador del individuo, y este se usa para extraer el resto de fotos de la base de datos.

El código de esta aplicación se encuentra en el apéndice E.

3.8. Análisis de los vectores de características

Ya sabemos que los vectores de características discriminan bien los distintos individuos, pero a diferencia de otros métodos, al utilizar redes neuronales no sabemos lo que estas hacen por dentro. Es decir, podemos conseguir un resultado óptimo, pero con la desventaja de no conocer el porqué de ese resultado. Esto provoca que el método sea difícil de mejorar. Por ejemplo, ¿cómo podemos deducir que algoritmo de clasificación utilizar si no sabemos que representan los datos de los que disponemos?

Por eso, se ha intentado entender que representan estos vectores de características, ya que sabiendo esto, quizá se pueda idear una mejora en el método de clasificación.

3.8.1. Análisis de clusters

Para buscar que es lo que mejor distinguen los vectores, se ha utilizado Kmeans, un algoritmo de clustering. Este nos separará el conjunto de individuos en grupos, en este caso 2 grupos, y se analizará el resultado para ver que atributo comparten los individuos de un mismo cluster, y cuales son diferentes entre los dos conjuntos. Primero se ha realizado esto para las imágenes preprocesadas, sin utlizar PMLF.

La base de datos proporciona el género, raza, y edad de cada individuo, por lo que estos serán los elementos que compararemos:

La figura 3.10 muestra la distribución de edades de ambos clusters. En los gráficos se puede ver que no hay diferencias notables entre ambas distribuciones, y dado que ambos conjuntos tienen una edad media parecida, 38.95 en el primer cluster y 41.24 en el segundo, podemos concluir que la edad no es la característica que la red neuronal mejor discrimina.

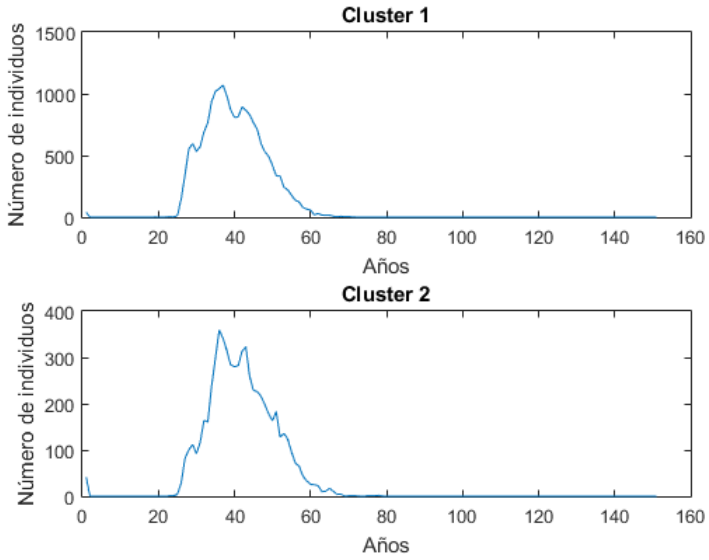

Figura 3.6: Distribución por edades al realizar clustering.

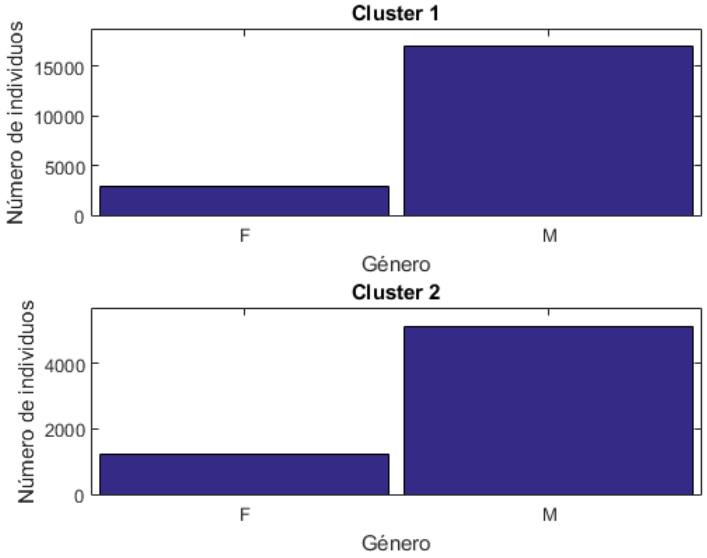

Figura 3.7: Número de hombres y mujeres en al realizar el clustering. F: Mujer, M: Hombre.

Con el género, ocurre de forma similar, tal como se ve en la figura 3.11. En este gráfico de barras tampoco se puede apreciar la diferencia entre ambos conjuntos.

Por último, las figuras 3.12 y 3.9 muestran los porcentajes de cada raza presentes en cada cluster. En este caso, si que se puede apreciar una clara diferencia entre ambos grupos.

Prácticamente la totalidad del primer cluster esta formada por individuos de raza negra, mientras que en el segundo predominan los individuos de raza blanca, aunque hay también un porcentaje de individuos de otras razas que no se puede despreciar. Esto indica que es muy probable que la red neuronal tome el color como una de las características más significativas de la imagen.

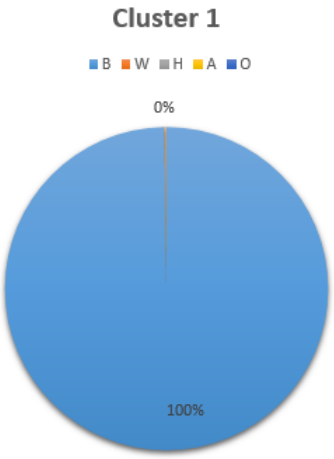

Figura 3.8: Distribución de razas en el primer cluster.

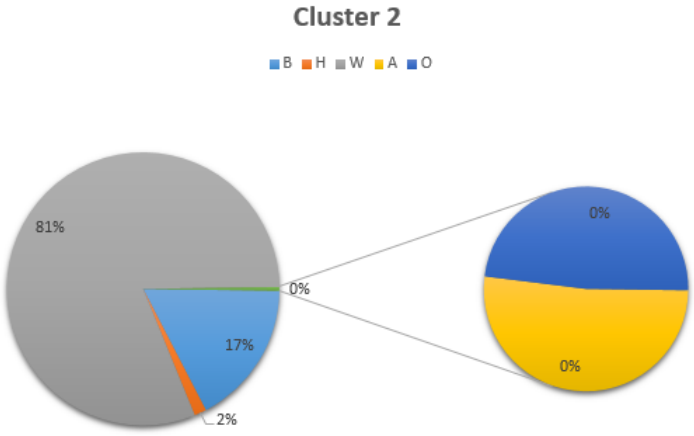

Figura 3.9: Distribución de razas en el segundo cluster.

A continuación, en las figuras 3.10, 3.11, 3.12 y 3.9, se muestra la misma información que en las cuatro figuras anteriores, pero esta vez utilizando PMLF de nivel 2.

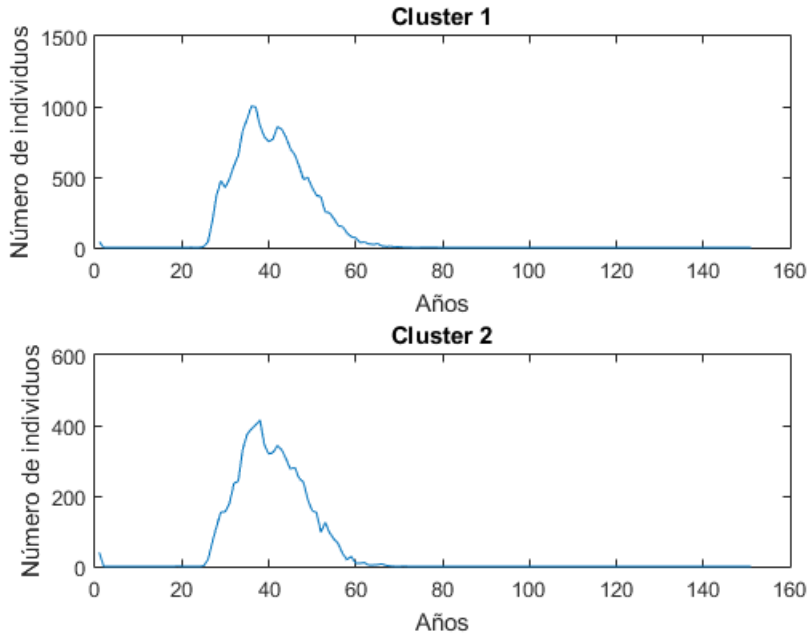

Figura 3.10: Distribución por edades al realizar clustering, con PMLF de nivel 2.

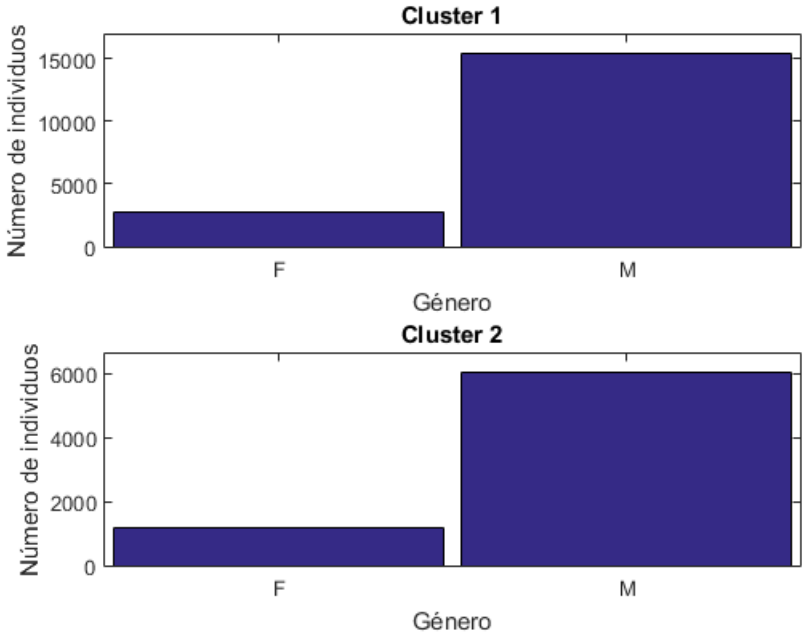

Figura 3.11: Número de hombres y mujeres en al realizar el clustering, con PMLF de nivel 2. F: Mujer, M: Hombre.

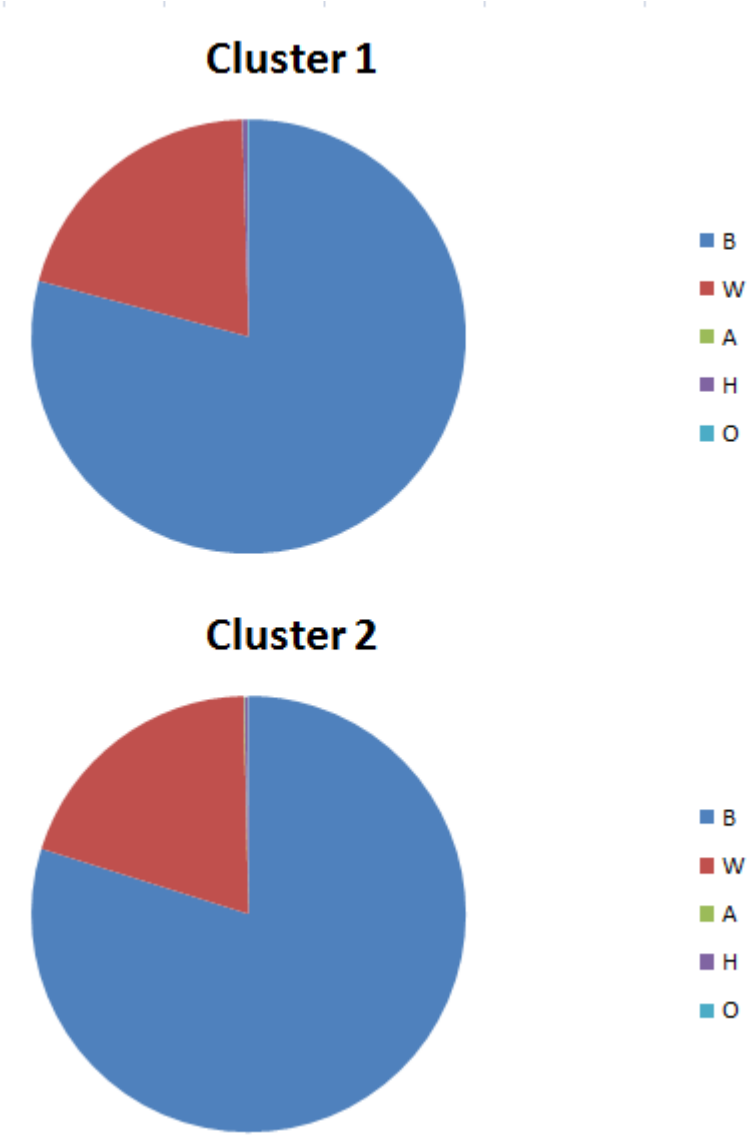

Figura 3.12: Distribución de razas, con PMLF de nivel 2.

En estas figuras no se puede observar ninguna diferencia entre ambos clusters, por lo que se refuerzan las conclusiones obtenidas en las pruebas de clasificación mediante KNN: combinar PMLF y VGG-Face no solo no mejora los resultados, sino que los empeora.

3.8.2. Análisis de clusters ideales

Se formarán los clusters a mano, de forma que cada cluster este formado por las imágenes de un solo individuo. Por eso nos referimos a estos clusters como ideales, porque son las

agrupaciones que se harían en caso de que los vectores discriminaran a la perfección. Es decir, una clasificación en la que las distancias entre los elementos intraclase son menores que las distancias a otras clases.

Primero, se ha tomado una muestra reducida del total de imágenes disponibles en la base de datos, y se han formado los clusters, de forma que cada cluster esta formado por todas las fotos disponibles de un mismo individuo.

Después, se han calculado las distancias máximas y mínimas entre individuos intragrupo (dentro de un mismo cluster) y extragrupo (entre dos clusters). La distancia mínima entre dos clusters es la distancia mínima existente entre dos de sus elementos. La distancia máxima entre dos clusters es la distancia máxima existente entre dos de sus elementos.

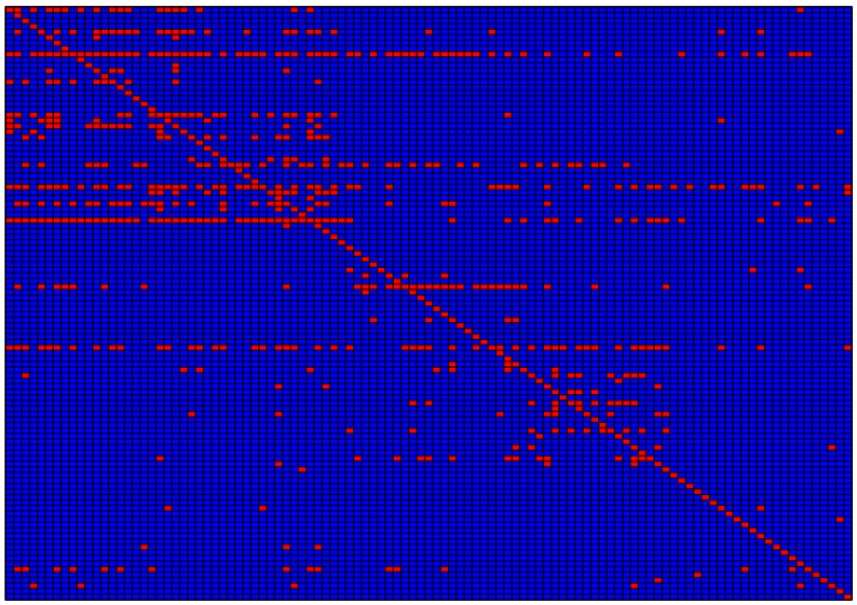

Figura 3.13: Distancias entre el máximo de un cluster y el mínimo del resto de clusters.

La figura 3.13 muestra las diferencia entre la distancia máxima intragrupo y la distancia mínima extragrupo. La casilla (i,j) es azul si la distancia mínima del grupo i al j es mayor que la distancia máxima dentro del grupo i y será roja en otro caso. En la imagen se puede ver claramente como predomina el color azul, y por lo tanto se puede decir que los clusters, en general, están bien diferenciados unos de otros.

Además, los clusters se ordenaron por razas, y en la figura también se pueden distinguir

dos zonas rectangulares alrededor de la diagonal donde la concentración de rojo es más alta. Estos rectángulos corresponden a la raza B (la de más arriba a la izquierda) y a la raza W.

En la siguiente página se puede ver una imagen que muestra las distancias entre individuos de la misma muestra reducida que se ha utilizado para el análisis de distancias entre clusters. Las distancias han sido normalizadas entre los valores 0 y 1. En esta imagen se puede ver como los individuos forman clusters diferenciables en la diagonal, donde las distancias son mas pequeñas. Además, fuera de esta diagonal predomina el color amarillo, que representa una distancia mas grande que el color rojo. Así, se puede concluir que la distancia entre individuos diferentes es, por lo general, alta.

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

3.8.3. Escalamiento multidimensional: representación en dos dimensiones

El escalamiento multidimensional tiene como finalidad crear una representación gráfica (mapa perceptual) que permita conocer la situación de los individuos en un conjunto de objetos por posicionamiento de cada uno en relación a los demás. [Pérez, 2004]

Con el mismo objetivo de entender los datos de salida de la red neuronal, se han intentado visualizar los individuos en un gráfico que represente el grado de similaridad de las imágenes. Para ello, primero se ha escogido un subconjunto de imágenes de la base de datos, intentando que haya representación de diferentes características. En la tabla 3.13 se puede ver el número de imágenes de la muestra según raza y género.

	B	**W**	**A**	**H**	**O**	**Tot.**
Mujeres	80	80	0	15	0	165
Hombres	80	80	13	46	11	230
Total	160	160	14	51	11	395

Tabla 3.13: Número de imágenes por género y raza.

Para poder visualizar la similaridad de las imágenes, se va a utilizar la distancia entre sus vectores de características. Por ello, primero se calcula la distancia entre todos los vectores, formando así una matriz de distancias. Esta se utiliza entonces como matriz de disimilaridades, para utilizarla como entrada de una función de escalado multidimensional. Esta función toma las distancias entre los individuos, y forma un sistema de dos dimensiones donde las distancias son lo más cercanas posibles a las reales. De esta manera, hemos reducido un sistema con vectores de 4096 variables a uno de 2, en los que las distancias son similares.

Este nuevo sistema no es totalmente fiel al original sin embargo. La fidelidad de este nuevo sistema se ve reflejada en el estrés de la misma (un valor entre 0 y 1). Cuanto más alto es el estrés, peor es el sistema resultante. En el caso que nos ocupa, el estrés es 0,3864. En las figuras 3.14 y 3.15 se puede ver la distancia entre individuos.

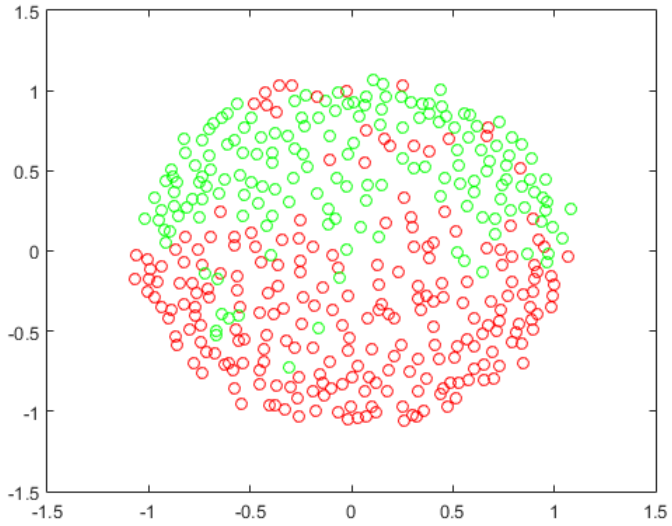

Figura 3.14: Distancias entre individuos resaltando razas.

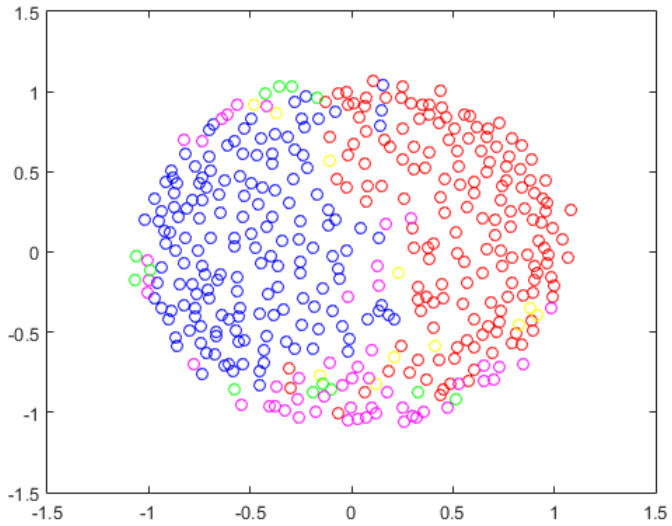

Figura 3.15: Distancias entre individuos resaltando el género.

Se puede ver cómo, pese a haber forzado vectores de 4096 variables a solo 2 variables, con la distorsión que ello conlleva a la hora de visualizar los datos, la distinción que se hace entre género y entre razas (sobre todo entre tonos oscuros y tonos claros de piel) es bastante buena. Esto confirma que los vectores resultantes de la red tienen una alta

probabilidad de acierto si se pretende distinguir un individuo de los demás por su género o color de piel.

Esto es muy interesante, porque quizá signifique que la escala se puede reducir considerablemente frente a ciertos problemas. Quizá no sea un método eficiente para distinguir individuos, pero como hemos visto, sí distingue bien entre razas y género. Esto supondría un tiempo inicial mayor, ya que habría que calcular las distancias entre vectores de características, que supone un tiempo considerable, pero después el entrenamiento del modelo de clasificación y su uso se vuelven mucho mas viables en cuanto a tiempo.

3.9. Problemas

Durante el proceso de desarrollo del proyecto han sucedido varios problemas. El primero de ellos ocurrió al tratar de preprocesar las imágenes. Se planeó escribir una función que realizara esta tarea, pero a la hora de intentarlo, no fue posible. El método que se había planteado inicialmente (utilizando el algoritmo Viola-Jones) no se consiguió llevar a cabo correctamente. Tras esto, se investigó acerca de otros métodos para conseguirlo. Al final, tal como se explica en el apartado 3.5.1 hubo que utilizar una función ya escrita. Esto retrasó el proyecto 2 semanas.

A la hora de extraer los vectores de características se interpretó de forma equivocada uno de los datos que proporcionaba la base de datos (en concreto, todos los vectores de características se extrajeron asignándoles un individuo erróneo), por lo que hubo que repetir todo el proceso. Se ha conseguido no retrasar el proyecto invirtiendo más tiempo diario del planificado.

Por último, el problema más grande con el que me he encontrado ha sido a la hora de estimar los tiempos de ejecución. Tanto a la hora de preprocesar, como de extraer los vectores de características, como de entrenar y probar los clasificadores el tiempo estimado al principio del proyecto fue mucho menor del que ha sido realmente. Esto ha provocado en algunos casos que haya que esperar días mientras el programa se ejecutaba sin poder avanzar demasiado con el proyecto ya que no se disponía de los datos necesarios para ello. Al final, se consiguió un ordenador adicional en el que ejecutar los programas que mas tiempo llevarían para poder trabajar en otras cuestiones mientras tanto, pero no se pudo evitar perder más tiempo del planeado.

Este último problema, el tiempo de ejecución, también ha provocado que no se haya podido probar el método PMLF para una pirámide de nivel 3. Debido a esto no se ha

probado con este más que la clasificación. Además los vectores de este nivel ocupaban un espacio en memoria que a menudo Matlab no era capaz de manejar.

4. CAPÍTULO

Conclusiones

4.1. Resultados del proyecto

Los resultados obtenidos en todas las pruebas anteriores han sido bastante esclarecedores. La red neuronal VGG-Face obtiene unos resultados muy buenos en cuanto al reconocimiento de individuos por sí sola, sin necesidad de utilizar métodos adicionales. Por desgracia, el método PMLF ha resultado no ser compatible con esta red, pero seria interesante probar este mismo método con otra red que extraiga características de imágenes en general, es decir, una red que no este entrenada para tratar con caras, sino con cualquier tipo de imagen. Esto quizá obtenga mejores resultados.

El análisis de clusters ha demostrado que la red funciona especialmente bien a la hora de separar las imágenes por su color. Por otra parte, el análisis de clusters ideales muestra que los vectores de características discriminan la información suficientemente bien para formar clusters de individuos distintos. Esto corrobora los buenos resultados del algoritmo k-nn y la aplicabilidad del mismo para recuperar las fotos de una persona a partir de otra que demos como entrada.

El escalamiento multidimensional, además de reforzar que los colores (y por consiguiente la raza negra de las demás) y los géneros son fácilmente clasificables por la red, muestra que es posible reducir la escala del problema a un número de variables mucho menor. Sería interesante seguir investigando acerca de esto, porque quizá se puedan simplificar algunos problemas que sin hacerlo serían demasiado costosos de solucionar.

4.2. Conclusiones personales

Tras realizar este proyecto, me queda más claro el coste que puede llegar a suponer trabajar con redes neuronales es alto, y su complejidad es también alta. He tratado específicamente con redes convolutivas, pero hay mas tipos de redes que pueden ser estudiadas, por lo que el campo de las redes neuronales es muy amplio y tiene muchas posibles aplicaciones.

La capacidad computacional de los ordenadores sigue siendo una limitación hoy en día, por lo que merece la pena investigar acerca de como acelerar los procesos que sabemos que son posibles, pero que por motivos de tiempo no podemos llevar a cabo. Gracias al proyecto ha aumentado mi interés en cuanto a la inteligencia artificial y todo lo que la rodea.

Alejándome del ámbito de la informática, he podido comprobar lo que supone llevar a cabo un proyecto de esta envergadura. Me he equivocado en muchas cosas a la hora de planificar, y sobre todo a la hora de llevar a cabo lo planificado. Aún así, estoy contento del trabajo que he hecho y de haber terminado a tiempo pese a los contratiempos.

Anexos

A. ANEXO

Código de la función de preprocesamiento con escalado aleatorio

```
function [ colorFace , res , width , height ] = Preprocess
    ( originalImage , lvl )
%   Rota y recorta la imagen segun la posición de los ojos
    .
%   La imagen de salida está en formato RGB.También
    devuelve
%   el alto y el ancho de la imagen La variable res tendrá
%   valor 1 si todo ha funcionado correctamente, 0 en caso
%   contrario. Si res = 0, los demas valores de salida ser
    án 0.

    try
    width = 0; height = 0; res = 0; colorFace = 0;
    orimagegray = rgb2gray(originalImage); %Imagen a
    escala de grises
    %Deteccion y recorte de la cara
    sc = 0;
    auxFace = originalImage;
    scale = 0.25;
```

```
15      while sc < 6
16          faceDetector = vision.CascadeObjectDetector;
17          bbox_face = step(faceDetector,orimagegray);
18          if length(bbox_face) < 4
19              faceDetector = vision.CascadeObjectDetector('
    FrontalFaceCART');
20              bbox_face = step(faceDetector,orimagegray);
21          end;
22          if length(bbox_face) == 4
23              sc = 6;
24          else                                                %
    Si no ha encontrado una cara, reescala la imagen
25              auxFace = imresize(originalImage,scale);    %
    hasta 6 veces
26              sc = sc + 1;
27              scale = scale + 0.25;
28              if scale == 1
29                  scale = scale + 0.25;
30              end;
31          end;
32      end;
33      if length(bbox_face) < 4                            %
    Si no se ha conseguido encontrar una cara aun
    reescalando
34          return;                                         %
    la función termina
35      end;
36      face = imcrop(orimagegray, bbox_face);              %
    Recorte de la imagen en escala de grises
37      colorFace = imcrop(originalImage, bbox_face);       %
    Mismo recorte, de la imagen a color
38
39      %Detección de los ojos y rotación de la imagen
40      sc = 0;
```

```
41      auxFace = face;
42      scale = 0.25;
43      while sc < 6
44          [leftEye_X, leftEye_Y, rightEye_X, rightEye_Y,
    nose_X, nose_Y, res] = detectEyesAndNose2(auxFace);
45          if res == 1
46              sc = 6;
47          else
48              auxFace = imresize(face,scale);
49                  sc = sc + 1;
50                  scale = scale + 0.25;
51              if scale == 1
52                  scale = scale + 0.25;
53              end;
54          end;
55      end;
56      if res > 0
57          angle = atan2(rightEye_Y-leftEye_Y,rightEye_X-
    leftEye_X);
58          B = imrotate(face,(-angle));
59          colorFace = imrotate(colorFace,(-angle));
60          [leftEye_X, leftEye_Y, rightEye_X, rightEye_Y,
    nose_X, nose_Y, res] = detectEyesAndNose(B);
61          if res > 0
62              angle = atan2(rightEye_Y-leftEye_Y,rightEye_X-
    leftEye_X);
63              B = imrotate(B,(-angle));
64              colorFace = imrotate(colorFace,(-angle));
65              [leftEye_X, leftEye_Y, rightEye_X, rightEye_Y,
     nose_X, nose_Y, res] = detectEyesAndNose(B);
66              if res > 0
67                  %Recorte de la imagen segun distancia
    entre los ojos
68                  L = leftEye_X - rightEye_X;
```

```
69                  top = uint16(L * 0.75);
70                  side = uint16(L * 0.5);
71                  bottom = uint16(L * 1.3);
72                  xmin = rightEye_X - side;
73                  ymin = rightEye_Y - top;
74                  width = side + L + side;
75                  height = top + bottom;
76                  m = mod(width,lvl);
77                  width = width + (lvl - m);
78                  m = mod(height,lvl);
79                  height = height + (lvl - m);
80                  colorFace = imcrop(colorFace, [xmin ymin
    width height]);
81              end
82          end
83      end
84      catch
85          width = 0;
86          height = 0;
87          res = 0;
88          colorFace = 0;
89      end
90  end
```

B. ANEXO

Código de la función de preprocesamiento evitando el escalado

```
function [registered2, res] = preprocessNoScale( img )
res = 0;
registered2 = 0;
try
    sc = 0;
    I = img;
    scale = 0.25;
    while sc < 6
        [LEX, LEY, REX, REY, NX, NY] = detectEyesAndNose(
   I );
        if LEX == 0
            I = imresize(img,scale);
            sc = sc + 1;
            scale = scale + 0.25;
            if scale == 1
                scale = scale + 0.25;
            end;
        else
            sc = 6;
```

```
19          end;
20      end;
21      if sc == 6 && scale == 0.25
22          scale = 1;
23      end;
24      L_Eye_X = LEX / scale;
25      L_Eye_Y = LEY / scale;
26      R_Eye_X = REX / scale;
27      R_Eye_Y = REY / scale;
28      Nose_X = NX / scale;
29      Nose_Y = NY / scale;
30
31      if L_Eye_X ~= 0
32
33          k1 = 0.5;
34          k2 = 0.75;
35          k3 = 1.5;
36          finalWidth = 224;
37          finalHeight = 224;
38          l = finalWidth / (2*k1+1);
39
40          fixed = uint8( zeros(finalWidth, finalHeight, 3) )
    ;
41          fixed_points = [k1* l, k2*l; (k1*l+l),k2*l;
    finalWidth/2, finalHeight/2];
42          Rfixed = imref2d( size( fixed ) );
43
44          moving_points = [R_Eye_X, R_Eye_Y;...
45              L_Eye_X, L_Eye_Y;...
46              Nose_X, Nose_Y];
47
48          mytform = fitgeotrans(moving_points, fixed_points,
     'similarity');
49
```

```
50          registered2 = imwarp( img, mytform,'FillValues',
    255,'OutputView',Rfixed);
51
52          res = 1;
53
54          %Eyes = [Database(kk).R_Eye_X Database(kk).L_Eye_X
    ;Database(kk).R_Eye_Y Database(kk).L_Eye_Y];
55          %faceImg(:,:,1) = alignement_crop(img(:,:,1), [0.5
     1 1.75],[224 224],Eyes);
56          %faceImg(:,:,2) = alignement_crop(img(:,:,2), [0.5
     1 1.75],[224 224],Eyes);
57          %faceImg(:,:,3) = alignement_crop(img(:,:,3), [0.5
     1 1.75],[224 224],Eyes);
58
59          %face = alignement(img,[0.5 1 1.75],[200 200],Eyes
    );
60          %imwrite(faceImg, [ './crop/' faces{kk}.filename])
    ;
61      else
62          disp([ 'Unable to detect eyes and nose on image']
    );
63      end
64  end;
65  end
```

C. ANEXO

Función de extracción del vector para una imagen

```
function trfeature = nn( net , im )
%   Extrae el vector de caracteristicas de la imagen

im_ = imresize(single(im), net.meta.normalization.
   imageSize(1:2)) ;

for i = 1:3
im_(:,:,i) = im_(:,:,i) - net.meta.normalization.
   averageImage(1,1,i) ;
end;

res = vl_simplenn(net, im_) ;

% extract 4K features
feat4K = reshape( res(end-2).x, 4096, 1);
% apply L2 norm to features
features = feat4K / norm( feat4K, 2 );
% Convert into column
trfeature = features';
end
```

D. ANEXO

Código del script de PMLF y juntar vectores en matriz

```
cd ../matconvnet-1.0-beta12
run matlab/vl_setupnn
net = load('vgg-face.mat') ; %Cargar la red de neuronas
cd ../Matlab
[num,txt,raw] = xlsread('morph_2008_nonCommercial.csv'); %
    55134 elementos


er = 0;
lvl = 2;
featSize = 0;
for i = 1:lvl
    featSize = featSize + (i^2);
end
featSize = (featSize * 4096) + 1;
featuresMatrix = zeros(10000,featSize); %Inicializa la
    matriz que contendrá las características
for i = 2:55135 %55135
     i
     try
         %Cargar la imagen
```

```
20          features = num(i-1,1);
21          imPath = char(txt(i,11));
22          imPath = strcat('AlbumPreprocess/', strrep(imPath,
    'Album2/',''));
23          face = imread(imPath);
24          %Calcular el tamaño de la imagen para que se
    ajuste al número
25          %de bloques  a recortar
26          [height, width, z] = size(face);
27          m = mod(width,lvl);
28          width = width - m;
29          m = mod(height,lvl);
30          height = height - m;
31          face = imresize(face, [height width]);
32
33          tic     %Empezar el contador de tiempo (por imagen
    )
34          blockSizeW = width/lvl;
35          blockSizeH = height/lvl;
36          w = width;
37          h = height;
38          for n = lvl:-1:1
39              xPositions = [1 blockSizeW:blockSizeW:w];
40              yPositions = [1 blockSizeH:blockSizeH:h];
41              for k = 1:n
42                  for j = 1:n
43                      %Recorte del bloque
44                      I = imcrop(face, [xPositions(j)
    yPositions(k) blockSizeW blockSizeH]);
45                      %Utilizar la red neuronal con el
    bloque como entrada
46                      f = nn( net, I );
47                      features = [features f];
48                  end
```

```
49              end;
50              w = w - blockSizeW;
51              h = h - blockSizeH;
52              if (w > 0) && (h > 0)
53                  face = imresize(face, [w,h]);
54              end;
55          end;
56          time(i-1) = toc; %Termina el contador de tiempo
57          featuresMatrix(i,:) = features; %Añadir el vector
    de características a la matriz
58      catch
59          er = er + 1;
60      end;
61  end;
62  meanTime = mean(time) %Calcular el tiempo medio de ejecuci
    ón
```

E. ANEXO

Código de la aplicación

```
function  id = recog( im )
disp('Cargando red neuronal')
cd ../matconvnet-1.0-beta12
run matlab/vl_setupnn
net = load('vgg-face.mat') ; %Cargar la red de neuronas
cd ../Matlab

disp('Cargando variables')
load appVars
f = imread(im);

disp('Preprocesando')
[ face, res, width, height] = Preprocess( f , 1 );
if res ~= 0
    disp('Ejecutando red')
    features = nn( net , face );

    disp('Clasificando')
    id = predict(Md1,features)
    for i = 2:25000
       if num(i-1,1) == id
```

```
            figure(i);
            imPath = char(txt(i,11));
            imshow(imPath);
        end
    end
else
    disp('No se ha podido preprocesar la imagen.')
end
end
```

Bibliografía

[DBW, 2008] (2008). MORPH Non-Commercial Release Whitepaper.

[MCN, 2017] (2017). MatConvNet. http://www.vlfeat.org/matconvnet/.

[MOR, 2017] (2017). MORPH Database (Academic Use Only). $https : //ebill.uncw.edu/C20231_{u}stores/web/product_{d}etail.jsp?PRODUCTID = 8$.

[KT, 2017] (2017). Tablero de Control Kanban Online para Empresas | Software de Gestión Visual de Proyectos | Kanban Tool. https://kanbantool.com/es/.

[Bekhouche et al., 2016] Bekhouche, S. E., Ouafi, A., Dornaika, F., Taleb-ahmed, A., and Hadid, A. (2016). Pyramid Multi-Level Features for Facial Demographic Estimation. pages 8–17.

[Belver, 2016] Belver, C. (2016). Comparative Study of Human Age Estimation Based on Hand-crafted and Deep Face Features.

[Kazemi and Sullivan, 2014] Kazemi, V. and Sullivan, J. (2014). One Millisecond Face Alignment with an Ensemble of Regression Trees.

[Lamport, 1986] Lamport, L. (1986). *LaTeX: User's Guide & Reference Manual*. Addison-Wesley.

[Lippmann, 2007] Lippmann, R. P. (2007). An Introduction to Computing with Neural Nets.

[N.V., 2017] N.V., T. (2017). Planificador de proyectos | Diagrama de Gantt Online | Tom's Planner. https://www.tomsplanner.es/.

[Parkhi et al., 2015] Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep Face Recognition.

[Pérez, 2004] Pérez, C. (2004). *Técnicas de Análisis Multivariante de Datos*. Pearson Educación.

[Roa, 2016] Roa, L. (2016). Analysis of facial expressions in children: Experiments based on the DB 2Child Affective Facial Expression.

[Sathyanarayana, 2014] Sathyanarayana, S. (2014). A Gentle Introduction to Backpropagation.

[Schmidhuber, 2014] Schmidhuber, J. (2014). Deep learning in neural networks: An overview. pages 86–100.

[Vedaldi and Lenc, 2015] Vedaldi, A. and Lenc, K. (2015). Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*.

[Wang, 2014] Wang, Y.-Q. (2014). An Analysis of the Viola-Jones Face Detection Algorithm.