



MEMORIA FINAL DEL ALUMNO

ALUMNO: Sebastián Meilán Bereciartúa			
NIF: 33556675 - C	TELEFONO: 605457025	E-MAIL: sebas.meilan.bereciartua@gmail.com	
TITULACIÓN: INGENIERÍA INFORMÁTICA			CURSO: 2020/21
FECHA DE INICIO/FIN REALIZACIÓN PRÁCTICAS: 1 ER CUATRIMESTRE 20/21			HORAS REALIZADAS: 150 - 200
ENTIDAD COLABORADORA: PROCONSI			LOCALIDAD: León
IDENTIFICACIÓN DE LAS APORTACIONES QUE, en materia de aprendizaje, han supuesto las prácticas:			
<p>Este proyecto se realizó en el lenguaje de programación Python, lenguaje que apenas se había desarrollado a lo largo de la carrera. Lenguaje que ahora domino de una manera más cómoda y fluida. Además, en cuanto a sistemas de inteligencia artificial y algoritmos para el reconocimiento facial mis conocimientos no se extendían más allá de lo que la investigación y la autoformación que había llevado a cabo.</p>			
VALORACIÓN PERSONAL: (Indicad vuestra valoración personal, puntos a favor y en contra. Este informe es confidencial, por lo que no dudéis en poner todo lo que queráis...)			
<p>Considero que las prácticas han sido un punto motivador para mi carrera profesional. Al ser un tema que no se había desarrollado tanto en los estudios, tema que ahora conozco y comprendo con más facilidad. Creo que he trabajado de una manera apropiada ya que le he dedicado bastante tiempo, posiblemente debido al interés que me generó en cuanto empecé a comprender más la materia.</p> <p>Por esto puedo decir que, aunque al principio la labor de investigación primero que tuvimos que realizar para poder comprender bien el proyecto fue más difícil y pesado, en cuanto empecé a poder trabajar de manera práctica y ver mis propios avances mejoró mi manera de hacerlo junto con los resultados que obtenía.</p>			
La Descripción detallada del trabajo (objetivos, tareas desarrolladas, así como la relación entre los problemas planteados y el procedimiento seguido para su resolución) se incluyen en la memoria del trabajo que se adjunta a continuación.			

León, a 23 de enero de 2021

Fdo.:

(Sebastián Meilán Bereciartúa)

Sebastián Meilán Bereciartúa



Escuela de Ingenierías Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICAS EXTERNAS

SISTEMAS DE VISIÓN ARTIFICIAL IDENTIFICACIÓN FACIAL

Autor: Sebastián Meilán Bereciartúa

(Enero, 2021)

Sebastián Meilán Bereciartúa

UNIVERSIDAD DE LEÓN
Escuela de Ingenierías Industrial, Informática y
Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

RESUMEN:

Este proyecto se desarrolló de manera online mayoritariamente debido a la situación que vivimos actualmente. Esto no influyó en ningún momento en que la comunicación con nuestros tutores fuese fluida y se realizase de manera cómoda. Dejando ya esto de lado, podríamos dividirlo en semanas donde las primeras se basaron principalmente en labores de investigación y comprensión de distintos algoritmos de identificación y reconocimiento facial, pasando a unas semanas intermedias en las que ya pude empezar a trabajar de manera práctica mostrando primero unos resultados funcionales, pero no del todo eficientes utilizando la librería Face_Recognition en Python. Por último, durante las últimas semanas se basaron en optimizar un nuevo sistema, más completo y que utiliza la librería OpenCV como base, para mejorar de esta manera su fiabilidad y su velocidad a la hora de producir resultados. La aplicación final consta de una interfaz gráfica simple, en la que el usuario de la misma podrá seleccionar tanto la imagen en la que quiere realizar el reconocimiento, como el directorio en el que se guardarán los resultados o el algoritmo que quiere utilizar. Todo el proceso se realiza sin que el usuario lo vea mostrándole por último la imagen con el resultado y, posteriormente, una tabla en la que se pueden ver los tiempos de la realización de todo el proceso. Durante la última semana mi labor principal fue el de retocar algún matiz final en el proyecto junto la realización de una presentación para constatar diferencias entre algoritmos y librerías utilizados.

Palabras clave: Investigación, algoritmos, reconocimiento, detección, OpenCV, Face_Recognition, Python, librerías.

Índice de contenidos

1. Introducción	6
1.1 Contexto del trabajo	6
1.2 Introducción a la temática	7
2. Objetivos	9
2.1 Objetivos de carácter personal	9
2.2 Objetivos técnicos	9
3. Planificación	10
3.1 Estructuración en reuniones	10
3.2 Tareas desarrolladas	11
4. Núcleo del trabajo	12
4.1 Desarrollo por fases	12
4.1.1 Fase de investigación	12
4.1.2 Desarrollo	12
4.1.3 Presentación y finalización del proyecto	17
5. Resultados	19
6. Conclusiones	20
6.1 Nivel personal	20
6.2 Nivel técnico	20
7. Bibliografía	21
8. Anexos	22
8.1 Dependencias de instalación	22
8.2 Librerías finales instaladas	22
8.3 Archivos precargados	23
8.4 Código fuente	24
8.4.1 Face_Recognition	25
8.4.2 OpenCV	32

Índice de figuras

Figura 1.1 Logo PROCONSI	6
Figura 1.2 Logo Microsoft Teams	6
Figura 1.3 Identificación Facial	7
Figura 1.4 Reconocimiento facial	8
Figura 3.1 Diagrama de Gantt	11
Figura 4.1 Imagen incluida en el dataset con su correspondiente nombre	14
Figura 4.2 Ejemplo de imagen de salida con el reconocimiento efectuado	15
Figura 4.3 Interfaz de usuario	16
Figura 4.4 Ejemplo en el análisis de tiempos	17
Figura 4.5 Diferencia en el tratamiento de datos	18
Figura 5.1 Resultado de ejecución	19
Figura 8.1: Librerías finales	23
Figura 8.2 Ejemplo de capa en el prototxt	24
Figura 8.3 t1.py	25
Figura 8.4 t2.py	26
Figura 8.5 t3.py	27
Figura 8.6 t4.py	28
Figura 8.7 t5.py	29
Figura 8.8 t6.py	30
Figura 8.9 t7.py	31
Figura 8.10 main.py	32
Figura 8.11 AnalyzeDatapool.py	33
Figura 8.12 Trainer.py	34
Figura 8.13 Recognize.py	35

1. Introducción

1.1 CONTEXTO DEL TRABAJO

Pese a la situación que se vive actualmente, la cual no permite realizar un contacto de manera habitual con la empresa, este se desarrolló de manera cómoda y fluida.



Figura 1.1 Logo PROCONSI

Todo el desarrollo del proyecto se ha realizado bajo un formato de supervisión y contacto constante aprovechando la aplicación **Microsoft Teams**, las cuales se realizaron todos los viernes. Durante todas las reuniones, ambos tutores estuvieron presentes en todo momento para observar los avances realizados y para realizar comentarios sobre lo presentado y dar consejos sobre los posibles avances a futuro. Además, ambos tutores me hicieron saber que si tenía dudas en cualquier momento podía consultarles cualquier duda



Figura 1.2 Logo Microsoft Teams

1.2 INTRODUCCIÓN A LA TEMÁTICA

La temática del proyecto consistió de un sistema de identificación y reconocimiento facial. El proyecto se realizó en el lenguaje Python, aprovechando el entorno de desarrollo *PyCharm Community*, y utilizando además una serie de librerías especializadas en este contexto, como son *Face_Recognition* y *OpenCV*. Además, se aplicaron diferentes algoritmos para el tratamiento de los datos obtenidos durante la comparativa, los dos principales y los que se expusieron son *linear* y *RBF*.

El primer sistema que se realizó realizaba una identificación, buscando caras en la imagen pasada.

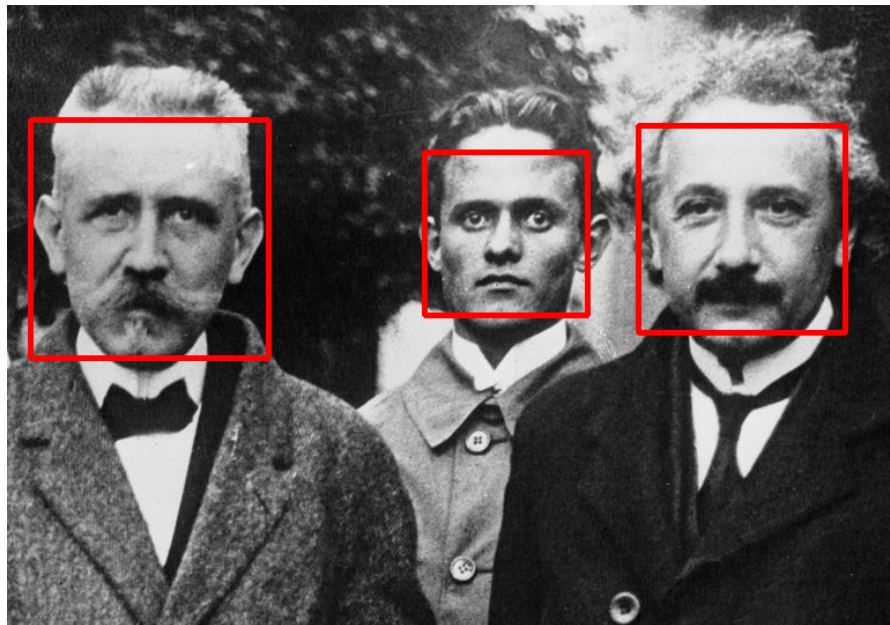


Figura 1.3 Identificación Facial

Viendo así una clara diferencia con el último sistema que genera un reconocimiento en base a las caras conocidas dentro de las que encuentra en la imagen. La diferencia base entre estos dos sistemas es la introducción de un cuadro de texto en el borde superior de la imagen, el cual identificará la cara en base a una lista de nombres.

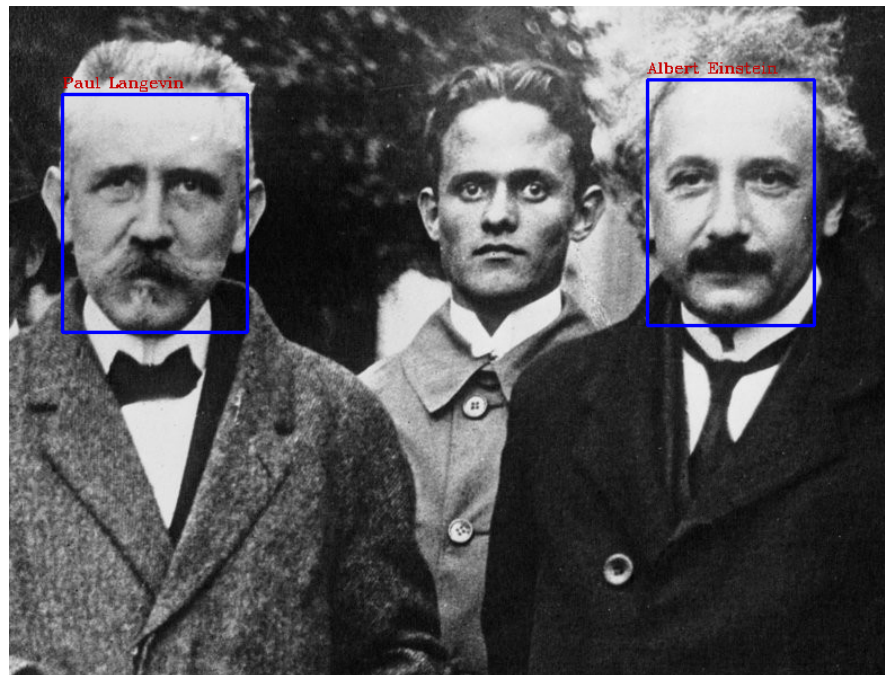


Figura 1.4 Reconocimiento facial

Durante el desarrollo de todo el proyecto se tratará mucho con temas relacionados con el aprendizaje automático en redes neuronales y diferentes métodos de entrenamiento de las mismas. De esta manera, se podrán identificar ciertas diferencias en la eficacia y eficiencia de cada uno de estos algoritmos.

2. Objetivos

El objetivo principal del proyecto es que fuese capaz de reconocer personas mediante la imagen de su cara. Además, el sistema debía estar pensado para que fuese sencillo cambiar el algoritmo de identificación o los parámetros del mismo para poder evaluar la eficiencia de las diferentes posibilidades. Podríamos diferenciar entre tipos de objetivos dentro del proyecto.

2.1 OBJETIVOS DE CARÁCTER PERSONAL

En cuanto al carácter personal, dentro del ámbito personal, uno de los objetivos principales era el de afianzar los conocimientos en cuanto al lenguaje informático *Python* y en cuanto al los diferentes algoritmos utilizables para el reconocimiento facial y la efectividad de cada uno de ellos.

Sumado a esto, podríamos considerar como objetivo personal el aprendizaje y la introducción en el mundo laboral, es decir, un contacto y una visión de como trabaja una empresa de este ámbito. Además, el hecho de tener que realizar avances constantes y ser capaz de mantener un ritmo de trabajo diario en un tema de este tipo.

2.2 OBJETIVOS TÉCNICOS

En el carácter técnico del proyecto, este debía ser capaz de realizar un reconocimiento facial de una imagen pasada en un tiempo aceptable (inferior a 1 décima de segundo).

Además, el propio programa, debía ofrecer posibilidades de personalización, por ejemplo, el usuario del sistema podría escoger la imagen de entrada mediante una interfaz gráfica simple y cómoda. Sumado a esto, y para poder desarrollar el programa de una manera eficiente, decidí implementar un sistema para poder elegir el algoritmo de entrenamiento del sistema.

3. Planificación

El proyecto se estructuró de manera semanal. Todos los viernes (a excepción de los festivos) se realizaba una reunión con David Jáñez y David García durante la cual por mi parte comentaba los problemas o dudas si existiesen junto con los avances realizados desde la última reunión. De esta manera dividió el proyecto en tres fases principales:

- ➔ Investigación (Primeras semanas).
- ➔ Desarrollo del programa y comparación de algoritmos (Tiempo intermedio).
- ➔ Presentación y entrega del proyecto.

3.1 ESTRUCTURACIÓN EN REUNIONES

Como ya he comentado las reuniones se realizaron semanalmente para facilitar el seguimiento del trabajo. La primera reunión se llevó a cabo el día 30 de octubre de 2020, en esta se nos puso al tanto de cómo se realizaría este seguimiento y además se nos dieron unas pautas a seguir junto con algunos objetivos a corto plazo.

Las dos primeras semanas consistieron en labor de investigación sobre posibles algoritmos a utilizar y su funcionamiento, comparar eficiencias al utilizarlos para reconocimiento facial o sobre librerías que podrían utilizarse para el desarrollo del proyecto.

A mediados del mes de noviembre esta labor de investigación se dio por concluida tomando algunos resultados como positivos y pude empezar a trabajar con los primeros bocetos de lo que sería el proyecto final.

Existió un pequeño parón en la realización de reuniones ya que tanto el día 25 de diciembre como el 1 de enero coincidieron con festivos nacionales, estas fechas coincidieron con los avances más significativos del proyecto debido a esta separación entre reuniones.

En el mes de enero el proyecto entró en su etapa final, que consistió en la preparación del mismo para ser presentado rematando así los últimos matices del mismo, así como el desarrollo de una pequeña interfaz que facilitaría su uso.

3.2 TAREAS DESARROLLADAS

Para marcar un claro desarrollo de las fases del proyecto decidí realizar un diagrama de Gantt. En la siguiente figura se puede ver claramente los cambios de etapa y las fases dentro de cada una ellas. Cada salto de etapa se vio definido por la realización de una reunión con ambos tutores presentes y la puesta de acuerdo tanto por su parte como por la mía. Se pueden ver claramente también las fechas de inicio del proyecto, coincidiendo con la primera reunión, y de entrega final marcada por la presentación del mismo.

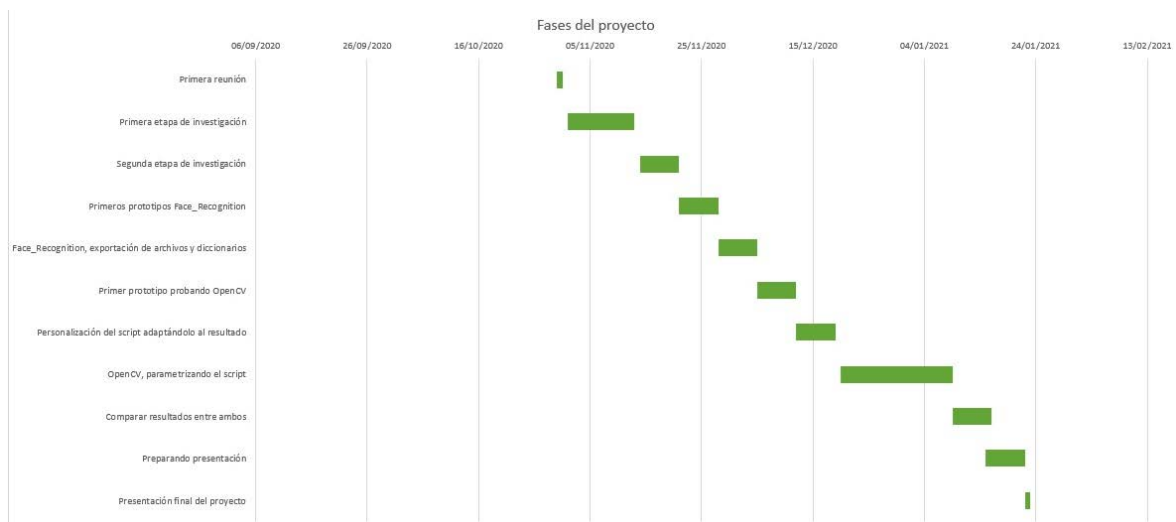


Figura 3.1 Diagrama de Gantt

4. Núcleo del trabajo

4.1 DESARROLLO POR FASES

Al comienzo del proyecto se planteó un problema principal, desarrollar un sistema capaz de reconocer caras dentro de una imagen en base a una base de datos local con caras previamente analizadas.

4.1.1 FASE DE INVESTIGACIÓN

Esta etapa consistió básicamente en la búsqueda y comprensión de distintos algoritmos y de su posible implementación en este proyecto, así como de diferentes librerías que implementasen dichos algoritmos. De esta manera, las librerías elegidas para el desarrollo fueron *Face_Recognition* y *OpenCV*, además los algoritmos que se compararán serán el *linear* y el *RBF*.

Destacar la diferencia entre identificación y reconocimiento facial ya que la primera solo reconoce la existencia de una cara en una imagen, pero no incluye la asociación de la propia con un nombre comparándola con un sistema de caras ya conocidas.

4.1.2 DESARROLLO

A) *FACE_RECOGNITION*

Antes de comentar el funcionamiento del sistema y diferentes detalles, considero que sería importante recalcar la estructura de trabajo que se siguió. Los diferentes test realizados con esta librería se fueron numerando de la forma ***##.py***.

Supuso el primer contacto con el reconocimiento facial. Es una librería simple, con métodos muy intuitivos y fáciles de utilizar.[1]

El funcionamiento es sencillo, primero se cargan las imágenes de caras conocidas, sobre las cuales se aplica un sistema de reconocimiento preestablecido para guardar una serie de características identificativas de las caras que encuentra en las imágenes proporcionadas, generando lo que se denomina *encoding* de cada una. Posteriormente, sobre la imagen en la que se quiera realizar el reconocimiento, se aplicará el mismo sistema para identificar las caras que existan en la misma y

se compararán los *encodings* de esta con los de la base de datos para ver si se puede estimar que una de las caras conocidas se encuentre en la imagen.

De esta manera podemos dividir los avances en esta librería de la siguiente manera:

- 1) A esta etapa se le asigna el desarrollo del **t1.py**, este script se constaba de un sistema de identificación facial, sin reconocimiento. Su funcionamiento es simple, ya que todos los parámetros de ejecución estaban introducidos de forma manual en el propio código, sin capacidad de personalización.
Se incluiría también el script **t2.py**, que ya contaba con un sistema de reconocimiento facial simple. Funciona de manera que tanto las imágenes del *dataset*, junto con los nombres correspondientes, como la imagen para realizar el reconocimiento se introducían como líneas de código de manera manual. Este script tampoco guardaba los resultados de su ejecución en ningún archivo externo.
- 2) En esta etapa empecé a trabajar con una subida masiva de archivos para formar el *dataset* del sistema. Incluiría los scripts **t3.py**, **t4.py** y **t5.py**. Los avances más destacables entre estos scripts se encuentran entre el 3 y 4, ya que, en este último, después de analizar el *dataset* se exportaba este resultado a un archivo externo para ahorrar tiempo al sistema; y, entre el 4 y el 5, la lista de nombres pasaba a generarse de manera automática junto con el análisis del *dataset*.
- 3) Esta etapa se basó en la personalización del sistema para poder ser aplicable a diferentes casos. De esta manera, en el script **t6.py** decidí dividir los distintos procesos existentes dentro del programa en diferentes métodos en vistas a posibles cambios o mejoras del mismo. Por último, y como script final utilizando esta librería, en el **t7.py**, implementé una parametrización de la última versión hasta el momento, donde el usuario introducía por línea de comandos los directorios con los que deseaba que el programa trabajase. Además, se solucionó un error en el método de reconocimiento a la hora de escribir el nombre correspondiente a cada cara reconocida (error que fue visible al aumentar el tamaño del *dataset*).

Ventajas: Fácil de entender y de usar, código simple.

Desventajas: Baja eficiencia (tiempos de espera elevados).

B) OPENCV

OpenCV es una librería mucho más compleja, supone una mayor dificultad a la hora de utilizarla, pero también un avance en el sistema de reconocimiento. [2]

Para este sistema implementé una nueva estructura de trabajo, que he seguido utilizando hasta el día de la entrega. Consta de 3 archivos principales junto con el *main.py* que es el que se encarga de lanzar la ejecución y mostrar por pantalla los resultados. Son los siguientes:

→ **AnalyzeDatapool.py.** Este *script* es el que se encarga de analizar la base de datos local de imágenes de caras generando sus denominados *embeddings*[3] (Funcionan de una manera similar a los *encodings* de *Face_Recognition*) los cuales guarda en un diccionario como valor relacionado a una clave que se corresponde con el nombre de la persona reconocida en esas imágenes. Para que cada cara se agregue a este diccionario debe superar un “umbral de confianza” establecido por mí en este caso, pero que se podría pasar como parámetro externo. Además, este resultado se guardará en un archivo de datos externo para no tener que volver a realizar este proceso cada vez que el programa se ejecute.

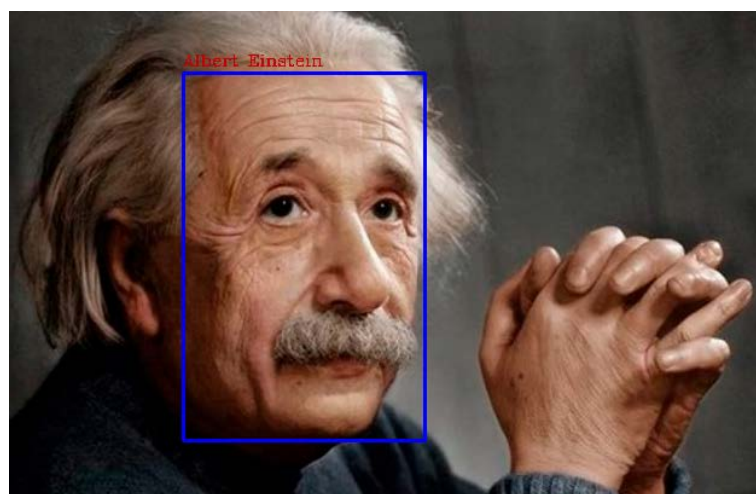


Figura 4.1 Imagen incluida en el dataset con su correspondiente nombre

- **Trainer.py:** Este *script* se encarga de entrenar el sistema de reconocimiento de caras. Para esto toma como entrada el archivo que se ha generado en el paso anterior junto con el *kernel* que seleccione el usuario (para este ejemplo he puesto o *linear*, *RBF* o *poly*)[4] y lo pasa por un SVC (*C-Support Vector Classification*) el cual actualizará el archivo en el cual, para el par cara – nombre que le hemos pasado generará un nuevo archivo, el cual contendrá la información correspondiente al sistema entrenado para el reconocimiento facial. Además, para los nombres he implementado un sistema *LabelEncoder* que generará una serie de clases correspondiente con cada uno de los nombres eliminando los repetidos entre ellos. Estos dos archivos se utilizarán en el último paso.[5]
- **Recognize.py:** Tanto para este *script* como para el primero se precargan dos archivos previos que son los que se utilizan para el reconocimiento facial (se mencionan más detalladamente en los anexos). Vuelve a generarse un reconocimiento sobre la imagen principal (es mucho más rápido del primero ya que es solo 1) y, después de comparar las caras que detecte con las que tiene almacenadas de los pasos previos devuelve unas estimaciones. Después dibujará un recuadro con el nombre correspondiente a la estimación más alta en la imagen.



Figura 4.2 Ejemplo de imagen de salida con el reconocimiento efectuado

Ventajas: Eficiencia y efectividad.

Desventajas: Algo más difícil de comprender, no demasiado.

C) INTERFAZ Y SUBIDA DE ARCHIVOS

Durante las últimas semanas, y para que el programa fuese más atractivo y cómodo de utilizar. Esto se centró en una sola clase, la clase **main.py**. Se centra en una ventana principal en la que el usuario tiene la opción de seleccionar tanto la carpeta de salida, como la carpeta en la que se encuentra la base de datos local y el algoritmo que se desea utilizar para entrenar el sistema. Además, se le pedirá, de manera obligatoria que elija una imagen en la que quiera realizar el reconocimiento. Posteriormente, se realizará todo el proceso comentado anteriormente y se mostrará la imagen en su tamaño original. Una vez el usuario cierre esta ventana con la imagen, se le mostrará una nueva ventana en la que podrá ver una tabla con los tiempos que ha tardado el programa en realizar cada uno de los pasos anteriores.[6]

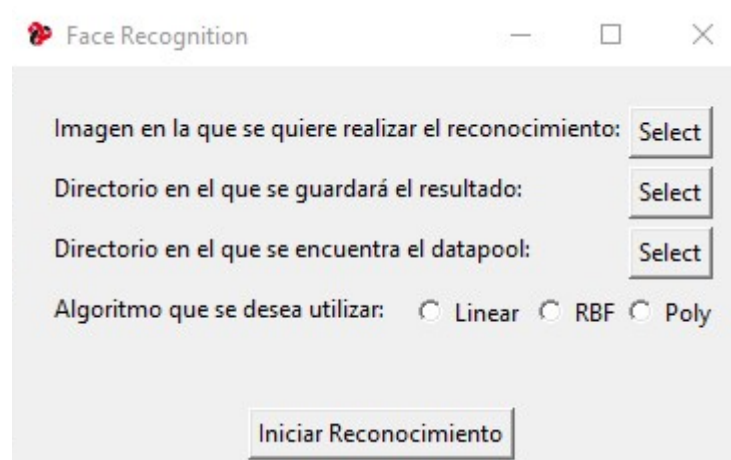
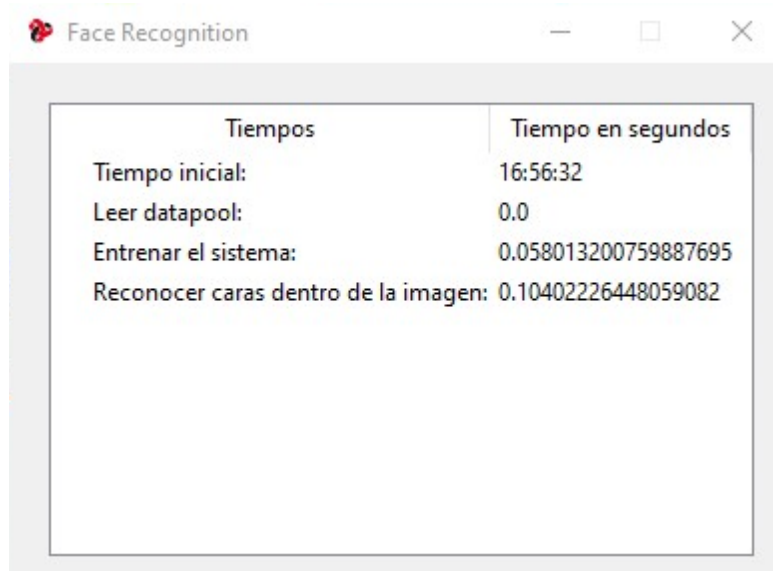


Figura 4.3 Interfaz de usuario

D) ANÁLISIS DE TIEMPOS

Para el análisis de tiempos, utilicé la librería *time*, que analiza los tiempos en segundos que han pasado desde el instante que se guarda hasta el tiempo *epoch* (1 de enero de 1970 a las 00:00:00 UTC). Guardando el punto justo antes de comenzar un proceso (como puede ser cuando se comienza a leer el *dataset*) y restándoselo al tiempo cuando ese proceso termina tenemos lo que ha tardado y es lo que se muestra en la tabla final.

Los tiempos guardados son: Instante en el que comienza la ejecución; tiempo que tarda el programa en analizar el *dataset*, tiempo que tarda en entrenarse y en reconocer las caras en la imagen.



Tiempos	Tiempo en segundos
Tiempo inicial:	16:56:32
Leer datapool:	0.0
Entrenar el sistema:	0.058013200759887695
Reconocer caras dentro de la imagen:	0.10402226448059082

Figura 4.4 Ejemplo en el análisis de tiempos

4.1.3 PRESENTACIÓN Y FINALIZACIÓN DEL PROYECTO

A) COMPARATIVA ENTRE ALGORITMOS

Comparando diferentes algoritmos podemos ver que la diferencia de tiempo de entrenamiento entre ellos es mínima. La diferencia destacable entre ellos es su eficacia a la hora de ofrecer resultados correctos. Para estudiar esto de una manera detallada se ejecutaron por separado cada uno de los algoritmos cien veces seguidas y se calculó un porcentaje de aciertos para los mismos.[7] Los resultados obtenidos fueron:

- ➔ Para el algoritmo de carácter lineal la tasa de acierto fue del 84%
- ➔ Para el algoritmo RBF la tasa de acierto fue del 90%
- ➔ Para el algoritmo polinomial la tasa de acierto fue del 87%

Estos resultados denotan algo similar a lo que pude recopilar durante la fase de investigación, el algoritmo lineal no es el más apropiado para el tratamiento de este tipo de datos que se reparten de una manera más radial en la gráfica y, por lo tanto, no es capaz de separarlos correctamente entre sus clases. Mientras, los algoritmos RBF y polinomial son capaces de reconocer formas más irregulares en la distribución de los datos.

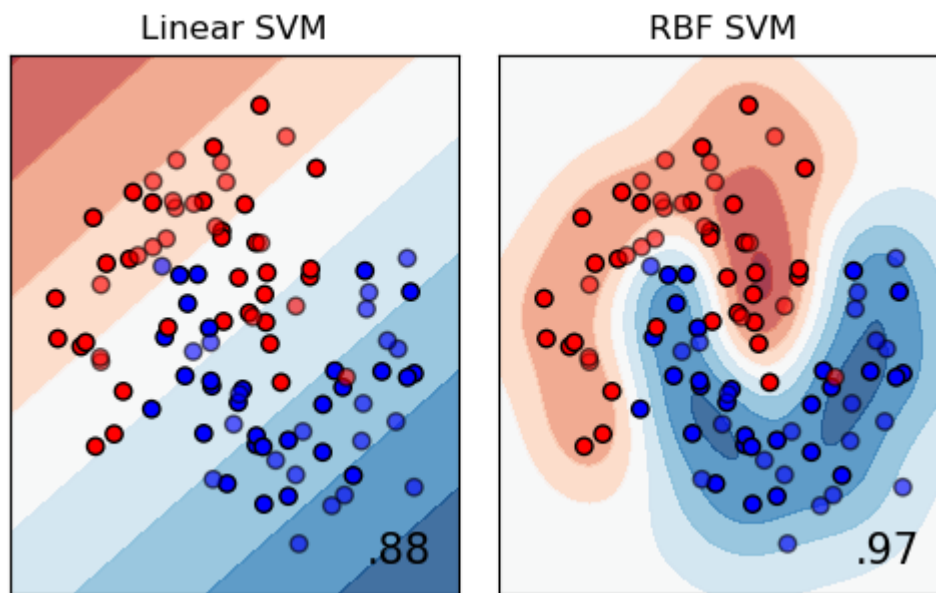


Figura 4.5 Diferencia en el tratamiento de datos

5. Resultados

Los resultados del proyecto los dividiremos también en las diferentes etapas que existen dentro de su ejecución.

- ➔ De esta manera, la etapa de análisis del *dataset* generará una serie de datos denominados *embeddings* que se exportarán a un archivo llamado "*embeddings.pickle*", junto con una lista correspondiente a los nombres. Esto se guardará en el formato de un diccionario de Python para asignar que imágenes se corresponden a cada uno de los nombres existentes.
- ➔ Durante el entrenamiento también se generarán 2 archivos externos, uno incluye el resultado del sistema entrenado, llamado "*recognizer.pickle*", y el otro incluiría un *labelEncoder*, sistema que se utiliza para ordenar listas y eliminar valores duplicados, este archivo se llamará "*le.pickle*".
- ➔ La última etapa del proyecto se encarga de, en base a los archivos generados en los pasos previos, realizar un reconocimiento facial en la imagen seleccionada por el usuario. Este paso solo generará como salida la propia imagen en la que se ha realizado el reconocimiento resaltando las caras que ha identificado y reconocido con un rectángulo y con el nombre escrito en la parte superior del mismo. De esta manera, las caras que no ha logrado reconocer no se marcarán en la imagen.

Por lo tanto, el resultado sería algo similar a la siguiente figura:

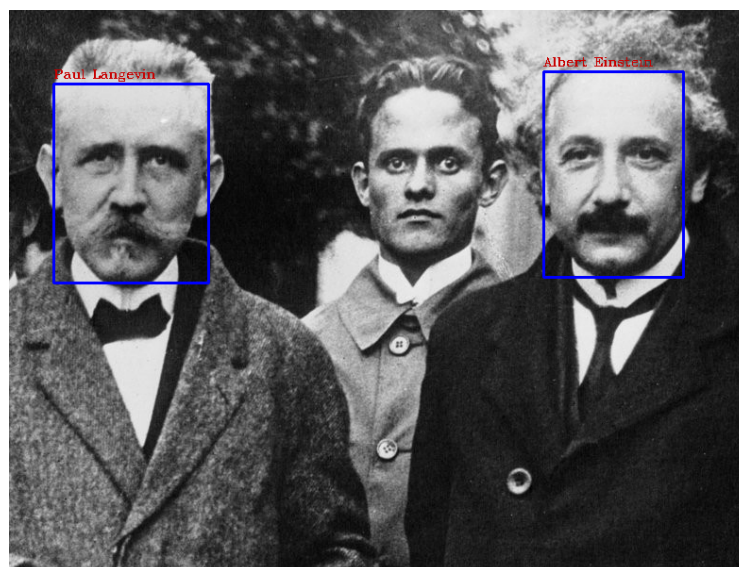


Figura 5.1 Resultado de ejecución

6. Conclusiones

Las conclusiones, al igual que los objetivos, también las podemos dividir en 2 apartados.

6.1 NIVEL PERSONAL

A nivel personal, antes de comenzar este proyecto, la Inteligencia Artificial y su desarrollo no suponía para mí más que un interés personal como un tema del que poder tener conocimientos, pero no como para dedicarme a nivel profesional o trabajar en proyectos en un futuro. Trabajar en él me ha demostrado que todas las ramas de la informática son interesantes, y que no hay que cerrar puertas a los proyectos en los que se pueden trabajar.

Además, en cuanto a la organización del trabajo, es importante destacar la diferencia que existió entre el trabajo de las prácticas y el modelo de trabajo que se acostumbra a nivel estudiante. En ningún momento resulta pesado el trabajar diariamente debido al interés que genera el proyecto junto con la libertad con la que se trabaja, al menos en mi casa, que los tutores me permitieron organizarme de la manera más cómoda para mí, pero manteniendo una presión constante al tener que presentar avances semanales.

6.2 NIVEL TÉCNICO

A nivel técnico, destacar que hasta ahora el trabajo con el lenguaje de programación Python durante la carrera había sido escaso, por lo tanto, mi comodidad y fluidez con el mismo han mejorado. Este proyecto supuso una labor de investigación, previo al comentado durante el proyecto, donde me autoformé en el propio lenguaje. [8]

Sumado a esto, el tema del proyecto, relacionado con el desarrollo de una inteligencia artificial, entrenarla, y aprovecharla para realizar, en este caso, un reconocimiento facial en una imagen, al igual que el uso del lenguaje, es un área en la que no se había trabajado demasiado a nivel estudios, más allá de una asignatura de segundo año (Introducción a los Sistemas Inteligentes).

El conjunto de estos dos puntos provocó un reto y un objetivo a nivel técnico, lo cual fomentó también el desarrollo del propio proyecto.

7. Bibliografía

- [1] A. Geitgey, “Face_Recognition API,” *ReadTheDocs*, 2017. <https://face-recognition.readthedocs.io/en/latest/index.html#> (accessed Jan. 23, 2021).
- [2] O. Authors, “OpenCV API,” *OpenCV Docs*, 2020. <https://docs.opencv.org/4.5.0/> (accessed Jan. 23, 2021).
- [3] O. Authors, “OpenCV Image Processing,” *OpenCV Docs*, 2020. https://docs.opencv.org/4.5.0/d7/dbd/group__imgproc.html (accessed Jan. 23, 2021).
- [4] S. L. Authors, “Support Vector Machines,” *Scikit Learn*, 2015. <https://scikit-learn.org/stable/modules/svm.html> (accessed Jan. 23, 2021).
- [5] S. L. Authors, “Classifier comparison,” *Scikit Learn*, 2015. https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html (accessed Jan. 23, 2021).
- [6] P. D. Authors, “Tkinter API,” *Python Docs*, 2018. <https://docs.python.org/3/library/tkinter.html> (accessed Jan. 23, 2021).
- [7] S. L. Authors, “Supervised Learnings API,” *Scikit Learn*, 2015. https://scikit-learn.org/stable/supervised_learning.html#supervised-learning (accessed Jan. 23, 2021).
- [8] P. D. Authors, “Python API,” *Python Docs*, 2018. <https://docs.python.org/3.8/> (accessed Jan. 23, 2021).
- [9] Ashwin Nanjappa, *Caffe2 Quick Start Guide*. 2019.

8. Anexos

8.1 DEPENDENCIAS DE INSTALACIÓN

Durante la primera etapa del proyecto en la que se realizó una investigación en la que se incluían algunas librerías que se utilizarían encontré una serie de requisitos previos a la instalación incluyendo algunas versiones específicas de librerías. Las más destacables:

- **Python 3.8**: Lenguaje en el que se programa, más estable que su siguiente versión y con menos problemas de compatibilidad de versiones.
- **Pip**: Utilidad para instalar programas y complementos de Python.
- **OpenCV-Contrib-Python**: Una de las principales, para poder instalar esta hay que instalar 2 previas: **numpy** (en su versión 1.19.3 para ser exactos), librería que se utiliza para trabajar con arrays; y **matplotlib** que es una librería para el trabajo y representación de gráficas de datos.
- **Face_Recognition**: La otra librería principal. Constaba de dos requisitos específicos, **cmake**, que se utiliza para controlar el proceso de compilación de un programa y **dlib** que incluye algoritmos de *machine learning*. Además, un último requisito es la instalación de **Visual Code C++** para poder instalar esta última.

8.2 LIBRERÍAS FINALES INSTALADAS

El entorno de desarrollo utilizado fue PyCharm Community, incluye una utilidad para agregar librerías al proyecto en el que se está trabajando, estas se guardan en una carpeta dentro del directorio raíz del mismo llamada **venv**.

Todas las librerías finales instaladas para el desarrollo de este proyecto se pueden ver desde la ventana de configuración del mismo y se mostrarían de la siguiente manera:

Package	Version	Latest version
Pillow	8.0.1	▲ 8.1.0
click	7.1.2	7.1.2
cmake	3.18.4.post1	3.18.4.post1
cycler	0.10.0	0.10.0
dlib	19.21.1	19.21.1
face-recognition	1.3.0	1.3.0
face-recognition-models	0.3.0	
imutils	0.5.3	▲ 0.5.4
joblib	1.0.0	1.0.0
kiwisolver	1.3.1	1.3.1
matplotlib	3.3.3	3.3.3
numpy	1.19.3	▲ 1.19.5
opencv-contrib-python	4.4.0.46	▲ 4.5.1.48
pip	20.3.3	20.3.3
pyparsing	2.4.7	2.4.7
python-dateutil	2.8.1	2.8.1
scikit-learn	0.23.2	▲ 0.24.1
scipy	1.5.4	▲ 1.6.0
setuptools	51.0.0	▲ 51.3.3
six	1.15.0	1.15.0
threadpoolctl	2.1.0	2.1.0

Figura 8.1: Librerías finales

Las librerías principales son *Face_Recognition* (y su versión *models*), *opencv-contrib-python* y *scikit-learn* que son las que incluyen los algoritmos de reconocimiento. Además, se ven también las librerías comentadas en el apartado de dependencias.

8.3 ARCHIVOS PRECARGADOS

Dentro de este apartado se incluyen los archivos que cargan el sistema de identificación y reconocimiento facial. Serán en los que se basará el sistema para generar los embeddings que se compararán tanto del *dataset* como de la imagen de entrada.[9] Estos archivos son:

➔ ***Deploy.prototxt***. Define la arquitectura del modelo. Su estructura consiste en una sucesión de diccionarios con unos parámetros internos obligatorios, como pueden ser el nombre, el tipo, y las dimensiones de los valores de entrada y salida junto con las conexiones entre las diferentes capas que

conformarán este modelo. Cuanto más completo este modelo mejor funcionará el sistema. Para el desarrollo de este proyecto hice uso de un archivo de modelo ya preconfigurado.

```
input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 300
  dim: 300
}

layer {
  name: "data_bn"
  type: "BatchNorm"
  bottom: "data"
  top: "data_bn"
  param {
    lr_mult: 0.0
  }
  param {
    lr_mult: 0.0
  }
  param {
    lr_mult: 0.0
  }
}
```

Figura 8.2 Ejemplo de capa en el prototxt

➔ **res10_300x300_ssd_iter_140000.caffemodel:** Un archivo binario preconfigurado que incluye la serialización de la red neuronal entrenada en el formato binario *ProtoBuf*. Este formato guarda valores de tipo numérico tanto decimal como enteros que incluyen el “peso” de cada una de las capas en el sistema.

8.4 CÓDIGO FUENTE

Todo el código fuente correspondiente al desarrollo del trabajo se encuentra en un repositorio de GitHub referenciado en la bibliografía del proyecto.

8.4.1 FACE_RECOGNITION

```
import cv2
def run():
    # Cargamos nuestro clasificador de Haar:
    cascada_rostro = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
    # Si utilizas otro clasificador o lo tienes guardado en un directorio diferente al de este script
    # python,
    # tendrás que cambiar 'haarcascade_frontalface_alt.xml' por el path a tu fichero xml.

    # Cargamos la imagen y la convertimos a grises:
    img = cv2.imread('Input\imagen_input.jpg')
    img_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Nota: la imagen de ejemplo que hemos utilizado para el tutorial ya está en blanco y negro,
    # por lo que no sería necesario convertirla. Lo he hecho igualmente por si más adelante queréis
    # probar con una imagen en color.

    # Buscamos los rostros:
    coordenadas_rostros = cascada_rostro.detectMultiScale(img_gris, 1.3, 5)
    # Nota 1: la función detectMultiScale() requiere una imagen en escala de grises. Esta es la razón
    # por la que hemos hecho la conversión de BGR a Grayscale.
    # Nota 2: '1.3' y '5' son parámetros estándar para esta función. El primero es el factor de escala
    # ('scaleFactor'): la
    # función intentará encontrar rostros escalando la imagen varias veces, y este factor indica en
    # cuánto se reduce la imagen
    # cada vez. El segundo parámetro se llama 'minNeighbours' e indica la calidad de las detecciones:
    # un valor elevado
    # resulta en menos detecciones pero con más fiabilidad.

    # Ahora recorremos el array 'coordenadas_rostros' y dibujamos los rectángulos sobre la imagen
    # original:
    for (x, y, ancho, alto) in coordenadas_rostros:
        cv2.rectangle(img, (x, y), (x + ancho, y + alto), (0, 0, 255), 3)

    # Abrimos una ventana con el resultado:
    cv2.imshow('Output', img)
    print("\nMostrando resultado. Pulsa cualquier tecla para salir.\n")
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Figura 8.3 t1.py

```
import cv2
import face_recognition

def run():
    # Cargamos las imágenes con los rostros que queremos identificar:
    imagen_einstein = face_recognition.load_image_file("Datapool\Albert Einstein\einstein.jpg")
    imagen_paul = face_recognition.load_image_file("Datapool\Paul Langevin\paul.jpg")
    imagen_planck = face_recognition.load_image_file("Datapool\Max Planck\planck.jpg")

    # El siguiente paso es extraer los 'encodings' de cada imagen.
    # Los encodings son las características únicas de cada rostro que permiten diferenciarlo de otros.
    einstein_encodings = face_recognition.face_encodings(imagen_einstein)[0]
    paul_encodings = face_recognition.face_encodings(imagen_paul)[0]
    planck_encodings = face_recognition.face_encodings(imagen_planck)[0]

    # Creamos un array con los encodings y otro con sus respectivos nombres:
    encodings_conocidos = [
        einstein_encodings,
        paul_encodings,
        planck_encodings
    ]
    nombres_conocidos = [
        "Albert Einstein",
        "Paul Langevin",
        "Max Planck"
    ]

    # Cargamos una fuente de texto:
    font = cv2.FONT_HERSHEY_COMPLEX

    # Cargamos la imagen donde hay que identificar los rostros:
    img = face_recognition.load_image_file('Input\imagen_input.jpg')
    # (Para probar la segunda imagen hay que cambiar el argumento de la función por 'Imagen_input2.jpg')

    # Definir tres arrays, que servirán para guardar los parámetros de los rostros que se encuentren en la imagen:
    loc_rostros = [] # Localización de los rostros en la imagen (contendrá las coordenadas de los recuadros que las contienen)
    encodings_rostros = [] # Encodings de los rostros
    nombres_rostros = [] # Nombre de la persona de cada rostro

    # Localizamos cada rostro de la imagen y extraemos sus encodings:
    loc_rostros = face_recognition.face_locations(img)
    encodings_rostros = face_recognition.face_encodings(img, loc_rostros)

    # Recorremos el array de encodings que hemos encontrado:
    for encoding in encodings_rostros:

        # Buscamos si hay alguna coincidencia con algún encoding conocido:
        coincidencias = face_recognition.compare_faces(encodings_conocidos, encoding)

        # El array 'coincidencias' es ahora un array de booleanos.
        # Si contiene algún 'True', es que ha habido alguna coincidencia:
        if True in coincidencias:
            # Buscamos el nombre correspondiente en el array de nombres conocidos:
            nombre = nombres_conocidos[coincidencias.index(True)]

        # Si no hay ningún 'True' en el array 'coincidencias', no se ha podido identificar el rostro:
        else:
            nombre = "???"

        # Añadimos el nombre de la persona identificado en el array de nombres:
        nombres_rostros.append(nombre)

    # Dibujamos un recuadro rojo alrededor de los rostros desconocidos, y uno verde alrededor de los conocidos:
    for (top, right, bottom, left), nombre in zip(loc_rostros, nombres_rostros):

        # Cambiar el color según el nombre:
        if nombre != "???":
            color = (0, 255, 0) # Verde
        else:
            color = (0, 0, 255) # Rojo

        # Dibujar los recuadros alrededor del rostro:
        cv2.rectangle(img, (left, top), (right, bottom), color, 2)
        cv2.rectangle(img, (left, bottom - 20), (right, bottom), color, -1)

        # Escribir el nombre de la persona:
        cv2.putText(img, nombre, (left, bottom - 6), font, 0.6, (0, 0, 0), 1)

    # Abrimos una ventana con el resultado:
    cv2.imshow('Output', img)
    print("\nMostrando resultado. Pulsa cualquier tecla para salir.\n")
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Figura 8.4 t2.py

```
# -*- coding: latin-1 -*-
import cv2
import face_recognition
import os

def run():
    # Cargamos todas las imágenes del datapool utilizando un bucle para recorrer las subcarpetas
    arrayImages = []
    faceEncodings = []
    i = 0
    pathing = r"C:\Users\Sbita\PycharmProjects\Test1\Datapool\\"
    pathfinal = os.listdir(pathing)
    for person in pathfinal:
        print(person)
        file = os.listdir(pathing + person)
        if (file.__len__() > 0):
            for imagen in file:
                arrayImages.append(face_recognition.load_image_file(pathing + person + "/" + imagen))

            # Para cada imagen generamos su encoding correspondiente
            faceEncodings.append(face_recognition.face_encodings(arrayImages[i]))
    i += 1

    # Creamos un array con los encodings y otro con sus respectivos nombres:
    nombres_conocidos = [
        "Ben Afflek",
        "Albert Einstein",
        "Elton John",
        "Jerry Seinfeld",
        "Madonna",
        "Mindy Kailing",
        "Paul Langevin",
        "Max Planck",
    ]

    # Cargamos una fuente de texto:
    font = cv2.FONT_HERSHEY_COMPLEX

    # Cargamos la imagen donde hay que identificar los rostros:
    img = face_recognition.load_image_file('Input\Imagen_Input.jpg')
    # [Para probar la segunda imagen hay que cambiar el argumento de la función por 'Imagen_Input2.jpg']

    # Definir tres arrays, que servirán para guardar los parámetros de los rostros que se encuentran en la imagen:
    loc_rostros = [] # Localización de los rostros en la imagen (contendrá las coordenadas de los recuadros que las contienen)
    encodings_rostros = [] # Encodings de los rostros
    nombres_rostros = [] # Nombre de la persona de cada rostro

    # Localizamos cada rostro de la imagen y extraemos sus encodings:
    loc_rostros = face_recognition.face_locations(img)
    encodings_rostros = face_recognition.face_encodings(img, loc_rostros)

    # Recorremos el array de encodings que hemos encontrado:
    for encoding in encodings_rostros:
        # Buscamos si hay alguna coincidencia con algún encoding conocido:
        coincidencias = face_recognition.compare_faces(faceEncodings, encoding, tolerance=0.48)

        # El array 'coincidencias' es ahora un array de booleanos,
        # Si contiene algún 'True', es que ha habido alguna coincidencia:
        if True in coincidencias:
            encontrado = coincidencias.index(True)
            index = 0
            for carpeta in os.listdir(pathing):
                if not os.path.isfile(carpeta):
                    encontrado = encontrado - os.listdir(pathing + carpeta).__len__()
                if encontrado < 0:
                    nombre = carpeta.__str__()
                    break

            # Si no hay ningún 'True' en el array 'coincidencias', no se ha podido identificar el rostro:
            else:
                nombre = "???"

            # Añadimos el nombre de la persona identificada en el array de nombres:
            nombres_rostros.append(nombre)

    # Dibujamos un recuadro rojo alrededor de los rostros desconocidos, y uno verde alrededor de los conocidos:
    for (top, right, bottom, left), nombre in zip(loc_rostros, nombres_rostros):
        # Cambiar el color según el nombre:
        if nombre != "???":
            color = (0, 255, 0) # Verde
        else:
            color = (0, 0, 255) # Rojo

        # Dibujar los recuadros alrededor del rostro:
        cv2.rectangle(img, (left, top), (right, bottom), color, 2)
        cv2.rectangle(img, (left, bottom - 20), (right, bottom), color, -1)

        # Escribir el nombre de la persona:
        cv2.putText(img, nombre, (left, bottom - 6), font, 0.6, (0, 0, 0), 1)

    # Abrimos una ventana con el resultado:
    cv2.imshow('Output', img)
    print("\nMostrando resultado. Pulsa cualquier tecla para salir.\n")
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Figura 8.5 t3.py

```
# -*- coding: latin-1 -*-
import cv2
import face_recognition
import os
import pickle

# Guardando el resultado de los encodings conocidos comprobando si ya existía antes
def run():
    pathing = "D:/Datos/Proyectos/Test1_Face_recognition/datapool/"
    # Comprobamos si el archivo ya existe:
    if not os.path.exists(pathing+'saved_encodings.dat'):
        # Cargamos todas las imágenes del datapool utilizando un bucle para recorrer
        # las subcarpetas.
        arrayImages = []
        faceEncodings = []
        i = 0
        pathfinal = os.listdir(pathing)
        for person in pathfinal:
            print(person)
            file = os.listdir(pathing + person)
            if (file.__len__() > 0):
                for imagen in file:
                    arrayImages.append(face_recognition.load_image_file(pathing +
                    person + "/" + imagen))

        # Para cada imagen generamos su encoding correspondiente
        faceEncodings.append(face_recognition.face_encodings(arrayImages[i])[0])
        i += 1

        with open(pathing+'saved_encodings.dat', 'wb') as f:
            pickle.dump(faceEncodings, f)
    else:
        with open(pathing+'saved_encodings.dat', 'rb') as f:
            faceEncodings = pickle.load(f)

    # Creamos un array con sus respectivos nombres:
    nombres_conocidos = [
        "Ben Afflek",
        "Albert Einstein",
        "Elton John",
        "Jerry Seinfeld",
        "Madonna",
        "Mindy Kalling",
        "Paul Langevin",
        "Max Planck",
    ]

    # Cargamos una fuente de texto:
    font = cv2.FONT_HERSHEY_COMPLEX

    # Cargamos la imagen donde hay que identificar los rostros:
    img = face_recognition.load_image_file('imagen_input.jpg')
    # (Para probar la segunda imagen hay que cambiar el argumento de la función por
    # 'imagen_input2.jpg')

    # Definir tres arrays, que servirán para guardar los parámetros de los rostros
    # que se encuentren en la imagen:
    loc_rostros = [] # Localización de los rostros en la imagen (contendrá las
    # coordenadas de los recuadros que los contienen)
    encodings_rostros = [] # Encodings de los rostros
    nombres_rostros = [] # Nombre de la persona de cada rostro

    # Localizamos cada rostro de la imagen y extraemos sus encodings:
    loc_rostros = face_recognition.face_locations(img)
    encodings_rostros = face_recognition.face_encodings(img, loc_rostros)

    # Recorremos el array de encodings que hemos encontrado:
    for encoding in encodings_rostros:
        # Buscamos si hay alguna coincidencia con algún encoding conocido:
        coincidencias = face_recognition.compare_faces(faceEncodings, encoding,
        tolerance=0.5)

        # El array 'coincidencias' es ahora un array de booleanos.
        # Si contiene algún 'True', es que ha habido alguna coincidencia:
        if True in coincidencias:
            # Buscamos el nombre correspondiente en el array de nombres conocidos:
            encontrado = coincidencias.index(True)
            index = 0
            for carpeta in os.listdir(pathing):
                if (encontrado+1 == os.listdir(pathing+carpeta).__len__()) <= 0):
                    nombre = nombres_conocidos[index]
                    break
            else:
                index +=1
                encontrado = encontrado -
                os.listdir(pathing+carpeta).__len__()

        # Si no hay ningún 'True' en el array 'coincidencias', no se ha podido
        # identificar el rostro:
        else:
            nombre = "???"

        # Añadimos el nombre de la persona identificada en el array de nombres:
        nombres_rostros.append(nombre)

    # Dibujamos un recuadro rojo alrededor de los rostros desconocidos, y uno verde
    # alrededor de los conocidos:
    for (top, right, bottom, left), nombre in zip(loc_rostros, nombres_rostros):
        # Cambiar el color según el nombre:
        if nombre != "???":
            color = (0, 255, 0) # Verde
        else:
            color = (0, 0, 255) # Rojo

        # Dibujar los recuadros alrededor del rostro:
        cv2.rectangle(img, (left, top), (right, bottom), color, 2)
        cv2.rectangle(img, (left, bottom - 20), (right, bottom), color, -1)

        # Escribir el nombre de la persona:
        cv2.putText(img, nombre, (left, bottom - 6), font, 0.6, (0, 0, 0), 1)

    # Abrimos una ventana con el resultado:
    cv2.imshow('Output', img)
    print("\nMostrando resultado. Pulsa cualquier tecla para salir.\n")
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Figura 8.6 t4.py

```
# -*- coding: latin-1 -*-
import cv2
import face_recognition
import os
import pickle
import time

def run():
    print("Sistema de reconocimiento facial T5.")
    print("Comienza el contador de tiempo.")
    arrayImages = []
    path = os.getcwd()
    pathingdatapool = path + "\\datapool"
    tiempo1 = time.time()
    if not os.path.exists(path + "\\output\\saved_encodings.dat"):
        print("Se van a generar encodings.")
        faceEncodings = encoding_generator(arrayImages, pathingdatapool, path)
        print("Han tardado " + str(time.time()-tiempo1) + " segundos en generarse los encodings.")
    else:
        with open(path + "\\output\\saved_encodings.dat", 'rb') as f:
            print("Los encodings ya estaban guardados en el archivo correspondiente.")
            faceEncodings = pickle.load(f)
            print("Han tardado " + str(time.time()-tiempo1) + " segundos en cargarse los encodings.")

    # Definimos la lista de nombres conocidos con el siguiente método.
    nombres_conocidos = namelist_generator(os.listdir(pathingdatapool))

    # Necesitamos una fuente para escribir el recuadro
    font = cv2.FONT_HERSHEY_COMPLEX

    # La imagen donde se buscan las caras por ahora se carga directamente
    image_bw = cv2.cvtColor(cv2.imread(path + "\\input\\imagen_input.jpg"), cv2.COLOR_BGR2GRAY)
    cv2.imwrite(path + "\\input\\imagen_inputbw.jpg", image_bw)
    img = face_recognition.load_image_file(path + "\\input\\imagen_inputbw.jpg")

    # Generamos los 3 arrays para los parámetros de las caras de la imagen
    loc_rostros = face_recognition.face_locations(img)
    encoding_rostros = face_recognition.face_encodings(img, loc_rostros)

    # Usamos el método que compara las caras encontradas con las que tenía almacenadas
    nombres_rostros = finding_faces(pathingdatapool, encoding_rostros, faceEncodings)

    # Dibujamos los cuadrados al rector
    draw_squares(img, font, loc_rostros, nombres_rostros)

    # Mostramos una ventana con el resultado
    cv2.imshow('Output', img)
    cv2.imwrite(path + "\\output\\Output.jpg", img)
    print("Mostrando resultado. Pulse cualquier tecla para salir.\n")
    cv2.waitKey()
    cv2.destroyAllWindows()

def encoding_generator(arrayImages, pathinghome, path):
    # Este método se encarga de generar el encoding de cada cara
    # y lo exporta a un archivo .dat
    faceEncodings = []
    i = 0
    pathfinal = os.listdir(pathinghome)
    for person in pathfinal:
        file = os.listdir(pathinghome + "\\\" + person)
        if (file.__len__() > 0):
            for imagen in file:
                arrayImages.append(face_recognition.load_image_file(pathinghome + "\\\" + person + "\\\" + imagen))
            # Para cada imagen generamos su encoding correspondiente.
            faceEncodings.append(face_recognition.face_encodings(arrayImages[i]))
        i += 1

    with open(path + "\\output\\saved_encodings.dat", 'wb') as f:
        pickle.dump(faceEncodings, f)
    return faceEncodings

def namelist_generator(directoryList):
    # Este método genera una lista de nombres conocidos en función de
    # las carpetas existentes en la carpeta raíz datapool
    print("Se va a generar una lista de nombres conocidos.")
    tiempo2 = time.time()
    namelist = []
    for carpeta in directoryList:
        namelist.append(carpeta)
    print("Se ha generado una lista de nombres conocidos tardando un total de " + str(time.time() - tiempo2) + " segundos.")
    return namelist

def finding_faces(pathinghome, encoding_rostros, faceencodings):
    print("Se van a buscar las caras conocidas.")
    tiempo3 = time.time()
    nombres_rostros = []
    for encoding in encoding_rostros:
        # Arrays de True y False en función de si encuentra las caras en las que ya conoce
        coincidencias = face_recognition.compare_faces(faceencodings, encoding, tolerance=0.5)
        if True in coincidencias:
            encontrado = coincidencias.index(True)
            index = 0
            for carpeta in os.listdir(pathinghome):
                if not os.path.isfile(carpeta):
                    encontrado = encontrado + 1
            os.listdir(pathinghome+"\\\"+carpeta)....len...()
            if encontrado < 0:
                nombre = carpeta.__str__()
                break
            else:
                nombre = "???"
            nombres_rostros.append(nombre)
        print("Han tardado " + str(time.time()-tiempo3) + " segundos en encontrar las caras.")
    return nombres_rostros

def draw_squares(img, font, loc_rostros, nombres_rostros):
    for (top, right, bottom, left), nombre in zip(loc_rostros, nombres_rostros):
        # Cambiar el color según el nombre
        if nombre != "???":
            color = (0, 255, 0) # Verde
        else:
            color = (0, 0, 255) # Rojo
        # Dibujar los recuadros alrededor del rostro
        cv2.rectangle(img, (left, top), (right, bottom), color, 2)
        cv2.rectangle(img, (left, bottom - 20), (right, bottom), color, -1)
        # Escribir el nombre de la persona
        cv2.putText(img, nombre, (left, bottom - 6), font, 0.6, (0, 0, 0), 1)
```

Figura 8.7 t5.py


```
import cv2
import face_recognition
import os
import pickle
import time

from itertools import paths

def run():
    imagen = os.getcwd() + "\\input\\imagen_input.jpg"
    input = os.getcwd() + "\\input"
    output = os.getcwd() + "\\output"
    datapool = os.getcwd() + "\\datapool"

    print("Sistema de reconocimiento facial v6.")
    print("Cadenza el contador de tiempo.")
    # Pasa de la carpeta input
    tBase = time.time()
    # Primera vez vamos a ver si se han generado los encodings previamente del
    # datapool.
    if not os.path.exists(output + "\\saved_encodings.dat"):
        print("Se generan los encodings para las imágenes de datapool.")
        faceEncodings = genEncodings(datapool, output)
        t1 = time.time() - tBase
        print("Se ha tardado {} segundos en generar las imágenes y crear el archivo
        dat.".format(t1))
    else:
        with open(output + "\\saved_encodings.dat", "rb") as f:
            print("Los encodings ya estaban guardados y se cargarán de un archivo
            externo.")
            faceEncodings = pickle.load(f)
            t1 = time.time() - tBase
            print("Se ha tardado {} en cargar los encodings.".format(t1))

    # Definimos una lista de nombres conocidos en base al datapool.
    # Esta lista a que en el datapool las carpetas se llaman así (lado) con el
    # nombre que queremos mostrar.
    names = gen_NamesList(datapool)

    # Queremos una fuente para escribir en el cuadrado.
    font = cv2.FONT_HERSHEY_COMPLEX

    # Cargamos la imagen en la que queremos reconocer caras.
    # Primero la pasamos a blanco y negro.
    bw = cv2.cvtColor(cv2.imread(imagen), cv2.COLOR_BGR2GRAY)
    # Si queremos guardarla antes de realizar el reconocimiento.
    cv2.imwrite(input + "\\\" + "inputBW.jpg", bw)

    # La cargamos al face_recognition
    img = face_recognition.load_image_file(input + "\\\" + "inputBW.jpg")

    # Creamos un diccionario donde almacenamos los arrays que necesitaremos
    recon = {}
    loc = face_recognition.face_locations(img)
    enc = face_recognition.face_encodings(img, loc)
    nom = find_Faces(datapool, enc, faceEncodings)

    recon.setdefault("loc", loc)
    recon.setdefault("enc", enc)
    recon.setdefault("nom", nom)

    # Dibujamos los cuadrados en la imagen
    draw(img, font, recon.get("loc"), recon.get("nom"))

    cv2.imshow("Output", img)
    cv2.imwrite(output + "\\output.jpg", img)
    print("Mostrando resultado, pulsa cualquier tecla para salir")
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def gen_Encodings(datapool, output):
    # Este método genera encodings para cada una en el datapool y exporta el resultado
    # final.
    encodings = []
    i = 0
    images = []
    for person in os.listdir(datapool):
        direc = "\\{\\path\\list_images(datapool + "\\\" + person)\\}
        for image in direc:
            images.append(face_recognition.load_image_file(image))
            # Aquí se generan los encodings.
            encodings.append(face_recognition.face_encodings(images[i])[0])
            i += 1

    # Se exportan
    with open(output + "\\saved_encodings.dat", "wb") as f:
        pickle.dump(encodings, f)
    return encodings

def gen_NamesList(datapool):
    # Este método genera una lista de nombres conocidos en función de
    # las carpetas existentes en la carpeta \\datapool.
    print("Se va a generar una lista de nombres conocidos.")
    t2 = time.time()
    namesList = []
    for carpeta in datapool:
        namesList.append(carpeta)
    print("Se ha generado una lista de nombres conocidos tardando un total de " +
    str(
        time.time() - t2) + " segundos.")
    return namesList

def find_Faces(datapool, enc_input, enc_datapool):
    print("Se van a buscar las caras conocidas en la imagen pasada.")
    nom = []
    name = ""
    totalImages = list(paths.list_images(datapool)).__len__()
    t3 = time.time()
    for enc in enc_input:
        # Generamos un array de True y False donde cada True representa si se ha
        # encontrado.
        coins = face_recognition.compare_faces(enc_datapool, enc, tolerance=0.44)
        # Ahora comprobamos si se ha encontrado y si es así guardamos su nombre.
        if True in coins:
            found = coins.index(True)
            index = 0
            for carpeta in os.listdir(datapool):
                if not os.path.isfile(carpeta):
                    tem = os.listdir(datapool + "\\\" + carpeta).__len__()
                    found = found - tem
                    if found == 0:
                        name = carpeta.__str__()
                        break
            else:
                name = "Unknown"
            nom.append(name)
    print("Se han tardado {} segundos en reconocer todas las caras de la
    imagen.".format(time.time()-t3))
    return nom

def draw(img, font, loc, nom):
    for (top, right, bottom, left), name in zip(loc, nom):
        # Cambiar el color según el nombre.
        if name != "???":
            color = (0, 255, 0) # Verde
        else:
            color = (0, 0, 255) # Rojo

        # Dibujar los rectángulos alrededor del rostro.
        cv2.rectangle(img, (left, top), (right, bottom), color, 2)
        cv2.rectangle(img, (left, bottom - 20), (right, bottom), color, -1)

        # Escribir el nombre de la persona.
        cv2.putText(img, name, (left, bottom - 6), font, 0.6, (0, 0, 0), 1)
```

Figura 8.8 t6.py

```
import cv2
import face_recognition
import os
import pickle
import time

from utils import paths

def run(data):
    root = os.getcwd()
    imagen = root + data["input"] + "\\imagen_input.jpg"
    input = root + data["input"]
    output = root + data["output"]
    datapool = root + data["datapool"]

    print("Sistema de reconocimiento facial T6.")
    print("Comienza el contador de tiempo.")
    # Path de la carpeta padre
    tBase = time.time()
    # Primero comprobamos si ya se han generado las encodings previamente del
    # datapool.
    if not os.path.exists(output + "\\saved_encodings.dat"):
        print("Se generan los encodings para las imágenes de datapool.")
        faceEncodings = gen_Encodings(datapool, output)
        t1 = time.time() - tBase
        print("Se ha tardado {} segundos en generar las imágenes y crear el archivo
        dat.".format(t1))
    else:
        with open(output + "\\saved_encodings.dat", "rb") as f:
            print("Los encodings ya estaban guardados y se cargarán de un archivo
            externo.")
            faceEncodings = pickle.load(f)
            t1 = time.time() - tBase
            print("Se ha tardado {} en cargar los encodings.".format(t1))

    # Definimos una lista de nombres conocidos en base al datapool.
    # Esta lista a que en el datapool las carpetas se tengan que llamar con el
    # nombre que queremos mostrar.
    names = gen_NameList(datapool)

    # Guardamos una fuente para escribir en el cuadrado
    font = cv2.FONT_HERSHEY_COMPLEX

    # Cargamos la imagen en la que queremos reconocer caras
    # Primeros los caminos de imagen y luego el nombre
    bw = cv2.cvtColor(cv2.imread(imagen), cv2.COLOR_BGR2GRAY)
    # Si necesitamos guardarlo antes de realizar el reconocimiento
    cv2.imwrite(input + "\\\" + "inputBW.jpg", bw)

    # Lo cargamos al face_recognition
    img = face_recognition.load_image_file(input + "\\\" + "inputBW.jpg")

    # Creamos un diccionario donde almacenamos los arrays que necesitaremos
    recon = {}
    loc = face_recognition.face_locations(img)
    enc = face_recognition.face_encodings(img, loc)
    nom = find_Faces(datapool, enc, faceEncodings)

    recon.setdefault("loc", loc)
    recon.setdefault("enc", enc)
    recon.setdefault("nom", nom)

    # Dibujamos los cuadrados de la imagen
    draw(img, font, recon.get("loc"), recon.get("nom"))

    cv2.imshow("Output", img)
    cv2.imwrite(output + "\\Output.jpg", img)
    print("Mostrando resultado, pulsa cualquier tecla para salir")
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def gen_Encodings(datapool, output):
    # Esta función genera encodings para cada un el datapool y exporta el resultado
    # final.
    encodings = []
    L = 0
    images = []
    for person in os.listdir(datapool):
        direc = list(paths.list_images(datapool + "\\\" + person))
        for image in direc:
            images.append(face_recognition.load_image_file(image))
            # Aquí se generan los encodings
            encodings.append(face_recognition.face_encodings(images[L])[0])
            L += 1

    # Lo exportamos
    with open(output + "\\saved_encodings.dat", "wb") as f:
        pickle.dump(encodings, f)
    return encodings

def gen_NameList(datapool):
    # Esta función genera una lista de nombres conocidos en función de
    # las carpetas existentes en la carpeta del datapool.
    print("Se va a generar una lista de nombres conocidos.")
    t2 = time.time()
    namelist = []
    for carpeta in datapool:
        namelist.append(carpeta)
    print("Se ha generado una lista de nombres conocidos tardando un total de " +
    str(
        time.time() - t2) + " segundos.")
    return namelist

def find_Faces(datapool, enc_input, enc_datapool):
    print("Se van a buscar las caras conocidas en la imagen pasada.")
    nom = []
    name = ""
    totalImages = list(paths.list_images(datapool)).__len__()
    t3 = time.time()
    for enc in enc_input:
        # Generamos un Array de True y False donde cada True representa si se ha
        # encontrado.
        colic = face_recognition.compare_faces(enc_datapool, enc, tolerance=0.44)
        # Ahora comprobamos si se ha encontrado y si es así guardamos su nombre.
        if True in colic:
            found = colic.index(True)
            index = 0
            for carpeta in os.listdir(datapool):
                if not os.path.isfile(carpeta):
                    len = os.listdir(datapool + "\\\" + carpeta).__len__()
                    found = found - len
                    if found == 0:
                        name = carpeta.__str__()
                        break
            else:
                name = "Unknown"
            nom.append(name)
    print("Se ha tardado {} segundos en reconocer todas las caras de la
    imagen.".format(time.time()-t3))
    return nom

def draw(img, font, loc, nom):
    for (top, right, bottom, left), name in zip(loc, nom):
        # Cambiamos el color según el nombre:
        if name != "7777":
            color = (0, 255, 0) # Verde
        else:
            color = (0, 0, 255) # Azul

        # Dibujar los rectángulos alrededor del rostro:
        cv2.rectangle(img, (left, top), (right, bottom), color, 2)
        cv2.rectangle(img, (left, bottom - 20), (right, bottom), color, -1)

        # Escribir el nombre de la persona:
        cv2.putText(img, name, (left, bottom - 6), font, 0.6, (0, 0, 0), 1)
```

Figura 8.9 t7.py

8.4.2 OPENCV

```
from tkinter import *
from tkinter import filedialog
from tkinter import ttk
import AnalizeDatapool, Trainer, Recognize
import time, datetime, os

rutaInput = ""
rutaDatapool = "C:/Users/Sbita/PycharmProjects/Test1/Datapool"
rutaOutput = "C:/Users/Sbita/PycharmProjects/Test1/Test Final - Interfaz Grafica/Output"

def ventanaTiempos(ts):
    root = Tk()
    root.title("Face Recognition")
    root.geometry("390x260")
    root.resizable(0, 0)
    root.iconbitmap("../Input/proconsl.ico")

    tiempos = ttk.Treeview(root, columns="ts")
    tiempos.heading("#0", text="Tiempos")
    tiempos.heading("ts", text="Tiempo en segundos")
    tiempos.column("#0", minwidth=0, width=220)
    tiempos.column("ts", minwidth=0, width=130)
    item = tiempos.insert("", "end", text="Tiempo inicial: ", values=str(ts["t0"]))
    item = tiempos.insert("", "end", text="Leer datapool: ", values=str(ts["t1"]))
    item = tiempos.insert("", "end", text="Entrenar el sistema: ", values=str(ts["t2"]))
    item = tiempos.insert("", "end", text="Reconocer caras dentro de la imagen: ", values=str(ts["t3"]))
    tiempos.place(x=20, y=20)

    root.mainloop()

def uploadInput():
    global rutaInput
    path = filedialog.askopenfilename(initialdir=os.getcwd()+"/../")
    rutaInput = path

def uploadDatapool():
    path = filedialog.askdirectory(initialdir=os.getcwd()+"/../")
    rutaDatapool = path

def uploadOutput():
    path = filedialog.askdirectory(initialdir=os.getcwd()+"/../")
    rutaOutput = path
    print(rutaOutput)

def start(kernel):
    t0 = time.time()
    tiempos = {"t0": datetime.datetime.now().strftime("%H:%M:%S"), "t1": "", "t2": "", "t3": ""}
    if not os.path.exists(rutaOutput + "/embeddings.pickle"):
        AnalizeDatapool.run("Face_Detection_Model/", "openface.nn4.small2.v1.t7", rutaDatapool + "/", 0.95, rutaOutput + "/embeddings.pickle")
    tiempos["t1"] = time.time() - t0
    t0 = time.time()
    if kernel.get() == 1:
        kern = "linear"
    elif kernel.get() == 2:
        kern = "rbf"
    else:
        kern = "poly"
    Trainer.run(rutaOutput + "/embeddings.pickle", rutaOutput + "/recognizer.pickle", rutaOutput + "/le.pickle", kern)
    tiempos["t2"] = time.time() - t0

    t0 = time.time()
    tfinal = Recognize.run("face_detection_model/", "openface.nn4.small2.v1.t7", rutaOutput + "/recognizer.pickle", rutaOutput + "/le.pickle", rutaInput, 0.95, rutaOutput, t0)
    tiempos["t3"] = tfinal
    ventanaTiempos(tiempos)

def main():
    root = Tk()
    root.title("Face Recognition")
    root.geometry("370x260")
    root.resizable(0, 0)
    root.iconbitmap("../Input/proconsl.ico")

    # Añadir los objetos que utilizaremos después
    labelInput = Label(root, text="Imagen en la que se quiere realizar el reconocimiento:")
    labelInput.place(x=20, y=20)

    buttonUploadFile = Button(root, text="Select", command=uploadInput)
    buttonUploadFile.place(x=310, y=20)

    labelOutput = Label(root, text="Directorio en el que se guardará el resultado:")
    labelOutput.place(x=20, y=50)
    buttonUploadFolder1 = Button(root, text="Select", command=uploadOutput)
    buttonUploadFolder1.place(x=310, y=50)

    labelDatapool = Label(root, text="Directorio en el que se encuentra el datapool:")
    labelDatapool.place(x=20, y=80)
    buttonUploadFolder2 = Button(root, text="Select", command=uploadDatapool)
    buttonUploadFolder2.place(x=310, y=80)

    labelDatapool = Label(root, text="Algoritmo que se desea utilizar:")
    labelDatapool.place(x=20, y=110)

    kernel = IntVar()
    radioLinear = Radiobutton(root, text="Linear", variable=kernel, value=1)
    radioLinear.place(x=200, y=110)

    radioRBF = Radiobutton(root, text="RBF", variable=kernel, value=2)
    radioRBF.place(x=260, y=110)

    radioPoly = Radiobutton(root, text="Poly", variable=kernel, value=3)
    radioPoly.place(x=305, y=110)

    buttonStart = Button(root, text="Iniciar Reconocimiento", command=lambda: start(kernel)).place(x=120, y=170)

    root.mainloop()

main()
```

Figura 8.10 main.py


```
import os
import cv2
from imutils import paths
import imutils
import numpy
import pickle

def run(pathDetector, model, datapool, umbralConfianza, pathEmbeddings):
    # Cargamos en base a los archivos precargados el detector
    protoPath = os.path.sep.join([pathDetector, "deploy.prototxt"])
    modelPath = os.path.sep.join([pathDetector,
                                   "res10_300x300_ssd_iter_140000.caffemodel"])
    detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
    # Ahora cargamos el sistema de reconocimiento
    embedder = cv2.dnn.readNetFromTorch(model)

    # Generamos una lista con las imagenes en el datapool
    imagePath = list(paths.list_images(datapool))

    # Generamos 2 listas además para guardar los nombres y sus correspondientes
    embeddings
    # Junto con el total de nombres
    embeddings = []
    nombres = []
    total = 0
    for (i, imagePath) in enumerate(imagePath):
        # Extraemos el nombre de cada persona asociada a la imagen
        nomb = imagePath.split(os.path.sep)[-2]
        nombre = nomb.split("/").pop()

        # Para cada imagen, la cargamos y la reescalamos en ancho para obtener las
        dimensiones finales
        imagen = cv2.imread(imagePath)
        imagen = imutils.resize(imagen, width=600)
        (h, w) = imagen.shape[:2]

        # Para cada imagen generamos su "blob"
        imageBlob = cv2.dnn.blobFromImage(cv2.resize(imagen, (300, 300)), 1.0, (300,
        300),
        (104.0, 177.0, 123.0), swapRB=False,
        crop=False)

        # Localizamos caras en las imagenes usando OpenCV
        detector.setInput(imageBlob)
        resultados = detector.forward()

        # Tiene que detectar minimo una cara en la imagen para seguir
        if len(resultados) > 0:
            # Suponemos que cada imagen del datapool tiene solo una cara
            # Escogemos la que tiene mayor probabilidad entonces
            i = numpy.argmax(resultados[0, 0, :, 2])
            confianza = resultados[0, 0, i, 2]

            # Esta detección tiene que tener un valor de confianza mayor que el
            umbral

            if confianza > umbralConfianza:
                # Cargamos las coordenadas extraidas
                box = resultados[0, 0, i, 3:7] * numpy.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")

                # Extraemos los datos ROI de la imagen con sus dimensiones
                cara = imagen[startY:endY, startX:endX]
                (fH, fW) = cara.shape[:2]

                # Tiene que tener unas dimensiones minimas
                if fW < 20 or fH < 20:
                    continue

                # Construimos el "blob" de las detecciones ROI y lo pasamos por le
                modelo

                # para obtener la cuantificacion de la cara en 128
                blob = cv2.dnn.blobFromImage(cara, 1.0 / 255, (96, 96), (0, 0, 0),
                swapRB=True, crop=False)
                embedder.setInput(blob)
                vec = embedder.forward()

                # Añadimos ahora el nombre junto con su embedding las listas y
                sumamos 1 al total
                nombres.append(nombre)
                embeddings.append(vec.flatten())
                total += 1

            # Ahora para guardarlo en un fichero construimos un diccionario en el
            que guardamos ambos arrays

            data = {
                "embeddings": embeddings,
                "nombres": nombres
            }
            f = open(pathEmbeddings, "wb")
            f.write(pickle.dumps(data))
            f.close()
```

Figura 8.11 AnalizeDatapool.py

```
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
import pickle

def run(emb, recon, inputle, kernel):
    # Cargamos los embeddings generados antes
    data = pickle.loads(open(emb, "rb").read())
    le = LabelEncoder()
    labels = le.fit_transform(data["nombres"])

    # Ahora entrenamos el modelo usando los embeddings para crear el sistema de
    reconocimiento
    recognizer = SVC(C=100, kernel=kernel, probability=True, gamma=1)
    recognizer.fit(data["embeddings"], labels)

    # Guardamos el sistema entrenado
    f = open(recon, "wb")
    f.write(pickle.dumps(recognizer))
    f.close()

    # Guardamos el label encoder también
    f = open(inputle, "wb")
    f.write(pickle.dumps(le))
    f.close()
```

Figura 8.12 Trainer.py

```
import numpy
import pickle
import cv2
import os
import time

def run(detect, model, recon, inputle, input, umbralConfianza, output, t0):
    # Cargamos el detector entrenado
    protoPath = os.path.sep.join([detect, "deploy.prototxt"])
    modelPath = os.path.sep.join([detect,
    "res10_300x300_ssd_iter_140000.caffemodel"])
    detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

    # Cargamos ahora el modelo
    embedder = cv2.dnn.readNetFromTorch(model)

    # Ahora cargamos el sistema de reconocimiento entrenado
    recognizer = pickle.loads(open(recon, "rb").read())
    le = pickle.loads(open(inputle, "rb").read())

    # Cargamos la imagen en la que vamos a realizar el reconocimiento
    imagen = cv2.imread(input)
    (h, w) = imagen.shape[:2]

    # Construimos el blob de esta imagen
    imageBlob = cv2.dnn.blobFromImage(
        cv2.resize(imagen, (300, 300)), 1.0, (300, 300),
        (104.0, 177.0, 123.0), swapRB=False, crop=False)

    # Ahora detectamos las caras en esta imagen y las guardamos
    detector.setInput(imageBlob)
    detections = detector.forward()

    # Todas estas detecciones las tenemos que comparar con el datapool
    for i in range(0, detections.shape[2]):
        confianza = detections[0, 0, i, 2]
        if confianza > umbralConfianza:
            box = detections[0, 0, i, 3:7] * numpy.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # extraemos los ROI
            face = imagen[startY:endY, startX:endX]
            (fH, fW) = face.shape[:2]
            # Comprobamos que cumpla el tamaño mínimo
            if fW < 20 or fH < 20:
                continue
            faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255, (96, 96), (0, 0, 0),
            swapRB=True, crop=False)
            embedder.setInput(faceBlob)
            vec = embedder.forward()

            # Clasificamos para reconocer las caras
            preds = recognizer.predict_proba(vec)[0]
            j = numpy.argmax(preds)
            proba = preds[j]
            nombre = le.classes_[j]

            # Por ultimo dibujamos la caja que lo contiene
            texto = "{: {:.2f}%}".format(nombre, proba * 100)
            y = startY - 10 if startY - 10 > 10 else startY + 10
            cv2.rectangle(imagen, (startX, startY), (endX, endY), (255, 0, 0), 2)
            cv2.putText(imagen, nombre, (startX, startY - 6),
            cv2.FONT_HERSHEY_COMPLEX, 0.45, (0, 0, 200), 1)

    # show the output image
    tfinal = time.time() - t0
    cv2.imshow("Image", imagen)
    cv2.imwrite(os.path.join(output, 'output.jpg'), imagen)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return tfinal
```

Figura 8.13 Recognize.py