



INFORME DE PRÁCTICAS

03/12/2020

Sebastián Meilán Bereciartúa
Ingeniería Informática
Identificación facial

ÍNDICE

Instalando las librerías	2
Requisitos Previos	2
OpenCV	2
Face_Recognition	2
Modelo de trabajo	3
T1.py	4
T2.py	4
T3.py	4
T4.py	5
T5.py	6
Futuro	7
Bibliografía	8

⁽¹⁾ Nota: El comando para instalar sería “*pip install librería(==versión)*”.

Instalando las librerías

Requisitos Previos

Primero que nada, para este proyecto y con las librerías que decidí trabajar necesité instalar otras librerías previas para que funcionasen correctamente.

Personalmente decidí instalar todas estas librerías en el python 3.8 oficial que se descarga en la web de python ([Descarga](#)) y después importar este *interpreter* en Pycharm Community. Esto lo hago porque en trabajos previos que realicé en la universidad siempre había utilizado el comando *pip* para instalar diferentes librerías en python.

Supongo que esto no hace falta que lo ponga pero es para que quede constancia por escrito y es que para instalar el pip se utiliza el [get-pip.py](#).

OpenCV

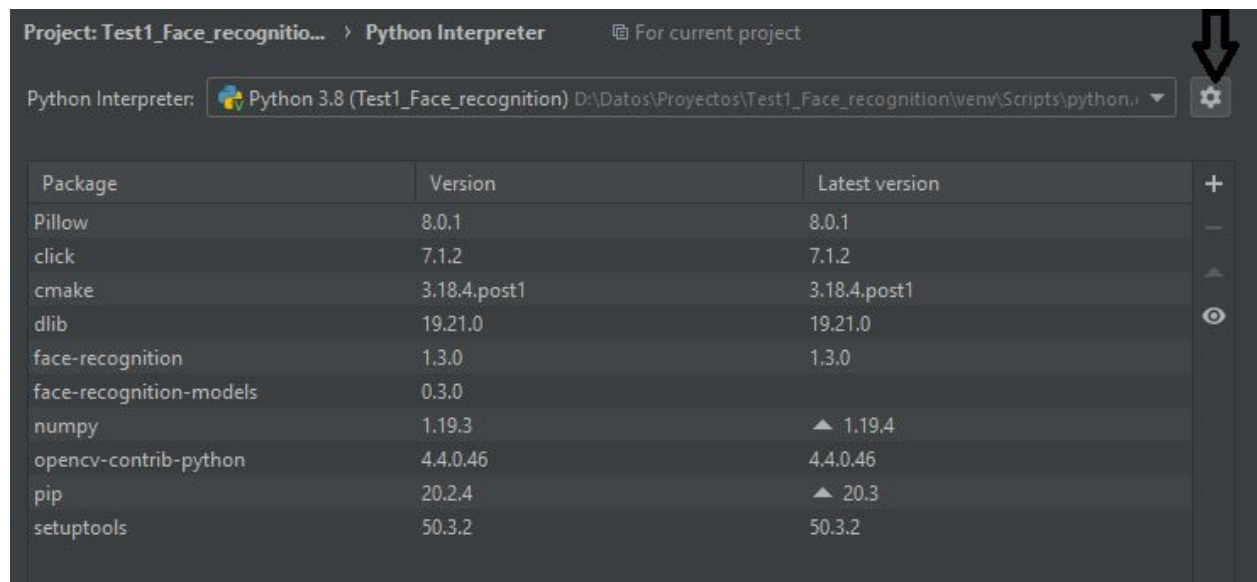
En cuanto a esta librería tiene una serie de requisitos previos, que son instalar las librerías **numpy** (en su versión 1.19.3 para evitar errores) y **matplotlib**. Además esta librería me respondía con una serie de errores en una DLL al importarla desde pycharm que solucioné instalando su versión **opencv-contrib-python** que es una versión completa de la misma. ⁽¹⁾

Face_Recognition

Face_Recognition también tiene una serie de requisitos. El primero, instalar **cmake**, que se puede descargar desde su [página oficial](#). Una vez tenemos esto, debemos instalar **dlib**, que se puede instalar también utilizando *pip*, pero previamente deberemos instalar **Visual Code C + +**, también desde su página.

⁽¹⁾ Nota: El comando para instalar sería "*pip install librería(==versión)*".

Una vez tenemos todo esto simplemente tendríamos que importar este interpreter en Pycharm y nos quedaría algo así:



(La flecha indica donde se añade un nuevo interpreter).

Modelo de trabajo

Como nos disteis mucha libertad a la hora de escoger cómo trabajar y cómo estructurar nuestro trabajo, y en vistas a lo que sería un futuro TFG, con documentación necesarias donde necesitaría poder ver los pasos que siguió el proyecto me decanté por un modelo de trabajo donde voy generando archivos a los que voy llamando **t#.py** donde, para pasar al siguiente debo añadir una modificación o trabajar un método diferente y así marcar claramente la evolución del proyecto.

Debido a esto, y si miráis en el repositorio de Github que tenemos compartido se puede apreciar que el archivo *t1.py* es un sistema muy simple donde simplemente se realiza una identificación facial en una foto sin un *datapool*, y, poco a poco se ve un avance hacia un sistema de reconocimiento facial más estructurado.

⁽¹⁾ Nota: El comando para instalar sería "*pip install librería(==versión)*".

T1.py

Este archivo está basado en una guía bastante simple que encontré y que realiza una identificación facial sin reconocimiento y muestra el resultado por pantalla explicando cómo funcionan los métodos de OpenCV.

T2.py

En este archivo ya se ejecuta un reconocimiento facial sobre una imagen utilizando como “*mini-datapool*” 3 imágenes que se cargan previamente y de manera manual para generar sus respectivos *encodings*, que también se generan manualmente y se almacenan en un array.

Además un detalle a destacar es que cuando se realiza el reconocimiento sobre la imagen es que cuando se reconoce una cara en base a las almacenadas el *index* sobre el array de nombres coincide con el *index* en el array de *Trues* y *Falses* que devuelve el método que los compara

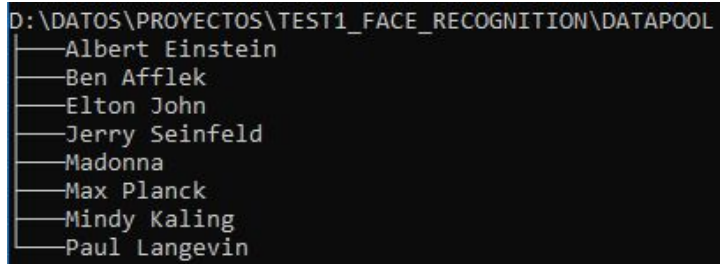
T3.py

A partir de este archivo soy yo trabajando sobre este último código e intentando mejorarlo para que sea más “aceptable” y óptimo para lo que se busca en este proyecto.

En esta parte me centré sobre todo en la automatización del proceso en general. Mi idea era ampliar el *datapool*, y me di cuenta de que conforme crecía se volvía más engorroso añadir imágenes al mismo y cargarlas. Por lo tanto decidí actualizar primero la estructura del árbol de directorios del proyecto para poder utilizar un bucle y recorrerlo entero.

⁽¹⁾ Nota: El comando para instalar sería “*pip install librería(==versión)*”.

Quedó algo así



```
D:\DATOS\PROYECTOS\TEST1_FACE_RECOGNITION\DATAPOOL
—Albert Einstein
—Ben Afflek
—Elton John
—Jerry Seinfeld
—Madonna
—Max Planck
—Mindy Kaling
—Paul Langevin
```

Una vez hecho esto el funcionamiento del bucle es muy simple (si que es verdad que en la última reunión os dije que me había encontrado algún problema, pero eso ya fue fallo mío, errores de novato), se recorre la carpeta general pasando por todas sus subcarpetas y generando un *encoding* de cada una de las imágenes que contiene y los va guardando en un array.

Además de esto, a la hora de comparar también añadí un bucle ya que, por ejemplo, en mi caso tenía un total de 87 imágenes y solo tenía 8 nombres conocidos, por lo tanto tenía que ir recorriendo las carpetas de nuevo para ver a qué nombre se correspondía, otra opción que se me ocurrió era utilizar un estructura bidireccional o matriz donde cada fila representa la carpeta y cada columna las fotos que contenía la misma, me decanté por esta sin ningún motivo en especial, pero bueno, no descarto utilizar otra en otras versiones.

T4.py

Después de la reunión pasada David me comentó que sería interesante que para mejorar este archivo sería útil que al cargar las imágenes del *datapool* se exportase este resultado a un archivo externo para no tener que cargarlo cada vez que lo lanzo.

El desarrollo fue sencillo, simplemente me aprovecho de la librería *os* para utilizar métodos como el **path** o el **exists** para comprobar si el archivo ya se creó anteriormente y de utilidad **with open** pasándole como parámetro 1 el archivo que quiero crear o cargar y como parámetro 2 **wb** cuando quiero crearlo y **rb** cuando solamente quiero cargarlo o leerlo.

⁽¹⁾ Nota: El comando para instalar sería “*pip install librería(==versión)*”.

T5.py

Esta es la última modificación de código hasta la fecha. Consideraba que este sistema de identificación era muy simple y mejorable obviamente, por lo tanto decidí reestructurar un poco el código y dividirlo en métodos para así, conforme añada o modifique cosas sea más comprensible y cómodo de hacer. Dividí el código en 5 métodos principales.

```
def run():...

def encoding_generator(arrayimages, pathinghome):...

def namelist_generator(directorylist):...

def finding_faces(pathinghome, encoding_rostros, faceencodings):...

def draw_squares(img, font, loc_rostros, nombres_rostros):..
```

- **Encoding_generator:** Se genera si es necesario un nuevo archivo de *encoding* basado en la carpeta *datapool*, el cual ahora estará alojado en la carpeta *output*, además, almacena el contenido de este mismo archivo en el array de *encodings* para trabajar con él.

- **Namelist_generator:** Viendo la actualización que había hecho en el t3, llegue a la conclusión que de igual manera también era engorroso crear manualmente una lista de nombres y cree este método nuevo que simplemente recorre la lista de carpetas del *datapool* y por cada una agrega un nuevo nombre. Dos cosas a destacar: 1. Sí, sé que esto lo podría hacer en el método anterior conforme género los *encodings* pero por ahora prefería dejarlo por separado, esto nos lleva al punto 2. Este método condiciona la estructura del *datapool* haciendo que las carpetas de imágenes se tengan que llamar con el nombre exacto que queremos que identifique a la persona, por eso está separado, a lo mejor en una futura versión se prefiere utilizar otro formato de archivo como *csv*.

⁽¹⁾ Nota: El comando para instalar sería “*pip install librería(==versión)*”.

- **Finding_faces:** Este método es el que se encarga de comparar los *encodings* de los rostros de la imagen con los almacenados en base al datapool y genera un array de coincidencias que es el que luego mira para generar un array de nombres para escribir en la imagen.

Como detalle, me dí cuenta al realizar esta modificación que el bucle que era el que se encargaba de dar el nombre a cada cara estaba mal y no lo hacía correctamente, ya está solucionado. Además, si la calidad de imagen no era demasiado buena confundía alguna cara y decidí bajarle 0,1 la tolerancia al método, ahora funciona correctamente.

- **Draw_squares:** Este método simplemente es el que dibuja el recuadro que engloba la cara de color rojo y con "???" por nombre si no conoce la cara y verde con su correspondiente nombre en caso contrario.

- **Run:** Este es el método principal del script, se encarga de ir llamando a los anteriores y pasándole como parámetro lo que le corresponde a cada uno. Además es el que se encarga de comprobar si el archivo del datapool existe, además carga la fuente que escribirá el nombre en la imagen y la exporta a un archivo externo (cosa que no se hacía en los anteriores).

Futuro

De cara al futuro me gustaría intentar una serie de implementaciones:

- Implementar el clasificador Haar para la identificación facial y combinarlo con lo que tengo hasta ahora.
- Retocar la manera en la que se estructura el datapool aprovechando una utilidad sobre la cual me estoy informando que es algo parecido al Hashmap de Java que genera una estructura con una key para representar a un campo o dato, creo que podría ser útil para usar esta key para representar el nombre y agregarle como valor un array de *encodings*, o simplemente añadir varios valores por separado.
- He empezado a mirar y a probar el enlace que nos habíais dejado en la primera documentación que nos entregasteis ahora que me veo más capacitado para comprenderlo y poder aplicarlo.
- Ver cómo generar documentación desde el propio Python.

⁽¹⁾ Nota: El comando para instalar sería "*pip install librería(==versión)*".

Bibliografía

OPENCV

[API principal](#)

[API imgproc](#)

[Post sobre como funciona OpenCV](#)

FACE_RECOGNITION

[API](#)

⁽¹⁾ Nota: El comando para instalar sería “*pip install librería(==versión)*”.