# Léeme

**Equipo:** You Only Lose Once

**IMPORTANTE:** leer este documento antes de comenzar a ver el resto del proyecto.

**Antes de mirar el código, os recomendamos leeros la sección Entornos, que se encuentra al final del documento.**

Aquí os aportamos información de utilidad sobre las partes por las que está compuesta nuestra solución. Tenemos las siguientes secciones:

## Documentación:

**Archivo**: *memoria.pdf*

Está dividida en los puntos que nos indicasteis. Os recomendamos leerla primero para entender cómo está dividido el código y cuáles son los modelos que os presentamos.

## Ejecutable con el que testear el funcionamiento del pipeline completo:

**Carpeta:** *Scripts*

Cuando leáis nuestra memoria, veréis que os hemos hecho dos propuestas de modelo:

- La primera es el código más potente que hemos conseguido, con una precisión del 68,93% en el apartado de Test. Sin embargo, se apoya en una red preentrenada y requiere un tiempo grande de entrenamiento unas 3h y media. La demo de este modelo se puede encontrar en la carpeta **Script1**.
- La segunda es un código menos potente (63,10% de precisión). Sin embargo, ha sido enteramente entrenada y ajustada por nosotros, y solo usando datos del dataset. Además, requiere mucho menos tiempo de entrenamiento, entorno a minuto y medio. La demo de este modelo se puede encontrar en **Script2**.

Para probarlos, tenemos todo el código necesario en la carpeta *Scriptx.* Ambas tienen la misma estructura, con los archivos:

- *codigo.py*: código que hay que ejecutar
- *haarcascade_frontalface_default.xml*: archivo con el clasificador para detectar las caras.
- *Red convolucional*: para extraer las features. Dependiendo del script usaremos una u otra:
    - En el Script1 usaremos el archivo ***VGGFace***
    - En el Script2 usaremos el archivo ***62_26***
- *Clasificador*: para etiquetar las emociones a partir de la featurización. Dependiendo del script usaremos una u otra:
    - En el Script1 usaremos el archivo ***SVM_68_93***
    - En el Script2 usaremos el archivo ***RF 62,26 Primera densa***

- *Imágenes*: carpeta con las imágenes que queremos probar en la demo (podéis poner las imágenes que queráis, os incluimos las que hemos probado en la memoria para que podáis verificar las predicciones).

**Todas las carpetas contienen ya todos los elementos para ser ejecutadas, simplemente hay que lanzar *codigo.py.* Para pasar entre las distintas imágenes, simplemente hay que cerrarlas (equis en la parte superior derecha).**

**Los archivos *codigo.py* están pensados para ser ejecutados en el entorno *PythonGPU* (explicado en el apartado *Entornos*). En *PythonCPU* también pueden ser ejecutados y se ven las predicciones, pero salta un error al final.**

## Código:

**Carpeta**: *Codigos*

En esta carpeta adjuntamos todos los códigos que hemos utilizado para la realización del proyecto. Todos ellos están en Jupyter, y se han ejecutado con el entorno que os indicamos a continuación. Suponemos que no vais a ejecutar la mayor parte el código, pero **parte de los códigos son necesarios para el apartado de comprobar la precisión.** Además, ya que os los proporcionamos, queremos que podáis probar también el entrenamiento si queréis. No queremos que penséis que los modelos han aparecido de forma mágica:

**Modelo 1** (correspondiente a *Script1*):

- *Clasificador red preentrenada.ipynb* (PythonCPU): en él cargamos la red preentrenada (VGGFace2) y ajustamos los diferentes clasificadores.

**Modelo 2** (correspondiente a *Script2*):

- *Entrenamiento base CNN.ipynb* (PythonGPU): en él entrenamos nuestras propias CNN que empleamos después como extractor de features.
- *CNN + SVM.ipynb* (PythonGPU): aquí cargamos la CNN entrenada en el fichero anterior y ajustamos una SVM como clasificador.
- *CNN + RF.ipynb* (PythonGPU): idéntico al anterior pero con un Random Forest como clasificador.
- *CNN + KNN.ipynb* (PythonGPU): idéntico a los dos anteriores pero con un K-nearest neighbours
- *Detector caras extractor+clasificador.ipynb* (PythonGPU): es la versión Notebook del archivo *codigo.py* del Script2.

Además de el código, se esperan las siguientes cosas en el mismo directorio, con la siguiente estructura.

- Datos de entrenamiento. Son los datos del conjunto FER2013, que utilizamos para entrenar. Se espera que haya una carpeta llamada *Data*, que contenga 2 subcarpetas: *train* y *test*, con los datos de entrenamiento y prueba respectivamente. **Os mandamos**

**de nuevo los datos, puesto que hemos modificado las imágenes de entrenamiento eliminando algunas corruptas.**

**La carpeta de test no la hemos modificado**, pero sentíos libres de eliminar la que os proporcionamos y coger la que nos pasasteis por Drive limpia para aseguraros de que no la hemos modificado (ya que os pasamos la de train, también para vuestra comodidad hemos incluido la de test).

- Extractores/Clasificadores en el caso de que queramos utilizar un modelo que ya hemos entrenado nosotros. Os proveemos de 4, dos extractores y dos clasificadores, que van en los archivos (y se corresponden con las propuestas 1 y 2):
  - VGGFace, extractor del modelo 1
  - SVM_68_93, clasificador del modelo 1
  - 62_26, extractor del modelo 2
  - RF 62,26 Primera densa, clasificador del modelo 2
- Carpeta *Detectores*, con el archivo *haarcascade_frontalface_default.xml* para detectar las caras (solo para el archivo *Detector caras extractor+clasificador.ipynb*).
- **Para evitar que tengáis que preprocesar todas las imágenes para verificar los resultados del Script1, os pasamos el archivo *emb_data*, con el preprocesado ya hecho por nosotros y guardado** (tarda alrededor de 1h).

En resumen, **la jerarquía esperada es la siguiente**:

```
Directorio del proyecto
        |---- Clasificador red preentrenada.ipynb
        |---- Entrenamiento base CNN.ipynb
        |---- CNN + SVM.ipynb
        |---- CNN + RF.ipynb
        |---- CNN + KNN.ipynb
        |---- Detector caras extractor+clasificador.ipynb
        |---- Data
        |       |---- Train
        |       |       |---- Fotografías de train del dataset proporcionado (modificadas)
        |       |---- Test
        |               |---- Fotografías de test del dataset (sin modificar)
        |---- VGGFace
        |---- SVM_68_93
        |---- 62_26
        |---- RF 62,26 Primera densa
        |---- Detectores
        |       |----haarcascade_frontalface_default.xml
        |---- emb_data
```

## Módulo que sea capaz, dado un conjunto de imágenes, el porcentaje de acierto, tanto de forma desagregada por clases como de forma agregada.

Estos módulos están incorporados en algunos de los códigos del apartado anterior. De hecho, **a pesar de que los códigos estuviesen pensados tanto para entrenar un modelo como para ver sus métricas, os los hemos dejado especialmente preparados para que solo sirvan para ver los resultados (con el resto de código comentado)**. De la jerarquía anterior solo necesitaréis los archivos del color correspondiente.

**Modelo 1**

Se corresponde a los archivos en amarillo. Tenéis que ejecutar el Notebook ***Clasificador red preentrenada.ipynb***. Además, necesitaréis los archivos *VGGFace*, *SVM_68_93* y *emb_data* (para ahorraros el paso de las imágenes por el modelo), **además de las *imágenes***.

**Evidentemente, si cargáis las imágenes del archivo emb_data no necesitáis las imágenes, puesto que ya las hemos pasado nosotros por la CNN y solo hay que pasarlas por el clasificador. Si queréis aseguraros de que las imágenes de test no han sido alteradas, tendréis que volver a pasarlas por el modelo. Para hacer esto, simplemente tenéis que descomentar la celda donde se carga X_test (indicado en el propio notebook).**

**Nosotros lo hemos ejecutado con el entorno PythonCPU.**

**Modelo 2**

Se corresponde a los archivos en verde. Tenéis que **ejecutar el Notebook *CNN+RF.ipynb***. Además, necesitaréis los archivos *62_26*, *RF 62,26 Primera densa* y las imágenes de *test*, en la carpeta correspondiente.

**Nosotros lo hemos ejecutado con el entorno PythonGPU, pero para hacer las pruebas de precisión se puede probar también con el entorno PythonCPU.**

## Entornos

**Carpeta**: *Entornos*

Como os hemos indicado anteriormente, hemos utilizado dos entornos:

- **PythonGPU**: para entrenar los modelos en la GPU
- **PythonCPU**: para entrenar los modelos en la CPU

Ambos entornos son exactamente iguales, salvo por las librerías de aceleración GPU que usamos en el primero. Como los códigos son algo delicados en lo que a versiones se refiere, os indicamos para cada código en qué entorno lo hemos ejecutado, y las consideraciones más importantes. Las versiones de las librerías más importantes están especificadas en la memoria.

**Para evitar problemas, os incluimos sendos archivos environment.yml, para que podáis exportar fácilmente ambos entornos. Los podéis encontrar dentro de las carpetas de *Entornos.***

**Nosotros utilizamos Anaconda. Para importar el entorno en Anaconda, simplemente tenéis que seleccionar a la izquierda la pestaña *Environments*, después la opción *Import* debajo y seleccionar como origen el archivo proporcionado.**

Esperamos que todo haya quedado lo suficientemente claro. No dudéis en preguntarnos cualquier duda / consulta técnica que tengáis.

Un saludo

Equipo YOLO