- Carlos Fernández de la Torre 10/11/2019

Laboratorio: Convolutional Neural Networks

En este laboratorio, vamos a trabajar con Convolutional Neural Networks para resolver un problema de clasificación de imágenes. En particular, vamos a clasificar imágenes de personajes de la conocida serie de los Simpsons.

Como las CNN profundas son un tipo de modelo bastante avanzado y computacionalmente costoso, se recomienda hacer la práctica en Google Colaboratory con soporte para GPUs. En este enlace se explica cómo activar un entorno con GPUs. *Nota: para leer las imágenes y estandarizarlas al mismo tamaño se usa la librería opencv. Esta Íbrería está ya instalada en el entorno de Colab, pero si trabajáis de manera local tendréis que instalarla.*

El dataset a utilizar consiste en imágenes de personajes de los Simpsons extraídas directamente de capítulos de la serie. Este dataset ha sido recopilado por Alexandre Attia y es más complejo que el dataset de Fashion MNIST que hemos utilizado hasta ahora. Aparte de tener más clases (vamos a utilizar los 18 personajes con más imágenes), los personajes pueden aparecer en distintas poses, en distintas posiciones de la imagen o con otros personajes en pantalla (si bien el personaje a clasificar siempre aparece en la posición predominante).

El dataset de training puede ser descargado desde aquí:

Training data (~500MB)

Por otro lado, el dataset de test puede ser descargado de aquí:

Test data (~10MB)

Antes de empezar la práctica, se recomienda descargar las imágenes y echarlas un vistazo.

Nueva sección

Carga de los datos

```
import cv2
import os
import numpy as np
import keras
import matplotlib.pyplot as plt
import glob
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
# Primero, bajamos los datos de entrenamiento
keras.utils.get_file(fname="simpsons_train.tar.gz",
                     origin="https://onedrive.live.com/download?cid=C506CF0A4F373B0F&resid=C506CF0A4F373B0F%219337&authkey=AMzI92bJPx

# Descomprimimos el archivo
!tar -xzf /root/.keras/datasets/simpsons_train.tar.gz -C /root/.keras/datasets
```

```
# Hacemos lo mismo con los datos de test
keras.utils.get_file(fname="simpsons_test.tar.gz",
                     origin="https://onedrive.live.com/download?cid=C506CF0A4F373B0F&resid=C506CF0A4F373B0F%219341&authkey=ANnjK3Uq1F
!tar -xzf /root/.keras/datasets/simpsons_test.tar.gz -C /root/.keras/datasets
```

```
Downloading data from https://onedrive.live.com/download?cid=C506CF0A4F373B0F&resid=C506CF0A4F373B0F%219337&authkey=AMzI92bJPx8S
523796480/523789527 [==============================] - 1646s 3us/step
Downloading data from https://onedrive.live.com/download?cid=C506CF0A4F373B0F&resid=C506CF0A4F373B0F%219341&authkey=ANnjK3Uq1Fhu
10665984/10658925 [==============================] - 0s 0us/step
```

```
# Esta variable contiene un mapeo de número de clase a personaje.
# Utilizamos sólo los 18 personajes del dataset que tienen más imágenes.
MAP_CHARACTERS = {
    0: 'abraham_grampa_simpson', 1: 'apu_nahasapeemapetilon', 2: 'bart_simpson',
    3: 'charles_montgomery_burns', 4: 'chief_wiggum', 5: 'comic_book_guy', 6: 'edna_krabappel',
    7: 'homer_simpson', 8: 'kent_brockman', 9: 'krusty_the_clown', 10: 'lisa_simpson',
    11: 'marge_simpson', 12: 'milhouse_van_houten', 13: 'moe_szyslak',
    14: 'ned_flanders', 15: 'nelson_muntz', 16: 'principal_skinner', 17: 'sideshow_bob'
}

# Vamos a standarizar todas las imágenes a tamaño 64x64
IMG_SIZE = 64


def load_train_set(dirname, map_characters, verbose=True):
    """Esta función carga los datos de training en imágenes.

    Como las imágenes tienen tamaños distintas, utilizamos la librería opencv
    para hacer un resize y adaptarlas todas a tamaño IMG_SIZE x IMG_SIZE.

    Args:
        dirname: directorio completo del que leer los datos
        map_characters: variable de mapeo entre labels y personajes
        verbose: si es True, muestra información de las imágenes cargadas

    Returns:
        X, y: X es un array con todas las imágenes cargadas con tamaño
              IMG_SIZE x IMG_SIZE
```

```
                IMG_SIZE x IMG_SIZE
            y es un array con las labels de correspondientes a cada imagen
    """
    X_train = []
    y_train = []
    for label, character in map_characters.items():
        files = os.listdir(os.path.join(dirname, character))
        images = [file for file in files if file.endswith("jpg")]
        if verbose:
          print("Leyendo {} imágenes encontradas de {}".format(len(images), character))
        for image_name in images:
            image = cv2.imread(os.path.join(dirname, character, image_name))
            X_train.append(cv2.resize(image,(IMG_SIZE, IMG_SIZE)))
            y_train.append(label)
    return np.array(X_train), np.array(y_train)


def load_test_set(dirname, map_characters, verbose=True):
    """Esta función funciona de manera equivalente a la función load_train_set
    pero cargando los datos de test."""
    X_test = []
    y_test = []
    reverse_dict = {v: k for k, v in map_characters.items()}
    for filename in glob.glob(dirname + '/*.*'):
        char_name = "_".join(filename.split('/')[-1].split('_')[:-1])
        if char_name in reverse_dict:
            image = cv2.imread(filename)
            image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
            X_test.append(image)
            y_test.append(reverse_dict[char_name])
    if verbose:
        print("Leídas {} imágenes de test".format(len(X_test)))
    return np.array(X_test), np.array(y_test)



# Cargamos los datos. Si no estás trabajando en colab, cambia los paths por
# los de los ficheros donde hayas descargado los datos.
DATASET_TRAIN_PATH_COLAB = "/root/.keras/datasets/simpsons"
```

```
DATASET_TEST_PATH_COLAB = "/root/.keras/datasets/simpsons_testset"

X, y = load_train_set(DATASET_TRAIN_PATH_COLAB, MAP_CHARACTERS)
X_t, y_t = load_test_set(DATASET_TEST_PATH_COLAB, MAP_CHARACTERS)
```

```
Leyendo 913 imágenes encontradas de abraham_grampa_simpson
Leyendo 623 imágenes encontradas de apu_nahasapeemapetilon
Leyendo 1342 imágenes encontradas de bart_simpson
Leyendo 1193 imágenes encontradas de charles_montgomery_burns
Leyendo 986 imágenes encontradas de chief_wiggum
Leyendo 469 imágenes encontradas de comic_book_guy
Leyendo 457 imágenes encontradas de edna_krabappel
Leyendo 2246 imágenes encontradas de homer_simpson
Leyendo 498 imágenes encontradas de kent_brockman
Leyendo 1206 imágenes encontradas de krusty_the_clown
Leyendo 1354 imágenes encontradas de lisa_simpson
Leyendo 1291 imágenes encontradas de marge_simpson
Leyendo 1079 imágenes encontradas de milhouse_van_houten
Leyendo 1452 imágenes encontradas de moe_szyslak
Leyendo 1454 imágenes encontradas de ned_flanders
Leyendo 358 imágenes encontradas de nelson_muntz
Leyendo 1194 imágenes encontradas de principal_skinner
Leyendo 877 imágenes encontradas de sideshow_bob
Leídas 890 imágenes de test
```

```
# Vamos a barajar aleatoriamente los datos. Esto es importante ya que si no
# lo hacemos y, por ejemplo, cogemos el 20% de los datos finales como validation
# set, estaremos utilizando solo un pequeño número de personajes, ya que
# las imágenes se leen secuencialmente personaje a personaje.
perm = np.random.permutation(len(X))
X, y = X[perm], y[perm]
```

Entregable

Utilizando Convolutional Neural Networks con Keras, entrenar un clasificador que sea capaz de reconocer personajes en imágenes de los Simpsons con una accuracy en el dataset de test de **85%**. Redactar un informe analizando varias de las alternativas probadas y los resultados obtenidos.

A continuación se detallan una serie de aspectos orientativos que podrían ser analizados en vuestro informe (no es necesario tratar todos ellos ni mucho menos, esto son ideas orientativas de aspectos que podéis explorar):

- Análisis de los datos a utilizar.
- Análisis de resultados, obtención de métricas de *precision* y *recall* por clase y análisis de qué clases obtienen mejores o peores resultados.
- Análisis visual de los errores de la red. ¿Qué tipo de imágenes o qué personajes dan más problemas a nuestro modelo?
- Comparación de modelos CNNs con un modelo de Fully Connected para este problema.
- Utilización de distintas arquitecturas CNNs, comentando aspectos como su profundidad, hiperparámetros utilizados, optimizador, uso de técnicas de regularización, *batch normalization*, etc.
- [*algo más difícil*] Utilización de *data augmentation*. Esto puede conseguirse con la clase ImageDataGenerator de Keras.

Notas:

- Recuerda partir los datos en training/validation para tener una buena estimación de los valores que nuestro modelo tendrá en los datos de test, así como comprobar que no estamos cayendo en overfitting. Una posible partición puede ser 80 / 20.
- No es necesario mostrar en el notebook las trazas de entrenamiento de todos los modelos entrenados, si bien una buena idea seria guardar gráficas de esos entrenamientos para el análisis. Sin embargo, **se debe mostrar el entrenamiento completo del mejor modelo obtenido y la evaluación de los datos de test con este modelo**.
- Las imágenes **no están normalizadas**. Hay que normalizarlas como hemos hecho en trabajos anteriores.
- El test set del problema tiene imágenes un poco más "fáciles", por lo que es posible encontrarse con métricas en el test set bastante mejores que en el training set.

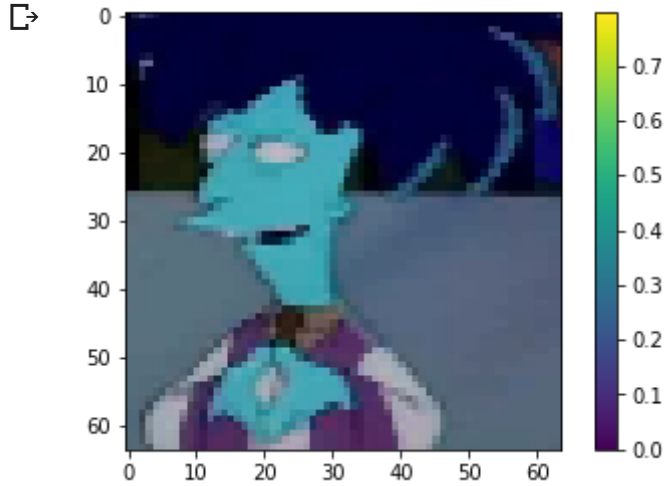
Normalización de las imágenes.

```
# Preprocessing normalización de las imágenes
# X, y = load_train_set(DATASET_TRAIN_PATH_COLAB, MAP_CHARACTERS)
# X_t, y_t = load_test_set(DATASET_TEST_PATH_COLAB, MAP_CHARACTERS)
X = X / 255.0
X_t = X_t / 255.0
```

Análisis de los datos a utilizar.

```
def visualize_example(x):
  plt.figure()
  plt.imshow(x)
  plt.colorbar()
  plt.grid(False)
  plt.show()
```

```
visualize_example(X[70]) # Visualiza la imagen normalizada con valores de 0 a 1
print(MAP_CHARACTERS[y[70]]) # Acceso al diccionario
print(X[70].shape[0:3]) # Dimensiones de la imagen tras resize con los 3 canales RGB
```




```
sideshow_bob
(64, 64, 3)
```

```
#batch_size=128
num_classes=18
epochs=5
img_rows,img_cols=IMG_SIZE,IMG_SIZE
input_shape=(img_rows,img_cols,3) #(64,64,3)
```

```
# convert class vectors to binary class matrices
y=keras.utils.to_categorical(y,num_classes)
y_t=keras.utils.to_categorical(y_t,num_classes)
```

```
X.shape
```

```
(18992, 64, 64, 3)
```

```
y.shape
```

```
(18992, 18)
```

Parámetros por defecto de la capa de convolución:

keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)

```
model = keras.Sequential([
keras.layers.Conv2D(32, kernel_size=(3, 3), data_format="channels_last",input_shape=(64, 64, 3), activation='relu'),
keras.layers.MaxPooling2D(pool_size=(2, 2)),
keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
keras.layers.MaxPooling2D(pool_size=(2, 2)),
keras.layers.Conv2D(512, kernel_size=(3, 3), activation='relu'),
keras.layers.MaxPooling2D(pool_size=(2, 2)),
keras.layers.MaxPooling2D(pool_size=(2, 2)),
keras.layers.Flatten(),
keras.layers.Dense(500, activation='relu'),
keras.layers.Dropout(0.5),
keras.layers.Dense(18, activation='softmax')
])
```

```
model.summary()
model.compile(optimizer='adam',
loss='categorical_crossentropy',
```

```
    metrics=['accuracy'])
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_13 (Conv2D)           (None, 62, 62, 32)        896       
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 31, 31, 32)        0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 29, 29, 128)       36992     
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 128)       0         
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 12, 12, 512)       590336    
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 6, 6, 512)         0         
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 3, 3, 512)         0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 4608)              0         
_________________________________________________________________
dense_9 (Dense)              (None, 500)               2304500   
_________________________________________________________________
dropout_5 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 18)                9018      
=================================================================
Total params: 2,941,742
Trainable params: 2,941,742
Non-trainable params: 0
_________________________________________________________________
```

Observamos que el modelo tiene cerca de 3 millones de parámetros, se ha visto una mejora importante al añadir más neuronas en la capa dense_9, otros modelos probados con 200 neuronas solo alcanzaban un 65% de precisión. Hacemos dos máxpooling seguidos para reducir de 600000 a 4600 las neuronas en la capa flatten_5 y que no se dispare el número de parámetros pudiendo así aumentar el número de neuronas de la capa dense_9 que hemos observado mejora la precisión. Se usó categorical crossentropy porque las variables dependientes se transformaron en categorías y se usa la función de activation softmax en la capa de salida.

Comparación de modelos CNNs con un modelo de Fully Connected para este problema.

El número de neuronas en la capa de entrada hace inviable entrenar un modelo Fully Connected que no usa capas convolucionales. Los filtros convolucionales permiten extraer características espaciales de las imágenes, frente a un modelo Fully Connected en el que no se aprovecha esta posibilidad.

Entrenamiento completo del mejor modelo.

División de los datos en training/validation 80/20 para tener una buena estimación de los valores que nuestro modelo tendrá en los datos de test, así como comprobar que no estamos cayendo en overfitting.

```
history=model.fit(X, y, epochs=7, validation_split = 0.2)
```

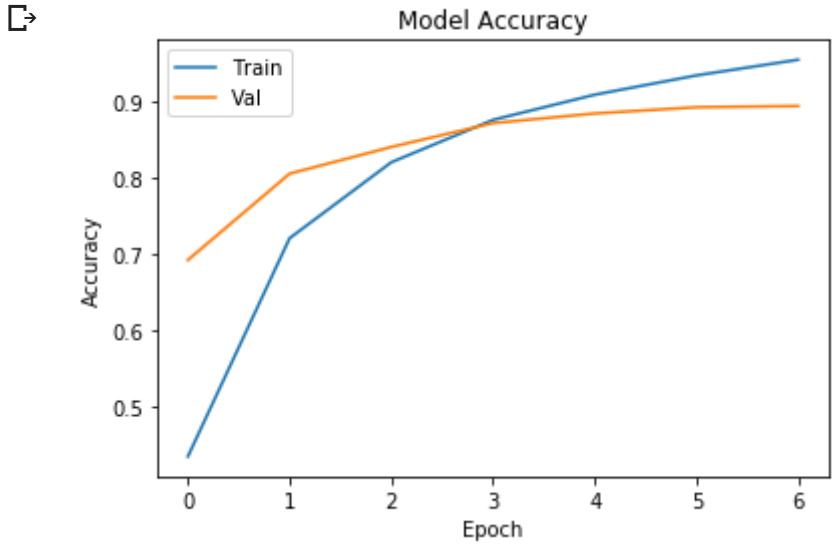
```
Train on 15193 samples, validate on 3799 samples
Epoch 1/7
15193/15193 [==============================] - 14s 937us/step - loss: 1.8561 - acc: 0.4347 - val_loss: 1.0270 - val_acc: 0.6923
Epoch 2/7
15193/15193 [==============================] - 13s 880us/step - loss: 0.9281 - acc: 0.7209 - val_loss: 0.6632 - val_acc: 0.8055
Epoch 3/7
15193/15193 [==============================] - 13s 885us/step - loss: 0.6003 - acc: 0.8204 - val_loss: 0.5339 - val_acc: 0.8405
Epoch 4/7
15193/15193 [==============================] - 13s 875us/step - loss: 0.4156 - acc: 0.8761 - val_loss: 0.4371 - val_acc: 0.8718
Epoch 5/7
15193/15193 [==============================] - 13s 884us/step - loss: 0.2970 - acc: 0.9090 - val_loss: 0.3988 - val_acc: 0.8844
Epoch 6/7
15193/15193 [==============================] - 13s 887us/step - loss: 0.2111 - acc: 0.9344 - val_loss: 0.3819 - val_acc: 0.8926
Epoch 7/7
15193/15193 [==============================] - 13s 882us/step - loss: 0.1464 - acc: 0.9549 - val_loss: 0.4178 - val_acc: 0.8942
```

Análisis visual de los errores de la red.

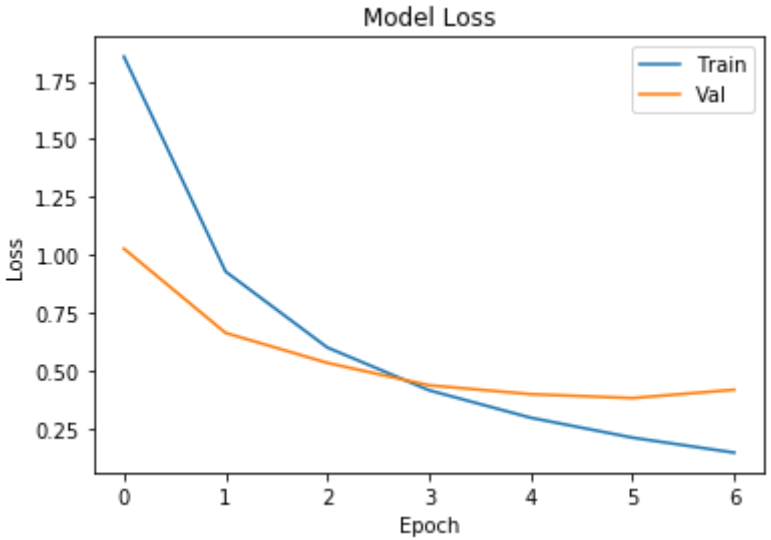
```
def plot_acc(history, title="Model Accuracy"):
  """Imprime una gráfica mostrando la accuracy por epoch obtenida en un entrenamiento"""
  plt.plot(history.history['acc'])
  plt.plot(history.history['val_acc'])
  plt.title(title)
  plt.ylabel('Accuracy')
```

```
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()
  def plot_loss(history, title="Model Loss"):
    """Imprime una gráfica mostrando la pérdida por epoch obtenida en un entrenamiento"""
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(title)
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper right')
    plt.show()


  plot_acc(history)
```




```
  plot_loss(history)
```



Viendo las gráficas observamos que con 5 épocas las líneas de loss y accuracy se vuelven prácticamente horizontales, por eso no seguimos entrenando.

Evaluación de los datos de test con este modelo.

```
model.evaluate(X_t, y_t, batch_size=32, verbose=1)
```

```
890/890 [==============================] - 0s 464us/step
[0.16797978566603713, 0.943820223915443]
```

Observamos que sobre los datos de test el loss es 0.17 y la precisión de un 94%

Almacenamos todas las predicciones para los datos de test.

```
prediccion=model.predict(X_t, batch_size=32, verbose=1)  # Almacenamos todas las predicciones para los datos de test
```

```
print(prediccion)
prediccion.shape
```

```
[[1.0436461e-05 4.0393564e-09 8.6248328e-05 ... 4.1252917e-09
  3.8302916e-09 1.1701950e-06]
 [4.2870491e-07 7.4545491e-12 1.6985687e-05 ... 2.1730613e-13
  1.3417210e-10 2.1972285e-11]
 [1.3970128e-05 1.3535890e-04 1.8186621e-04 ... 1.3414168e-04
  3.1769969e-02 2.9720401e-05]
 ...
 [5.3526263e-19 1.4045362e-21 2.8902658e-14 ... 2.2493529e-22
  4.5590745e-24 1.3800931e-23]
 [8.3708455e-04 3.1794378e-04 5.7568890e-03 ... 9.0936762e-01
  3.1456554e-03 4.5966716e-03]
 [1.4757951e-07 1.6051710e-10 1.5710880e-07 ... 1.0878345e-05
  1.0580720e-05 2.7247069e-07]]
(890, 18)
```

```
print(prediccion[0]) # Ejemplo de predicción para el primer elemento del test.
predicho = np.argmax(prediccion[0])  # Nos quedamos con la posición del valor máximo de las estimaciones de probabilidad para cada un
print(predicho)
print(MAP_CHARACTERS[predicho]) # Acceso al diccionario para mostrar el nombre del personaje predicho.
```

```
[1.0436461e-05 4.0393564e-09 8.6248328e-05 9.9808145e-01 4.5345655e-06
 6.2164123e-05 5.6259819e-09 1.7311788e-03 4.5602128e-10 4.9568888e-07
 2.0808417e-05 8.2596607e-08 4.4503784e-07 8.5214401e-07 4.0693501e-10
 4.1252917e-09 3.8302916e-09 1.1701950e-06]
3
charles_montgomery_burns
```

```
y_t.shape
```

```
(890, 18)
```

Análisis de resultados, obtención de métricas de precision y recall por clase y análisis de qué clases obtienen mejores o peores resultados.

```
from sklearn.metrics import confusion_matrix, classification_report
informe = classification_report(np.argmax(y_t,axis=1), np.argmax(prediccion,axis=1))
print(informe)
```

```
              precision    recall  f1-score   support

           0       0.97      0.79      0.87        48
           1       0.96      1.00      0.98        50
           2       0.94      0.94      0.94        50
           3       0.90      0.90      0.90        48
           4       1.00      0.94      0.97        50
           5       0.96      0.92      0.94        49
           6       0.91      0.98      0.94        50
           7       0.85      1.00      0.92        50
           8       0.92      0.98      0.95        50
           9       0.98      1.00      0.99        50
          10       0.92      0.92      0.92        50
          11       0.98      0.98      0.98        50
          12       0.92      0.94      0.93        49
          13       0.94      0.90      0.92        50
          14       0.94      0.98      0.96        49
          15       0.98      0.90      0.94        50
          16       1.00      0.94      0.97        50
          17       0.96      0.98      0.97        47

    accuracy                           0.94       890
   macro avg       0.95      0.94      0.94       890
weighted avg       0.95      0.94      0.94       890
```

Vemos que la menor precisión es del 85% al clasificar las imágenes de Homer Simpson (número 7), mientras que la mayor precisión se consigue al clasificar las imágenes correspondientes a Chief Wiggum (número 4) y Principal Skinner (número 16).

Mejoras

Podríamos hacer data augmentation mediante el preprocesamiento de las imágenes predichas con menor precisión usando la clase

ImageDataGenerator. Para ello podríamos fijarnos si están descentradas, oscuras etc. realizando transformaciones de rotación, desplazamientos y cambios de brillo según el caso.