



PREDICCIÓN O RECONOCIMIENTO DE GÉNERO DE UNA PERSONA



- **AUTORES:** ALEJANDRO SÁNCHEZ CORREDOR
PABLO RODRÍGUEZ ROYO
ROBERTO IBÁÑEZ GÓMEZ
- **ASIGNATURA:** VISIÓN ARTIFICIAL Y RECONOCIMIENTO DE PATRONES
- **FECHA:** 5 DE ENERO DE 2024

Tabla de contenido

1.	Contexto.....	3
2.	Datos y Librerías	3
3.	Análisis exploratorio de datos	5
3.1.	Conclusiones Análisis Exploratorio de datos	8
4.	Preparación de datos "finales"	8
5.	Pre - procesamiento.....	9
5.1.	Aumento de datos.....	12
5.2.	Callbacks.....	14
6.	Bibliografía	15

1. Contexto

El objetivo de este proyecto la detección de género utilizando técnicas de aprendizaje automático y procesamiento de imágenes. El objetivo principal es desarrollar un modelo capaz de **identificar el género** (masculino o femenino) a partir de imágenes faciales.

Hemos obtenido un dataset que contiene **imágenes faciales** etiquetadas con el género correspondiente. Cada imagen está asociada con una etiqueta de la variable clase **"Male"**, muchas veces nos referiremos a los registros de la variable clase como **"Male"** o **"No male"**.

Para conseguir nuestro objetivo, hemos optado por utilizar técnicas de aprendizaje profundo, específicamente utilizando **redes neuronales convolucionales (CNN)**. Entrenaremos un modelo CNN utilizando **TensorFlow/Keras** para aprender patrones y características que distingan entre los géneros a partir de las imágenes faciales.

2. Datos y Librerías

A lo largo de este proyecto vamos a utilizar la base de datos de CelebFaces para realizar una predicción del género de un famoso a partir de una imagen. Debido a limitación de memoria, tiempo, etc, no vamos a poder entrenar todas las imágenes del dataframe, por ello, solo trabajaremos con una muestra.

En próximos apartados realizaremos un análisis exploratorio de la base de datos, pero para introducir, esta base de datos contiene un archivo con todas las imágenes de los famosos. Por otro lado, tenemos dos .csv importantes que vamos a utilizar.

- **"list_attr_celeba.csv"**: El cual contiene todos los atributos, que serán binarios, representando con un 1 si una imagen contiene dicha característica y un 0, en caso contrario.
- **"list_eval_partition.csv"**: El cual contiene como debemos partir la base de datos en, entrenamiento, validación, y prueba.

Estas son las librerías que vamos a utilizar:

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import random
from keras.preprocessing.image import load_img, img_to_array, ImageDataGenerator
from keras.applications.inception_v3 import preprocess_input, InceptionV3
import numpy as np
from keras import utils
import cv2
from keras.layers import Dropout, Dense, GlobalAveragePooling2D, BatchNormalization
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.models import Model
from keras.optimizers import SGD, Adam
import tensorflow as tf
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

✓ 0.0s

```
raiz_base = 'C:\\Users\\Alejandro\\proyecto_vision\\'
imagenes_totales = os.path.join(raiz_base, 'img_align_celeba\\')
df = pd.read_csv(os.path.join(raiz_base, 'list_attr_celeba.csv'))
```

En “df” tendremos toda la base de datos “grande”, no la que vamos a entrenar finalmente. Al final del análisis exploratorio de datos explicaremos por qué esta base de datos nos ha sido útil.

- Ejemplo del dataset “grande”:

df.sample(5)

✓ 0.0s

	5_o_Clock_Shadow	Arched_Eyebrows	Attractive	Bags_Under_Eyes	Bald	Bangs	Big_Lips	Big_Nose	Black_Hair	Blond_Hair	...	Sideburns	Smiling	Str
image_id														
166708.jpg	0	0	1	0	0	0	0	0	0	0	...	0	0	
180506.jpg	1	0	0	1	0	0	0	1	1	0	...	0	1	
186986.jpg	0	0	1	0	0	0	1	1	1	0	...	0	1	
138290.jpg	0	1	0	0	0	1	0	0	0	0	...	0	1	
037570.jpg	0	0	0	1	0	0	0	0	1	0	...	0	0	

El paso final antes de comenzar el análisis exploratorio de datos será particionar para realizar el análisis exploratorio de datos solo con la parte de entrenamiento para trabajar de manera honesta y más adelante no cometer fuga de datos.

```
def partition_EDA(df, partition_path): # Dividimos en entrenamiento, validación o test, estas particiones serán utilizadas solo
    # para en análisis exploratorio de datos

    df_partition = pd.read_csv(partition_path) # Cargamos el .csv de las particiones

    # Ahora partimos segun el .csv de particiones y juntamos con el df original para tener todas las columnas
    df_merged_train = pd.merge(df, df_partition[df_partition['partition'] == 0], on='image_id')
    df_merged_train.set_index('image_id', inplace=True)

    df_merged_val = pd.merge(df, df_partition[df_partition['partition'] == 1], on='image_id')
    df_merged_val.set_index('image_id', inplace=True)

    df_merged_test = pd.merge(df, df_partition[df_partition['partition'] == 2], on='image_id')
    df_merged_test.set_index('image_id', inplace=True)

    return df_merged_train, df_merged_val, df_merged_test

✓ 0.0s
```

```
def divide_X_y(dataframe, variable_objetivo='Male'):

    X = dataframe.drop(columns=[variable_objetivo])
    y = dataframe[variable_objetivo]

    return X, y

✓ 0.0s
```

```
csv_path = 'C:\\Users\\Alejandro\\practica_vision_ens\\list_eval_partition.csv'
train_data, val_data, test_data = partition_EDA(df, csv_path)
```

```
train_data.drop(columns=['partition'], inplace=True)
val_data.drop(columns=['partition'], inplace=True)
test_data.drop(columns=['partition'], inplace=True)
```

```
x_train_expl, y_train_expl = divide_X_y(train_data)
```

Ahora que tenemos nuestros subconjuntos de datos de entrenamiento para poder realizar el análisis exploratorio de datos, vamos con ello.

3. Análisis exploratorio de datos

```

for i, columna in enumerate(x_train_expl.columns.values):
    print(i, columna)
✓ 0.0s
0 5_o_Clock_Shadow
1 Arched_Eyebrows
2 Attractive
3 Bags_Under_Eyes
4 Bald
5 Bangs
6 Big_Lips
7 Big_Nose
8 Black_Hair
9 Blond_Hair
10 Blurry
11 Brown_Hair
12 Bushy_Eyebrows
13 Chubby
14 Double_Chin
15 Eyeglasses
16 Goatee
17 Gray_Hair
18 Heavy_Makeup
19 High_Cheekbones
20 Mouth_Slightly_Open
21 Mustache
22 Narrow_Eyes
23 No_Beard
24 Oval_Face
...
35 Wearing_Lipstick
36 Wearing_Necklace
37 Wearing_Necktie
38 Young

```

Nuestro subconjunto de datos “X” contiene **39 variables**.

```

y_train_expl.values
✓ 0.0s
array([0, 0, 1, ..., 1, 1, 0], dtype=int64)

```

Nuestro subconjunto de datos “y” solo tendrá la variable “**Male**”

Ahora procedemos a realizar un análisis de los estadísticos:

`x_train_expl.describe()`
✓ 0.1s

	5_o_Clock_Shadow	Arched_Eyebrows	Attractive	Bags_Under_Eyes	Bald	Bangs	Big_Lips	Big_Nose	Black_Hair	Blond_H
count	162770.000000	162770.000000	162770.000000	162770.000000	162770.000000	162770.000000	162770.000000	162770.000000	162770.000000	162770.000000
mean	0.111673	0.265884	0.513627	0.204460	0.022811	0.151656	0.240910	0.235553	0.239024	0.149000
std	0.314965	0.441804	0.499816	0.403308	0.149302	0.358688	0.427637	0.424345	0.426489	0.356000
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

`y_train_expl.describe()`
✓ 0.0s

count162770.000000

mean0.419371

std0.493458

min0.000000

25%0.000000

50%0.000000

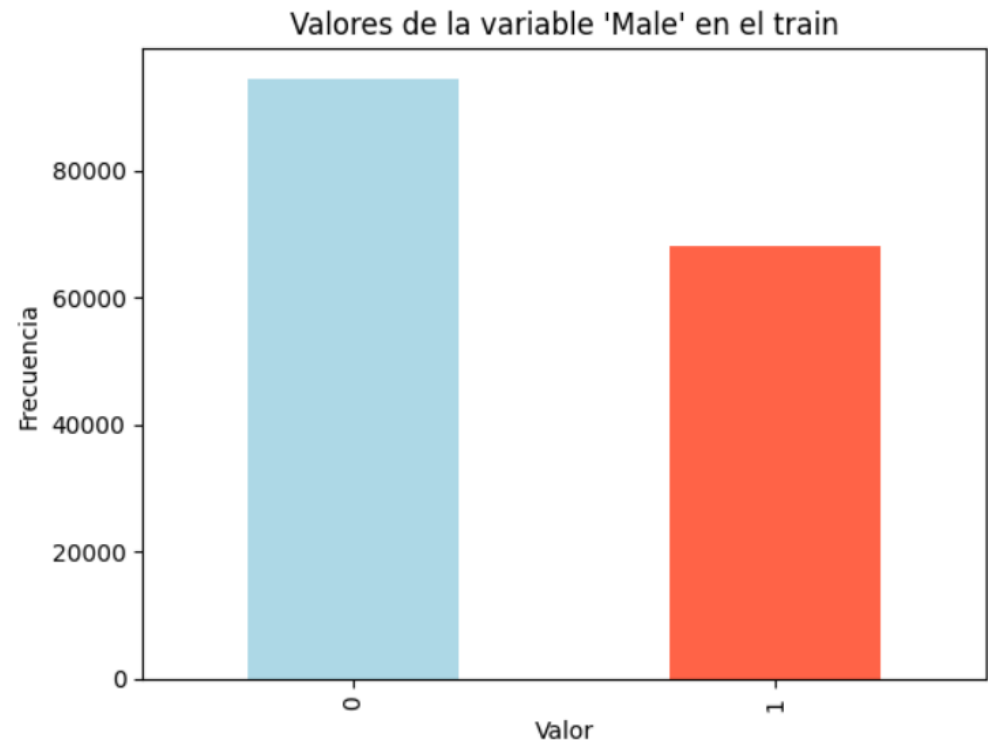
75%1.000000

max1.000000

Name: Male, dtype: float64

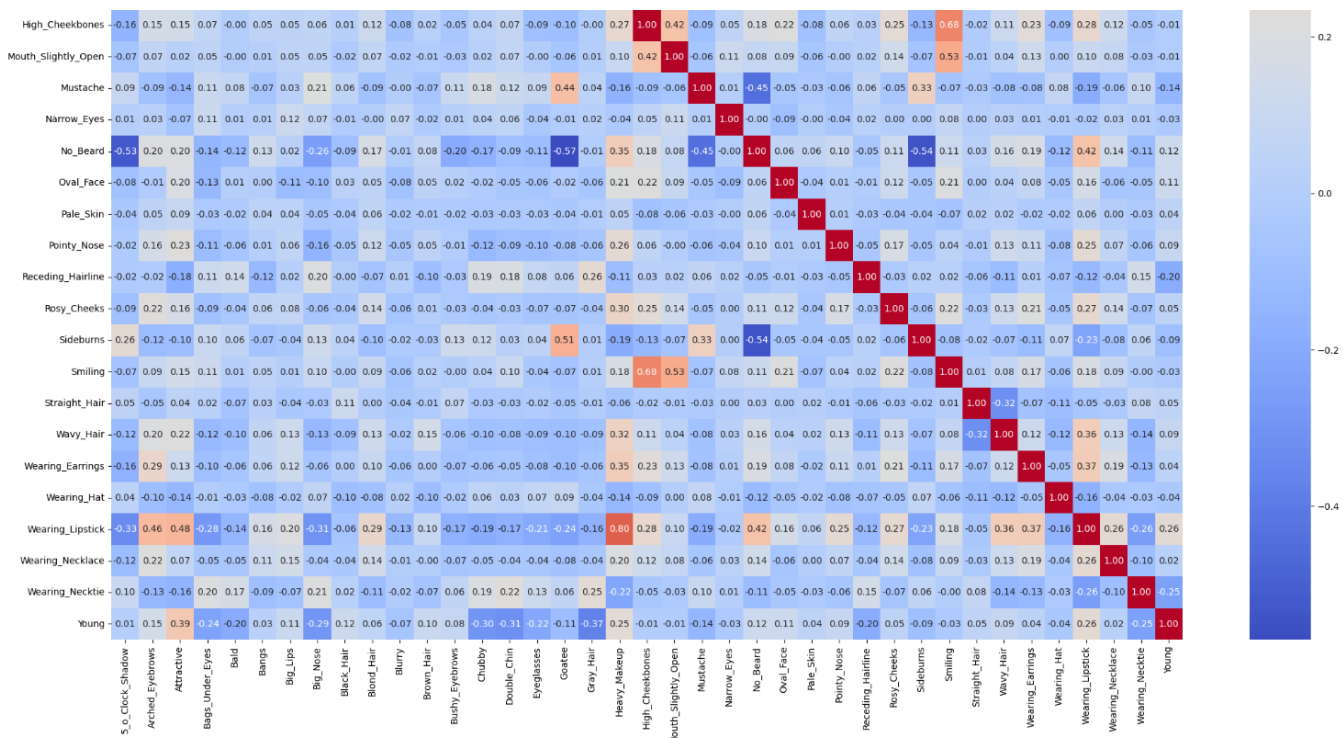
Podemos ver que tenemos **162770** instancias.

A continuación, vamos a realizar un gráfico de barras para comprobar si la variable clase está **balanceada**.








Podemos ver que estamos ante un problema **desbalanceado**, en la que hay más famosas que hombres.

El siguiente paso será realizar una **correlación entre las variables predictoras** para ver que relación hay entre ellas. Enseñaremos solo una pequeña parte ya que la matriz es demasiado grande.



Vemos **interesantes relaciones** como por ejemplo hay una correlación muy fuerte (**0.68**) entre **“High_Cheekbones”** y **“Smiling”**, esto quiere decir que cuanto mientras una persona más esté sonriendo, más altos tendrá los pómulos. Otra correlación muy fuerte entre **“Heavy_makeup”** y **“Wearing_Lipstick”** (**0.80**) que nos dice que las personas que lleven mucho maquillaje, tendrán a llevar pintalabios. Otro tipo de correlación negativa del **-0.54** existe entre **“No_Beard”** y **“Goatee”**, esta última es un tipo de barba lo que significa que, si una persona no tiene barba, lógicamente no puede tener ningún tipo de barba.

Ahora mostraremos unos ejemplos de imágenes y algunos de sus atributos:

<p>Imagen: 134988.jpg</p> <p>Atributos:</p> <p>Male 0</p> <p>Young 1</p> <p>Smiling 0</p> <p>No_Beard 1</p> <p>Heavy_Makeup 0</p> <p>Name: 134988.jpg, dtype: int64</p> 	<p>Imagen: 064983.jpg</p> <p>Atributos:</p> <p>Male 1</p> <p>Young 1</p> <p>Smiling 0</p> <p>No_Beard 1</p> <p>Heavy_Makeup 0</p> <p>Name: 064983.jpg, dtype: int64</p> 	<p>Imagen: 049453.jpg</p> <p>Atributos:</p> <p>Male 0</p> <p>Young 1</p> <p>Smiling 0</p> <p>No_Beard 1</p> <p>Heavy_Makeup 1</p> <p>Name: 049453.jpg, dtype: int64</p> 	<p>Imagen: 154577.jpg</p> <p>Atributos:</p> <p>Male 1</p> <p>Young 1</p> <p>Smiling 0</p> <p>No_Beard 1</p> <p>Heavy_Makeup 0</p> <p>Name: 154577.jpg, dtype: int64</p> 	<p>Imagen: 121231.jpg</p> <p>Atributos:</p> <p>Male 0</p> <p>Young 1</p> <p>Smiling 1</p> <p>No_Beard 1</p> <p>Heavy_Makeup 1</p> <p>Name: 121231.jpg, dtype: int64</p> 
---	---	---	--	---

3.1. Conclusiones Análisis Exploratorio de datos

Nuestro objetivo con este análisis exploratorio de datos ha sido comprender las características presentes en las imágenes de nuestro conjunto de datos y cómo van a ser las imágenes que vamos a entrenar en pasos posteriores, ya que, seguramente un futuro modelo serán este tipo de características las que extraiga de las imágenes.

4. Preparación de datos "finales"

Hasta ahora, hemos estado trabajando con una base de datos que contenía una gran cantidad de variables, más de las que necesitamos para entrenar nuestro modelo. Ahora obtendremos el dataset con el que sí entrenaremos nuestro modelo, el dataset que realmente utilizaremos.

Nos centraremos en las **imágenes** y en la variable **"Male"**, ya que esta última es clave para nuestra tarea de clasificación de género. El conjunto de datos resultante contendrá únicamente las imágenes y la variable objetivo Male, así como la **información de partición** que utilizaremos para dividir nuestros datos en conjuntos de entrenamiento, validación y prueba.

```
def model_partitions(training_samples, validation_samples, test_samples, seed=None): # Dividimos en entrenamiento, validación o test, estas particio

    training_samples_per_class = training_samples // 2 # Para obtener una muestra estratificada, tendremos que tener los mismos casos de Male == 0 q
    validation_samples_per_class = validation_samples // 2
    test_samples_per_class = test_samples // 2

    # Partición para 'Male' == 0
    df_train_male_0 = model_df.loc[(model_df['partition'] == 0) & (model_df['Male'] == 0)].sample(training_samples_per_class, random_state=seed)
    df_val_male_0 = model_df.loc[(model_df['partition'] == 1) & (model_df['Male'] == 0)].sample(validation_samples_per_class, random_state= seed)
    df_test_male_0 = model_df.loc[(model_df['partition'] == 2) & (model_df['Male'] == 0)].sample(test_samples_per_class, random_state= seed)

    # Partición para 'Male' == 1
    df_train_male_1 = model_df.loc[(model_df['partition'] == 0) & (model_df['Male'] == 1)].sample(training_samples_per_class, random_state= seed)
    df_val_male_1 = model_df.loc[(model_df['partition'] == 1) & (model_df['Male'] == 1)].sample(validation_samples_per_class, random_state= seed)
    df_test_male_1 = model_df.loc[(model_df['partition'] == 2) & (model_df['Male'] == 1)].sample(test_samples_per_class, random_state= seed)

    # Unir los DataFrames para cada conjunto
    df_train = pd.concat([df_train_male_0, df_train_male_1])
    df_val = pd.concat([df_val_male_0, df_val_male_1])
    df_test = pd.concat([df_test_male_0, df_test_male_1])

    return df_train, df_val, df_test
```

Este es un método importante con el que obtendremos el dataframe con el que vamos a entrenar el modelo, bueno, no del todo, posteriormente la aplicaremos unos pasos de preprocesamiento. Además, como mencionamos al principio de la práctica, no vamos a coger todas las imágenes, solo unas **muestras**. Ahora partiremos este conjunto de datos en **entrenamiento, test y validación**. También conseguimos obtener un conjunto de datos **estratificado** (tener los mismos casos de Male == 0 que de Male ==1) lo que será importante para intentar obtener un **buen rendimiento** en nuestro modelo.


```

training_samples = 14000
validation_samples = 6000
test_samples = 6000
#batch_size = 32
batch_size = 128
df_train, df_val, df_test = model_partitions(training_samples, validation_samples, test_samples, seed=120)

```

PONER BATCH SIZE FINAL!!

Df_train, df_val y df_test serán los dataframes con los que trabajaremos a partir de ahora.

- Ejemplo de solo uno de ellos:

	partition	Male
image_id		
105701.jpg	0	0
123153.jpg	0	0
143076.jpg	0	0
101300.jpg	0	0
108812.jpg	0	0
...
090084.jpg	0	1
060586.jpg	0	1
093308.jpg	0	1
081580.jpg	0	1
043679.jpg	0	1

Llegados a este punto tenemos, nuestros **3 dataframes** (subconjuntos) **estratificados**, con el mismo número de Male == 0 que Male == 1, esto es especialmente importante cuando hay un **desequilibrio en la distribución de clases** y queremos **realizar clasificación binaria**. Además, hemos conseguido coger una **muestra más pequeña** de datos para entrenamiento, validación y test para así solucionar futuros problemas computacionales.

5. Pre - procesamiento

Procedemos ahora a realizar una serie de **transformaciones** a los datos para dejarlo tal y como el modelo quiere tenerlo y además tal y como mejor rendimiento tengan para el mismo.

El objetivo de este método será preparar los datos de **entrenamiento y validación** para ajustarlos a como quiere el modelo que le entren. Vamos a seguir una serie de pasos:

- 1.1. **Normalizamos** los píxeles de una imagen.
- 1.2. **Añadimos una dimensión** más. (*Explicación1)
- 1.3. **Repetimos** para todas las imágenes del dataframe.
- 1.4. **Añadimos** todas a un array de NumPy.
- 1.5. **Ajustamos** cada imagen al formato correcto (*Explicación2)

Con todo lo anterior tendríamos el subconjunto **"X"** preparado. Para el subconjunto **"y"** aplicamos one-hot a la variable objetivo que será **"Male"**. (*Explicación3)

- (***Explicación1**): `x.reshape((1,) + x.shape)`: Añade una dimensión adicional al principio de la tupla. Si `x.shape` es (altura, ancho, canales), entonces `(1,) + x.shape` se convierte en (1, altura, ancho, canales). **Transformamos en un tensor 4D** donde la primera dimensión tiene tamaño 1.

Hemos tenido que realizar este paso ya que posteriormente trabajaremos con **lotes de datos** (imágenes), entonces el modelo de aprendizaje necesita una **dimensión extra** que represente el número de lotes que va a tener nuestro dataframe, ya que este esperará que tengamos estos lotes de datos.

```
X = X.reshape(X.shape[0], 218, 178, 3) # Explicación2
```

- (***Explicación2**): `x_.shape[0]` representará las imágenes totales. Como vimos en el EDA, las imágenes tendrán un tamaño de 218 x 178 y 3 canales, entonces transformamos ese array de imágenes para que tengan esa forma y así el modelo pueda aprenderlas.
- (***Explicación3**): Transformamos las etiquetas para que tengan formato One-Hot, es decir, en vez de tener un variable “Male” con 0s y 1s, tendremos dos variables, podrían ser, por poner un ejemplo, “No Male” y “Male”, que tendrán 1s dependiendo si en la imagen aparece un hombre o una mujer.
Nos hemos estado informado de que necesitamos hacer este paso ya que es requerido para entrenar modelos que utilizan **funciones de pérdida categóricas o softmax** en la **capa de salida**, y en efecto, los modelos que utilizamos lo tienen.
Ya que la **función softmax** produce una distribución de **probabilidad sobre las clases**, por lo que es necesario codificar las etiquetas en formato one-hot.

Así quedarán nuestros datos de **entrenamiento**:

```
x_train
✓ 0.0s

array([[[[0.74509805, 0.6627451 , 0.44705883],
         [0.7490196 , 0.6666667 , 0.4509804 ],
         [0.76862746, 0.6862745 , 0.47843137],
         ...,
         [0.8352941 , 0.7411765 , 0.5529412 ],
         [0.78431374, 0.6901961 , 0.54901963],
         [0.79607844, 0.69803923, 0.5764706 ]],

        [[0.7137255 , 0.627451 , 0.43137255],
         [0.6901961 , 0.6039216 , 0.40784314],
         [0.67058825, 0.58431375, 0.39215687],
         ...,
```

```

y_train
✓ 0.0s
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [0., 1.],
       [0., 1.],
       [0., 1.]], dtype=float32)

```

De forma similar se realizarán los de **validación**. La gran diferencia está en como hemos tratado los datos de **test**.

Para preparar nuestro conjunto de datos de prueba seguiremos estos pasos:

- 1.6. **Cargamos** cada imagen.
- 1.7. Pasamos a **RGB** en vez de BGR.
- 1.8. **Redimensionamos** a nuestro tamaño de imágenes que es 178 x 218.
- 1.9. **Normalizamos** el valor de los píxeles.
- 1.10. **Añadimos una dimensión extra** al array para transformarlo en un tensor 4D. Por último, añadimos la etiqueta al atributo de la clase.

```

def prepare_test_df(df):
    x = []
    y = []

    for idx, clase in df.iterrows():
        img = cv2.imread(imagenes_totales + idx)
        img = cv2.resize(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), (178, 218)).astype(np.float32) / 255.0
        x.append(np.expand_dims(img, axis=0))
        y.append(clase['Male'])

    return x,y

```

```

x_test
✓ 0.5s
[array([[[[0.972549 , 0.90588236, 0.8352941 ],
          [0.96862745, 0.9019608 , 0.83137256],
          [0.96862745, 0.9019608 , 0.83137256],
          ...,
          [0.99215686, 0.8980392 , 0.79607844],
          [0.99215686, 0.8980392 , 0.79607844],
          [0.99215686, 0.8980392 , 0.79607844]],

        [[0.972549 , 0.90588236, 0.8352941 ],
          [0.96862745, 0.9019608 , 0.83137256],
          [0.96862745, 0.9019608 , 0.83137256],
          ...,

```

```
y_test
✓ 0.0s
[0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0]
```

La diferencia por la que preprocesamos diferente el conjunto de entrenamiento y validación contra el de test es que, durante el entrenamiento, es común proporcionar **lotes de datos** para **mejorar la eficiencia computacional**.

Por otro lado, durante la evaluación, a menudo se desea evaluar el modelo en el conjunto de prueba completo, por lo que de cierta manera se "deja un poco de lado" la eficiencia, al querer **las imágenes una por una**.

Además de que si, por ejemplo, hacemos el mismo preprocesamiento al conjunto de test, a la hora de predecir **tendríamos 2 clases objetivo, "No male" y "Male"** porque habríamos hecho one-hot, y ese no es nuestro objetivo. Nosotros queremos tener una clase objetivo y **hacer clasificación binaria**.

Hacemos esta reflexión ya que habíamos pensado hacer las mismas transformaciones a los 3 conjuntos de datos ya que temíamos que si hacíamos transformaciones diferentes tal vez cometeríamos **fuga de datos**, pero nos hemos encontrado con el problema anterior así que hemos decidido realizarlo así.

5.1. Aumento de datos

Nuestro siguiente paso de pre-procesamiento es aplicar la técnica conocida como "**Data Augmentation**" (Aumento de datos). El objetivo de esta técnica es tener **más datos para entrenar** a partir de las imágenes originales. ¿Cómo conseguimos esto? A partir de una imagen creamos otras diferentes, rotando la inicial, cambiándole el ángulo, la posición, etc. Así conseguiremos que el modelo tenga un mejor rendimiento al disponer de las mismas imágenes en diferentes posiciones ángulos, etc.

Destacar que este pre-procesamiento **solo se aplicará** al conjunto de datos de **ENTRENAMIENTO**. Pudimos realizar esta técnica como Luis lo hace que es a partir de un Sequential e introducirlo como una capa más al modelo que queramos entrenar.

Pero, leímos que cuando se utiliza la técnica de **Transfer Learning**, que más tarde explicaremos, es recomendable crear un "nuevo" train que será todo el conjunto junto, tanto las imágenes originales, como estas rotadas, cambiadas de ángulo, etc. Por ello decidimos aplicar esta técnica con un "**ImageDataGenerator**".

Hemos pensado que para nuestro proyecto tiene sentido realizar este pre-procesamiento ya que si nosotros rotamos, giramos, etc, alguna imagen, sigue siendo **la misma persona**, pero, si estuviésemos realizando una **detección de números**, por ejemplo, esta técnica no tendría mucho sentido ya que podría ser que, si rotamos un numero o le damos la vuelta, puede llegar a ser **otro número diferente** y no será nada beneficioso para el modelo.

En la libreta hemos realizado esta técnica para una sola imagen, con el objetivo de comprobar si realizábamos correctamente esta técnica. En la memoria hemos decidido no mostrar este paso y mostrar directamente como lo aplicamos a todo el conjunto de entrenamiento con el fin de no extender la memoria más de lo necesario. Solo mostraremos el resultado aplicado a una imagen.



Aumento de datos para una sola imagen de ejemplo



Ahora sí, vamos a ver como hemos aplicado esta técnica a todo el conjunto de entrenamiento.

```
train_data_aug = ImageDataGenerator( preprocessing_function = preprocess_input,
rotation_range = 10, # Rango en el que se rotarán las imágenes aleatoriamente
#width_shift_range = 0.15, # Rango dentro el cual se puede ensanchar o estrechar la imagenes horizontalmente
#height_shift_range = 0.15, # Rango dentro el cual se puede ensanchar o estrechar la imagenes verticalmente
#shear_range = 0.15, # Rango en el que se "inclinará" la imagen
zoom_range = 0.2, # Rango en el que se aplicará zoom a la imagen
horizontal_flip = True, # Se aplicará o no un giro horizontal
vertical_flip = True, # Se aplicará o no un giro vertical
)
```

PONER Y EXPLICAR LAS DEFINITIVAS!!!!!!!!!!!!!!!!!!!!!!

Procedemos a entrenar el modelo con el conjunto de imágenes de entrenamiento y después guardamos los lotes en "lotes_train_data_aug". Esta variable contendrá tanto las imágenes originales como estas con todos los cambios que le introduzcamos.

```
train_data_aug.fit(X_train)

lotes_train_data_aug = train_data_aug.flow( X_train, y_train, batch_size=batch_size,seed=120)
```

5.2. Callbacks

Cuando finaliza el número de epochs establecido se guarda es el historial del modelo, pero a través de la clase **Callback** podemos monitorizar los diferentes resultados sobre el conjunto de validación. Para ello **guardaremos** en un archivo **el mejor modelo** sobre el conjunto de validación y estableceremos un número **máximo de épocas** sin mejoras en la tasa de acierto para interrumpir el entrenamiento y no tener que continuar entrenando en el caso de que no pueda mejorar más el modelo.

Crearemos un método que reciba por parámetro el **nombre del archivo del modelo a guardar** y cree los callbacks devolviendo una lista que los contenga. Además, le pasaremos el **número máximo de épocas** sin mejora antes de detener el entrenamiento.

```
def create_callbacks(filepath, no_improve):

    model_checkpoint = ModelCheckpoint(
        filepath=filepath,
        verbose=2,
        save_best_only=True)
    # Vamos guardando el modelo que tenga mejor accuracy al validar
    # Guardamos con extensión .hdf5, formato de archivo estándar de Keras/TensorFlow para guardar modelos.

    early_stopping = EarlyStopping(
        patience=no_improve, # Máximas épocas que pasará sin mejorar
        verbose=1
    )

    callbacks = [model_checkpoint, early_stopping]
    return callbacks
```

6. Transfer Learning

Pasamos ahora al entrenamiento del modelo que vamos a usar para detectar si un famoso es un hombre o una mujer. Hemos estado investigando y llegamos a un modelo llamado **“Inception v3”**. Este es un modelo de reconocimiento de imágenes que ya ha sido utilizado con grandes bases de datos como **“ImageNet”**.

Comprobamos que este modelo suele ser utilizado para las tareas que justo necesitábamos para este proyecto, **clasificación y reconocimiento de imágenes** y al ver que proporcionaba muy buenos resultados decidimos utilizarlo, con algunas modificaciones que explicaremos posteriormente.

Pensamos en este modelo ya que la capacidad de reconocer una gran cantidad de características visuales podría ser crucial para identificar atributos de género en las imágenes.

Investigamos también por qué tenía tantas y tantas capas de convolución y descubrimos que, esta **nueva versión** del modelo utiliza en su mayoría convoluciones **1x1** para reducir la cantidad de cálculos necesarios y con ello el coste computacional. En **primeras versiones** se usaban menos capas de convolución, pero de mayor tamaño, como **5x5**. Es por eso por lo que esta versión es muchísimo más eficiente que las demás.

Procederemos a aplicar la técnica **“Transfer Learning”** ya que este modelo ha sido pre-entrenado con la base de datos anteriormente mencionada, entonces tiene la capacidad de reconocer una amplia gama de características. Por ello, usaremos este modelo pre-entrenado que **contendrá los pesos de la red neuronal ya entrenada**.

```
modelo_inception = InceptionV3(weights='C:\\Users\\Alejandro\\proyecto_vision\\inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5',
                                include_top=False, input_shape=(218, 178, 3)) # Ponemos aquí el input para no tener que ponerlo en cada uno de los tipos de cap
```

Importante mencionar que, **pasamos el input por parámetro** para no tener que ponerlo como capa en cada uno de los modelos que probemos.

Como ya hemos mencionado, este modelo fue pre-entrenado en la tarea de clasificación de Imágenes de **ImageNet**, pero nosotros los usaremos para una finalidad diferente, por lo que **no debemos incluir las últimas capas del modelo**. **"include_top=False"**.

Necesitamos **añadir nuevas capas** y sobre todo la capa de salida con **activación "softmax"** ya que este siempre es altamente recomendable para trabajar con clasificaciones de 2 clases o más, como es nuestro caso ("**No male**" y "**Male**").

7. Bibliografía

Incluir aquí si se ha consultado alguna bibliografía en particular.

