```
from google.colab import drive
import pandas as pd
from functools import reduce
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
import numpy as np
from functools import reduce
```

```
from tensorflow.keras import layers, Model
from tensorflow.keras.layers import Layer
from tensorflow.keras.utils import Sequence, load_img, img_to_array, array_to_img
from keras.callbacks import ModelCheckpoint
import tensorflow_hub as hub
import tensorflow as tf
```

```
epochs = 25
batch_size = 32
num_classes = 80
margin = 1
```

```
resnset50 = 'https://tfhub.dev/google/imagenet/resnet_v2_50/classification/5'
```

```
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
img_dir = "/content/drive/My Drive/TFG_Xarxes_neuronals_siameses/main_img/"
img_size = (178,218)
```

```
def visualize_pair(pair, pred=None):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
  fig.suptitle('Sim: {}'.format(pair['Sim']), size=16)
  ax1.imshow(load_img(img_dir + pair['Image1']))
  ax1.set_axis_off()
  ax2.imshow(load_img(img_dir + pair['Image2']))
  ax2.set_axis_off()
  plt.show()
```

```
def pairs(df):
  def f(row):
    return [[row['Image'], df[df['Class']==row['Class']].sample(n=1).iloc[0,0], 1],
            [row['Image'], df[df['Class']!=row['Class']].sample(n=1).iloc[0,0], 0]]
  return pd.DataFrame(reduce(lambda x, y: x + y, df.apply(f,axis=1).to_list(), []),
                      columns=['Image1','Image2','Sim'])
```

```
def get_celebrities(n_celebrities:int, refactor_class=False):
    df1, df2 = get_max_aparicions(get_identities(), n_celebrities)
    #df.reset_index(inplace=True, drop=True)
    if refactor_class:
        df1 = refactor_identity(df1, 'Identity')
    return df1, df2

def get_max_aparicions(df:pd.DataFrame, n_celebrities:int):
    aparicions_df = df["Identity"].value_counts()
    aparicions_df = aparicions_df.reset_index()
    aparicions_df = aparicions_df.rename(columns={'Identity':'aparicions', 'index':'Identi
    aparicions_df = aparicions_df.iloc[:n_celebrities]
    return df[df['Identity'].isin(aparicions_df['Identity'])].reset_index(drop=True), df[~

def get_identities():
    identity_df = pd.read_csv("/content/drive/My Drive/TFG_Xarxes_neuronals_siameses/main_
    identity_df = identity_df.rename(columns={0:"Image_name", 1:"Identity"})
    return identity_df

def refactor_identity(df, traget_col):
    d = get_dic_index(df, traget_col)
    df["Class"] = df.apply(lambda row: d[row[traget_col]], axis=1)
    return df

def get_dic_index(df, traget_col):
    df = df[traget_col].value_counts()
    dic_index = df.to_dict()
    i = 0
    for c in dic_index:
        dic_index[c] = i
        i += 1
    return dic_index
def get_tvt(df:pd.DataFrame, target_name:str, train_size:int):
    total_classes = df[target_name].nunique()
    n_df_col = df.shape[0]
    df = shuffle(df)
    train_df = df.iloc[:int(n_df_col*train_size)]
    val_df = df.iloc[int(n_df_col*train_size):int(n_df_col*( train_size + (1-train_size)/2
    test_df = df.iloc[int(n_df_col*( train_size + (1-train_size)/2 )):]
    while True:
        if train_df[target_name].nunique() == total_classes and val_df[target_name].nuniqu
            return train_df, val_df, test_df

        df = shuffle(df)
        train_df = df.iloc[:int(n_df_col*train_size)]
        val_df = df.iloc[int(n_df_col*train_size):int(n_df_col*( train_size + (1-train_siz
        test_df = df.iloc[int(n_df_col*( train_size + (1-train_size)/2 )):]


identity_df, rest = get_celebrities(num_classes, refactor_class=True)
identity_df = identity_df[["Image_name", "Class"]].rename(columns={"Image_name": "Image"})
identity_df.head()
```

	Image	Class
0	000001.jpg	79
1	000096.jpg	8
2	000116.jpg	14
3	000150.jpg	9

```
train_df, val_df, test_df = get_tvt(identity_df, "Class", 0.8)
```

```
pairsTrain = pairs(train_df)
pairsVal = pairs(val_df)
pairsTest = pairs(test_df)
```

```
pairsTrain.head()
```

	Image1	Image2	Sim
0	107629.jpg	047600.jpg	1
1	107629.jpg	029240.jpg	0
2	085304.jpg	124276.jpg	1
3	085304.jpg	099122.jpg	0
4	011738.jpg	016156.jpg	1

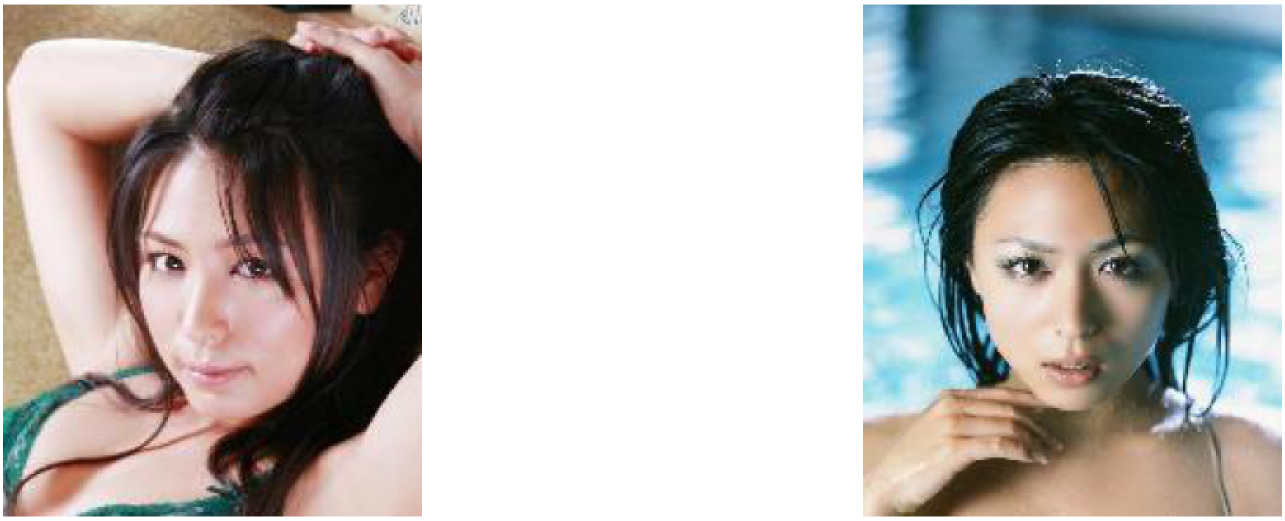
```
visualize_pair(pairsTrain.loc[0])
```

```
visualize_pair(pairsTrain.loc[1])
```

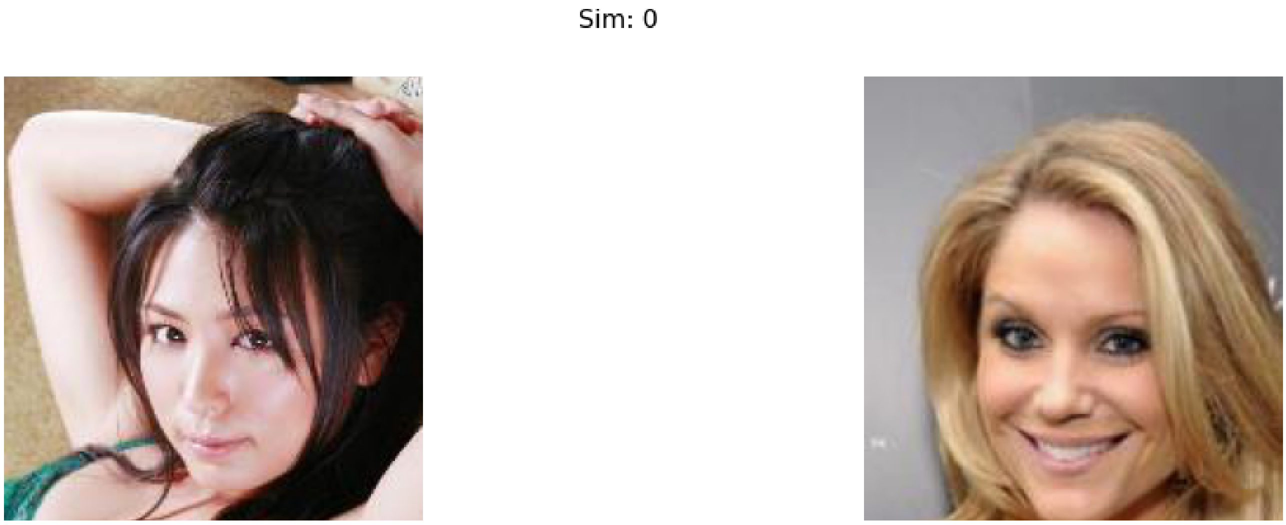
Sim: 0

```
visualize_pair(pairsTrain.loc[pairsTrain.shape[0]-2])
```

Sim: 1



```
visualize_pair(pairsTrain.loc[pairsTrain.shape[0]-1])
```



```
class DataGenerator(Sequence):
  def __init__(self, df, batch_size=32, img_size=(200,200), img_dir= './', shuffle=True):
    self.batch_size = batch_size
    self.img_size = img_size
    self.img_dir = img_dir
    self.df = df
    self.indices = self.df.index.tolist()
    self.shuffle = shuffle
    self.on_epoch_end()

  def __len__(self):
    return len(self.indices) // self.batch_size

  def __getitem__(self, index):
    index = self.index[index * self.batch_size:(index + 1) * self.batch_size]
    batch = [self.indices[k] for k in index]

    X1, X2, y = self.__get_data(batch)
    return [X1, X2], y

  def on_epoch_end(self):
    self.index = np.arange(len(self.indices))
    if self.shuffle:
      np.random.shuffle(self.index)

  def __get_data(self, batch):
    X1 = np.zeros((self.batch_size,) + self.img_size + (3,), dtype="float32")
    X2 = np.zeros((self.batch_size,) + self.img_size + (3,), dtype="float32")
    y = np.zeros((self.batch_size,), dtype="float32")
    for i, idx in enumerate(batch):
      X1[i] = img_to_array(load_img(self.img_dir + self.df.loc[idx,'Image1'], target_size=
      X2[i] = img_to_array(load_img(self.img_dir + self.df.loc[idx,'Image2'], target_size=
      y[i] = 1 - self.df.loc[idx,'Sim']
    return X1, X2, y


trainGenerator = DataGenerator(pairsTrain, batch_size=batch_size, img_size=img_size, img_d
valGenerator = DataGenerator(pairsVal, batch_size=batch_size, img_size=img_size, img_dir=i
testGenerator = DataGenerator(pairsTest, batch_size=1, img_size=img_size, img_dir=img_dir,
class EuclideanDLayer(Layer):
    def __init__(self, **kwargs):
        super().__init__()

    def call(self, x, y):
        sum_square = tf.math.reduce_sum(tf.math.square(x - y), axis=1, keepdims=True)
        return tf.math.sqrt(tf.math.maximum(sum_square, tf.keras.backend.epsilon()))


pretrained_base = hub.KerasLayer(resnset50, trainable = False)


input = layers.Input(img_size + (3,))
x = layers.Rescaling(1./255)(input)
x = pretrained_base(x)
x = layers.Flatten()(x)
embedding_network = Model(input, x)

input_1 = layers.Input(img_size + (3,))
input_2 = layers.Input(img_size + (3,))


tower_1 = embedding_network(input_1)
tower_2 = embedding_network(input_2)

siamese_layer = EuclideanDLayer()
siamese_layer._name = 'distance'

distance = siamese_layer(tower_1, tower_2)

#merge_layer = layers.Lambda(euclidean_distance)([tower_1, tower_2])
features = layers.BatchNormalization()(distance)
features = layers.Dense(512, activation="relu")(features)
features = layers.Dropout(0.2)(features)
features = layers.Dense(512, activation="relu")(features)
features = layers.Dense(256, activation="relu")(features)
features = layers.Dense(128, activation="relu")(features)

output_layer = layers.Dense(1, activation="sigmoid")(features)


siamese = Model(inputs=[input_1, input_2], outputs=output_layer)


def loss(margin=1.0):
    def contrastive_loss(y_true, y_pred):
        square_pred = tf.math.square(y_pred)
        margin_square = tf.math.square(tf.math.maximum(margin - (y_pred), 0))
        return tf.math.reduce_mean(
            (1 - y_true) * square_pred + (y_true) * margin_square
        )

    return contrastive_loss


siamese.compile(loss=loss(margin=margin), optimizer="adam", metrics=["accuracy"])
siamese.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 178, 218, 3  0           []
                                )]

 input_3 (InputLayer)           [(None, 178, 218, 3  0           []
                                )]

 model (Functional)             (None, 1001)         25615849    ['input_2[0][0]',
                                                                  'input_3[0][0]']

 distance (EuclideanDLayer)     (None, 1)            0           ['model[0][0]',
                                                                  'model[1][0]']

 batch_normalization (BatchNorm  (None, 1)           4           ['distance[0][0]']
 alization)

 dense (Dense)                  (None, 512)          1024        ['batch_normalizatio

 dropout (Dropout)              (None, 512)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 512)          262656      ['dropout[0][0]']

 dense_2 (Dense)                (None, 256)          131328      ['dense_1[0][0]']

 dense_3 (Dense)                (None, 128)          32896       ['dense_2[0][0]']

 dense_4 (Dense)                (None, 1)            129         ['dense_3[0][0]']

==================================================================================================
Total params: 26,043,886
Trainable params: 428,035
Non-trainable params: 25,615,851
__________________________________________________________________________________________________
```

```
callbacks = [ModelCheckpoint('/content/drive/My Drive/TFG_Xarxes_neuronals_siameses/Weight
             save_best_only=True)]
```

```
history = siamese.fit(trainGenerator,
                    validation_data = valGenerator,
                    callbacks=callbacks,
                    epochs=epochs,
                    batch_size=batch_size)
```

```
Epoch 1/25
121/121 [==============================] - 732s 6s/step - loss: 0.2279 - accuracy: 0
Epoch 2/25
121/121 [==============================] - 35s 285ms/step - loss: 0.2276 - accuracy:
Epoch 3/25
121/121 [==============================] - 36s 299ms/step - loss: 0.2269 - accuracy:
Epoch 4/25
```

```
121/121 [==============================] - 24s 200ms/step - loss: 0.2304 - accuracy:
Epoch 5/25
121/121 [==============================] - 25s 207ms/step - loss: 0.2286 - accuracy:
Epoch 6/25
121/121 [==============================] - 24s 196ms/step - loss: 0.2272 - accuracy:
Epoch 7/25
121/121 [==============================] - 24s 196ms/step - loss: 0.2263 - accuracy:
Epoch 8/25
121/121 [==============================] - 25s 205ms/step - loss: 0.2267 - accuracy:
Epoch 9/25
121/121 [==============================] - 24s 197ms/step - loss: 0.2274 - accuracy:
Epoch 10/25
121/121 [==============================] - 25s 202ms/step - loss: 0.2266 - accuracy:
Epoch 11/25
121/121 [==============================] - 24s 196ms/step - loss: 0.2280 - accuracy:
Epoch 12/25
121/121 [==============================] - 24s 202ms/step - loss: 0.2276 - accuracy:
Epoch 13/25
121/121 [==============================] - 24s 200ms/step - loss: 0.2262 - accuracy:
Epoch 14/25
121/121 [==============================] - 23s 193ms/step - loss: 0.2276 - accuracy:
Epoch 15/25
121/121 [==============================] - 24s 200ms/step - loss: 0.2268 - accuracy:
Epoch 16/25
121/121 [==============================] - 23s 193ms/step - loss: 0.2262 - accuracy:
Epoch 17/25
121/121 [==============================] - 24s 200ms/step - loss: 0.2268 - accuracy:
Epoch 18/25
121/121 [==============================] - 24s 199ms/step - loss: 0.2262 - accuracy:
Epoch 19/25
121/121 [==============================] - 23s 192ms/step - loss: 0.2277 - accuracy:
Epoch 20/25
121/121 [==============================] - 25s 202ms/step - loss: 0.2256 - accuracy:
Epoch 21/25
121/121 [==============================] - 24s 197ms/step - loss: 0.2272 - accuracy:
Epoch 22/25
121/121 [==============================] - 23s 193ms/step - loss: 0.2254 - accuracy:
Epoch 23/25
121/121 [==============================] - 24s 195ms/step - loss: 0.2267 - accuracy:
Epoch 24/25
121/121 [==============================] - 23s 192ms/step - loss: 0.2261 - accuracy:
Epoch 25/25
121/121 [==============================] - 24s 200ms/step - loss: 0.2262 - accuracy:
```

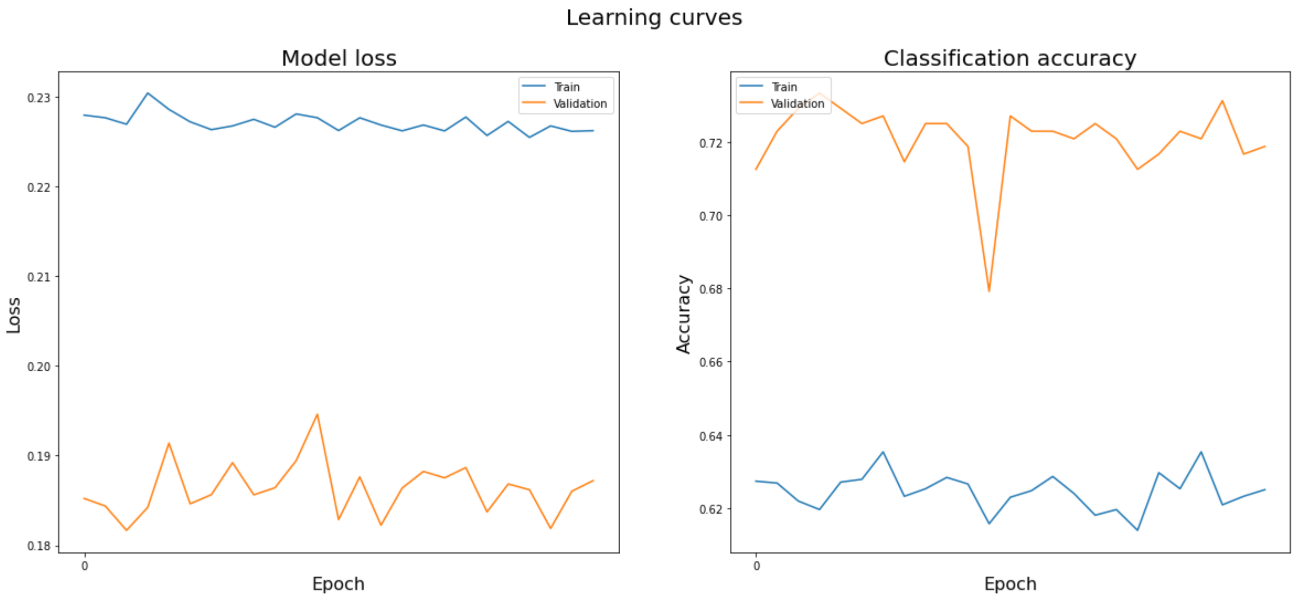
```
siamese.evaluate(testGenerator)
```

```
488/488 [==============================] - 90s 183ms/step - loss: 0.2034 - accuracy:
[0.20342084765434265, 0.6598360538482666]
```

```
siamese.save('/content/drive/My Drive/TFG_Xarxes_neuronals_siameses/models/Final.test_8')
```

```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize = (20, 8))

ax1.plot(history.history['loss'])
```

```
ax1.plot(history.history['val_loss'])
ax1.set_title('Model loss', size=20)
ax1.set_ylabel('Loss', size=16)
ax1.set_xlabel('Epoch', size=16)
ax1.set_xticks(range(0, 6, 25))
ax1.legend(['Train', 'Validation'], loc='upper right')

ax2.plot(history.history['accuracy'])
ax2.plot(history.history['val_accuracy'])
ax2.set_title('Classification accuracy', size=20)
ax2.set_ylabel('Accuracy', size=16)
ax2.set_xlabel('Epoch', size=16)
ax2.set_xticks(range(0, 6, 25))
ax2.legend(['Train', 'Validation'], loc='upper left')

fig.suptitle('Learning curves', size=20)

plt.show()
```



```
pairsTest['Preds'] = siamese.predict(testGenerator)
```

```
488/488 [==============================] - 11s 22ms/step
```

```
pairsTest.to_csv('/content/drive/My Drive/TFG_Xarxes_neuronals_siameses/Preds_test_8.txt')
```

```
from sklearn.metrics import r2_score, mean_absolute_error
```

```
r2_score(pairsTest['Sim'], pairsTest['Preds'])
```

```
    -0.6142602423545471
```

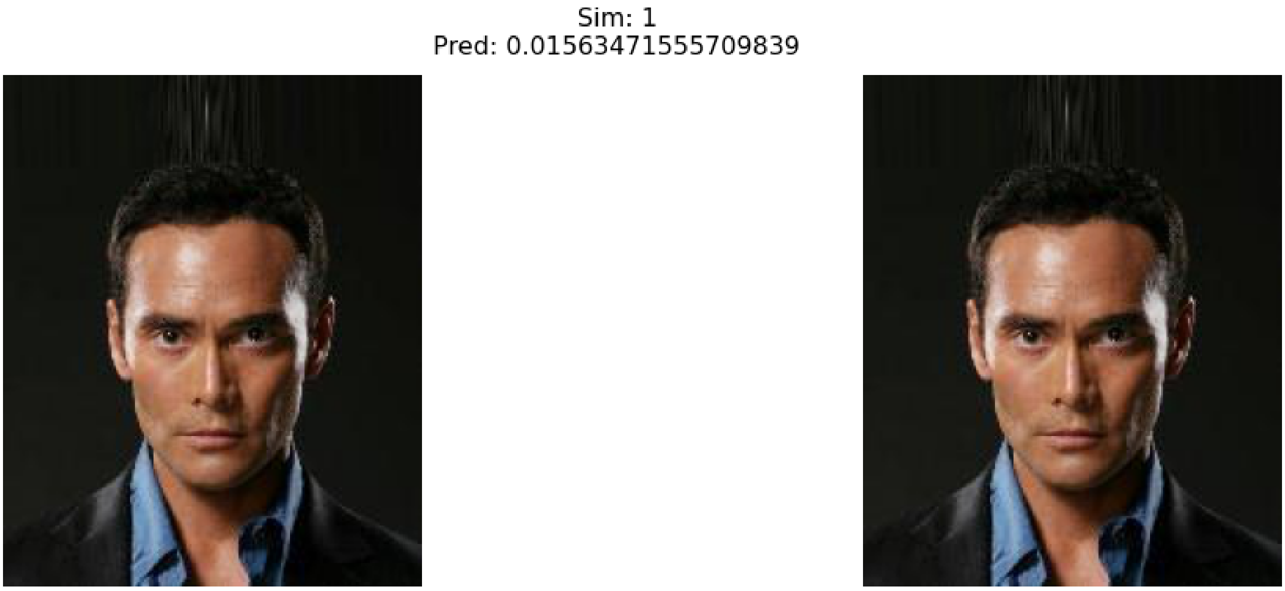
```
mean_absolute_error(pairsTest['Sim'], pairsTest['Preds'])
```

```
    0.6000721326983366
```

```
def visualize_pair(pair, pred=None):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
  fig.suptitle('Sim: {}\nPred: {}'.format(pair['Sim'], pair['Preds']), size=16)
  ax1.imshow(load_img(img_dir + pair['Image1']))
  ax1.set_axis_off()
  ax2.imshow(load_img(img_dir + pair['Image2']))
  ax2.set_axis_off()
  plt.show()
```

```
index = 12
```

```
visualize_pair(pairsTest.loc[index])
```



```
visualize_pair(pairsTest.loc[index+1])
```

Sim: 0
Pred: 0.5415585041046143