Introducción

Etapa I - Reconocimiento Facial

Instalar dependencias

Descargamos y descomprimos un archivo de datos de puntos de referencia faciales (face landmarks) desde la web. El archivo descargado es el modelo shape_predictor_68_face_landmarks.dat, que se utiliza para localizar puntos clave en rostros (como ojos, nariz, boca, etc.) y es esencial para tareas de reconocimiento y alineación facial en aplicaciones de visión por computadora.

```
import bz2
import os

from urllib.request import urlopen

def download_landmarks(dst_file):
    url = 'http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2'
    decompressor = bz2.BZ2Decompressor()

    with urlopen(url) as src, open(dst_file, 'wb') as dst:
        data = src.read(1024)
        while len(data) > 0:
            dst.write(decompressor.decompress(data))
            data = src.read(1024)

dst_dir = 'models'
dst_file = os.path.join(dst_dir, 'landmarks.dat')

if not os.path.exists(dst_file):
    os.makedirs(dst_dir)
    download_landmarks(dst_file)

!pip install tensorflow
!pip install --upgrade tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorf
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.10/dist-packages (from tensorfl
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.5.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from tensorfl
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.5.0->tensorflow) (
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.5.0->tensorflow)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.5.0->tensorflow)
```

```
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from request
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.19
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.19
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->kera
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->ke
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorf
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.10/dist-packages (from tensorfl
```

```
from google.colab import drive
drive.mount('/content/drive')
```

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

Definimos funciones de ayuda para la construccion del modelo

Este código define una serie de funciones y constantes en TensorFlow y NumPy para construir y cargar pesos en una red neuronal profunda, en una red tipo Inception paraposteriormente ser utlizado en la creacion del modelo.

Se definen variables y funciones básicas

```
import tensorflow as tf
import numpy as np
import os

from numpy import genfromtxt
from tensorflow.keras.layers import Conv2D, ZeroPadding2D, Activation, BatchNormalization

_FLOATX = 'float32'

def variable(value, dtype=_FLOATX, name=None):
  v = tf.Variable(np.asarray(value, dtype=dtype), name=name)
  _get_session().run(v.initializer)
  return v

def shape(x):
  return x.get_shape()

def square(x):
  return tf.square(x)

def zeros(shape, dtype=_FLOATX, name=None):
  return variable(np.zeros(shape), dtype, name)

def concatenate(tensors, axis=-1):
  if axis < 0:
      axis = axis % len(tensors[0].get_shape())
  return tf.concat(axis, tensors)
```

Definimos la normalización de respuesta local o LRN2D (Local Response Normalization en 2 dimensiones).

Es una técnica que utilizamos en la red neuronal para mejorar el rendimiento y precisión. Funciona destacando características más fuertes y reduciendo la influencia de características menos destacadas. Esta normalización compara cada activación en la capa con las activaciones vecinas y ajusta su valor en función de las activaciones circundantes.

```
def LRN2D(x):
  return tf.nn.lrn(x, alpha=1e-4, beta=0.75)
```

Definimos una función conv2d_bn.

La función conv2d_bn aplica una combinación de capas a una entrada x, realiza convoluciones, normalización por lotes y activación (ReLU), y opcionalmente aplica relleno (padding) en la entrada.

```
def conv2d_bn(
  x,
  layer=None,
  cv1_out=None,
  cv1_filter=(1, 1),
  cv1_strides=(1, 1),
  cv2_out=None,
  cv2_filter=(3, 3),
  cv2_strides=(1, 1),
  padding=None,
):
  num = '' if cv2_out == None else '1'
  tensor = Conv2D(cv1_out, cv1_filter, strides=cv1_strides, name=layer+'_conv'+num)(x)
  tensor = BatchNormalization(axis=3, epsilon=0.00001, name=layer+'_bn'+num)(tensor)
  tensor = Activation('relu')(tensor)
  if padding == None:
    return tensor
  tensor = ZeroPadding2D(padding=padding)(tensor)
  if cv2_out == None:
    return tensor
  tensor = Conv2D(cv2_out, cv2_filter, strides=cv2_strides, name=layer+'_conv'+'2')(tensor)
  tensor = BatchNormalization(axis=3, epsilon=0.00001, name=layer+'_bn'+'2')(tensor)
  tensor = Activation('relu')(tensor)
  return tensor
```

Cargamos pesos de capas convolucionales, capas de normalización por lotes y una capa densa, y los organizamos en un diccionario de Python para su uso posterior en el modelo.

```
weights = [
  'conv1', 'bn1', 'conv2', 'bn2', 'conv3', 'bn3',
  'inception_3a_1x1_conv', 'inception_3a_1x1_bn',
  'inception_3a_pool_conv', 'inception_3a_pool_bn',
  'inception_3a_5x5_conv1', 'inception_3a_5x5_conv2', 'inception_3a_5x5_bn1', 'inception_3a_5x5_bn2',
  'inception_3a_3x3_conv1', 'inception_3a_3x3_conv2', 'inception_3a_3x3_bn1', 'inception_3a_3x3_bn2',
  'inception_3b_3x3_conv1', 'inception_3b_3x3_conv2', 'inception_3b_3x3_bn1', 'inception_3b_3x3_bn2',
  'inception_3b_5x5_conv1', 'inception_3b_5x5_conv2', 'inception_3b_5x5_bn1', 'inception_3b_5x5_bn2',
  'inception_3b_pool_conv', 'inception_3b_pool_bn',
  'inception_3b_1x1_conv', 'inception_3b_1x1_bn',
  'inception_3c_3x3_conv1', 'inception_3c_3x3_conv2', 'inception_3c_3x3_bn1', 'inception_3c_3x3_bn2',
  'inception_3c_5x5_conv1', 'inception_3c_5x5_conv2', 'inception_3c_5x5_bn1', 'inception_3c_5x5_bn2',
  'inception_4a_3x3_conv1', 'inception_4a_3x3_conv2', 'inception_4a_3x3_bn1', 'inception_4a_3x3_bn2',
  'inception_4a_5x5_conv1', 'inception_4a_5x5_conv2', 'inception_4a_5x5_bn1', 'inception_4a_5x5_bn2',
  'inception_4a_pool_conv', 'inception_4a_pool_bn',
  'inception_4a_1x1_conv', 'inception_4a_1x1_bn',
  'inception_4e_3x3_conv1', 'inception_4e_3x3_conv2', 'inception_4e_3x3_bn1', 'inception_4e_3x3_bn2',
  'inception_4e_5x5_conv1', 'inception_4e_5x5_conv2', 'inception_4e_5x5_bn1', 'inception_4e_5x5_bn2',
  'inception_5a_3x3_conv1', 'inception_5a_3x3_conv2', 'inception_5a_3x3_bn1', 'inception_5a_3x3_bn2',
  'inception_5a_pool_conv', 'inception_5a_pool_bn',
  'inception_5a_1x1_conv', 'inception_5a_1x1_bn',
  'inception_5b_3x3_conv1', 'inception_5b_3x3_conv2', 'inception_5b_3x3_bn1', 'inception_5b_3x3_bn2',
  'inception_5b_pool_conv', 'inception_5b_pool_bn',
  'inception_5b_1x1_conv', 'inception_5b_1x1_bn',
  'dense_layer'
]

conv_shape = {
  'conv1': [64, 3, 7, 7],
  'conv2': [64, 64, 1, 1],
  'conv3': [192, 64, 3, 3],
  'inception_3a_1x1_conv': [64, 192, 1, 1],
  'inception_3a_pool_conv': [32, 192, 1, 1],
  'inception_3a_5x5_conv1': [16, 192, 1, 1],
  'inception_3a_5x5_conv2': [32, 16, 5, 5],
  'inception_3a_3x3_conv1': [96, 192, 1, 1],
  'inception_3a_3x3_conv2': [128, 96, 3, 3],
  'inception_3b_3x3_conv1': [96, 256, 1, 1],
  'inception_3b_3x3_conv2': [128, 96, 3, 3],
  'inception_3b_5x5_conv1': [32, 256, 1, 1],
  'inception_3b_5x5_conv2': [64, 32, 5, 5],
  'inception_3b_pool_conv': [64, 256, 1, 1],
  'inception_3b_1x1_conv': [64, 256, 1, 1],
  'inception_3c_3x3_conv1': [128, 320, 1, 1],
  'inception_3c_3x3_conv2': [256, 128, 3, 3],
  'inception_3c_5x5_conv1': [32, 320, 1, 1],
  'inception_3c_5x5_conv2': [64, 32, 5, 5],
  'inception_4a_3x3_conv1': [96, 640, 1, 1],
  'inception_4a_3x3_conv2': [192, 96, 3, 3],
  'inception_4a_5x5_conv1': [32, 640, 1, 1,],
  'inception_4a_5x5_conv2': [64, 32, 5, 5],
  'inception_4a_pool_conv': [128, 640, 1, 1],
  'inception_4a_1x1_conv': [256, 640, 1, 1],
  'inception_4e_3x3_conv1': [160, 640, 1, 1],
  'inception_4e_3x3_conv2': [256, 160, 3, 3],
  'inception_4e_5x5_conv1': [64, 640, 1, 1],
  'inception_4e_5x5_conv2': [128, 64, 5, 5],
  'inception_5a_3x3_conv1': [96, 1024, 1, 1],
  'inception_5a_3x3_conv2': [384, 96, 3, 3],
  'inception_5a_pool_conv': [96, 1024, 1, 1],
  'inception_5a_1x1_conv': [256, 1024, 1, 1],
  'inception_5b_3x3_conv1': [96, 736, 1, 1],
  'inception_5b_3x3_conv2': [384, 96, 3, 3],
  'inception_5b_pool_conv': [96, 736, 1, 1],
  'inception_5b_1x1_conv': [256, 736, 1, 1],
}
```

```
def load_weights():
  weightsDir = './weights'
  fileNames = filter(lambda f: not f.startswith('.'), os.listdir(weightsDir))
  paths = {}
  weights_dict = {}

  for n in fileNames:
    paths[n.replace('.csv', '')] = weightsDir + '/' + n

  for name in weights:
    if 'conv' in name:
      conv_w = genfromtxt(paths[name + '_w'], delimiter=',', dtype=None)
      conv_w = np.reshape(conv_w, conv_shape[name])
      conv_w = np.transpose(conv_w, (2, 3, 1, 0))
      conv_b = genfromtxt(paths[name + '_b'], delimiter=',', dtype=None)
      weights_dict[name] = [conv_w, conv_b]
    elif 'bn' in name:
      bn_w = genfromtxt(paths[name + '_w'], delimiter=',', dtype=None)
      bn_b = genfromtxt(paths[name + '_b'], delimiter=',', dtype=None)
      bn_m = genfromtxt(paths[name + '_m'], delimiter=',', dtype=None)
      bn_v = genfromtxt(paths[name + '_v'], delimiter=',', dtype=None)
      weights_dict[name] = [bn_w, bn_b, bn_m, bn_v]
    elif 'dense' in name:
      dense_w = genfromtxt(weightsDir+'/dense_w.csv', delimiter=',', dtype=None)
      dense_w = np.reshape(dense_w, (128, 736))
      dense_w = np.transpose(dense_w, (1, 0))
      dense_b = genfromtxt(weightsDir+'/dense_b.csv', delimiter=',', dtype=None)
      weights_dict[name] = [dense_w, dense_b]

  return weights_dict
```

Arquitectura del modelo

La arquitectura de la CNN utilizada es una variante de la arquitectura Inception. Más precisamente, es una variante de la arquitectura NN4 e identificada como el modelo nn4.small2 en el proyecto OpenFace. Este notebook utiliza una implementación en Keras de ese modelo, cuya definición se tomó del proyecto Keras-OpenFace.

Los detalles de la arquitectura no son demasiado importantes para nuestro caso, solo es útil saber que hay una capa completamente conectada con 128 unidades ocultas, seguida de una capa de normalización L2 sobre la base convolucional. Estas dos capas superiores se denominan la capa de embeddings, de la cual se pueden obtener los vectores de embeddings de 128 dimensiones. El modelo completo está definido en model.py y una visión gráfica se encuentra en model.png adjuntado en la entrega final.

Una versión en Keras del modelo nn4.small2 se puede crear con create_model().

```
# ----------------------------------------------------------------
# Code taken from https://github.com/iwantooxxoox/Keras-OpenFace
# ----------------------------------------------------------------
import tensorflow

from tensorflow.keras.layers import Conv2D, ZeroPadding2D, Activation, Input, concatenate
from tensorflow.keras.layers import Lambda, Flatten, Dense
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import MaxPooling2D, AveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K

def create_model():
    myInput = Input(shape=(96, 96, 3))

    x = ZeroPadding2D(padding=(3, 3), input_shape=(96, 96, 3))(myInput)
    x = Conv2D(64, (7, 7), strides=(2, 2), name='conv1')(x)
    x = BatchNormalization(axis=3, epsilon=0.00001, name='bn1')(x)
    x = Activation('relu')(x)
    x = ZeroPadding2D(padding=(1, 1))(x)
    x = MaxPooling2D(pool_size=3, strides=2)(x)
    x = Lambda(LRN2D, name='lrn_1')(x)
```

```
x = Conv2D(64, (1, 1), name='conv2')(x)
x = BatchNormalization(axis=3, epsilon=0.00001, name='bn2')(x)
x = Activation('relu')(x)
x = ZeroPadding2D(padding=(1, 1))(x)
x = Conv2D(192, (3, 3), name='conv3')(x)
x = BatchNormalization(axis=3, epsilon=0.00001, name='bn3')(x)
x = Activation('relu')(x)
x = Lambda(LRN2D, name='lrn_2')(x)
x = ZeroPadding2D(padding=(1, 1))(x)
x = MaxPooling2D(pool_size=3, strides=2)(x)

# Inception3a
inception_3a_3x3 = Conv2D(96, (1, 1), name='inception_3a_3x3_conv1')(x)
inception_3a_3x3 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3a_3x3_bn1')(inception_3a_3x3)
inception_3a_3x3 = Activation('relu')(inception_3a_3x3)
inception_3a_3x3 = ZeroPadding2D(padding=(1, 1))(inception_3a_3x3)
inception_3a_3x3 = Conv2D(128, (3, 3), name='inception_3a_3x3_conv2')(inception_3a_3x3)
inception_3a_3x3 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3a_3x3_bn2')(inception_3a_3x3)
inception_3a_3x3 = Activation('relu')(inception_3a_3x3)

inception_3a_5x5 = Conv2D(16, (1, 1), name='inception_3a_5x5_conv1')(x)
inception_3a_5x5 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3a_5x5_bn1')(inception_3a_5x5)
inception_3a_5x5 = Activation('relu')(inception_3a_5x5)
inception_3a_5x5 = ZeroPadding2D(padding=(2, 2))(inception_3a_5x5)
inception_3a_5x5 = Conv2D(32, (5, 5), name='inception_3a_5x5_conv2')(inception_3a_5x5)
inception_3a_5x5 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3a_5x5_bn2')(inception_3a_5x5)
inception_3a_5x5 = Activation('relu')(inception_3a_5x5)

inception_3a_pool = MaxPooling2D(pool_size=3, strides=2)(x)
inception_3a_pool = Conv2D(32, (1, 1), name='inception_3a_pool_conv')(inception_3a_pool)
inception_3a_pool = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3a_pool_bn')(inception_3a_pool)
inception_3a_pool = Activation('relu')(inception_3a_pool)
inception_3a_pool = ZeroPadding2D(padding=((3, 4), (3, 4)))(inception_3a_pool)

inception_3a_1x1 = Conv2D(64, (1, 1), name='inception_3a_1x1_conv')(x)
inception_3a_1x1 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3a_1x1_bn')(inception_3a_1x1)
inception_3a_1x1 = Activation('relu')(inception_3a_1x1)

inception_3a = concatenate([inception_3a_3x3, inception_3a_5x5, inception_3a_pool, inception_3a_1x1], axis=3)

# Inception3b
inception_3b_3x3 = Conv2D(96, (1, 1), name='inception_3b_3x3_conv1')(inception_3a)
inception_3b_3x3 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3b_3x3_bn1')(inception_3b_3x3)
inception_3b_3x3 = Activation('relu')(inception_3b_3x3)
inception_3b_3x3 = ZeroPadding2D(padding=(1, 1))(inception_3b_3x3)
inception_3b_3x3 = Conv2D(128, (3, 3), name='inception_3b_3x3_conv2')(inception_3b_3x3)
inception_3b_3x3 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3b_3x3_bn2')(inception_3b_3x3)
inception_3b_3x3 = Activation('relu')(inception_3b_3x3)

inception_3b_5x5 = Conv2D(32, (1, 1), name='inception_3b_5x5_conv1')(inception_3a)
inception_3b_5x5 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3b_5x5_bn1')(inception_3b_5x5)
inception_3b_5x5 = Activation('relu')(inception_3b_5x5)
inception_3b_5x5 = ZeroPadding2D(padding=(2, 2))(inception_3b_5x5)
inception_3b_5x5 = Conv2D(64, (5, 5), name='inception_3b_5x5_conv2')(inception_3b_5x5)
inception_3b_5x5 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3b_5x5_bn2')(inception_3b_5x5)
inception_3b_5x5 = Activation('relu')(inception_3b_5x5)

inception_3b_pool = AveragePooling2D(pool_size=(3, 3), strides=(3, 3))(inception_3a)
inception_3b_pool = Conv2D(64, (1, 1), name='inception_3b_pool_conv')(inception_3b_pool)
inception_3b_pool = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3b_pool_bn')(inception_3b_pool)
inception_3b_pool = Activation('relu')(inception_3b_pool)
inception_3b_pool = ZeroPadding2D(padding=(4, 4))(inception_3b_pool)

inception_3b_1x1 = Conv2D(64, (1, 1), name='inception_3b_1x1_conv')(inception_3a)
inception_3b_1x1 = BatchNormalization(axis=3, epsilon=0.00001, name='inception_3b_1x1_bn')(inception_3b_1x1)
inception_3b_1x1 = Activation('relu')(inception_3b_1x1)

inception_3b = concatenate([inception_3b_3x3, inception_3b_5x5, inception_3b_pool, inception_3b_1x1], axis=3)

# Inception3c
inception_3c_3x3 = conv2d_bn(inception_3b,
                             layer='inception_3c_3x3',
```

```
                                    cv1_out=128,
                                    cv1_filter=(1, 1),
                                    cv2_out=256,
                                    cv2_filter=(3, 3),
                                    cv2_strides=(2, 2),
                                    padding=(1, 1))

    inception_3c_5x5 = conv2d_bn(inception_3b,
                                    layer='inception_3c_5x5',
                                    cv1_out=32,
                                    cv1_filter=(1, 1),
                                    cv2_out=64,
                                    cv2_filter=(5, 5),
                                    cv2_strides=(2, 2),
                                    padding=(2, 2))

    inception_3c_pool = MaxPooling2D(pool_size=3, strides=2)(inception_3b)
    inception_3c_pool = ZeroPadding2D(padding=((0, 1), (0, 1)))(inception_3c_pool)

    inception_3c = concatenate([inception_3c_3x3, inception_3c_5x5, inception_3c_pool], axis=3)

    #inception 4a
    inception_4a_3x3 = conv2d_bn(inception_3c,
                                    layer='inception_4a_3x3',
                                    cv1_out=96,
                                    cv1_filter=(1, 1),
                                    cv2_out=192,
                                    cv2_filter=(3, 3),
                                    cv2_strides=(1, 1),
                                    padding=(1, 1))
    inception_4a_5x5 = conv2d_bn(inception_3c,
                                    layer='inception_4a_5x5',
                                    cv1_out=32,
                                    cv1_filter=(1, 1),
                                    cv2_out=64,
                                    cv2_filter=(5, 5),
                                    cv2_strides=(1, 1),
                                    padding=(2, 2))

    inception_4a_pool = AveragePooling2D(pool_size=(3, 3), strides=(3, 3))(inception_3c)
    inception_4a_pool = conv2d_bn(inception_4a_pool,
                                     layer='inception_4a_pool',
                                     cv1_out=128,
                                     cv1_filter=(1, 1),
                                     padding=(2, 2))
    inception_4a_1x1 = conv2d_bn(inception_3c,
                                    layer='inception_4a_1x1',
                                    cv1_out=256,
                                    cv1_filter=(1, 1))
    inception_4a = concatenate([inception_4a_3x3, inception_4a_5x5, inception_4a_pool, inception_4a_1x1], axis=3)

    #inception4e
    inception_4e_3x3 = conv2d_bn(inception_4a,
                                    layer='inception_4e_3x3',
                                    cv1_out=160,
                                    cv1_filter=(1, 1),
                                    cv2_out=256,
                                    cv2_filter=(3, 3),
                                    cv2_strides=(2, 2),
                                    padding=(1, 1))
    inception_4e_5x5 = conv2d_bn(inception_4a,
                                    layer='inception_4e_5x5',
                                    cv1_out=64,
                                    cv1_filter=(1, 1),
                                    cv2_out=128,
                                    cv2_filter=(5, 5),
                                    cv2_strides=(2, 2),
                                    padding=(2, 2))
    inception_4e_pool = MaxPooling2D(pool_size=3, strides=2)(inception_4a)
    inception_4e_pool = ZeroPadding2D(padding=((0, 1), (0, 1)))(inception_4e_pool)

    inception_4e = concatenate([inception_4e_3x3, inception_4e_5x5, inception_4e_pool], axis=3)
```

```
    #inception5a
    inception_5a_3x3 = conv2d_bn(inception_4e,
                                 layer='inception_5a_3x3',
                                 cv1_out=96,
                                 cv1_filter=(1, 1),
                                 cv2_out=384,
                                 cv2_filter=(3, 3),
                                 cv2_strides=(1, 1),
                                 padding=(1, 1))

    inception_5a_pool = AveragePooling2D(pool_size=(3, 3), strides=(3, 3))(inception_4e)
    inception_5a_pool = conv2d_bn(inception_5a_pool,
                                  layer='inception_5a_pool',
                                  cv1_out=96,
                                  cv1_filter=(1, 1),
                                  padding=(1, 1))
    inception_5a_1x1 = conv2d_bn(inception_4e,
                                 layer='inception_5a_1x1',
                                 cv1_out=256,
                                 cv1_filter=(1, 1))

    inception_5a = concatenate([inception_5a_3x3, inception_5a_pool, inception_5a_1x1], axis=3)

    #inception_5b
    inception_5b_3x3 = conv2d_bn(inception_5a,
                                 layer='inception_5b_3x3',
                                 cv1_out=96,
                                 cv1_filter=(1, 1),
                                 cv2_out=384,
                                 cv2_filter=(3, 3),
                                 cv2_strides=(1, 1),
                                 padding=(1, 1))
    inception_5b_pool = MaxPooling2D(pool_size=3, strides=2)(inception_5a)
    inception_5b_pool = conv2d_bn(inception_5b_pool,
                                  layer='inception_5b_pool',
                                  cv1_out=96,
                                  cv1_filter=(1, 1))
    inception_5b_pool = ZeroPadding2D(padding=(1, 1))(inception_5b_pool)

    inception_5b_1x1 = conv2d_bn(inception_5a,
                                 layer='inception_5b_1x1',
                                 cv1_out=256,
                                 cv1_filter=(1, 1))
    inception_5b = concatenate([inception_5b_3x3, inception_5b_pool, inception_5b_1x1], axis=3)

    av_pool = AveragePooling2D(pool_size=(3, 3), strides=(1, 1))(inception_5b)
    reshape_layer = Flatten()(av_pool)
    dense_layer = Dense(128, name='dense_layer')(reshape_layer)
    norm_layer = Lambda(lambda  x: K.l2_normalize(x, axis=1), name='norm_layer')(dense_layer)

    return Model(inputs=[myInput], outputs=norm_layer)
```

Explicación del Código de Redes Neuronales Siamesas con Pérdida de Tripletas Este código implementa un modelo de red neuronal basado en la arquitectura de redes neuronales siamesas, utilizando un mecanismo de pérdida llamado pérdida de tripletas (triplet loss). La pérdida de tripletas se usa comúnmente en tareas de aprendizaje de similitud, como la verificación facial y la detección de identidad. A continuación, se explica cada uno de los conceptos y su relación con este modelo:

Concepto de Imágenes Ancla, Positivas y Negativas

- Imagen Ancla (in_a): Es una imagen de referencia utilizada para la comparación. El modelo intenta aprender una representación (embedding) de esta imagen que capture sus características únicas.
- Imagen Positiva (in_p): Es una imagen que pertenece a la misma clase o identidad que la imagen ancla. El objetivo del modelo es aprender representaciones similares para la imagen ancla y la imagen positiva, minimizando la distancia entre sus embeddings.

- Imagen Negativa (in_n): Es una imagen de una clase o identidad diferente a la de la imagen ancla. El objetivo es maximizar la distancia entre el embedding de la imagen ancla y el embedding de la imagen negativa, asegurando que el modelo pueda diferenciar correctamente entre clases distintas.

Objetivo del Entrenamiento del Modelo

El objetivo del entrenamiento es aprender una función de embedding (f(x)) para una imagen (x), tal que:

1. La **distancia cuadrada L2** entre todas las imágenes de la misma identidad sea pequeña.
2. La distancia entre un par de imágenes de diferentes identidades sea grande.

Pérdida de Tripleta

Para lograr este objetivo, se utiliza una **pérdida de tripleta** (L) que se minimiza bajo las siguientes condiciones:

- Sea (x^a_i) una imagen ancla (anchor) de una identidad, (x^p_i) una imagen positiva de la misma identidad y (x^n_i) una imagen negativa de una identidad diferente.
- La pérdida se minimiza cuando la distancia entre la imagen ancla y la imagen positiva en el espacio de embeddings es menor que la distancia entre la imagen ancla y la imagen negativa, por un margen (\alpha).

Fórmula de la Pérdida de Tripleta

La pérdida de tripleta se define matemáticamente como:

$$L = \sum_{i=1}^{m} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

Componentes de la Fórmula

- $\|f(x_i^a) - f(x_i^p)\|_2^2$: Este término calcula la distancia cuadrada entre el embedding de la imagen ancla y el embedding de la imagen positiva.
- $\|f(x_i^a) - f(x_i^n)\|_2^2$: Este término calcula la distancia cuadrada entre el embedding de la imagen ancla y el embedding de la imagen negativa.
- α : Es un margen que se añade para asegurar que la distancia entre el ancla y la negativa sea al menos un cierto valor mayor que la distancia entre el ancla y la positiva.
- $[z]_+$: Esta notación significa que se toma el máximo entre z y 0, es decir, si el resultado es negativo, se considera como 0. Esto se utiliza para asegurar que solo se penalizan las condiciones que no se cumplen (es decir, si la distancia entre el ancla y la positiva no es suficientemente menor que la distancia entre el ancla y la negativa).

Implementación en Keras

Se menciona que la pérdida de tripleta en Keras se implementa mejor como una capa personalizada porque la función de pérdida no sigue el patrón habitual loss(input, target). En lugar de eso, la capa personalizada:

- Llama al método self.add_loss() para añadir la pérdida de tripleta al modelo. Esto permite que la pérdida se integre correctamente en el flujo de entrenamiento del modelo.

```
import numpy as np

# triplet_generator() crea un generador que devuelve continuamente
# tuplas ([a_batch, p_batch, n_batch], None) donde a_batch, p_batch
# y n_batch son lotes de imágenes RGB ancla, positiva y negativa,
# cada una con una forma de (batch_size, 96, 96, 3).

def triplet_generator(batch_size=32):
    ''' Dummy triplet generator for API usage demo only. '''
    while True:
        a_batch = np.random.rand(batch_size, 96, 96, 3)
        p_batch = np.random.rand(batch_size, 96, 96, 3)
```

```
        n_batch = np.random.rand(batch_size, 96, 96, 3)
        yield (tf.convert_to_tensor(a_batch, dtype=tf.float32),
               tf.convert_to_tensor(p_batch, dtype=tf.float32),
               tf.convert_to_tensor(n_batch, dtype=tf.float32)), tf.zeros((batch_size,), dtype=tf.float32)
```

```
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Layer

# Crea el modelo nn4_small2
nn4_small2 = create_model()

# Entrada para imágenes ancla, positivas y negativas
in_a = Input(shape=(96, 96, 3))
in_p = Input(shape=(96, 96, 3))
in_n = Input(shape=(96, 96, 3))

# Salida para vectores de embedding ancla, positivos y negativos
# La instancia del modelo nn4_small se comparte (red neuronal siamés)
emb_a = nn4_small2(in_a)
emb_p = nn4_small2(in_p)
emb_n = nn4_small2(in_n)

class TripletLossLayer(Layer):
    def __init__(self, alpha, **kwargs):
        self.alpha = alpha
        super(TripletLossLayer, self).__init__(**kwargs)

    def triplet_loss(self, inputs):
        a, p, n = inputs
        p_dist = K.sum(K.square(a-p), axis=-1)
        n_dist = K.sum(K.square(a-n), axis=-1)
        return K.sum(K.maximum(p_dist - n_dist + self.alpha, 0), axis=0)

    def call(self, inputs):
        loss = self.triplet_loss(inputs)
        self.add_loss(loss)
        return loss

# Capa que calcula la pérdida de tripletas a partir de vectores de embedding ancla, positivos y negativos
triplet_loss_layer = TripletLossLayer(alpha=0.2, name='triplet_loss_layer')([emb_a, emb_p, emb_n])

# Modelo que se puede entrenar con imágenes ancla, positivas y negativas
nn4_small2_train = Model([in_a, in_p, in_n], triplet_loss_layer)
```

Entrenamiento

Ejecutamos la funcion del generador de lotes tripletes de imagenes RGB.

```
# triplet_generator() creates a generator that continuously returns
# ([a_batch, p_batch, n_batch], None) tuples where a_batch, p_batch
# and n_batch are batches of anchor, positive and negative RGB images
# each having a shape of (batch_size, 96, 96, 3).

#batch_size = 4
#generator = triplet_generator(batch_size)
```

Entrenar

```
# nn4_small2_train.compile(optimizer='adam')
# nn4_small2_train.fit(dataset, epochs=10, steps_per_epoch=100)
#nn4_small2_train.fit_generator(generator, epochs=10, steps_per_epoch=100)

# Please note that the current implementation of the generator only generates
# random image data. The main goal of this code snippet is to demonstrate
```

```
# the general setup for model training. In the following, we will anyway
# use a pre-trained model so we don't need a generator here that operates
# on real training data. I'll maybe provide a fully functional generator
# later.
```

El fragmento de código anterior debería mostrar únicamente cómo configurar el entrenamiento del modelo. Sin embargo, en lugar de entrenar un modelo desde cero, utilizaremos un modelo preentrenado, ya que entrenar desde cero es muy costoso y requiere conjuntos de datos enormes para lograr un buen rendimiento de generalización.

El proyecto OpenFace proporciona modelos preentrenados que fueron entrenados con los conjuntos de datos públicos de reconocimiento facial FaceScrub y CASIA-WebFace. El proyecto Keras-OpenFace convirtió los pesos del modelo preentrenado nn4.small2.v1 a archivos CSV, que luego fueron convertidos aquí a un formato binario que Keras puede cargar con load_weights.

```
nn4_small2.load_weights('/content/drive/MyDrive/Trabajo Integrador Redes Neuronales/weights/nn4.small2.v1.h5')
```

Prueba con un el dataset VGGFace2

Este código carga metadatos de un conjunto de imágenes de un directorio de base, organizando la información en objetos que representan cada imagen. La clase IdentityMetadata y la función load_metadata ayudan a estructurar y obtener los datos de las imágenes de forma ordenada.

```
import os
import numpy as np

class IdentityMetadata():
    def __init__(self, base, name, file):
        # directorio base del dataset
        self.base = base
        # nombre de la identidad
        self.name = name
        # nombre del archivo de imagen
        self.file = file

    def __repr__(self):
        return self.image_path()

    def image_path(self):
        # construye y devuelve la ruta completa de la imagen
        return os.path.join(self.base, self.name, self.file)

def load_metadata(path):
    metadata = []
    for i in sorted(os.listdir(path)):
        for f in sorted(os.listdir(os.path.join(path, i))):
            # Verifica la extensión del archivo. Solo permite archivos jpg/jpeg/png.
            ext = os.path.splitext(f)[1]
            if (ext in ['.jpg', '.jpeg', '.png']) and 'png_inverted.png' not in f and 'jpg_inverted.png' not in f and
                metadata.append(IdentityMetadata(path, i, f))
    return np.array(metadata)

# Carga los metadatos de las imágenes en el dataset
#metadata = load_metadata('/content/drive/MyDrive/Trabajo Integrador Redes Neuronales/Dataset - VGGFace2/train')
metadata = load_metadata('/content/drive/MyDrive/Trabajo Integrador Redes Neuronales/FotosCiro')
```

```
import cv2

def load_image(path):
    img = cv2.imread(path, 1)
    # OpenCV loads images with color channels
    # in BGR order. So we need to reverse them
    return img[...,::-1]
```

Alineacion de rostros

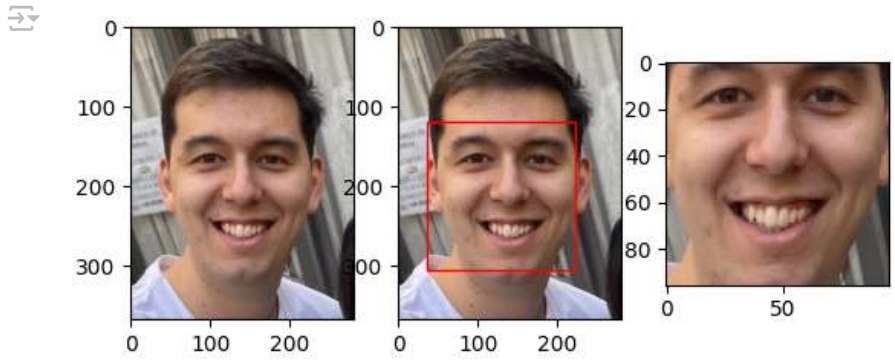
El modelo nn4.small2 fue entrenado con imágenes de rostros alineadas, por lo que las imágenes faciales del conjunto de datos personalizado también deben estar alineadas. Aquí usamos Dlib para la detección de rostros y OpenCV para la transformación y recorte de imágenes, con el fin de producir imágenes faciales RGB alineadas de 96x96. Al utilizar la utilidad AlignDlib del proyecto OpenFace, este proceso es sencillo:

```
# Copyright 2015-2016 Carnegie Mellon University
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Module for dlib-based alignment."""

import cv2
import dlib
import numpy as np

TEMPLATE = np.float32([
    (0.0792396913815, 0.339223741112), (0.0829219487236, 0.456955367943),
    (0.0967927109165, 0.575648016728), (0.122141515615, 0.691921601066),
    (0.168687863544, 0.800341263616), (0.239789390707, 0.895732504778),
    (0.325662452515, 0.977068762493), (0.422318282013, 1.04329000149),
    (0.531777802068, 1.06080371126), (0.641296298053, 1.03981924107),
    (0.738105872266, 0.972268833998), (0.824444363295, 0.889624082279),
    (0.894792677532, 0.792494155836), (0.939395486253, 0.681546643421),
    (0.96111933829, 0.562238253072), (0.970579841181, 0.441758925744),
    (0.971193274221, 0.322118743967), (0.163846223133, 0.249151738053),
    (0.21780354657, 0.204255863861), (0.291299351124, 0.192367318323),
    (0.367460241458, 0.203582210627), (0.4392945113, 0.233135599851),
    (0.586445962425, 0.228141644834), (0.660152671635, 0.195923841854),
    (0.737466449096, 0.182360984545), (0.813236546239, 0.192828009114),
    (0.8707571886, 0.235293377042), (0.51534533827, 0.31863546193),
    (0.516221448289, 0.396200446263), (0.517118861835, 0.473797687758),
    (0.51816430343, 0.553157797772), (0.433701156035, 0.604054457668),
    (0.475501237769, 0.62076344024), (0.520712933176, 0.634268222208),
    (0.565874114041, 0.618796581487), (0.607054002672, 0.60157671656),
    (0.252418718401, 0.331052263829), (0.298663015648, 0.302646354002),
    (0.355749724218, 0.303020650651), (0.403718978315, 0.33867711083),
    (0.352507175597, 0.349987615384), (0.296791759886, 0.350478978225),
    (0.631326076346, 0.334136672344), (0.679073381078, 0.29645404267),
    (0.73597236153, 0.294721285802), (0.782865376271, 0.321305281656),
    (0.740312274764, 0.341849376713), (0.68499850091, 0.343734332172),
    (0.353167761422, 0.746189164237), (0.414587777921, 0.719053835073),
    (0.477677654595, 0.706835892494), (0.522732900812, 0.717092275768),
    (0.569832064287, 0.705414478982), (0.635195811927, 0.71565572516),
    (0.69951672331, 0.739419187253), (0.639447159575, 0.805236879972),
    (0.576410514055, 0.835436670169), (0.525398405766, 0.841706377792),
    (0.47641545769, 0.837505914975), (0.41379548902, 0.810045601727),
    (0.380084785646, 0.749979603086), (0.477955996282, 0.74513234612),
    (0.523389793327, 0.748924302636), (0.571057789237, 0.74332894691),
    (0.672409137852, 0.744177032192), (0.572539621444, 0.776609286626),
    (0.5240106503, 0.783370783245), (0.477561227414, 0.778476346951)])

TPL_MIN, TPL_MAX = np.min(TEMPLATE, axis=0), np.max(TEMPLATE, axis=0)
MINMAX_TEMPLATE = (TEMPLATE - TPL_MIN) / (TPL_MAX - TPL_MIN)


class AlignDlib:
```

```
    """
    Use `dlib's landmark estimation <http://blog.dlib.net/2014/08/real-time-face-pose-estimation.html>`_ to align face

    The alignment preprocess faces for input into a neural network.
    Faces are resized to the same size (such as 96x96) and transformed
    to make landmarks (such as the eyes and nose) appear at the same
    location on every image.

    Normalized landmarks:

    .. image:: ../images/dlib-landmark-mean.png
    """

    #: Landmark indices.
    INNER_EYES_AND_BOTTOM_LIP = [39, 42, 57]
    OUTER_EYES_AND_NOSE = [36, 45, 33]

    def __init__(self, facePredictor):
        """
        Instantiate an 'AlignDlib' object.

        :param facePredictor: The path to dlib's
        :type facePredictor: str
        """
        assert facePredictor is not None

        self.detector = dlib.get_frontal_face_detector()
        self.predictor = dlib.shape_predictor(facePredictor)

    def getAllFaceBoundingBoxes(self, rgbImg):
        """
        Find all face bounding boxes in an image.

        :param rgbImg: RGB image to process. Shape: (height, width, 3)
        :type rgbImg: numpy.ndarray
        :return: All face bounding boxes in an image.
        :rtype: dlib.rectangles
        """
        assert rgbImg is not None

        try:
            return self.detector(rgbImg, 1)
        except Exception as e:
            print("Warning: {}".format(e))
            # In rare cases, exceptions are thrown.
            return []

    def getLargestFaceBoundingBox(self, rgbImg, skipMulti=False):
        """
        Find the largest face bounding box in an image.

        :param rgbImg: RGB image to process. Shape: (height, width, 3)
        :type rgbImg: numpy.ndarray
        :param skipMulti: Skip image if more than one face detected.
        :type skipMulti: bool
        :return: The largest face bounding box in an image, or None.
        :rtype: dlib.rectangle
        """
        assert rgbImg is not None

        faces = self.getAllFaceBoundingBoxes(rgbImg)
        if (not skipMulti and len(faces) > 0) or len(faces) == 1:
            return max(faces, key=lambda rect: rect.width() * rect.height())
        else:
            return None

    def findLandmarks(self, rgbImg, bb):
        """
        Find the landmarks of a face.

        :param rgbImg: RGB image to process. Shape: (height, width, 3)
        :type rgbImg: numpy.ndarray
```

```
            :param bb: Bounding box around the face to find landmarks for.
            :type bb: dlib.rectangle
            :return: Detected landmark locations.
            :rtype: list of (x,y) tuples
            """
            assert rgbImg is not None
            assert bb is not None

            points = self.predictor(rgbImg, bb)
            return list(map(lambda p: (p.x, p.y), points.parts()))

        def align(self, imgDim, rgbImg, bb=None,
                  landmarks=None, landmarkIndices=INNER_EYES_AND_BOTTOM_LIP,
                  skipMulti=False):
            r"""align(imgDim, rgbImg, bb=None, landmarks=None, landmarkIndices=INNER_EYES_AND_BOTTOM_LIP)

            Transform and align a face in an image.

            :param imgDim: The edge length in pixels of the square the image is resized to.
            :type imgDim: int
            :param rgbImg: RGB image to process. Shape: (height, width, 3)
            :type rgbImg: numpy.ndarray
            :param bb: Bounding box around the face to align. \
                       Defaults to the largest face.
            :type bb: dlib.rectangle
            :param landmarks: Detected landmark locations. \
                              Landmarks found on `bb` if not provided.
            :type landmarks: list of (x,y) tuples
            :param landmarkIndices: The indices to transform to.
            :type landmarkIndices: list of ints
            :param skipMulti: Skip image if more than one face detected.
            :type skipMulti: bool
            :return: The aligned RGB image. Shape: (imgDim, imgDim, 3)
            :rtype: numpy.ndarray
            """
            assert imgDim is not None
            assert rgbImg is not None
            assert landmarkIndices is not None

            if bb is None:
                bb = self.getLargestFaceBoundingBox(rgbImg, skipMulti)
                if bb is None:
                    return

            if landmarks is None:
                landmarks = self.findLandmarks(rgbImg, bb)

            npLandmarks = np.float32(landmarks)
            npLandmarkIndices = np.array(landmarkIndices)

            H = cv2.getAffineTransform(npLandmarks[npLandmarkIndices],
                                       imgDim * MINMAX_TEMPLATE[npLandmarkIndices])
            thumbnail = cv2.warpAffine(rgbImg, H, (imgDim, imgDim))

            return thumbnail


    import cv2
    import matplotlib.pyplot as plt
    import matplotlib.patches as patches


    %matplotlib inline

    def load_image(path):
        img = cv2.imread(path, 1)
        # OpenCV loads images with color channels
        # in BGR order. So we need to reverse them
        return img[...,::-1]

    # Initialize the OpenFace face alignment utility
    alignment = AlignDlib('models/landmarks.dat')
```

```
# Load an image of Jacques Chirac
jc_orig = load_image(metadata[48].image_path())

# Detect face and return bounding box
bb = alignment.getLargestFaceBoundingBox(jc_orig)

# Transform image using specified face landmark indices and crop image to 96x96
jc_aligned = alignment.align(96, jc_orig, bb, landmarkIndices=AlignDlib.OUTER_EYES_AND_NOSE)

# Show original image
plt.subplot(131)
plt.imshow(jc_orig)

# Show original image with bounding box
plt.subplot(132)
plt.imshow(jc_orig)
plt.gca().add_patch(patches.Rectangle((bb.left(), bb.top()), bb.width(), bb.height(), fill=False, color='red'))

# Show aligned image
plt.subplot(133)
plt.imshow(jc_aligned);
```




```
def align_image(img):
    return alignment.align(96, img, alignment.getLargestFaceBoundingBox(img),
                           landmarkIndices=AlignDlib.OUTER_EYES_AND_NOSE)
```

Verifiquemos en un solo ejemplo de tripleta que la distancia L2 al cuadrado entre su par ancla-positiva es menor que la distancia entre su par ancla-negativa.

Evaluación del modelo

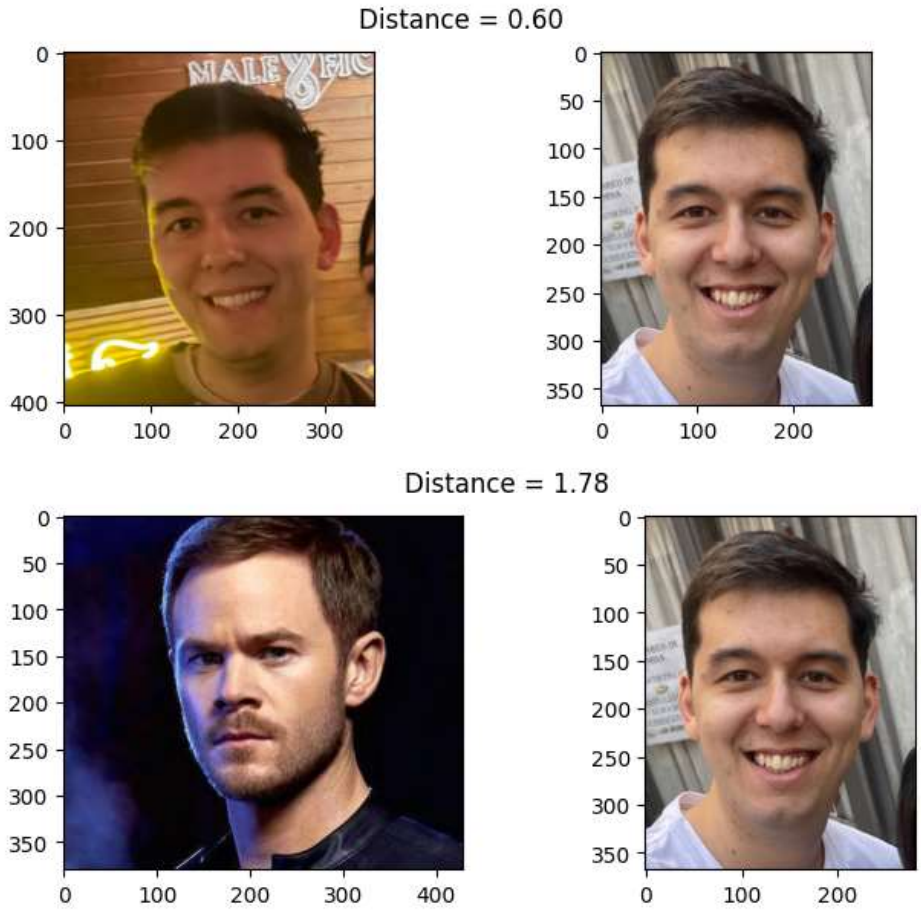
Este código implementa un proceso de clasificación para un conjunto de datos utilizando dos algoritmos: K-Nearest Neighbors (KNN) y una máquina de vectores de soporte lineal (SVM). Los datos contienen representaciones (embeddings) de imágenes de rostros, con las que se intenta clasificar a las personas basándose en sus identidades.

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score, accuracy_score

# Se obtienen los nombres de las identidades de las imágenes en el metadata
targets = np.array([m.name for m in metadata])

# Inicializa el codificador de etiquetas para transformar los nombres en valores numéricos
encoder = LabelEncoder()
encoder.fit(targets)

# Codificación numérica de las identidades
```

```
y = encoder.transform(targets)

# Selecciona índices para dividir el dataset en entrenamiento y prueba (pares para prueba, impares para entrenamiento)
train_idx = np.arange(metadata.shape[0]) % 2 != 0
test_idx = np.arange(metadata.shape[0]) % 2 == 0

# 50 ejemplos de entrenamiento y 50 de testeo de 10 identidades (5 ejemplos cada uno)
X_train = embedded[train_idx]
X_test = embedded[test_idx]

# Etiquetas de entrenamiento y prueba
y_train = y[train_idx]
y_test = y[test_idx]

# Inicializa el clasificador KNN con 1 vecino y métrica de distancia euclidiana
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
# Entrena el modelo KNN con los datos de entrenamiento
knn.fit(X_train, y_train)
# Calcula la precisión del modelo KNN en los datos de prueba
acc_knn = accuracy_score(y_test, knn.predict(X_test))

# Inicializa el clasificador SVM con un núcleo lineal
svc = LinearSVC()
# Entrena el modelo SVM con los datos de entrenamiento
svc.fit(X_train, y_train)
# Calcula la precisión del modelo SVM en los datos de prueba
acc_svc = accuracy_score(y_test, svc.predict(X_test))

# Imprime la precisión de ambos clasificadores
print(f'KNN accuracy = {acc_knn}, SVM accuracy = {acc_svc}')
```

```
KNN accuracy = 0.9310344827586207, SVM accuracy = 0.9310344827586207
```

```
def distance(emb1, emb2):
    return np.sum(np.square(emb1 - emb2))

def show_pair(idx1, idx2):
    plt.figure(figsize=(8,3))
    plt.suptitle(f'Distance = {distance(embedded[idx1], embedded[idx2]):.2f}')
    plt.subplot(121)
    plt.imshow(load_image(metadata[idx1].image_path()))
    plt.subplot(122)
    plt.imshow(load_image(metadata[idx2].image_path()));

show_pair(49, 48)
show_pair(25, 48)
```

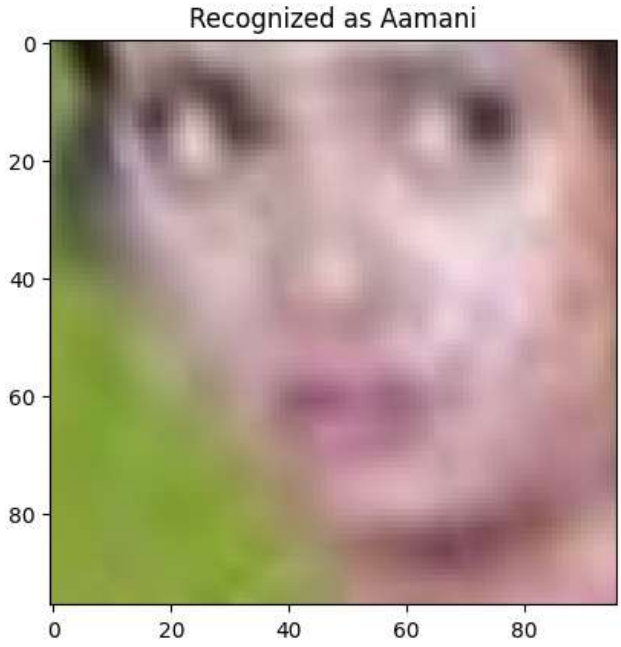


Clasificación y Recorte de rostros

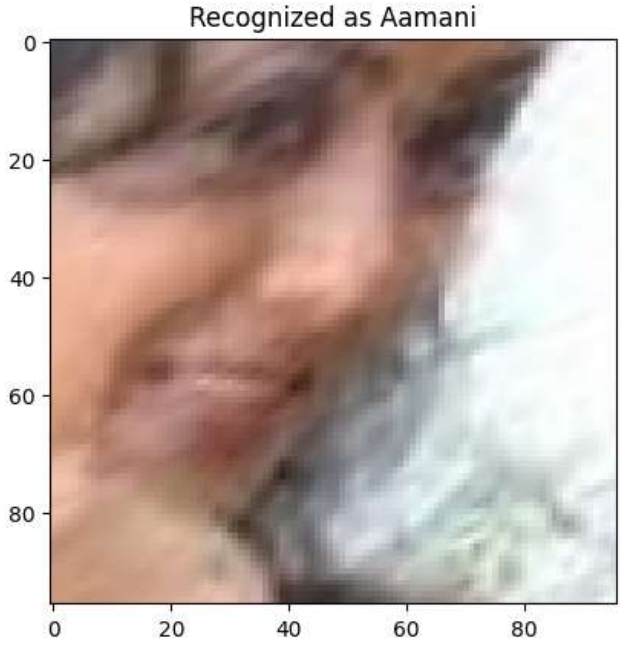
```
import numpy as np
import os
import warnings
from matplotlib import pyplot as plt
import cv2

# Suprimir advertencias
warnings.filterwarnings('ignore')

# Crear una matriz para almacenar los embeddings
embedded = np.zeros((metadata.shape[0], 128))

# Directorio principal donde se guardarán las imágenes clasificadas
output_dir = '/content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed_ciro'

# Crear el directorio de salida si no existe
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Procesar y clasificar las imágenes
for i, m in enumerate(metadata):
    img = load_image(m.image_path())
    # Alineacion
    if img is None:
        print(f"Failed to load image at {m.image_path()}")
        continue

    img = align_image(img)
    if img is None:
        print(f"Failed to align image at {m.image_path()}")
        continue

    # Normalizar la imagen y calcular el embedding
    img = (img / 255.).astype(np.float32)
```

```
        embedded[i] = nn4_small2.predict(np.expand_dims(img, axis=0))[0]

        # Ajustar la forma del embedding para la predicción
        embedding = embedded[i].reshape(1, -1)

        # Realizar la predicción de identidad usando el modelo `svc`
        prediction = svc.predict(embedding)
        identity = encoder.inverse_transform(prediction)[0]

        # Crear carpeta para la identidad si no existe
        identity_dir = os.path.join(output_dir, identity)
        if not os.path.exists(identity_dir):
            os.makedirs(identity_dir)

        # Convertir la imagen de normalizada (0-1) a formato BGR para guardar
        img_bgr = cv2.cvtColor((img * 255).astype(np.uint8), cv2.COLOR_RGB2BGR)

        # Guardar la imagen recortada y alineada en la carpeta correspondiente
        output_image_path = os.path.join(identity_dir, f"{os.path.basename(m.image_path())}")
        cv2.imwrite(output_image_path, img_bgr)  # Guardar en formato BGR

        # Mostrar la imagen con la etiqueta predicha
        plt.imshow(img)
        plt.title(f'Recognized as {identity}')
        plt.show()

        print(f"Imagen recortada y clasificada guardada en: {output_image_path}")
```

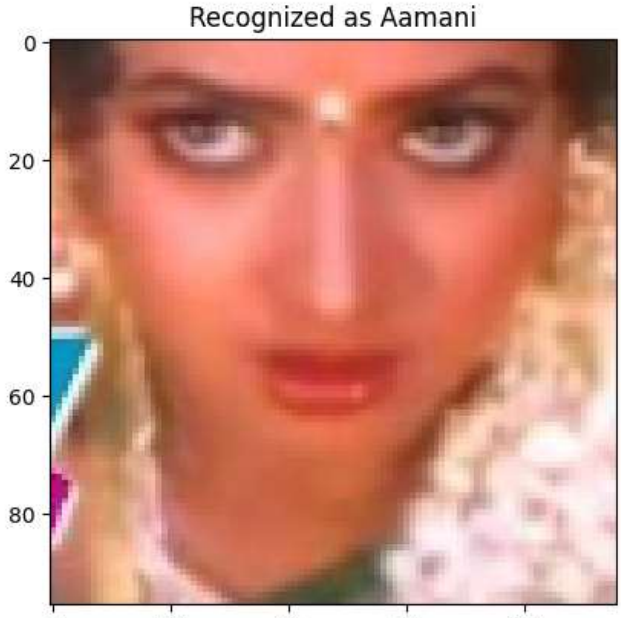
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
```

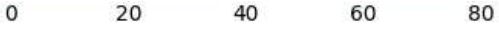



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
```



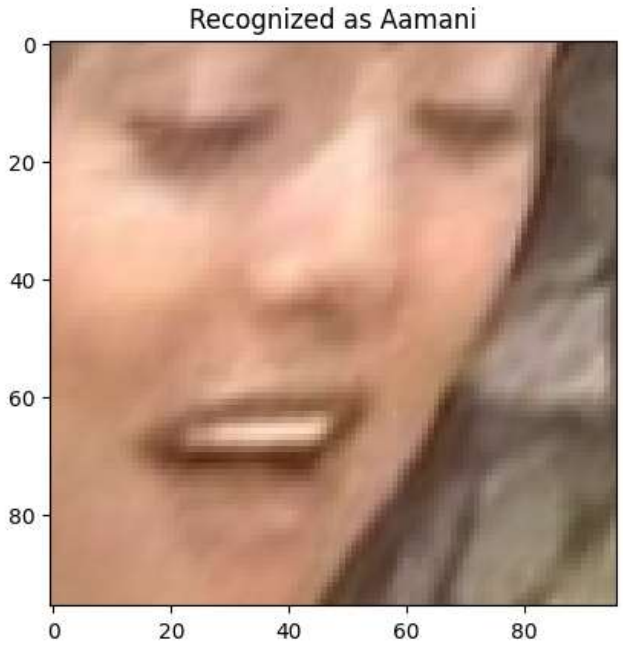

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
```



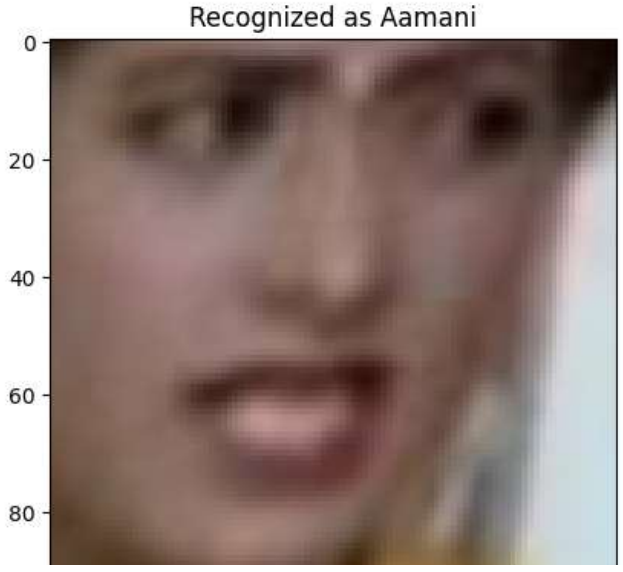



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```

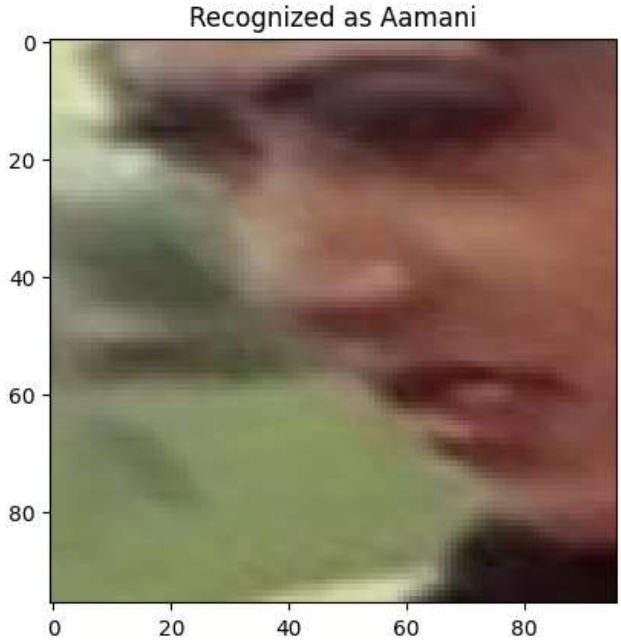

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```




```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

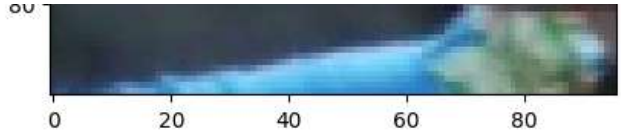



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

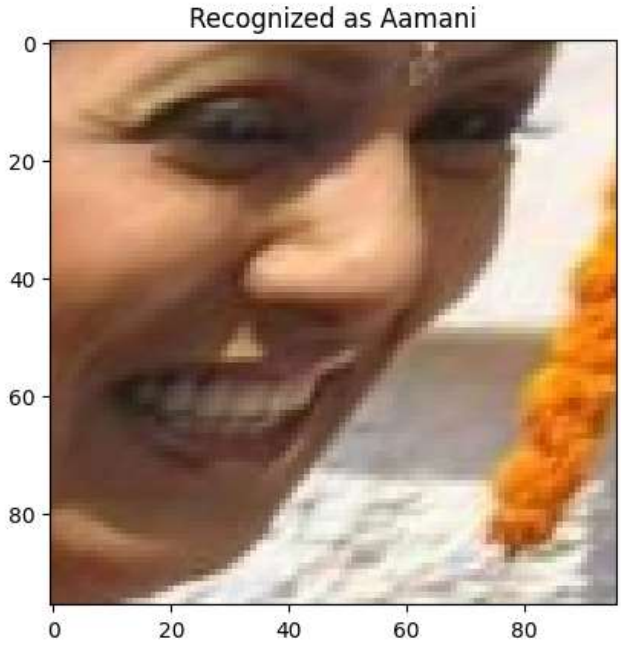



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

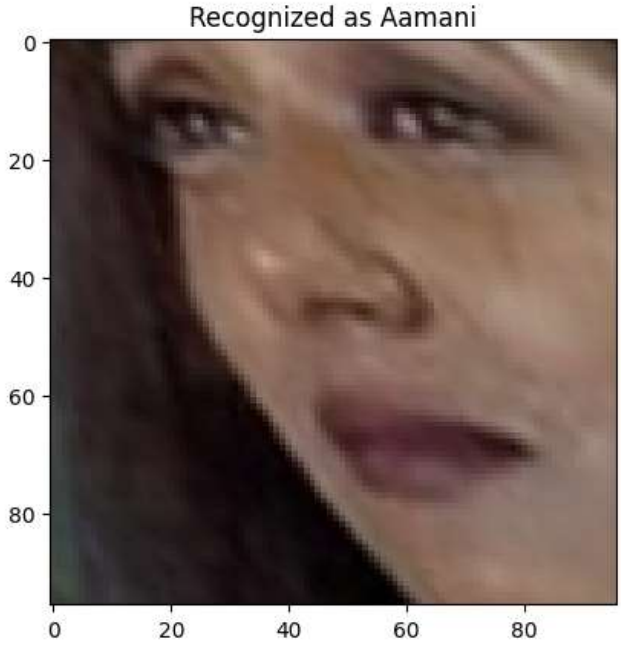

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



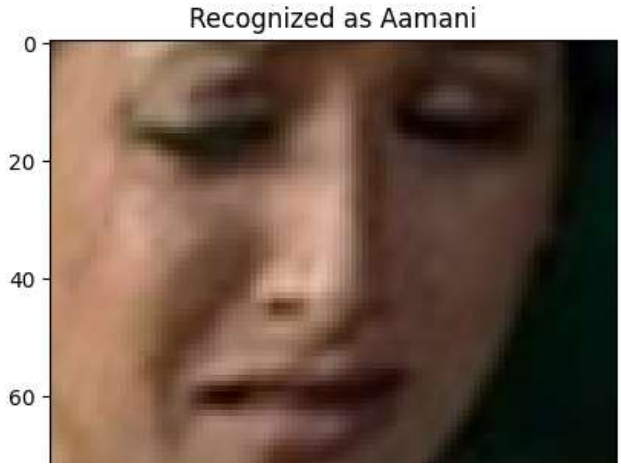

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

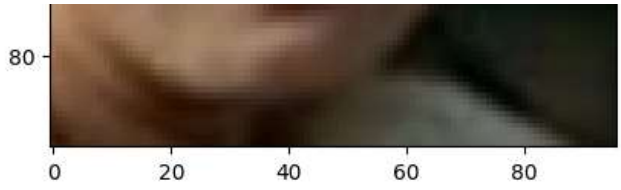



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

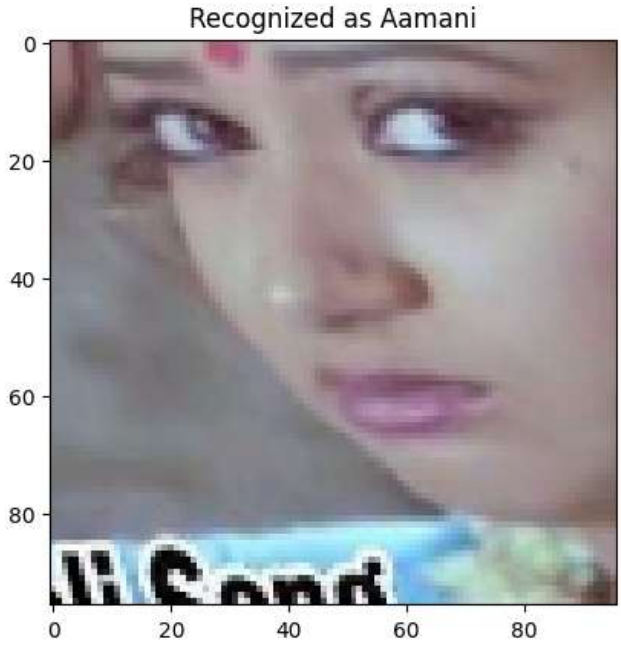



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

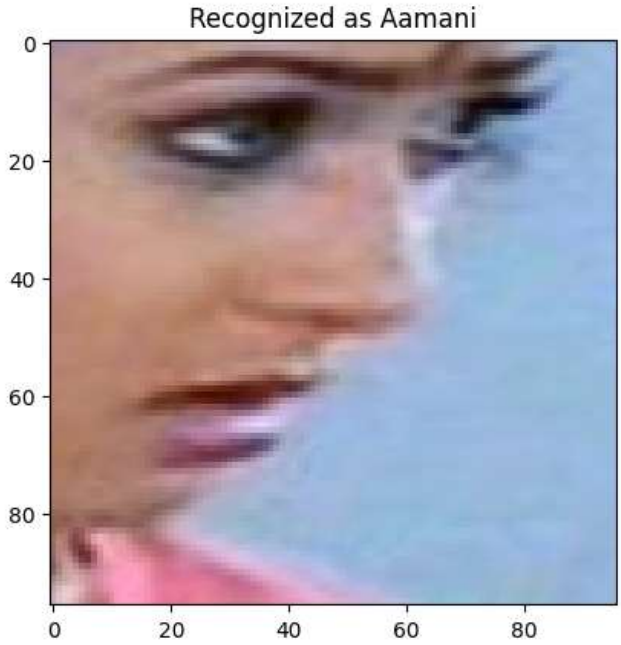




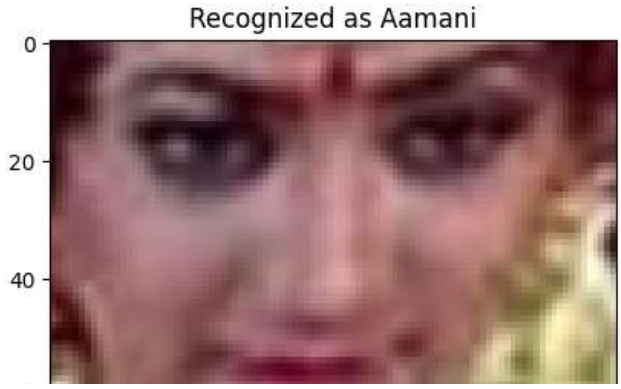

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

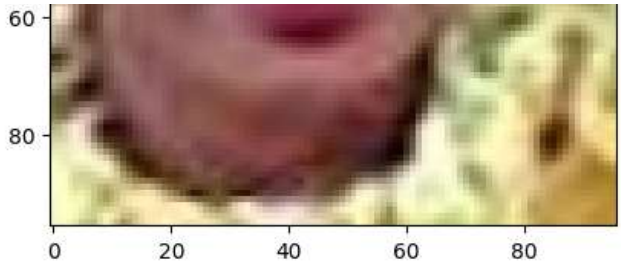



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
Failed to align image at /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/FotosCiro/Aamani/Copia de 0220
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



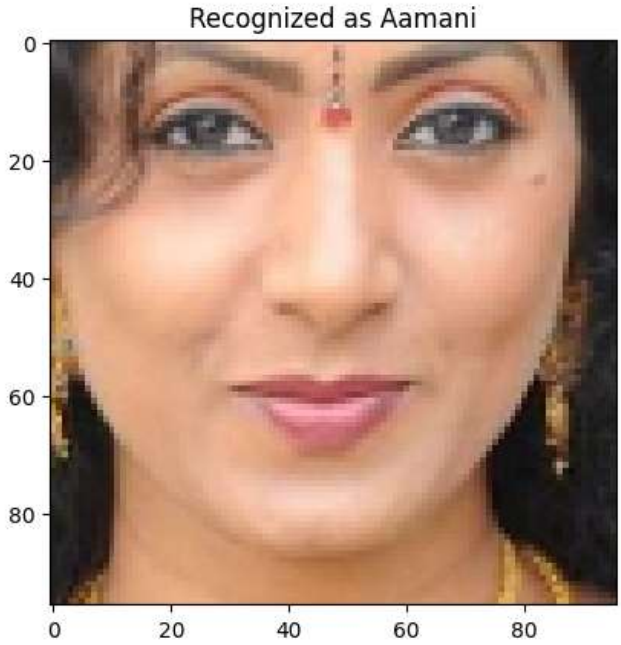

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



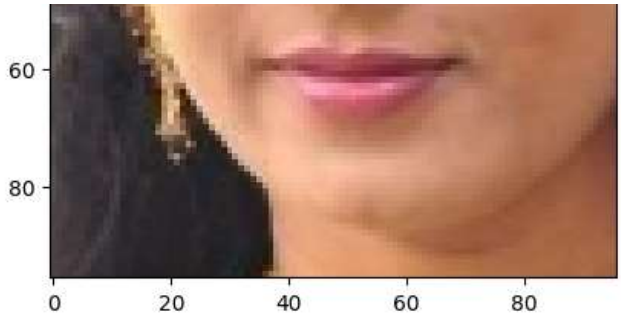


```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```


```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

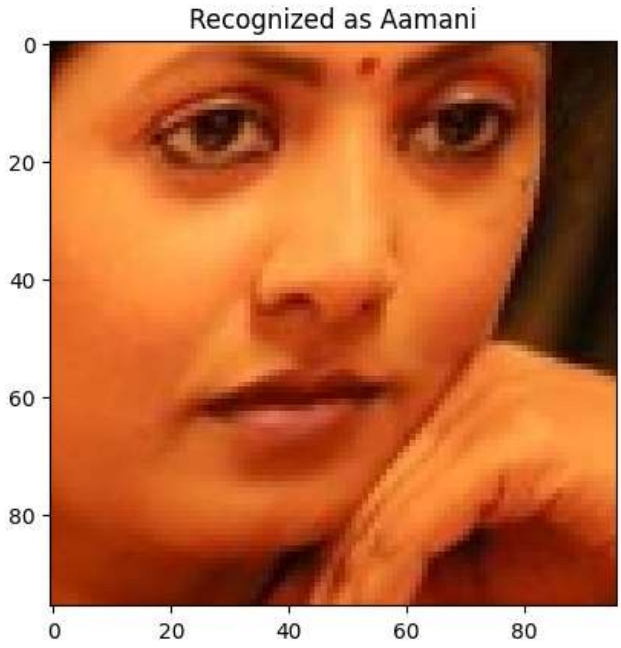



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

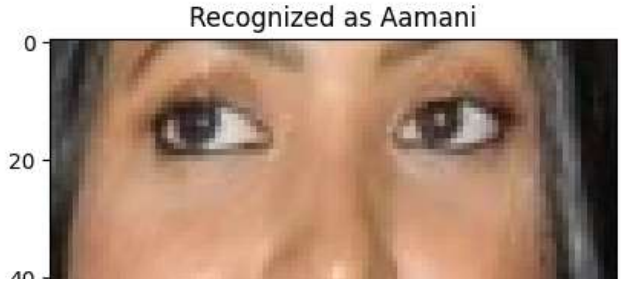


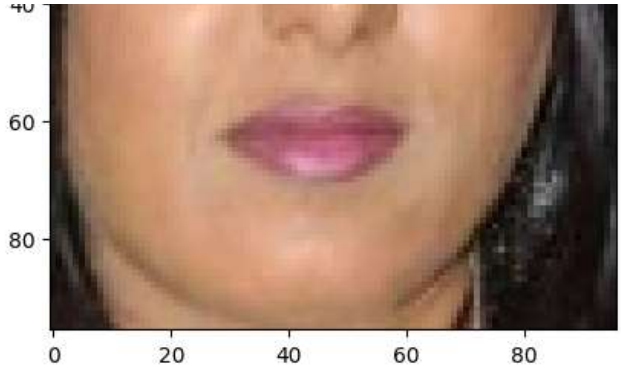
```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```


```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
```

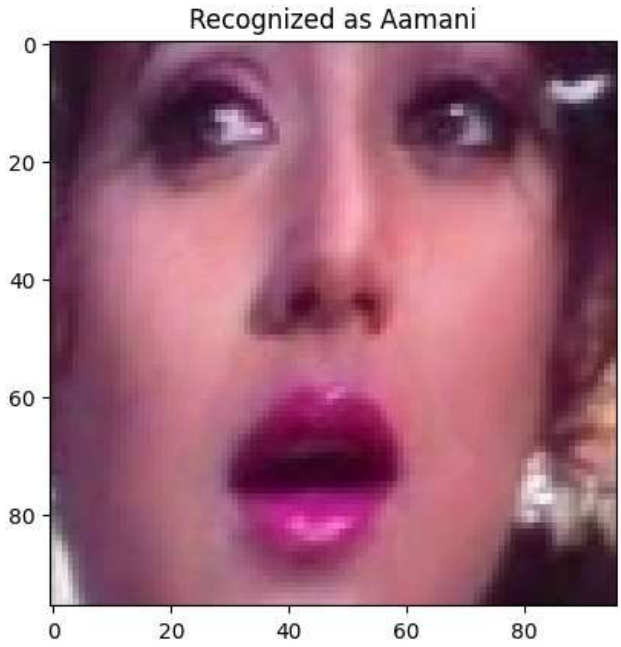



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

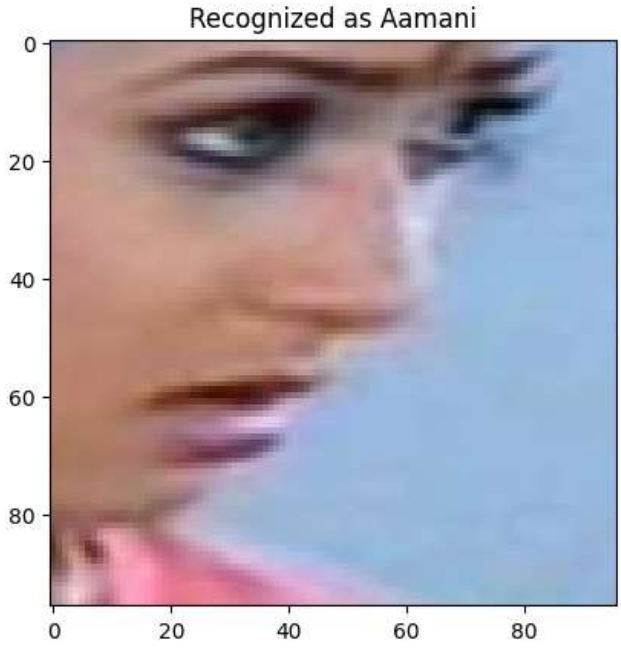




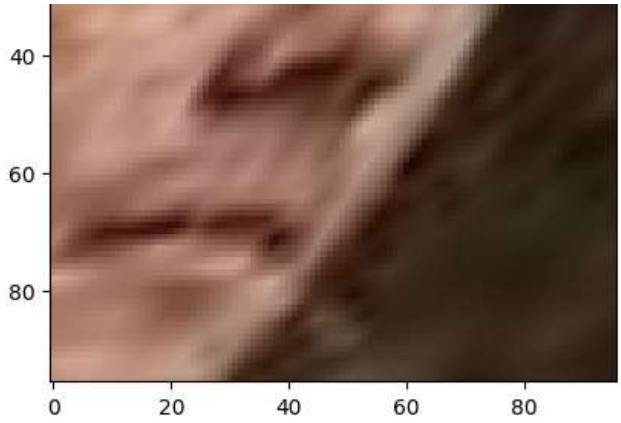

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```



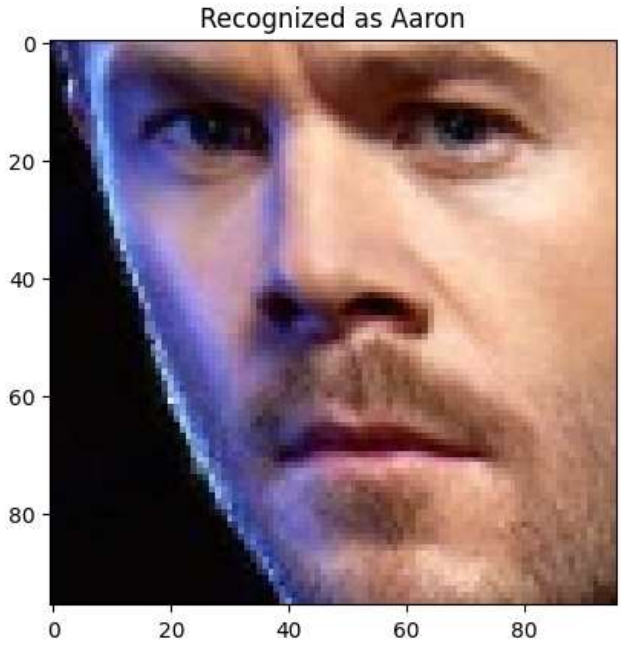

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
```




```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

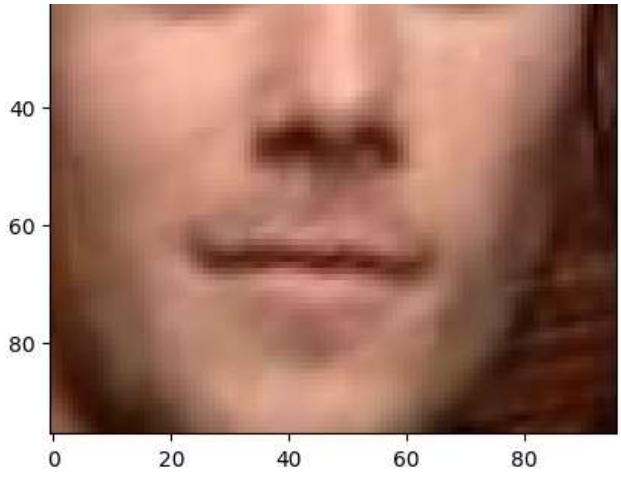


```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
```




```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
```

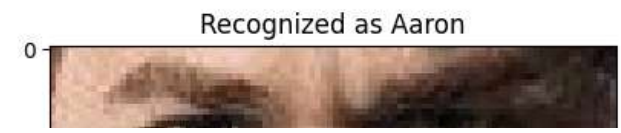

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```

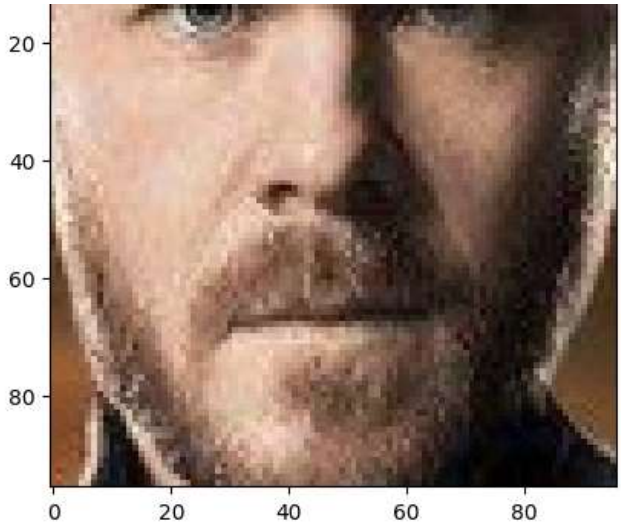



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

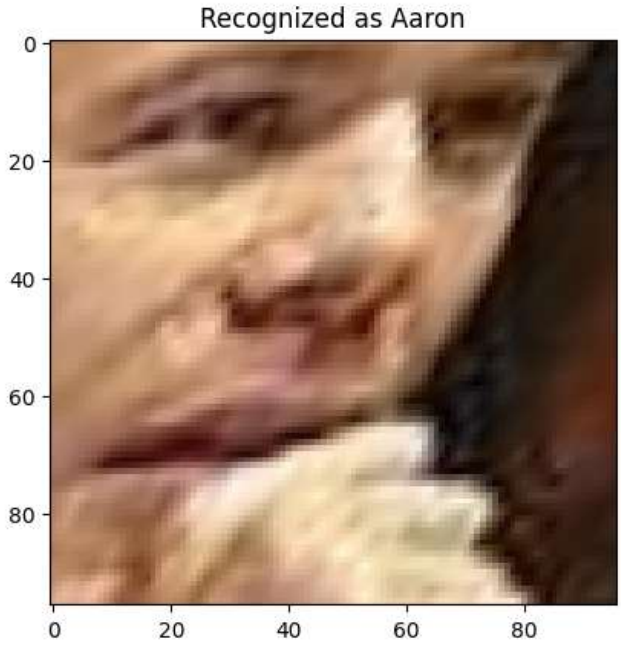

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
```


```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
```

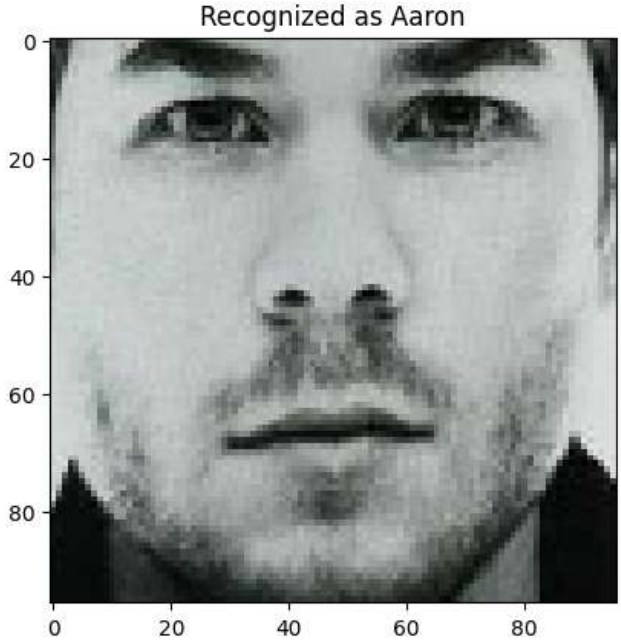




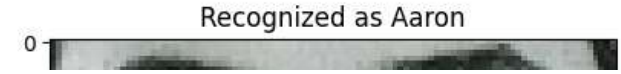
```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

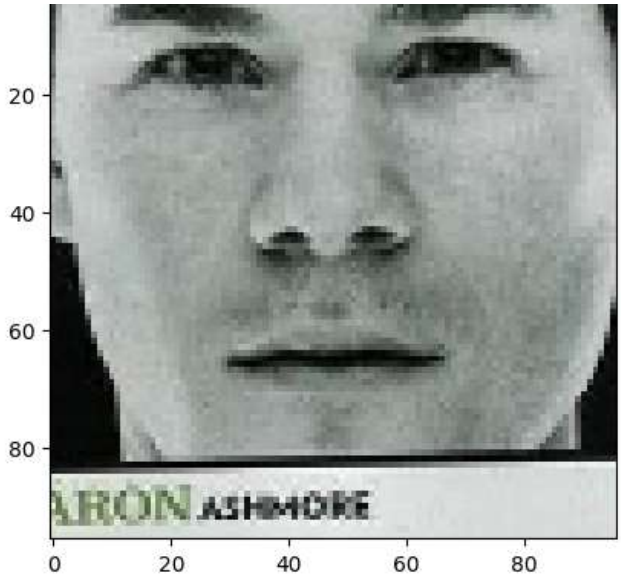



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

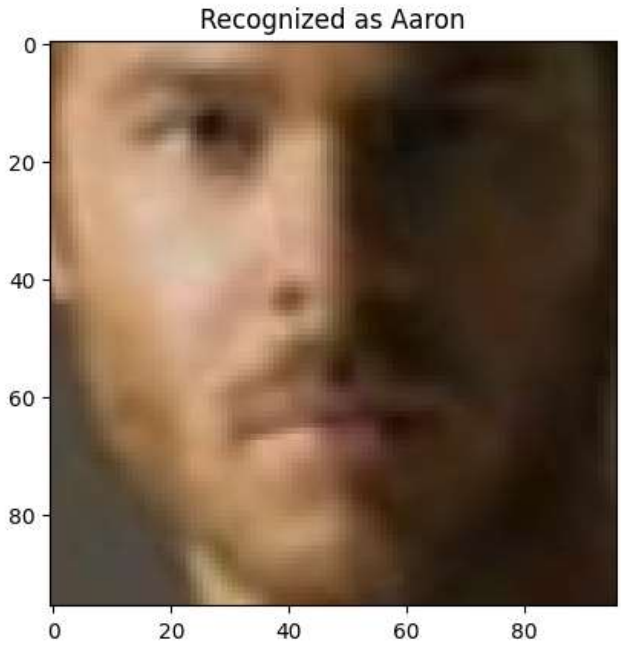



```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```





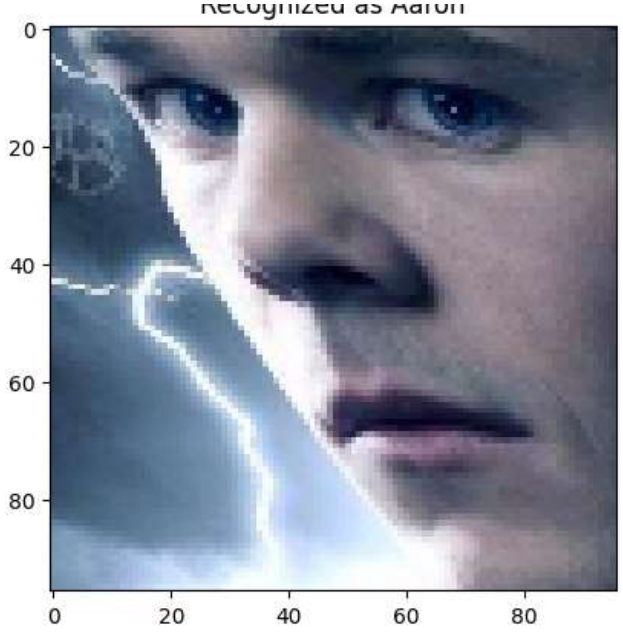

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```



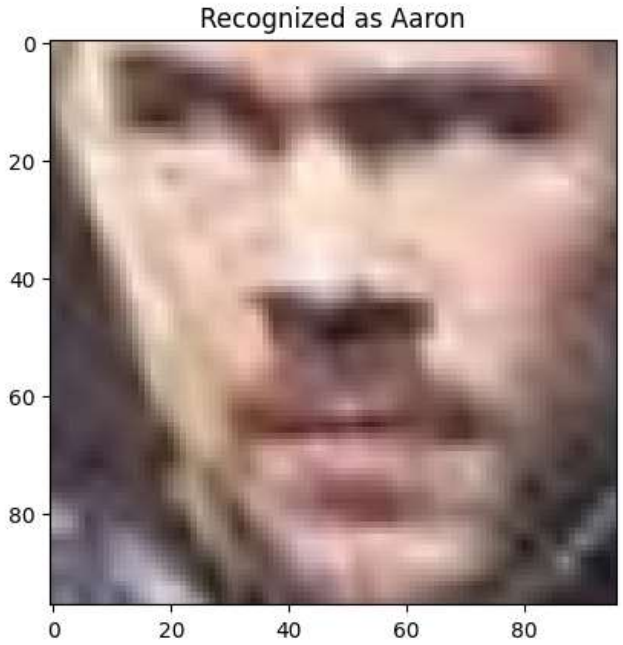

```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```


```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

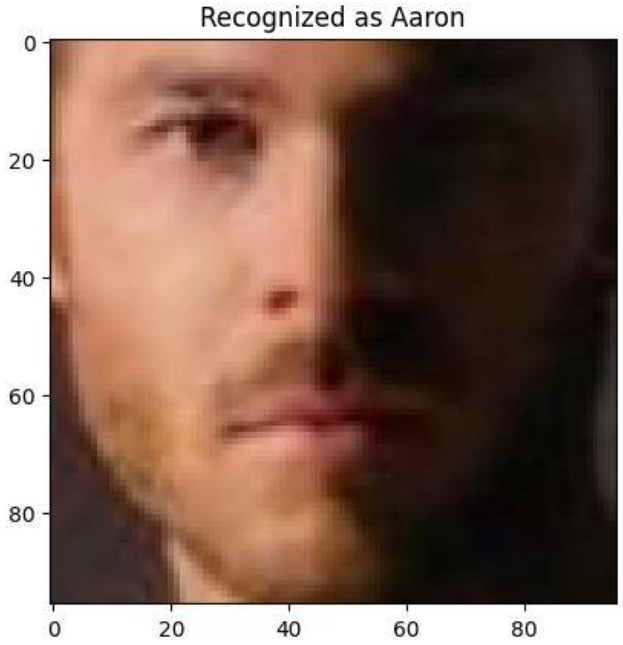
Recognized as Aaron




```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```




```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```




```
Imagen recortada y clasificada guardada en: /content/drive/MyDrive/Trabajo Integrador Redes Neuronales/preprocesed
```