Reconocimiento Facial en Python Mediante Modelos de Deep Learning

Contenido

Introducción

El reconocimiento de caras es un problema trivial para los humanos, sin embargo, para las máquinas no es algo tan fácil.

Aun así, se ha resuelto razonablemente bien mediante técnicas basadas en características, como los clasificadores en cascada.

La conexión en cascada es un caso particular de aprendizaje en conjunto basado en la concatenación de varios clasificadores, utilizando toda la información recopilada de la salida de un clasificador dado como información adicional para el siguiente clasificador en la cascada.

Más recientemente, los métodos de aprendizaje profundo (Deep Learning) han logrado resultados de vanguardia en conjuntos de datos de detección de rostros de referencia estándar. Un ejemplo es la red neuronal convolucional multitarea en cascada, o MTCNN para abreviar.

Reconocimiento Facial

La detección de rostros o reconocimiento facial es un problema de visión por computadora que consiste en localizar y situar uno o más rostros en una fotografía.

Situar una cara en una fotografía se refiere a encontrar las coordenadas de la cara en la imagen, mientras que localizar se refiere a delimitar la extensión de la cara, a menudo mediante un cuadro delimitador alrededor de esta.

Detectar rostros en una fotografía es algo fácil de resolver para los humanos, pero históricamente ha sido un desafío para las computadoras dada la naturaleza dinámica de los rostros. Por ejemplo, los rostros deben detectarse independientemente de la orientación o el ángulo en el que se encuentren, independientemente del color del cabello, del vello facial, del maquillaje, la edad, etc.

Dada una fotografía, un sistema de detección de rostros generará cero o más cuadros delimitadores que contengan rostros en su interior. Las caras detectadas se pueden proporcionar como entrada de a un sistema posterior, como un sistema de reconocimiento facial.

Existen **dos enfoques** principales para el reconocimiento facial: métodos basados en características que utilizan **filtros hechos a mano para buscar y detectar rostros**, y **métodos basados en imágenes que aprenden** de manera integral **cómo extraer rostros de la imagen completa.**

Fotografías de prueba

Para probar la eficacia de nuestros modelos, usaremos fotos en las que aparezcan rostros de frente. Usaremos dos fotos, una con una o dos caras y otra con un grupo de personas mirando a cámara.

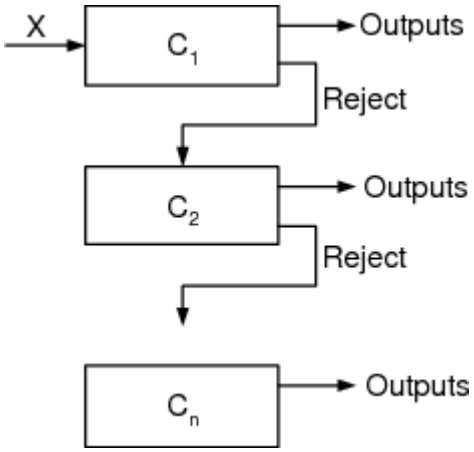
Reconocimiento Facial mediante la biblioteca OpenCV

Los algoritmos de detección de rostros basados en características son rápidos y efectivos y se han utilizado con éxito durante décadas.

Probablemente el ejemplo más exitoso es la técnica llamada clasificadores en cascada, descrita por primera vez por Paul Viola y Michael Jones en su paper de 2001 Rapid Object Detection using a Boosted Cascade of Simple Features.

En este caso las características son aprendidas mediante un algoritmo llamado AdaBoost y se hace uso de varios modelos organizados en forma de hilera o cascada.

El Boosting es una técnica de conjunto que intenta crear un clasificador fuerte a partir de una serie de clasificadores débiles.



En este estudio el modelo AdaBoost se usa para aprender una serie de características simples o débiles de cada cara, todas estas juntas hacen un clasificador robusto. Los modelos se organizan en forma de cascada o hilera, aumentando su complejidad.

Los clasificadores más simples operan directamente en regiones de caras candidatas, actuando como un filtro grueso, mientras que los clasificadores complejos operan solo en aquellas regiones candidatas que muestran más probabilidad de ser caras.

El resultado es un algoritmo de reconocimiento facial rápido y efectivo que ha sido usado en productos como cámaras de fotos.

Una implementación más moderna del Clasificador en Cascada viene incluida en la biblioteca **OpenCV**. Esta es una biblioteca de visión por computadora en C ++ que proporciona una interfaz en Python. El beneficio de esta implementación es que **proporciona modelos de detección de rostros previamente entrenados y proporciona una interfaz para entrenar un modelo con tu propio conjunto de datos.**

Código

OpenCV proporciona la clase CascadeClassifier que se puede utilizar para crear un clasificador en cascada para la detección de rostros.

El constructor puede tomar un nombre de archivo como argumento que especifica el archivo XML para un modelo previamente entrenado.

OpenCV también proporciona varios modelos previamente entrenados como parte de la instalación. Descargaremos un modelo previamente entrenado para la detección de rostros frontales desde (aquí) y lo importaremos en nuestro directorio de trabajo actual con el nombre de archivo "haarcascade_frontalface_default.xml".

```
import cv2
from cv2 import CascadeClassifier
from cv2 import imread
from cv2 import rectangle
from cv2 import imshow
from cv2 import waitKey
from cv2 import destroyAllWindows
#print(cv2.__version__)

#Cargamos un modelo pre-entrenado para reconocimiento facial y creamos un modelo en cascada con el
classifier = CascadeClassifier('haarcascade_frontalface_default.xml')

#cargamos una foto de prueba
pixels = imread('image/foto grupal.jpg')

#realizamos el reconocimiento facial. 1.1 significa ampliar la foto un 10% y 3 indica el nivel de fiabilidad de la deteccion
bboxes = classifier.detectMultiScale(pixels, 1.1, 3)
#representamos los cuadros delimitadores de los rostros detectados
for box in bboxes:
    print(box)
    # para mostrar la imagen original con los cuadros delimitadores dibujados encima haremos lo siguiente:
    # extraemos cada propiedad del resultado (x e y de la esquina inferior izq del cuadro delimitador y su altura y anchura)
    x, y, width, height = box
    x2, y2 = x + width, y + height
    # dibujamos un rectangulo sobre los pixels
    rectangle(pixels, (x, y), (x2, y2), (0, 0, 255), 1)

#imprimimos la imagen con la deteccion y la mantenamos abierta hasta que el usuario pulse una tecla
#mostramos la imagen
imshow('face detection', pixels)
#mantenemos la ventana abierta hasta que se pulse una tecla
waitKey(0)
#cerramos la ventana
destroyAllWindows()
```

Ejecutar el ejemplo primero carga la fotografía, luego carga y configura el clasificador en cascada; se detectan caras y se imprime cada cuadro delimitador.

Cada cuadro enumera las coordenadas X e Y para la esquina inferior izquierda del cuadro delimitador, así como el ancho y el alto.

Soluciones con OpenCV

La solución nos indica la posición y dimensiones del cuadro o cuadros delimitadores alrededor de los rostros detectados. Podemos imprimir la imagen y los cuadros delimitadores para contemplar la solución de manera visual.

```
[229 152  52  52]
[592  28  43  43]
[470  32  44  44]
[508 155  54  54]
[623 159  54  54]
[381 161  52  52]
[173  95  48  48]
[706  43  46  46]
[260  46  45  45]
[747 190  52  52]
[353  61  46  46]
```

Nota: Los resultados pueden variar a causa de la naturaleza estocástica del algoritmo o del proceso de evaluación. Podría lanzarse el ejemplo numerosas veces y hacer una media de la salida para refinar el resultado.

Refinar resultado

La función detectMultiScale () proporciona algunos argumentos para ayudar a ajustar el uso del clasificador. Dos parámetros importantes son scaleFactor y minNeighbors.

La función *scaleFactor* controla cómo la imagen de entrada está escalada antes de la detección, puede escalarse más grande o más pequeña.

El valor predeterminado es 1,1 (aumento del 10%), aunque puede reducirse a valores como 1,05 (aumento del 5%) o elevarse a valores como 1,4 (aumento del 40%).

La función *minNeighbors* determina cómo de fiable tiene que ser la detección para que salga adelante.

El valor predeterminado es 3, pero se puede reducir a 1 para detectar muchas más caras y probablemente aumentará los falsos positivos (como puede observarse abajo en la imagen), o aumentará a 6 o más para requerir mucha más confianza antes de que se detecte una cara.

Una estrategia rápida puede ser reducir (o aumentar para fotos pequeñas) el scaleFactor hasta que se detecten todas las caras, luego aumentar el minNeighbors hasta que desaparezcan todos los falsos positivos o cerca de él.

Reconocimiento facial con Deep Learning

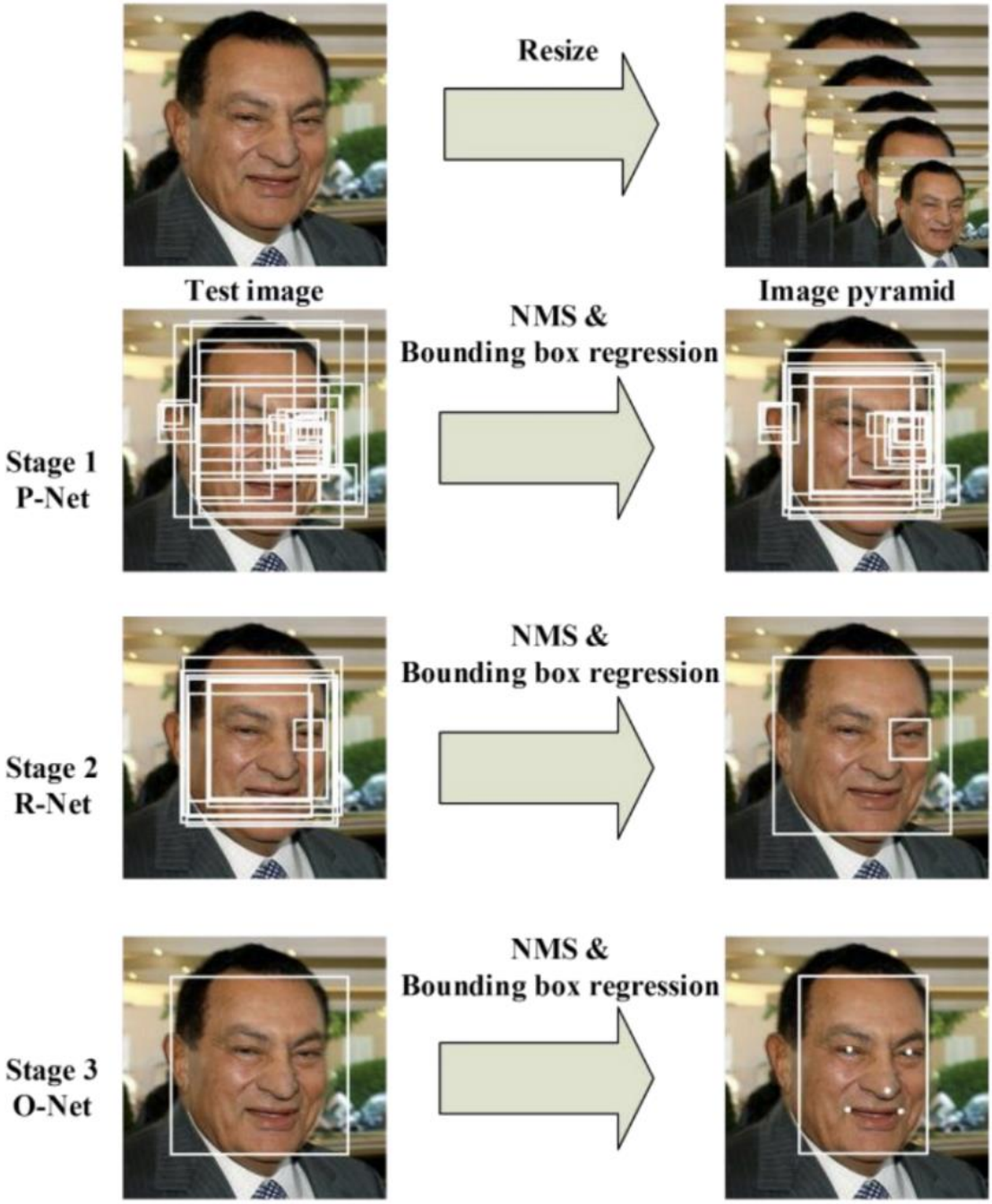
Se han desarrollado y demostrado varios métodos de Deep Learning o aprendizaje profundo para la detección de rostros en imágenes. Quizás uno de los enfoques más populares es el llamado “Red Neuronal Multitarea Convolucional en Cascada” o

MTCNN, descrita por Kaipeng Zhang en 2016 en el paper “Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks”.

MTCNN es popular por conseguir resultados de vanguarda en conjuntos de datos de referencia y por ser capaz de reconocer además características faciales como ojos y boca (denominado puntos de referencia).

La red utiliza una estructura en cascada con tres redes; Primero la imagen se reescala a diferentes tamaños (llamado pirámide de imágenes), después el primer modelo (Porposal Network o P-Net) propone regiones faciales candidatas, el segundo modelo (Refine Network o R-Net) filtra los cuadros delimitadores resultantes de estas propuestas y el tercer modelo (Output Network o O-Net) propone puntos de referencia faciales.

En la imagen tenemos una representación del proceso resultante de usar este modelo.



El modelo se llama red multitarea porque cada uno de los tres modelos por los que está compuesto (P-Net, R-Net y O-Net) están entrenados en tres tareas, p. Ej. hacer tres tipos de predicciones; son: clasificación de rostros, regresión de cuadro delimitador y localización de puntos de referencia faciales.

Los tres modelos no están conectados directamente; en cambio, las salidas de la etapa anterior se alimentan como entrada a la siguiente etapa. Esto permite realizar un procesamiento adicional entre etapas; por ejemplo, la supresión no máxima (NMS) se utiliza para filtrar los cuadros delimitadores candidatos propuestos por el P-Net de la primera etapa antes de proporcionarlos al modelo R-Net de la segunda etapa.

La arquitectura MTCNN es relativamente compleja de implementar. Afortunadamente, existen implementaciones de código abierto de la arquitectura que se pueden entrenar con nuevos conjuntos de datos, así como modelos previamente entrenados que se pueden usar directamente para la detección de rostros. Cabe destacar el lanzamiento oficial del algoritmo MTCNN que incluye el código y los modelos utilizados en el paper original de Kaipeng Zhang y que se nos proporciona aquí.

Aun así, el mejor proyecto MTCNN de terceros que podemos encontrar basado en Python sea el de Iván de Paz Centeno. Este proyecto es llamado MTCNN y está disponible bajo una licencia permisiva de código abierto del MIT.

Al ser un proyecto de código abierto, está sujeto a futuros cambios, por lo que nosotros vamos a trabajar con la versión del proyecto que se nos proporciona aquí.

El proyecto MTCNN, al que nos referiremos como ipazc / MTCNN para diferenciarlo del nombre de la red, proporciona una implementación de la arquitectura MTCNN utilizando TensorFlow y OpenCV. Hay dos beneficios principales de este proyecto; primero, proporciona un modelo pre-entrenado de alto rendimiento y el segundo es que se puede instalar como una biblioteca lista para usar en su propio código.

De forma predeterminada, la biblioteca utilizará el modelo previamente entrenado, aunque puede especificar su propio modelo a través del argumento "weights_file" y especificar una ruta o URL, por ejemplo:

```
filename = 'image/foto grupal.jpg'
#Usamos la funcion imread de pyplot en lugar de la de openv
pixels = pyplot.imread(filename)
#cargamos el modelo pre-entrenado que nos proporciona la libreria
detector = MTCNN()
#Detectamos los rostros en la imagen usando el modelo
faces = detector.detect_faces(pixels)
for face in faces:
    print(face)
```

El tamaño de cuadro mínimo para detectar una cara se puede especificar mediante el argumento "min_face_size", cuyo valor predeterminado es 20 píxeles. El constructor también proporciona un argumento "scale_factor" para especificar el factor de escala de la imagen de entrada, cuyo valor predeterminado es 0,709. Una vez que el modelo está configurado y cargado, se puede usar directamente para detectar caras en fotografías llamando a la función detect_faces (). Esto devuelve una lista de objetos en forma de diccionario, cada uno proporcionando una cantidad de claves para los detalles de cada rostro detectado, que incluyen:

- "Box": Proporciona la x, y de la parte inferior izquierda del cuadro delimitador, así como el ancho y alto del cuadro.
- "Confidence": la probabilidad de confianza de la predicción.

- “Keypoints”: Proporciona un diccionario con puntos para el "ojo_izquierdo", "ojo_derecho", "nariz", "boca_izquierda" y "boca_derecha".

Código

```
import mtcnn
from mtcnn.mtcnn import MTCNN
#print(mtcnn.__version__)
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from matplotlib.patches import Circle

filename = 'image/foto grupal.jpg'
#Usamos la funcion imread de pyplot en lugar de la de openv
pixels = pyplot.imread(filename)
#cargamos el modelo pre-entrenado que nos proporciona la libreria
detector = MTCNN()
#Detectamos los rostros en la imagen usando el modelo
faces = detector.detect_faces(pixels)
for face in faces:
    print(face)
#usamos una funci n para mostrar todos los datos por pantalla, tenemos que pasar la imagen original y las caras detectadas
#draw_image_with_boxes(filename, faces)
#tambien podemos crear una funcion para solo mostrar los rostros detectados
draw_result_faces(filename, faces)
```

Código principal

```
def draw_image_with_boxes(filename, result_list):
    #cargamos la imagen
    data = pyplot.imread(filename)
    #imprimimos la imagen
    pyplot.imshow(data)
    #obtenemos el contexto para dibujar los cuadros delimitadores
    ax = pyplot.gca()
    #imprimimos cada cuadro delimitador de la solucion
    for result in result_list:
        #obtener las coordenadas
        x, y, width, height = result['box']
        #crear los cuadros
        rect = Rectangle((x,y), width, height, fill=False, color='white')
        #dibujamos los cuadros
        ax.add_patch(rect)
        #a adimos los puntos de ojos, nariz y boca
        for key, value in result['keypoints'].items():
            #creamos y dibujamos los puntos
            dot = Circle(value, radius=2, color='white')
            ax.add_patch(dot)
    #mostramos por pantalla
    pyplot.show()
```

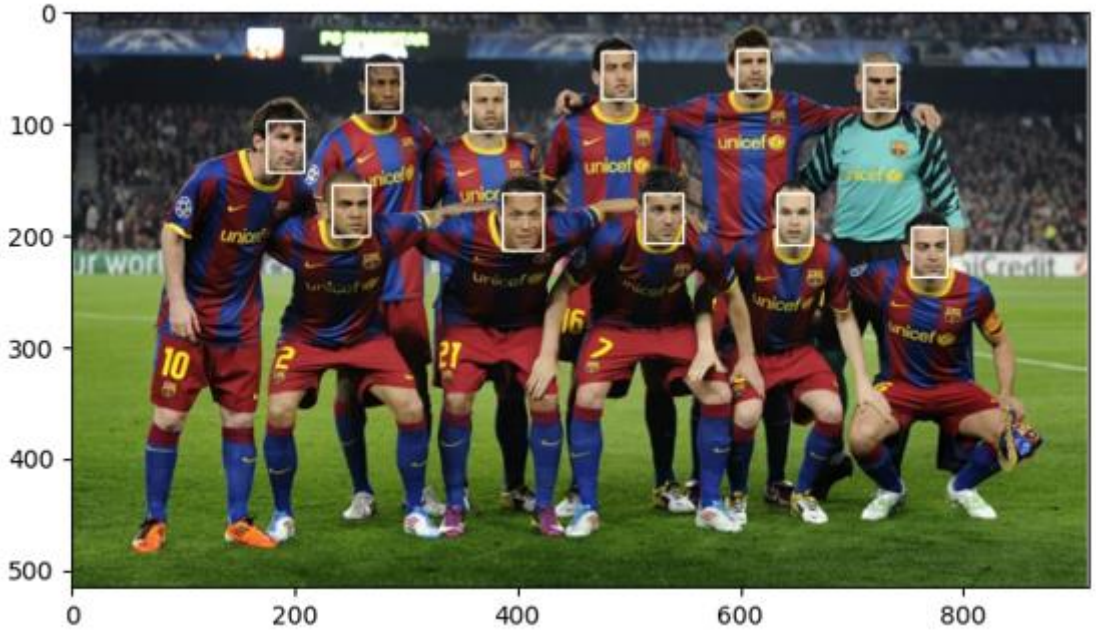
Código de la función draw_image_with_boxes

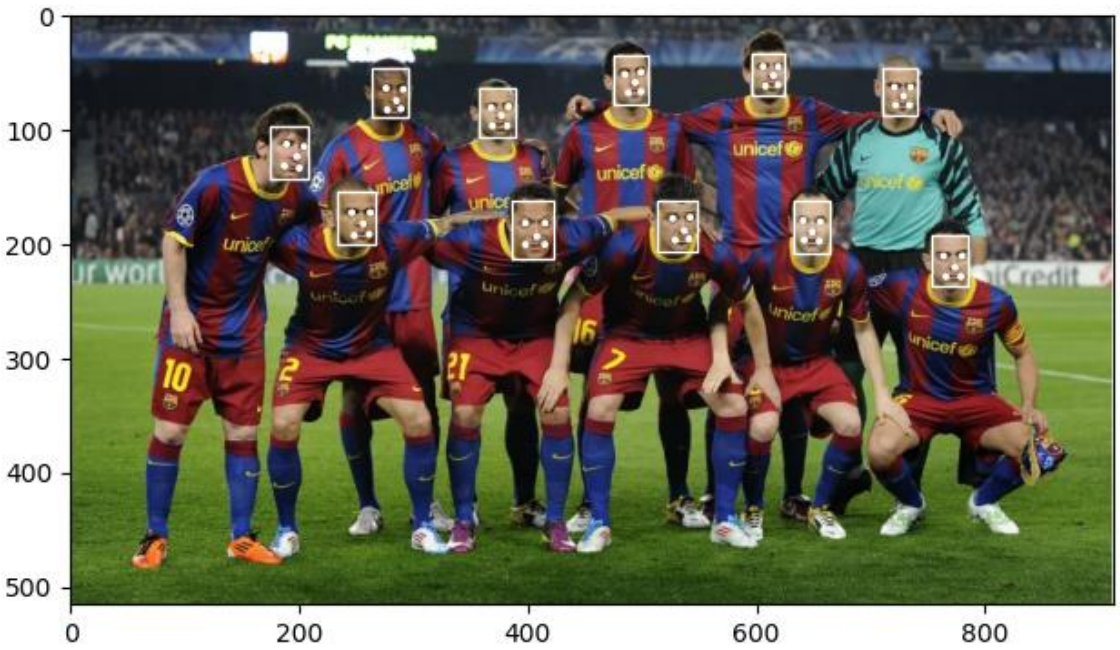
Es posible que queramos extraer las caras detectadas y pasarlas como entrada a otro sistema. Esto se puede lograr extrayendo los datos de píxeles directamente de la fotografía. Para ello crearemos la siguiente función.

```
def draw_result_faces(filename, result_list):
    #cargamos la imagen
    data = pyplot.imread(filename)
    for i in range(len(result_list)):
        #obtener las coordenadas
        x, y, width, height = result_list[i]['box']
        x2, y2 = x + width, y + height
        #los parametros de subplot son filas, columnas e indice
        pyplot.subplot(1, len(result_list), i+1)
        pyplot.axis('off')
        pyplot.imshow(data[y:y2, x:x2])
    #mostramos por pantalla
    pyplot.show()
```

Código de la función draw_result_faces

Soluciones con Deep Learning





En las imágenes de arriba puede verse la representación visual de las soluciones de detección de rostros para una sola cara, varias y varias con detección de ojos, nariz y boca. Abajo podemos ver una muestra de la solución.

{'box': [358, 63, 31, 43], 'confidence': 0.9999887943267822, 'keypoints': {'left_eye': (368, 80), 'right_eye': (383, 79), 'nose': (377, 87), 'mouth_left': (369, 96), 'mouth_right': (382, 96)}}

Por último, la solución de la extracción de los rostros de la imagen.

Cómo importar y ejecutar el proyecto.

Para poder correr el proyecto es necesario tener un entorno con Python, TensorFlow, Keras, OpenCV y MTCNN. En la sección *issues* del repositorio puede encontrarse las instrucciones para hacer esto mediante Conda.

Para importarlo simplemente hay que descargar o clonar el repositorio con *git clone* y abrirlo con algún IDE como Pycharm.

haarcascade_frontalface_default.xml es el modelo pre-entrenado que usamos con OpenCV. *faceDetectionOpenCV.py* es el código para reconocimiento facial haciendo uso de la librería OpenCV y *faceDetectionDeepLearning.py* es el código para reconocimiento facial haciendo uso de Deep Learning.