

Person recognition : Face detection

Ihor Volokhovych, 1st grade master student, MMAI-1

Abstract. In the paper we propose a generalized method for detecting face of a person on a picture using SVM from keras, Multi-task Cascaded Convolutional Networks (MTCNN) and PIL (Python Image Library)

1. Introduction

Problem settlement. Face recognition is the problem of identifying and verifying people in a photograph by their face.

It is a task that is trivially performed by humans, even under varying light and when faces are changed by age or obstructed with accessories and facial hair. Nevertheless, it remained a challenging computer vision problem for decades until recently.

Deep learning methods are able to leverage very large datasets of faces and learn rich and compact representations of faces, allowing modern models to first perform as-well and later to outperform the face recognition capabilities of humans.

Analysis or recent studies. Face detection is a computer technology that determines the location and size of a human face in the digital image. The facial features are detected and any other objects like trees, buildings and bodies are ignored from the digital image. It can be regarded as a specific case of object-class detection, where the task is finding the location and sizes of all objects in an image that belongs to a given class. Face detection can be seen as a more general case of face localization. In face localization, the task is to identify the locations and sizes of a known number of faces (usually one). Basically, there are two types of approaches to detect facial parts in the given digital image i.e. feature based and image based approach. Feature based approach tries to extract features of the image and match it against the knowledge of the facial features. While image based approach tries to get the best match between training and testing images. The following methods are commonly used to detect the faces from a still image or a video sequence.

Challenges. Challenges in face detection are the reasons which reduce the accuracy and detection rate of face detection. These challenges include a complex background, too many faces in images, odd expressions, illuminations, less resolution, face occlusion, skin color, distance and orientation etc. (Figure 1). • Odd expressions Human face in an image may have odd expressions unlike normal, which is a challenge for face detection. • Face occlusion Face occlusion is hiding face by any object. It may be glasses, scarf, hand, hairs, hats and any other object etc. It also reduces the face detection rate. • Illuminations Lighting effects may not be uniform in the image. Some parts of the image may have very high illumination and others may have very low

illumination. • **Complex background** Complex background means a lot of objects present in the image, which reduces the accuracy and rate of face detection. • **Too many faces in the image** means the image contains too many human faces, which is a challenge for face detection. • **Less resolution** Resolution of image may be very poor, which is also challenging for face detection. • **Skin color** Skin-color changes with geographical locations. Skin color of Chinese is different from African and skin-color of African is different from American and so on. Changing skin-color is also challenging for face detection.



Figure 1. Various categories of challenges for face detection

Applications.

- **Document control and access control.** Control can be imposed to document access with a face identification system.
- **Gender classification.** Gender information can be found from human images.
- **Human computer interaction system.** It is design and use of computer technology, focusing particularly on the interfaces between users and computers.
- **Biometric attendance.** It is system of taking attendance of people by their fingerprints or face etc.
- **Photography** Some recent digital cameras use face detection for autofocus. Face detection is also useful for selecting regions of interest in photo slideshows.
- **Facial feature extraction** Facial features like nose, eyes, mouth, skin-color etc. can be extracted from image.
- **Face recognition.** A facial recognition system is a process of identifying or verifying a person from a digital image or a video frame. One of the ways to do this is by comparing selected facial features from the image and a facial database. It is typically used in security systems.
- **Marketing.** Face detection is gaining the interest of marketers. A webcam can be integrated into a television and detect any face that walks by. The system then calculates the race,

gender, and age range of the face. Once the information is collected, a series of advertisements can be played that is specific towards the detected race/gender/age.

Purpose of an article. To develop and show capabilities of face recognition computer vision libraries by 2021 year.

2. Main part

Introduction. There is often a need to automatically recognize the people in a photograph. There are many reasons why we might want to automatically recognize a person in a photograph.

For example:

- We may want to restrict access to a resource to one person, called face authentication.
- We may want to confirm that the person matches their ID, called face verification.
- We may want to assign a name to a face, called face identification.

Generally, we refer to this as the problem of automatic “face recognition” and it may apply to both still photographs or faces in streams of video. Humans can perform this task very easily. We can find the faces in an image and comment as to who the people are, if they are known. We can do this very well, such as when the people have aged, are wearing sunglasses, have different colored hair, are looking in different directions, and so on. We can do this so well that we find faces where there aren’t any, such as in clouds.

Nevertheless, this remains a hard problem to perform automatically with software, even after 60 or more years of research. Until perhaps very recently.

Algorithms. Face recognition can be divided into multiple steps. The image below shows an example of a face recognition pipeline.



Figure 2. Face recognition pipeline.

There are various ways to implement each of the steps in a face recognition pipeline. In this post we’ll focus on popular deep learning approaches where we perform face detection using MTCNN, feature extraction using FaceNet and classification using Softmax.

MTCNN. **MTCNN** or Multi-Task Cascaded Convolutional Neural Networks is a neural network which detects faces and facial landmarks on images. It was published in 2016 by Zhang et al.

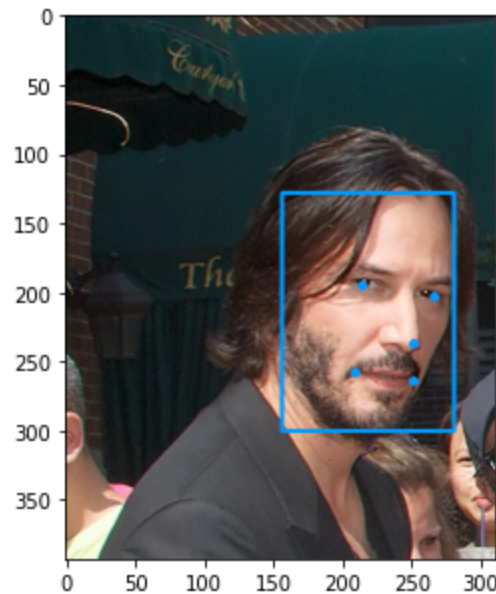


Figure 3. MTCNN Output example

MTCNN is one of the most popular and most accurate face detection tools today. It consists of 3 neural networks connected in a cascade.

Facenet. **FaceNet** is a deep neural network used for extracting features from an image of a person's face. It was published in 2015 by Google researchers Schroff et al.

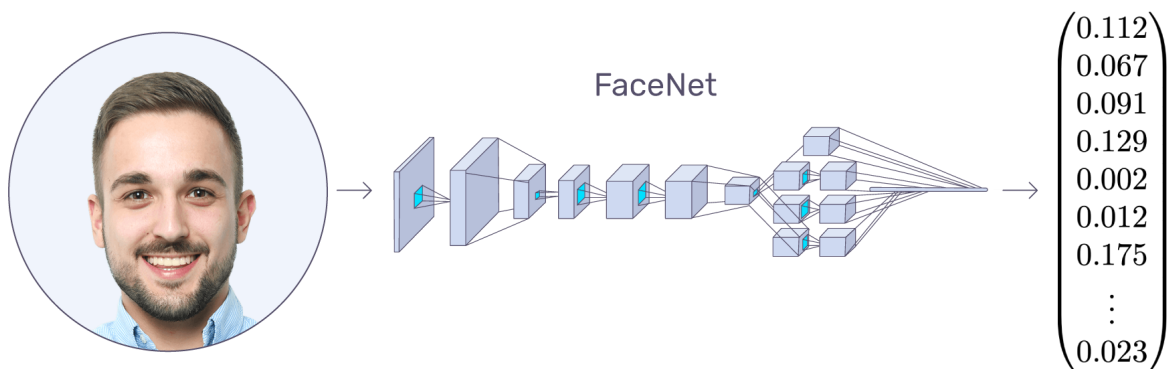


Figure 4. FaceNet algorithm

Implementation.

```
Підготовка

Імпорт потрібних ліб

In [ ]:
import os
import sys
import random
import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from pathlib import Path

# Піллов (Пітонівська ліба для картинок)
from PIL import Image
from keras.models import load_model

# Класифікатор SVM з СВЦ для класифікації на різних обличчях
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder, Normalizer
from sklearn.metrics import accuracy_score

from platform import python_version

Імпорт MTCNN для детекції обличчя та рис

In [ ]:
# Multi-task Cascaded Convolutional Networks (MTCNN)
!pip install MTCNN
from mtcnn.mtcnn import MTCNN
detector = MTCNN()
# Ваги - звичайні дефолтні

Requirement already satisfied: MTCNN in /usr/local/lib/python3.7/dist-packages (0.1.1)
Requirement already satisfied: keras>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from MTCNN) (2.7.0)
Requirement already satisfied: opencv-python>=4.1.0 in /usr/local/lib/python3.7/dist-packages (from MTCNN) (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python>=4.1.0->MTCNN) (1.19.5)
```

Figure 5. Imported libs

First of all, we import needed libraries to work on our project. In our case, we need numpy, matplotlib (for graphs), PIL (Pillow Image Library) to open, convert, resize and reshape images, Sklearn kit to process and build ML models, and MTCNN for training and detection.

Датасет

Для роботи було використано датасет з ресурсу kaggle

```
In [ ]:
# original dataset folder, you can see above. Костыль из прошлой работы
# We are using this dataset already splitted into train and valid databsets because google drive and colab are very bad with each other.
dataset_name = '5-celebrity-faces-dataset'
dirStr = '/content/drive/MyDrive/5-celebrity-faces-dataset'
base_dir = Path(dirStr)
print (base_dir)

sub_dir_name = [dirs for dirs in os.listdir(base_dir) if os.path.isdir(os.path.join(base_dir, dirs))]
print(f"sub_directories are {sub_dir_name}")

/content/drive/MyDrive/5-celebrity-faces-dataset
sub_directories are ['data', 'val', 'train', 'keras']
keras facenet models

In [ ]:
path_saved_model_string = dirStr + "/keras/model/facenet_keras.h5"
path_saved_model = Path(path_saved_model_string)
print (path_saved_model)

facenet_model = load_model(path_saved_model)

print(f"Input: {facenet_model.inputs}")
print(f"Output: {facenet_model.outputs}")

/content/drive/MyDrive/5-celebrity-faces-dataset/keras/model/facenet_keras.h5
WARNING:tensorflow:No training configuration found in the save file, so the model was "not" compiled. Compile it manually.
Input: [KerasTensor: shape=(None, 160, 160, 3) dtype=float32 (created by layer 'input_1')]
Output: [KerasTensor: shape=(None, 128) dtype=float32 (created by layer 'Bottleneck_BatchNorm')]
```

Figure 6. Importing dataset and keras FaceNet model

Next step is searching for an appropriate dataset, preparing data, distinguishing train and test data. This will ensure robust output.

Next step - processing data. In our case, we have database of celebrities photos in good quality so we need to assign paths to each folder in a variables

Процесинг даних:

```
In [ ]: base_dir_train = os.path.join(base_dir, 'train'+ '/')
base_dir_data_sting = dirStr + "/data/"
base_dir_data = os.path.join(base_dir_data_sting)
print (base_dir_data_sting)
base_dir_val = os.path.join(base_dir, 'val'+ '/')

print (base_dir_train)
sub_sub_dir = os.listdir(os.path.join(base_dir,base_dir_train))

base_dir_anne = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'anne' in fn])+ '/')
base_dir_arnold = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'arnold' in fn])+ '/')
base_dir_ben = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'ben' in fn])+ '/')
base_dir_dwayne = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'dwayne' in fn])+ '/')
base_dir_elton = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'elton' in fn])+ '/')
base_dir_jerry = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'jerry' in fn])+ '/')
base_dir_kate = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'kate' in fn])+ '/')
base_dir_keanu = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'keanu' in fn])+ '/')
base_dir_lauren = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'lauren' in fn])+ '/')
base_dir_madonna = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'madonna' in fn])+ '/')
base_dir_mindy = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'mindy' in fn])+ '/')
base_dir_simon = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'simon' in fn])+ '/')
base_dir_sofia = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'sofia' in fn])+ '/')
base_dir_will = os.path.join(base_dir_train, '.join([str(fn) for fn in sub_sub_dir if 'will' in fn])+ '/')
```

Figure 7. Data processing

Let's look at graph on what images we do have and frequencies

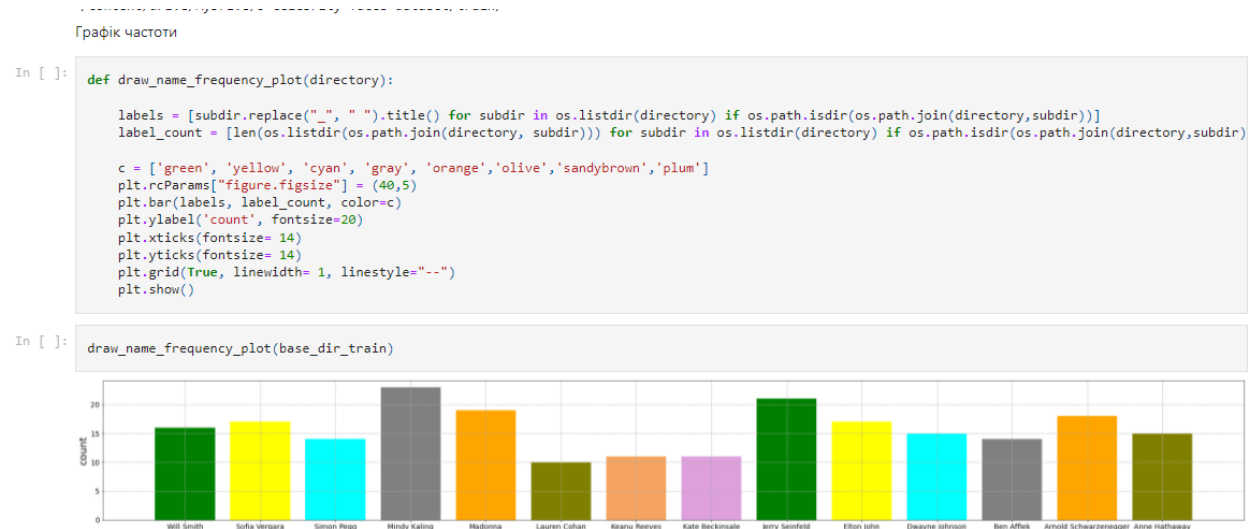


Figure 8. Photo frequencies graph

Next is face features extraction. The detector returns a list of JSON objects. Each JSON object contains three main keys: 'box', 'confidence', and 'keypoints'

- The bounding box is formatted as [x, y, width, height] in the "box" variable.
- Confidence is the probability that the restrictive box will match the face
- Key points that are formatted into a JSON object with the keys "left_eye", "right_eye", "nose", "mouth_left", "mouth_right". Each key point is determined by the position of the Pixel (x, y).

```
In [ ]: def draw_face_boundaries(image, feature_coordinate):
    filename = cv2.cvtColor(cv2.imread(image), cv2.COLOR_BGR2RGB)
    bounding_box = feature_coordinate[0]['box']
    keypoints = feature_coordinate[0]['keypoints']
    cv2.rectangle(filename,
                  (bounding_box[0], bounding_box[1]),
                  (bounding_box[0]+bounding_box[2], bounding_box[1] + bounding_box[3]),
                  (0,155,255),
                  2)
    cv2.circle(filename, (keypoints['left_eye']), 2, (0,155,255), 2)
    cv2.circle(filename, (keypoints['right_eye']), 2, (0,155,255), 2)
    cv2.circle(filename, (keypoints['nose']), 2, (0,155,255), 2)
    cv2.circle(filename, (keypoints['mouth_left']), 2, (0,155,255), 2)
    cv2.circle(filename, (keypoints['mouth_right']), 2, (0,155,255), 2)
    cv2.imwrite(image, cv2.cvtColor(filename, cv2.COLOR_RGB2BGR))
    plt.imshow(filename)

In [ ]: def extract_face(filename, required_size=(160, 160), print_features=False, show_features=False):
    # pre-processing on file image
    image = Image.open(filename)
    image = image.convert("RGB")
    pixels = np.asarray(image)

    face_coordinates = detector.detect_faces(pixels)

    if show_features:
        draw_face_boundaries(filename, face_coordinates)
    if print_features:
        print(f"> Extracted features: \n {face_coordinates}")

    x1, y1, width, height = face_coordinates[0]['box']
    x1, y1 = abs(x1), abs(y1)
    x2, y2 = x1 + width, y1 + height

    # extract the face
    face = pixels[y1:y2, x1:x2]

    # resize pixels to the model size
    image = Image.fromarray(face)
    image = image.resize(required_size)
    face_array = np.asarray(image)

    return face_array
```

Figure 9. Face features extraction

After this step, we are able to extract face features and output results.

```
In [ ]: def extract_face(filename,required_size=(160, 160), print_features=False, show_features=False):

    # pre-processing on file image
    image = Image.open(filename)
    image = image.convert('RGB')
    pixels = np.asarray(image)

    face_coordinates = detector.detect_faces(pixels)

    if show_features:
        draw_face_boundaries(filename, face_coordinates)
    if print_features:
        print(f"> Extracted features: \n {face_coordinates}")

    x1, y1, width, height = face_coordinates[0]['box']
    x1, y1 = abs(x1), abs(y1)
    x2, y2 = x1 + width, y1 + height

    # extract the face
    face = pixels[y1:y2, x1:x2]

    # resize pixels to the model size
    image = Image.fromarray(face)
    image = image.resize(required_size)
    face_array = np.asarray(image)

    return face_array
```

```
In [ ]: # Завантаження фото та екстракція обличчя
pixels = extract_face(base_dir_keanu + os.listdir(base_dir_keanu)[0], show_features=True)
```

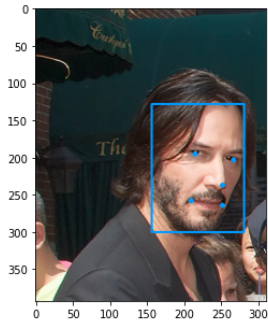


Figure 9. Example of extracted features

As we see, the algorithm detected face features (eyes, nose, mouth) and selected the viewbox of (in this case) Keanu Reeves's face.

Modelling. For modelling FaceNet was used. For this first pixel scaling was done. That they were normalized across all channels (RGB). Then we transformed normalized pixels into one sample and started to predict and return embedding dataset

Facenet

```
In [ ]: def get_128vectorEmbedding_singleImage(model, face_pixels):  
        # скейлинг пікселів  
        face_pixels = face_pixels.astype('float32')  
  
        # нормалізація пікселів по всіх каналах  
        face_pixels_normalized = (face_pixels - face_pixels.mean()) / face_pixels.std()  
  
        # трансформування обличчя в один приклад  
        samples = np.expand_dims(face_pixels_normalized, axis=0)  
  
        # предикш для ембеддингу  
        yhat = model.predict(samples)  
  
        return yhat[0]  
  
In [ ]: # вертає фейс ембединг для всього трейн датасету  
def get_embedding_dataset(model, train_X, test_X):  
  
    embTrain_X = [get_128vectorEmbedding_singleImage(model, face_pixels) for face_pixels in train_X]  
    embTest_X = [get_128vectorEmbedding_singleImage(model, face_pixels) for face_pixels in test_X]  
  
    print(f'embTrain_X shape: {np.asarray(embTrain_X).shape}, embTest_X shape: {np.asarray(embTest_X).shape}')  
  
    return np.asarray(embTrain_X), np.asarray(embTest_X)  
  
In [ ]: embTrain_X, embTest_X = get_embedding_dataset(model=facenet_model, train_X=trainX, test_X=testX)  
  
embTrain_X shape: (221, 128), embTest_X shape: (70, 128)
```

Figure 9. FaceNet

SVM. Then for similar processes SVM was used. We normalized input vectors, provided one hot encoding of incoming labels and created an svm classifier. After this showed the accuracy of the model.

```
def normalize_vectors(vectors):
    """ normalize input vectors """
    normalizer = Normalizer(norm='l2')
    vectors = normalizer.transform(vectors)

    return vectors
```

```
def labels_encoder(labels):
    """provide one hot encoding of incoming labels"""
    out_encoder = LabelEncoder()
    out_encoder.fit(labels)
    labels = out_encoder.transform(labels)
    return out_encoder, labels
```

```
def svm_classifier(train_x_embedding, train_y, test_x_embedding, test_y):

    # normalize input vectors
    train_x_norm_embedding = normalize_vectors(train_x_embedding)
    test_x_norm_embedding = normalize_vectors(test_x_embedding)

    # label encode targets
    out_encoder, train_y_class = labels_encoder(train_y)
    out_encoder, test_y_class = labels_encoder(test_y)

    # fit model
    model = SVC(kernel='linear', probability=True)
    model.fit(train_x_norm_embedding, train_y_class)

    yhat_train = model.predict(train_x_norm_embedding)
    yhat_test = model.predict(test_x_norm_embedding)

    score_train = accuracy_score(train_y_class, yhat_train)
    score_test = accuracy_score(test_y_class, yhat_test)
    print(f'Accuracy: train={score_train*100}, test={score_test*100}')

    return model
```

```
svm_model= svm_classifier(embTrain_X, trainy, embTest_X, testy)
```

Accuracy: train=100.0, test=98.57142857142858

Figure 9. Result of SVM with accuracy.

After this we saved SVM and started testing.

```
def random_testing_test_data(test_data, test_label, embedding_model, classification_model) :
    # test model on a random example from the test dataset
    selection = random.choice([i for i in range(test_data.shape[0])])

    random_face_pixels = get_128vectorEmbedding_singleImage(embedding_model, test_data[selection])
    random_face_pixels = np.expand_dims(random_face_pixels, axis=0)
    random_face_emb = normalize_vectors(random_face_pixels)

    label_encoder, test_y_class = labels_encoder(test_label)
    random_face_class = test_y_class[selection]
    random_face_name = label_encoder.inverse_transform([random_face_class])

    # prediction for the face
    yhat_class = classification_model.predict(random_face_emb)
    yhat_prob = classification_model.predict_proba(random_face_emb)

    # get name
    class_index = yhat_class[0]
    class_probability = yhat_prob[0,class_index] * 100
    predict_names = label_encoder.inverse_transform(yhat_class)
    print(f'Predicted: {predict_names[0].replace("_", " ").title()} confidence:({round(class_probability,2)})')
    print(f'Expected: {random_face_name[0].replace("_", " ").title()}')

    # plotting
    plt.imshow(test_data[selection])
    title = f'{predict_names[0].replace("_", " ").title()} ({round(class_probability,2)})'
    plt.title(title)
    plt.show()
```

```
random_testing_test_data(testX, testy, facenet_model, svm_model)
```

Predicted: Anne Hathaway confidence:(67.87)

Expected: Anne Hathaway

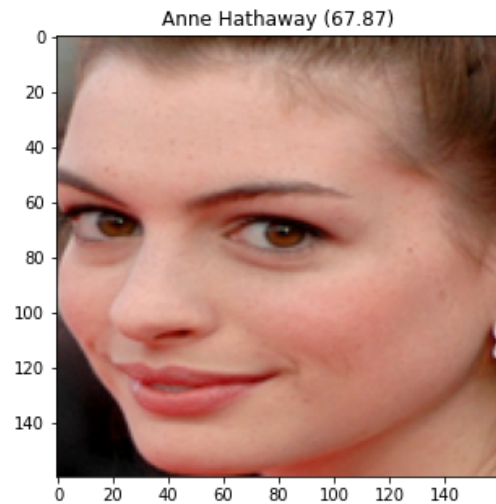



Figure 10. Testing True

```

imageFolder = os.listdir(base_dir_data)[0]
print(imageFolder)
image = os.listdir(base_dir_data_sting + imageFolder)[1]
print(image)
image_path = base_dir_data_sting + imageFolder + '/' + image
print(image_path)
externally_uploaded_face(image_path, labels, facenet_model, svm_model, threshold=80)

ben_afflek
6171279833f4b300189ad407.jpg
/content/drive/MyDrive/5-celebrity-faces-dataset/data/ben_afflek/6171279833f4b300189ad407.jpg
[ -0.57661045  0.984959   0.9533627 -1.3408836  0.49824405  1.4863181
 -0.12790625  0.5118755  1.0946969 -0.807067   0.9247993  -0.40200526
 -1.5290177   0.32466838 -0.18045413  0.1440936  -2.6962852  -1.6525437
 -0.3396457   0.15732956  1.7583699 -0.08185119  0.67029715 -0.5785131
  0.84059083  -1.127218   -0.27447107 -1.8759671  -0.78052396 -1.189114
 -0.30976108  -2.0517113  0.48046318  0.3577767  -0.35455173  0.3195674
 -0.25214708  -0.596499   1.6482406  -0.26628786 -0.2759446  -0.7887566
 -1.462276    -0.88654065 -1.6394535  0.05955991 -1.5491467   0.9561626
 -0.34041992  -0.46903732 -1.1940792  2.3642046  1.1458207  -0.8263962
  0.20701224  -0.3612144  1.8492688  -0.6505212  0.27692357  0.8578901
  0.98277706  -0.17959812 -0.83618027  0.1101831  1.032986   -0.6819553
  0.44555643  0.08470138 -2.1952958  0.87539583 -0.61047935 -0.4207352
 -1.595079    -1.6185563  0.80977035  0.3478353  0.3186685  1.7766808
 -1.3017918   0.73409486 -0.55692303 -0.30370668  0.7589213  -0.06660283
  1.1310468   0.19368717 -1.3448237  0.43624508 -0.7022747  -1.2481657
  1.1589547   -0.8792931  1.0797482  -0.3627025  0.4695284  0.31006858
 -1.5037436   0.06053814 -0.27362922 -0.27695745  0.03071409 -2.0812871
 -0.76885307  1.4365932  0.12786463 -0.3328045  0.2225872  0.4689239
 -0.12747768  0.9566745  -0.02091792 -1.2982825  1.0033681  -0.3645784
  0.12591122  -0.21677817  0.12019357  1.1501852  0.71412754  1.0380036
  0.9085091   -0.8979267  -0.6461048  0.0468697  1.33009   -1.9664285
 -0.3776332   2.116708   ]
Predicted: Unknown confidence:(52.08)
Unknown (52.08)


```

Figure 11. Testing False