机器学习系列（8）

人脸识别--原理及Python实现 ¶

人脸验证&识别：

- 验证和识别
- oneshot学习
- siamese网络
- Triplet损失
- 面部验证与二分类
- 总结

Python实现：

- 见文章内容

申明

本文由LSayhi完成，供学习参考，可转载；代码实现部分的框架由Coursera提供，由LSayhi完成，详细数据及代码可在github查阅。

github: https://github.com/LSayhi/DeepLearning (https://github.com/LSayhi/DeepLearning)

CSDN博客：https://blog.csdn.net/LSayhi (https://blog.csdn.net/LSayhi)

微信公众号：AI有点可ai

感谢您的关注

一、 人脸验证和识别原理

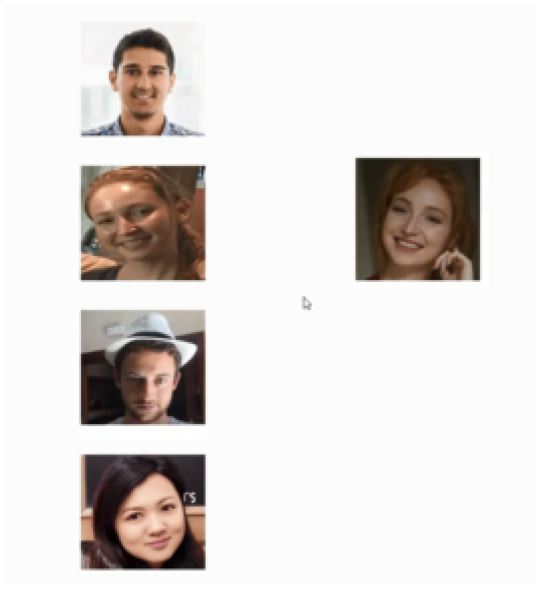
1.人脸验证与人脸识别：

- 人脸验证与人脸识别。顾名思义，人脸验证举例，假设输入一张某人图片和其ID，人脸验证只需要对该ID进行验证，判断是否是该ID对应的人脸。而人脸识别则复杂得多，它只允许输入一张图片，然后系统要根据这张输入判断此人是否是公司数据库中的员工以及其ID。

身份检测闸机

1.one-shot学习：

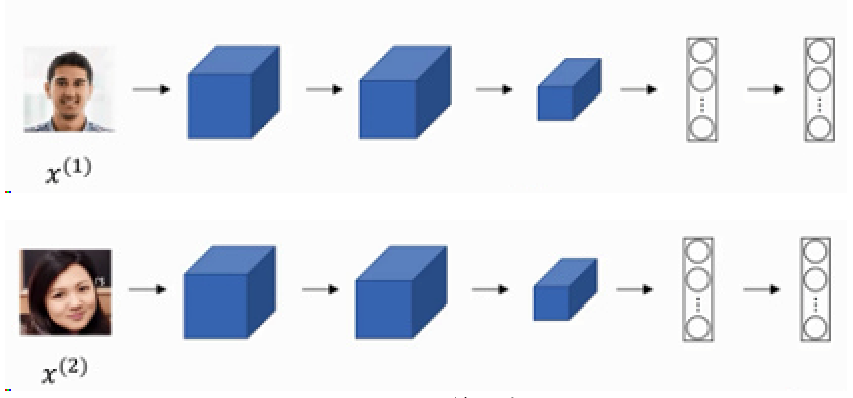
- 假设要构造一个公司人员验证闸机，你会怎么设计这个系统呢？前面我们CNN，也许我们可以利用员工的照片训练一个卷积神经网络，然后摄像头采集员工进入闸机前的图像，再由softmax层判断这个人是否是公司员工以及是哪位员工，听起来像是一个可行的办法。仔细想想，那可能需要每个人很多张照片，而且如何公司新加入了员工，还得重新训练一个网络，那么这样也许不是一个好的做法。
- one-shot学习就是要解决这一问题，每张员工的只需一张提供照片，然后将摄像头采集到的另一张照片与其进行比对。进而判断是否是公司员工以及是哪一位员工。具体来说，one-shot learning 是学习衡量图像之间的相似度（差异），两张图片之间的差异大小可以用distance(img1,img2)来表示，如何训练好一个学习器可以判断两张照片的差异，那我们就不需要每次都训练新的网络了。


one-shot learning

- 现在假设公司数据库中有四张员工照片ABCD，摄像头采集到一张来访者的照片E，one-shot学习先将E与A代入distance(E,A)，当差异distance的值大于某个阈值时，认为E不是A，反之则认为是A，同理，对BCD同样操作，最终获得E是否是公司的员工，以及是哪位员工。所以，问题的重心在于如何设计和优化出合适的distance(img1,img2)函数，此部分内容下节描述。

2.siamese网络：

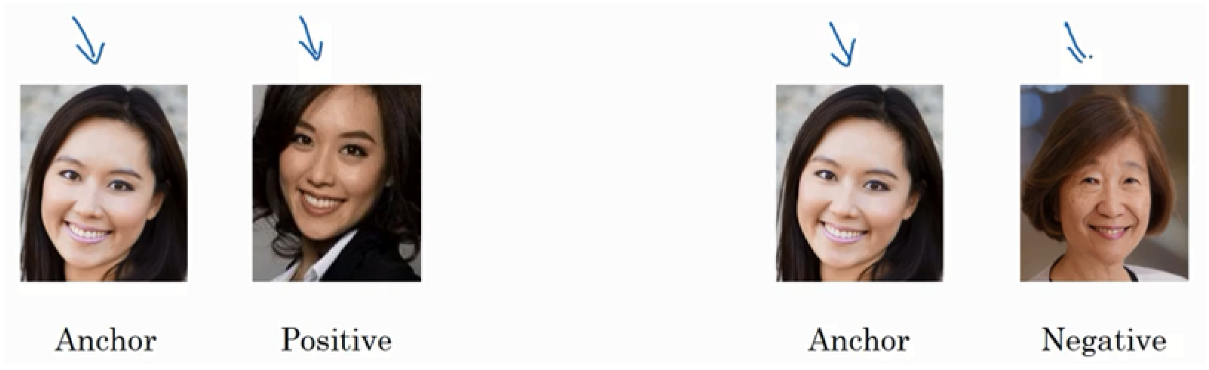
- siamese网络。我们经常将一张图片输入到CNN中，然后经过卷积层池化层全连接层等得到一个输出向量，如果我们将这个输出向量保留成为特征向量，它表示了图像的某些特征。siamese网络就是将两张图片分别输入同一个网络（参数也相同），然后比较特征向量间的差异，我们用特征向量的范数来表示两张图片之间的“距离”，训练这个网络，使得其对不同的人脸拥有较大的范数，对同一个人脸特征向量拥有较小的范数。



siamese网络思想

3.Triplet损失：

- 要想通过神经网络学习人脸编码的优质方法，使用Triplet损失是一种方式。Triplet损失，可以看出肯定是同时对三张图片操作。我们在siamese网络中讲到，对于相同的人脸，其范数“距离”要小，对于不同的人脸，其范数“距离”应该要大，Triplet损失很好地综合了这两点。如下图所示，Triplet损失是同时对三张图片的损失函数，三张图片分别是图片Anchor,图片positive（和Anchor属于同一个人的脸）,图片negative（和anchor属于不同人的脸）。



Anchor,positive,negative

- 我们要达到的效果是，anchor与positive之间的范数平方小(distance（A,P）小)，anchor与negative之间的范数平方大(distance（A,N）小)，也可表达为，distance(A,P)<distance(A,N)。当然为了增加这两者之间的差值，提高鲁棒性，我们应该增加一个margin（间隔α），将目标改为：

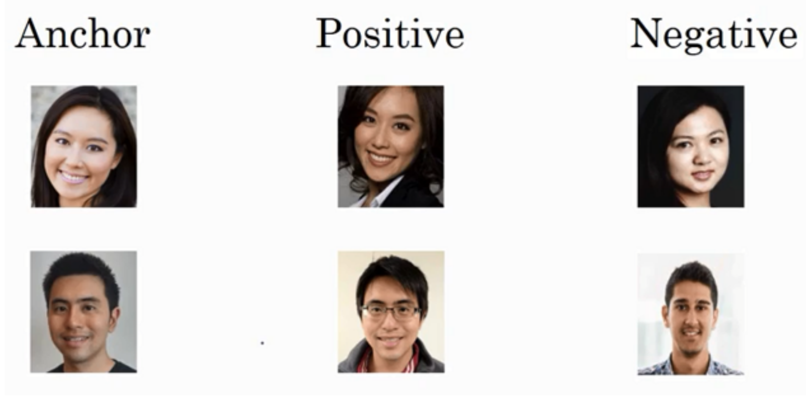
$$distance(A,P)+\alpha<distance(A,N)$$

即

$$||f(A)-f(P)||^2-||f(A)-f(N)||^2+\alpha<0$$

其中f()为神经网络对人脸的编码（可理解为对图片提取特征向量的方式）。

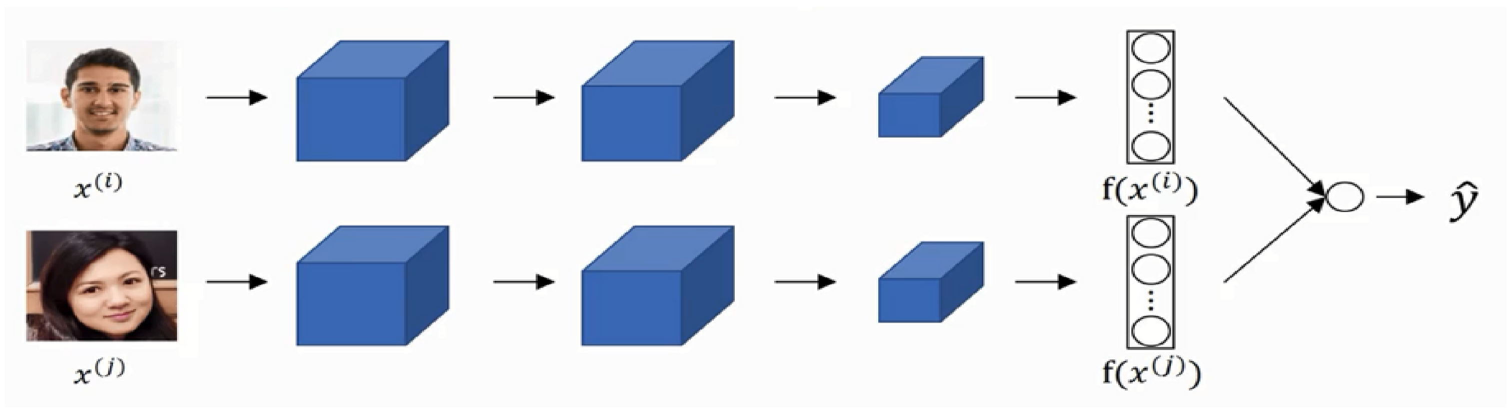
- 根据以上分析，我们可以定义损失函数Tripletloss了,其表达式如下：
$$TripletLoss=max\ (||f(A)-f(P)||^2-||f(A)-f(N)||^2+\alpha,0)$$
- 当满足anchor与positive距离（包括margin）小于等于anchor与negative时，TriletLoss函数第一个参数小于等于0，那么TripletLoss=0，不满足时，TripletLoss大于0。当我们对TripletLoss函数运用梯度下降等方式时，经过反向传播过程获得CNN的合适参数，从而降低loss。在实际中，为了获得更优的参数，我们对positive和negative的选择不完全随机，而是让positive不那么“positive”,negative不那么“negative”，即||f(A)-f(P)||^2 接近||f(A)-f(N)||^2，这样能加大训练的难度，从而让训练完成后的网络参数具更好的效果，下图为一示例：



Anchor,positive,negative选择

4.面部验证与二分类:

- 面部识别也可当作二分类问题来解决。比如，通过siamese网络对两张图片进行特征提取，再经过一个sigmoid单元预测两张图片是否是同一个人，如果两张人脸相似，输出为1，否则输出为0，如下图:



$$\hat{y} = \sigma(\sum_{k=1}^{128} w_i |f(x^{(i)})_k - f(x^{(j)})_k| + b)$$

二分类求解方式

- 在一些变体中，还可以将上式中的范数部分改为其它的表达式，如卡方公式等。
- 有个小技巧可以运用在人脸验证或者识别上，那就是我们可以预先存储员工照片的特征向量，而不是仅仅存照片，避免每次的前向传播过程的重复，节省大量的运算时间以及空间，这样可以提高部署终端的效率。

python实现 Face Recognition for the Happy House

Welcome to the first assignment of week 4! Here you will build a face recognition system. Many of the ideas presented here are from FaceNet (https://arxiv.org/pdf/1503.03832.pdf). In lecture, we also talked about DeepFace (https://research.fb.com/wp-content/uploads/2016/11/deepface-closing-the-gap-to-human-level-performance-in-face-verification.pdf).

Face recognition problems commonly fall into two categories:

- **Face Verification** - "is this the claimed person?". For example, at some airports, you can pass through customs by letting a system scan your passport and then verifying that you (the person carrying the passport) are the correct person. A mobile phone that unlocks using your face is also using face verification. This is a 1:1 matching problem.
- **Face Recognition** - "who is this person?". For example, the video lecture showed a face recognition video (https://www.youtube.com/watch?v=wr4rx0Spihs (https://www.youtube.com/watch?v=wr4rx0Spihs)) of Baidu employees entering the office without needing to otherwise identify themselves. This is a 1:K matching problem.

FaceNet learns a neural network that encodes a face image into a vector of 128 numbers. By comparing two such vectors, you can then determine if two pictures are of the same person.

In this assignment, you will:

- Implement the triplet loss function 完成triplet loss 函数
- Use a pretrained model to map face images into 128-dimensional encodings 使用训练好的模型转化图像到128维向量
- Use these encodings to perform face verification and face recognition 利用此编码方案进行人脸验证和人脸识别

In this exercise, we will be using a pre-trained model which represents ConvNet activations using a "channels first" convention, as opposed to the "channels last" convention used in lecture and previous programming assignments. In other words, a batch of images will be of shape (m, n_C, n_H, n_W) instead of (m, n_H, n_W, n_C). Both of these conventions have a reasonable amount of traction among open-source implementations; there isn't a uniform standard yet within the deep learning community.

Let's load the required packages.

In [3]:
```
from keras.models import Sequential
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input, concatenate
from keras.models import Model
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D, AveragePooling2D
from keras.layers.merge import Concatenate
from keras.layers.core import Lambda, Flatten, Dense
from keras.initializers import glorot_uniform
from keras.engine.topology import Layer
from keras import backend as K
K.set_image_data_format('channels_first')
import cv2
import os
import numpy as np
from numpy import genfromtxt
import pandas as pd
import tensorflow as tf
from fr_utils import *
from inception_blocks import *

%matplotlib inline
%load_ext autoreload
%autoreload 2

np.set_printoptions(threshold=np.nan)
```

0 - Naive Face Verification

In Face Verification, you're given two images and you have to tell if they are of the same person. The simplest way to do this is to compare the two images pixel-by-pixel. If the distance between the raw images are less than a chosen threshold, it may be the same person!

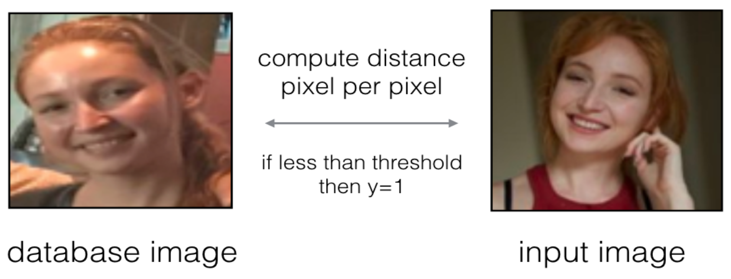


Figure 1

Of course, this algorithm performs really poorly, since the pixel values change dramatically due to variations in lighting, orientation of the person's face, even minor changes in head position, and so on.

You'll see that rather than using the raw image, you can learn an encoding $f(img)$ so that element-wise comparisons of this encoding gives more accurate judgements as to whether two pictures are of the same person.

1 - Encoding face images into a 128-dimensional vector

1.1 - Using an ConvNet to compute encodings

The FaceNet model takes a lot of data and a long time to train. So following common practice in applied deep learning settings, let's just load weights that someone else has already trained. The network architecture follows the Inception model from Szegedy *et al.* (https://arxiv.org/abs/1409.4842). We have provided an inception network implementation. You can look in the file `inception_blocks.py` to see how it is implemented (do so by going to "File->Open..." at the top of the Jupyter notebook).

The key things you need to know are:

- This network uses 96x96 dimensional RGB images as its input. Specifically, inputs a face image (or batch of m face images) as a tensor of shape $(m, n_C, n_H, n_W) = (m, 3, 96, 96)$
- It outputs a matrix of shape $(m, 128)$ that encodes each input face image into a 128-dimensional vector

Run the cell below to create the model for face images.

In [4]:
```
FRmodel = faceRecoModel(input_shape=(3, 96, 96))
```

In [5]:
```
print("Total Params:", FRmodel.count_params())
```

```
Total Params: 3743280
```

** Expected Output **

Total Params: 3743280

By using a 128-neuron fully connected layer as its last layer, the model ensures that the output is an encoding vector of size 128. You then use the encodings the compare two face images as follows:

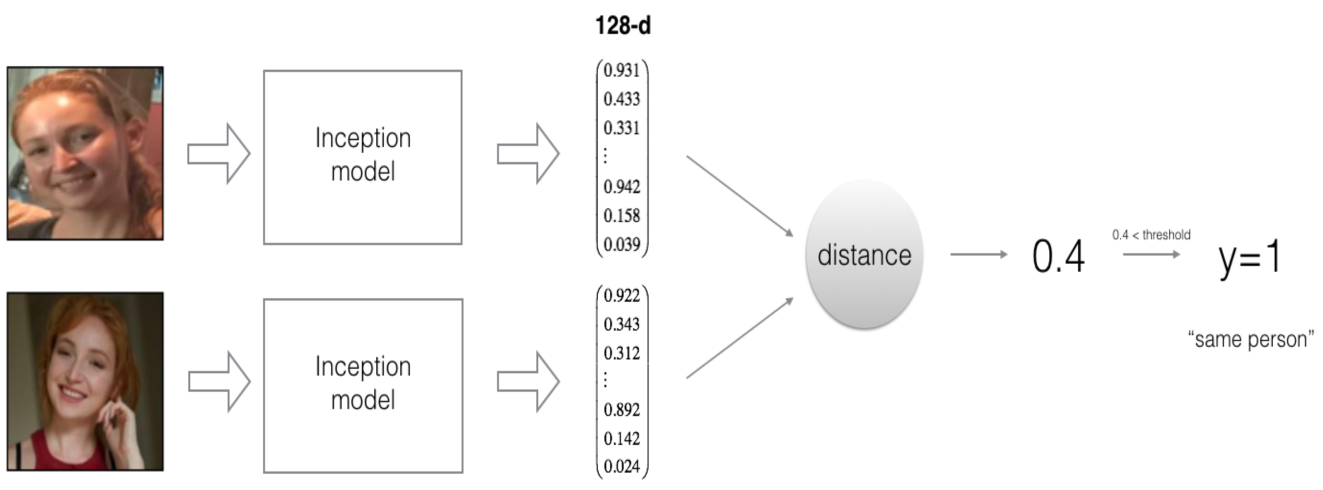


Figure 2:
By computing a distance between two encodings and thresholding, you can determine if the two pictures represent the same person

So, an encoding is a good one if:

- The encodings of two images of the same person are quite similar to each other
- The encodings of two images of different persons are very different

The triplet loss function formalizes this, and tries to "push" the encodings of two images of the same person (Anchor and Positive) closer together, while "pulling" the encodings of two images of different persons (Anchor, Negative) further apart.

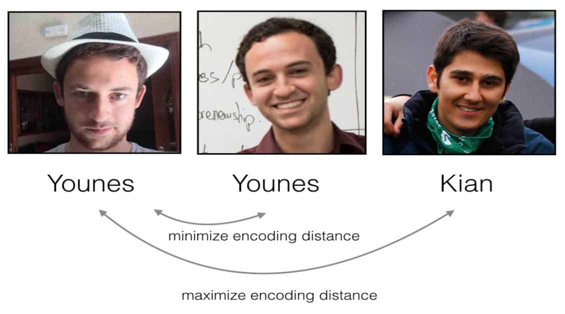
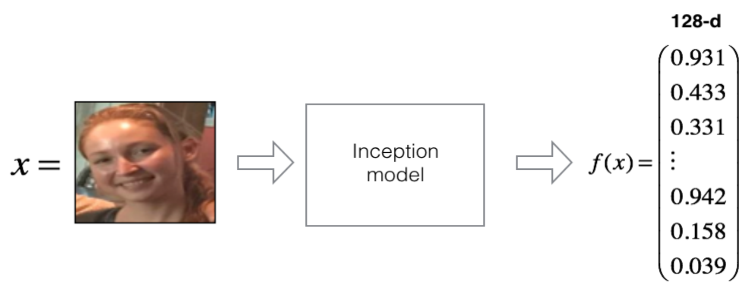


Figure 3:
In the next part, we will call the pictures from left to right: Anchor (A), Positive (P), Negative (N)

1.2 - The Triplet Loss

For an image x, we denote its encoding $f(x)$, where f is the function computed by the neural network.



Training will use triplets of images (A, P, N):

- A is an "Anchor" image--a picture of a person.
- P is a "Positive" image--a picture of the same person as the Anchor image.
- N is a "Negative" image--a picture of a different person than the Anchor image.

These triplets are picked from our training dataset. We will write $(A^{(i)}, P^{(i)}, N^{(i)})$ to denote the i-th training example.

You'd like to make sure that an image $A^{(i)}$ of an individual is closer to the Positive $P^{(i)}$ than to the Negative image $N^{(i)}$) by at least a margin α:

$$\mid\mid f(A^{(i)}) - f(P^{(i)}) \mid\mid_2^2 + \alpha < \mid\mid f(A^{(i)}) - f(N^{(i)}) \mid\mid_2^2$$

You would thus like to minimize the following "triplet cost":

$$\mathcal{J} = \sum^{N}_{i=1} \large[\small \underbrace{\mid \mid f(A^{(i)}) - f(P^{(i)}) \mid \mid_2^2}_\text{(1)} - \underbrace{\mid \mid f(A^{(i)}) - f(N^{(i)}) \mid \mid_2^2}_\text{(2)} + \alpha \large] \small_+ \tag{3}$$

Here, we are using the notation "$[z]_+$" to denote $max(z,0)$.

Notes:

- The term (1) is the squared distance between the anchor "A" and the positive "P" for a given triplet; you want this to be small.
- The term (2) is the squared distance between the anchor "A" and the negative "N" for a given triplet, you want this to be relatively large, so it thus makes sense to have a minus sign preceding it.
- α is called the margin. It is a hyperparameter that you should pick manually. We will use $\alpha = 0.2$.

Most implementations also normalize the encoding vectors to have norm equal one (i.e., $\mid \mid f(img)\mid \mid_2$=1); you won't have to worry about that here.

Exercise: Implement the triplet loss as defined by formula (3). Here are the 4 steps:

1. Compute the distance between the encodings of "anchor" and "positive": $\mid \mid f(A^{(i)}) - f(P^{(i)}) \mid \mid_2^2$

2. Compute the distance between the encodings of "anchor" and "negative": $\mid\mid f(A^{(i)}) - f(N^{(i)}) \mid\mid_2^2$
3. Compute the formula per training example: $\mid\mid f(A^{(i)}) - f(P^{(i)}) \mid - \mid\mid f(A^{(i)}) - f(N^{(i)}) \mid\mid_2^2 + \alpha$
4. Compute the full formula by taking the max with zero and summing over the training examples:

$$\mathcal{J} = \sum_{i=1}^{N} \left[\mid\mid f(A^{(i)}) - f(P^{(i)}) \mid\mid_2^2 - \mid\mid f(A^{(i)}) - f(N^{(i)}) \mid\mid_2^2 + \alpha \right]_+ \tag{3}$$

Useful functions: `tf.reduce_sum()` , `tf.square()` , `tf.subtract()` , `tf.add()` , `tf.reduce_mean` , `tf.maximum()` .

In [8]:
```
# GRADED FUNCTION: triplet_loss

def triplet_loss(y_true, y_pred, alpha = 0.2):
    """
    Implementation of the triplet loss as defined by formula (3)

    Arguments:
    y_true -- true labels, required when you define a loss in Keras, you don't need it in this function.
    y_pred -- python list containing three objects:
            anchor -- the encodings for the anchor images, of shape (None, 128)
            positive -- the encodings for the positive images, of shape (None, 128)
            negative -- the encodings for the negative images, of shape (None, 128)

    Returns:
    loss -- real number, value of the loss
    """

    anchor, positive, negative = y_pred[0], y_pred[1], y_pred[2]

    ### START CODE HERE ### (≈ 4 lines)
    # Step 1: Compute the (encoding) distance between the anchor and the positive
    pos_dist = tf.reduce_sum(tf.square(tf.subtract(y_pred[0],y_pred[1])))
    # Step 2: Compute the (encoding) distance between the anchor and the negative
    neg_dist = tf.reduce_sum(tf.square(tf.subtract(y_pred[0],y_pred[2])))
    # Step 3: subtract the two previous distances and add alpha.
    basic_loss = tf.add(tf.subtract(pos_dist,neg_dist),alpha)
    # Step 4: Take the maximum of basic_loss and 0.0. Sum over the training examples.
    loss = tf.reduce_sum(tf.maximum(basic_loss,0.0))
    ### END CODE HERE ###

    return loss
```

In [9]:
```
with tf.Session() as test:
    tf.set_random_seed(1)
    y_true = (None, None, None)
    y_pred = (tf.random_normal([3, 128], mean=6, stddev=0.1, seed = 1),
              tf.random_normal([3, 128], mean=1, stddev=1, seed = 1),
              tf.random_normal([3, 128], mean=3, stddev=4, seed = 1))
    loss = triplet_loss(y_true, y_pred)

    print("loss = " + str(loss.eval()))
```

```
loss = 350.027
```

Expected Output:

loss 350.026

2 - Loading the trained model

FaceNet is trained by minimizing the triplet loss. But since training requires a lot of data and a lot of computation, we won't train it from scratch here. Instead, we load a previously trained model. Load a model using the following cell; this might take a couple of minutes to run.

In [10]:
```
FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics = ['accuracy'])
load_weights_from_FaceNet(FRmodel)
```

Here're some examples of distances between the encodings between three individuals:

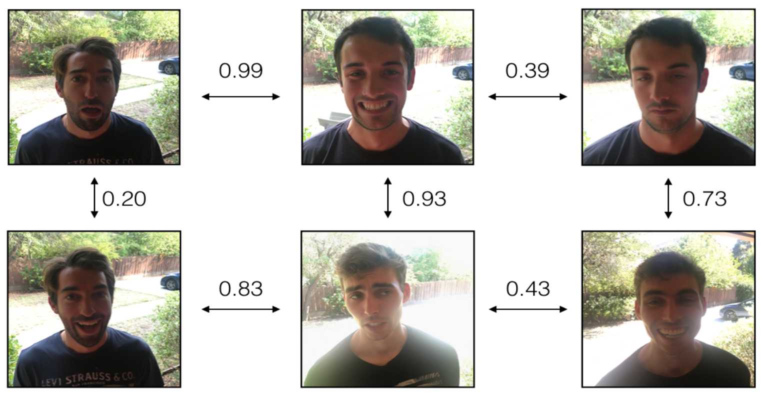


Figure 4:
Example of distance outputs between three individuals' encodings

Let's now use this model to perform face verification and face recognition!

3 - Applying the model

Back to the Happy House! Residents are living blissfully since you implemented happiness recognition for the house in an earlier assignment.

However, several issues keep coming up: The Happy House became so happy that every happy person in the neighborhood is coming to hang out in your living room. It is getting really crowded, which is having a negative impact on the residents of the house. All these random happy people are also eating all your food.

So, you decide to change the door entry policy, and not just let random happy people enter anymore, even if they are happy! Instead, you'd like to build a **Face verification** system so as to only let people from a specified list come in. To get admitted, each person has to swipe an ID card (identification card) to identify themselves at the door. The face recognition system then checks that they are who they claim to be.

3.1 - Face Verification

Let's build a database containing one encoding vector for each person allowed to enter the happy house. To generate the encoding we use `img_to_encoding(image_path, model)` which basically runs the forward propagation of the model on the specified image.

Run the following code to build the database (represented as a python dictionary). This database maps each person's name to a 128-dimensional encoding of their face.

In [11]:
```
database = {}
database["danielle"] = img_to_encoding("images/danielle.png", FRmodel)
database["younes"] = img_to_encoding("images/younes.jpg", FRmodel)
database["tian"] = img_to_encoding("images/tian.jpg", FRmodel)
database["andrew"] = img_to_encoding("images/andrew.jpg", FRmodel)
database["kian"] = img_to_encoding("images/kian.jpg", FRmodel)
database["dan"] = img_to_encoding("images/dan.jpg", FRmodel)
database["sebastiano"] = img_to_encoding("images/sebastiano.jpg", FRmodel)
database["bertrand"] = img_to_encoding("images/bertrand.jpg", FRmodel)
database["kevin"] = img_to_encoding("images/kevin.jpg", FRmodel)
database["felix"] = img_to_encoding("images/felix.jpg", FRmodel)
database["benoit"] = img_to_encoding("images/benoit.jpg", FRmodel)
database["arnaud"] = img_to_encoding("images/arnaud.jpg", FRmodel)
```

Now, when someone shows up at your front door and swipes their ID card (thus giving you their name), you can look up their encoding in the database, and use it to check if the person standing at the front door matches the name on the ID.

Exercise: Implement the verify() function which checks if the front-door camera picture (`image_path`) is actually the person called "identity". You will have to go through the following steps:

1. Compute the encoding of the image from image_path
2. Compute the distance about this encoding and the encoding of the identity image stored in the database
3. Open the door if the distance is less than 0.7, else do not open.

As presented above, you should use the L2 distance (np.linalg.norm). (Note: In this implementation, compare the L2 distance, not the square of the L2 distance, to the threshold 0.7.)

In [12]:
```python
# GRADED FUNCTION: verify

def verify(image_path, identity, database, model):
    """
    Function that verifies if the person on the "image_path" image is "identity".

    Arguments:
    image_path -- path to an image
    identity -- string, name of the person you'd like to verify the identity. Has to be a resident of the Happy house.
    database -- python dictionary mapping names of allowed people's names (strings) to their encodings (vectors).
    model -- your Inception model instance in Keras

    Returns:
    dist -- distance between the image_path and the image of "identity" in the database.
    door_open -- True, if the door should open. False otherwise.
    """

    ### START CODE HERE ###

    # Step 1: Compute the encoding for the image. Use img_to_encoding() see example above. (≈ 1 line)
    encoding = img_to_encoding(image_path, model)

    # Step 2: Compute distance with identity's image (≈ 1 line)
    dist = np.linalg.norm(encoding-database[identity],ord=2)

    # Step 3: Open the door if dist < 0.7, else don't open (≈ 3 lines)
    if dist < 0.7:
        print("It's " + str(identity) + ", welcome home!")
        door_open =True
    else:
        print("It's not " + str(identity) + ", please go away")
        door_open = False

    ### END CODE HERE ###

    return dist, door_open
```

Younes is trying to enter the Happy House and the camera takes a picture of him ("images/camera_0.jpg"). Let's run your verification algorithm on this picture:

In [13]:
```python
verify("images/camera_0.jpg", "younes", database, FRmodel)
```

```
It's younes, welcome home!
```

Out[13]:
```
(0.65938449, True)
```

Expected Output:

It's younes, welcome home! (0.65939283, True)

Benoit, who broke the aquarium last weekend, has been banned from the house and removed from the database. He stole Kian's ID card and came back to the house to try to present himself as Kian. The front-door camera took a picture of Benoit ("images/camera_2.jpg). Let's run the verification algorithm to check if benoit can enter.

In [ ]:
```python
verify("images/camera_2.jpg", "kian", database, FRmodel)
```

Expected Output:

It's not kian, please go away (0.86224014, False)

3.2 - Face Recognition

Your face verification system is mostly working well. But since Kian got his ID card stolen, when he came back to the house that evening he couldn't get in!

To reduce such shenanigans, you'd like to change your face verification system to a face recognition system. This way, no one has to carry an ID card anymore. An authorized person can just walk up to the house, and the front door will unlock for them!

You'll implement a face recognition system that takes as input an image, and figures out if it is one of the authorized persons (and if so, who). Unlike the previous face verification system, we will no longer get a person's name as another input.

Exercise: Implement `who_is_it()` . You will have to go through the following steps:

1. Compute the target encoding of the image from image_path 计算摄像头看到的人脸的编码
2. Find the encoding from the database that has smallest distance with the target encoding.找出数据库中与此编码距离最近的人脸
    - Initialize the `min_dist` variable to a large enough number (100). It will help you keep track of what is the closest encoding to the input's encoding.
    - Loop over the database dictionary's names and encodings. To loop use `for (name, db_enc) in database.items()` .
        - Compute L2 distance between the target "encoding" and the current "encoding" from the database.
        - If this distance is less than the min_dist, then set min_dist to dist, and identity to name.

In [16]:
```python
# GRADED FUNCTION: who_is_it

def who_is_it(image_path, database, model):
    """
    Implements face recognition for the happy house by finding who is the person on the image_path image.

    Arguments:
    image_path -- path to an image
    database -- database containing image encodings along with the name of the person on the image
    model -- your Inception model instance in Keras

    Returns:
    min_dist -- the minimum distance between image_path encoding and the encodings from the database
    identity -- string, the name prediction for the person on image_path
    """

    ### START CODE HERE ###

    ## Step 1: Compute the target "encoding" for the image. Use img_to_encoding() see example above. ## (≈ 1 line)
    encoding = img_to_encoding(image_path, model)

    ## Step 2: Find the closest encoding ##

    # Initialize "min_dist" to a large value, say 100 (≈1 line)
    min_dist = 100

    # Loop over the database dictionary's names and encodings.
    for (name, db_enc) in database.items():

        # Compute L2 distance between the target "encoding" and the current "emb" from the database. (≈ 1 line)
        dist =np.linalg.norm(encoding-db_enc,ord=2)

        # If this distance is less than the min_dist, then set min_dist to dist, and identity to name. (≈ 3 lines)
        if dist < min_dist:
            min_dist = dist
            identity = name

    ### END CODE HERE ###

    if min_dist > 0.7:
        print("Not in the database.")
    else:
        print ("it's " + str(identity) + ", the distance is " + str(min_dist))

    return min_dist, identity
```

Younes is at the front-door and the camera takes a picture of him ("images/camera_0.jpg"). Let's see if your who_it_is() algorithm identifies Younes.

In [17]:
```python
who_is_it("images/camera_0.jpg", database, FRmodel)
```

```
it's younes, the distance is 0.659384
```

Out[17]:
```
(0.65938449, 'younes')
```

Expected Output:

it's younes, the distance is 0.659393 (0.65939283, 'younes')

You can change " `camera_0.jpg` " (picture of younes) to " `camera_1.jpg` " (picture of bertrand) and see the result.

Your Happy House is running well. It only lets in authorized persons, and people don't need to carry an ID card around anymore!

You've now seen how a state-of-the-art face recognition system works.

Although we won't implement it here, here're some ways to further improve the algorithm:

- Put more images of each person (under different lighting conditions, taken on different days, etc.) into the database. Then given a new image, compare the new face to multiple pictures of the person. This would increae accuracy.

- Crop the images to just contain the face, and less of the "border" region around the face. This preprocessing removes some of the irrelevant pixels around the face, and also makes the algorithm more robust.

What you should remember: - Face verification solves an easier 1:1 matching problem; face recognition addresses a harder 1:K matching problem. - The triplet loss is an effective loss function for training a neural network to learn an encoding of a face image. - The same encoding can be used for verification and recognition. Measuring distances between two images' encodings allows you to determine whether they are pictures of the same person.

Congrats on finishing this assignment!

References:

- Florian Schroff, Dmitry Kalenichenko, James Philbin (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering (https://arxiv.org/pdf/1503.03832.pdf)
- Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, Lior Wolf (2014). DeepFace: Closing the gap to human-level performance in face verification (https://research.fb.com/wp-content/uploads/2016/11/deepface-closing-the-gap-to-human-level-performance-in-face-verification.pdf)
- The pretrained model we use is inspired by Victor Sy Wang's implementation and was loaded using his code: https://github.com/iwantooxxoox/Keras-OpenFace (https://github.com/iwantooxxoox/Keras-OpenFace).
- Our implementation also took a lot of inspiration from the official FaceNet github repository: https://github.com/davidsandberg/facenet (https://github.com/davidsandberg/facenet)