Face Recognition using FaceNet

In [1]:
```
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_v2 import *

%matplotlib inline
%load_ext autoreload
%autoreload 2

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

```
Using TensorFlow backend.
```

1 - Encoding face images into a 128-dimensional vector

1.1 - Using a ConvNet to compute encodings

The FaceNet model takes a lot of data and a long time to train. So we 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).

- 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

In [2]:
```
FRmodel = faceRecoModel(input_shape=(3, 96, 96))
```

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. These encodings can then be used to compare two face images as:

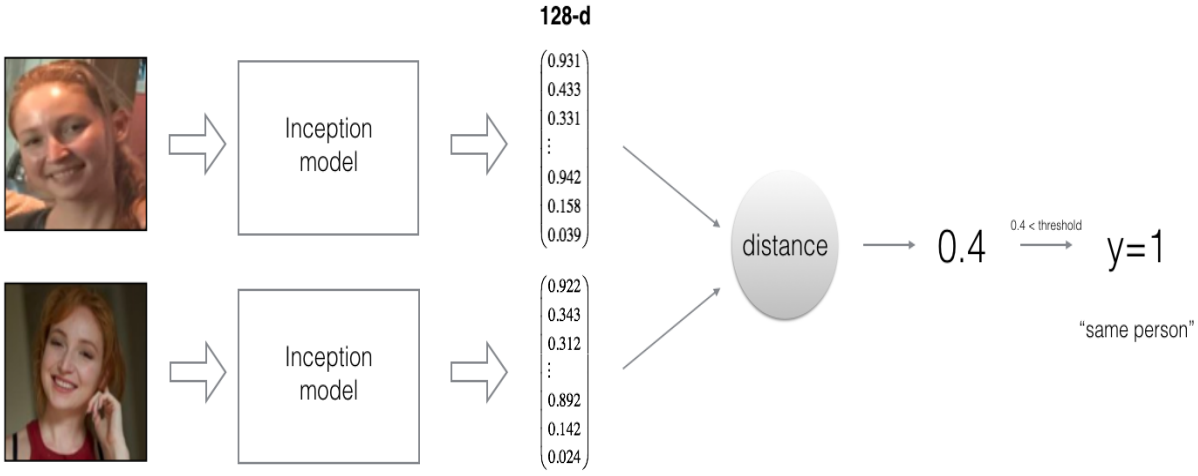


Figure 2:

By computing the 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.

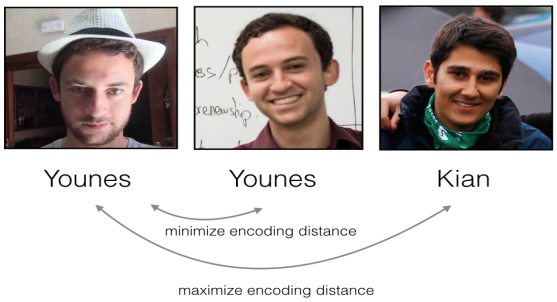
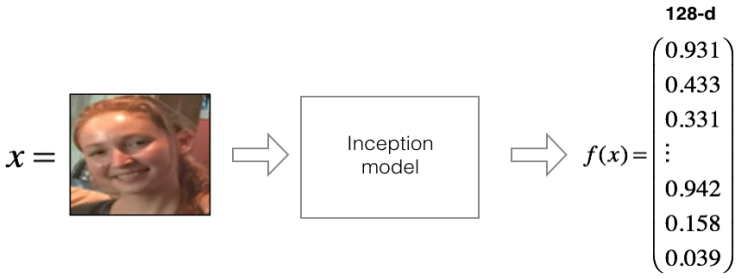


Figure 3:

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.

We have 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:

$$|| f(A^{(i)}) - f(P^{(i)}) ||_2^2 + \alpha < || f(A^{(i)}) - f(N^{(i)}) ||_2^2$$

We thus minimize the following "triplet cost":

$$\mathcal{J} = \sum_{i=1}^{m} \Big[\underbrace{|| f(A^{(i)}) - f(P^{(i)}) ||_2^2}_{(1)} - \underbrace{|| f(A^{(i)}) - f(N^{(i)}) ||_2^2}_{(2)} + \alpha \Big]_+ \tag{3}$$

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. It has a minus sign preceding it because minimizing the negative of the term is the same as maximizing that term.
- α is called the margin. It is a hyperparameter that you pick manually. We will use $\alpha = 0.2$.

In [3]:
```
def triplet_loss(y_true, y_pred, alpha = 0.2):  #Computation of triplet loss

    anchor, positive, negative = y_pred[0], y_pred[1], y_pred[2]
    pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,positive)),-1)  # dista
    neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,negative)),-1)  # dista

    basic_loss = pos_dist - neg_dist + alpha
    loss = tf.reduce_sum(tf.maximum(0.,basic_loss))

    return loss
```

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.

In [4]:
```
FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics = ['accuracy'])
load_weights_from_FaceNet(FRmodel)
```

Here are some examples of distances between the encodings between three individuals:

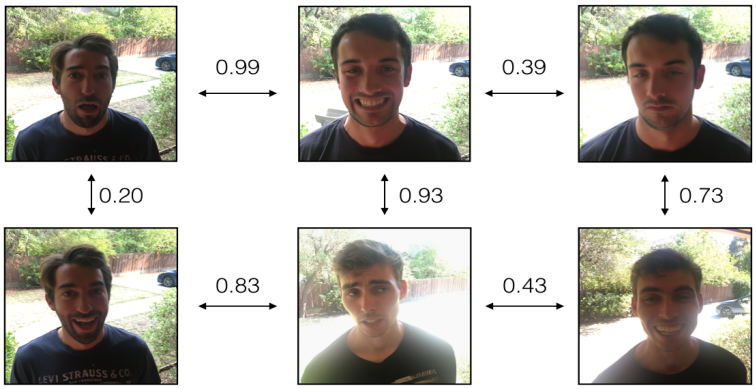


Figure 4:
Example of distance outputs between three individuals' encodings

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

2 - Applying the model

2.1 - Face Verification

Let's build a database first, containing one encoding vector for each person. To generate the encoding we use `img_to_encoding(image_path, model)`, which runs the forward propagation of the model on the specified image. This database maps each person's name to a 128-dimensional encoding of their face.

In [5]:
```
database = {}
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)
database["shivi"] = img_to_encoding("images/shivi.jpg", FRmodel)
database["maa"] = img_to_encoding("images/maa.jpg", FRmodel)
database["bhaiya"] = img_to_encoding("images/bhaiya.jpg", FRmodel)
database["papa"] = img_to_encoding("images/papa.jpg", FRmodel)
database["mama"] = img_to_encoding("images/mama.jpg", FRmodel)
database["nisha"] = img_to_encoding("images/nisha.jpg", FRmodel)
```

In [6]:
```
def verify(image_path, identity, database, model):  # face verification of input

    encoding = img_to_encoding(image_path, model)
    dist = np.linalg.norm(database[identity]-encoding)

    if dist<0.6:
        print("Matched")
        res = True
    else:
        print("Not Matched")
        res = False

    return dist, res
```

Now let's check if the images of the following two same persons is correctly verified or not

In [7]:
```
verify("images/shivi_test.jpg", "shivi", database, FRmodel)
```

```
Matched
```

Out[7]:
```
(0.53321266, True)
```

Now let's check for the following two images of different persons, if they are correctly verified.

In [8]:
```
verify("images/camera_0.jpg", "kian", database, FRmodel)
```

```
Not Matched
```

Out[8]:
```
(0.85352063, False)
```

2.2 - Face Recognition

The face verification system is mostly working well.

Now, we'd like to change the face verification system to 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).

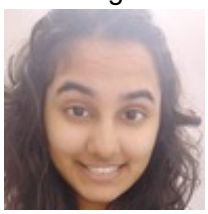
In [9]:
```
def recognize(image_path, database, model):  # recognizes the person in the given

    encoding = img_to_encoding(image_path, model)

    min_dist = 100
    for (name, db_enc) in database.items():
        dist = np.linalg.norm(encoding-db_enc)
        if dist<min_dist:
            min_dist = dist
            identity = name
    if min_dist > 0.6:
        print("Not in the database.")
    else:
        print ("it's " + str(identity))

    return min_dist
```

Let's check if the function correctly recognizes image of shivi.



In [10]:
```
recognize("images/shivi_test.jpg", database, FRmodel)
```

```
it's shivi
```

Out[10]:
```
0.53321266
```

Let's check if the function correctly recognizes image of danielle which is not in our database.

In [11]:
```
recognize("images/danielle.png", database, FRmodel)
```

```
Not in the database.
```

Out[11]:
```
0.60710424
```