Face Recognition

In this assignment, you will build a face recognition system. Many of the ideas presented here are from [FaceNet (https://arxiv.org/pdf/1503.03832.pdf)](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)](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
- Use a pretrained model to map face images into 128-dimensional encodings
- Use these encodings to perform face verification and face recognition

Channels-first notation

- 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.

Updates

If you were working on the notebook before this update...

- The current notebook is version "3a".
- You can find your original work saved in the notebook with the previous version name ("v3")
- To view the file directory, go to the menu "File->Open", and this will open a new tab that shows the file directory.

List of updates

- `triplet_loss` : Additional Hints added.
- `verify` : Hints added.
- `who_is_it` : corrected hints given in the comments.
- Spelling and formatting updates for easier reading.

Load packages

Let's load the required packages.

```python
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.

0 - Naive Face Verification

In Face Verification, you're given two images and you have to determine 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, $f(img)$.
- By using an encoding for each image, an element-wise comparison produces a more accurate judgement as to whether two pictures are of the same person.

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 following common practice in applied deep learning, let's 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_v2.py` to see how it is implemented (do so by going to "File->Open..." at the top of the Jupyter notebook. This opens the file directory that contains the '.py' file).

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

```
In [3]:  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 to compare two face images as follows:

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.
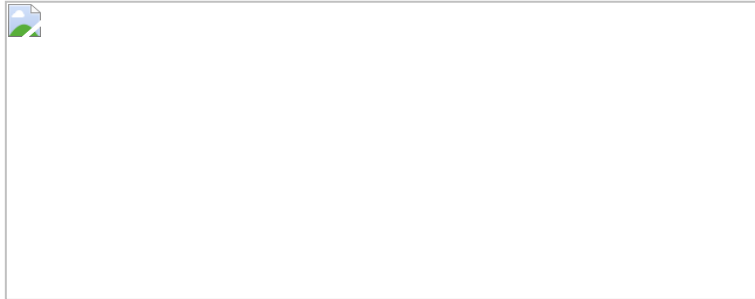

Figure 3:
In the next part, we will call the pictures from left to right: Anchor (A), Positive (P), Negative (N)

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.

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:

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

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

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

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 $\alpha = 0.2$.

Most implementations also rescale the encoding vectors to haven L2 norm equal to one (i.e., $\| f(img) \|_2 = 1$); you won't have to worry about that in this assignment.

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": $\| f(A^{(i)}) - f(P^{(i)}) \|_2^2$
2. Compute the distance between the encodings of "anchor" and "negative": $\| f(A^{(i)}) - f(N^{(i)}) \|_2^2$
3. Compute the formula per training example: $\| f(A^{(i)}) - f(P^{(i)}) \|_2^2 - \| f(A^{(i)}) - f(N^{(i)}) \|_2^2 + \alpha$
4. Compute the full formula by taking the max with zero and summing over the training examples:

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

Hints

- Useful functions: `tf.reduce_sum()`, `tf.square()`, `tf.subtract()`, `tf.add()`, `tf.maximum()`.
- For steps 1 and 2, you will sum over the entries of $\| f(A^{(i)}) - f(P^{(i)}) \|_2^2$ and $\| f(A^{(i)}) - f(N^{(i)}) \|_2^2$.
- For step 4 you will sum over the training examples.

Additional Hints

- Recall that the square of the L2 norm is the sum of the squared differences: $||x - y||_2^2 = \sum_{i=1}^{N} (x_i - y_i)^2$
- Note that the `anchor`, `positive` and `negative` encodings are of shape `(m,128)`, where m is the number of training examples and 128 is the number of elements used to encode a single example.
- For steps 1 and 2, you will maintain the number of `m` training examples and sum along the 128 values of each encoding. [tf.reduce_sum (https://www.tensorflow.org/api_docs/python/tf/math/reduce_sum)](https://www.tensorflow.org/api_docs/python/tf/math/reduce_sum) has an `axis` parameter. This chooses along which axis the sums are applied.
- Note that one way to choose the last axis in a tensor is to use negative indexing (`axis=-1`).
- In step 4, when summing over training examples, the result will be a single scalar value.
- For `tf.reduce_sum` to sum across all axes, keep the default value `axis=None`.

```
In [4]:   # 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 (Non
          e, 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(anchor - positive), axis = -1)
              # Step 2: Compute the (encoding) distance between the anchor and the
          negative
              neg_dist = tf.reduce_sum(tf.square(anchor - negative), axis = -1)
              # Step 3: subtract the two previous distances and add alpha.
              basic_loss = (pos_dist - neg_dist) + alpha
              # Step 4: Take the maximum of basic_loss and 0.0. Sum over the train
          ing examples.
              loss = tf.reduce_sum(tf.maximum(basic_loss, 0.0), axis=0)
              ### END CODE HERE ###

              return loss
```

```
In [5]:   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 = 528.143
```

Expected Output:

<div align="center">

loss   528.143

</div>

2 - Loading the pre-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 [6]: FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics = ['acc
        uracy'])
        load_weights_from_FaceNet(FRmodel)
```

```
------------------------------------------------------------------------
----
InvalidArgumentError                      Traceback (most recent call l
ast)
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/comm
on_shapes.py in _call_cpp_shape_fn_impl(op, input_tensors_needed, input
_tensors_as_shapes_needed, debug_python_shape_fn, require_shape_fn)
    670             graph_def_version, node_def_str, input_shapes, input_
tensors,
--> 671             input_tensors_as_shapes, status)
    672     except errors.InvalidArgumentError as err:

/opt/conda/lib/python3.6/contextlib.py in __exit__(self, type, value, t
raceback)
     88             try:
---> 89                 next(self.gen)
     90             except StopIteration:

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/erro
rs_impl.py in raise_exception_on_not_ok_status()
    465             compat.as_text(pywrap_tensorflow.TF_Message(status)),
--> 466             pywrap_tensorflow.TF_GetCode(status))
    467     finally:

InvalidArgumentError: Invalid reduction dimension 0 for input with 0 di
mensions. for 'loss/lambda_1_loss/Sum_2' (op: 'Sum') with input shapes:
[], [] and with computed input tensors: input[1] = <0>.

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call l
ast)
<ipython-input-6-7c25b79864e6> in <module>()
----> 1 FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metric
s = ['accuracy'])
      2 load_weights_from_FaceNet(FRmodel)

/opt/conda/lib/python3.6/site-packages/keras/engine/training.py in comp
ile(self, optimizer, loss, metrics, loss_weights, sample_weight_mode, w
eighted_metrics, target_tensors, **kwargs)
    848                 with K.name_scope(self.output_names[i] + '_los
s'):
    849                     output_loss = weighted_loss(y_true, y_pred,
--> 850                                                 sample_weight,
 mask)
    851                 if len(self.outputs) > 1:
    852                     self.metrics_tensors.append(output_loss)

/opt/conda/lib/python3.6/site-packages/keras/engine/training.py in weig
hted(y_true, y_pred, weights, mask)
    448         """
    449         # score_array has ndim >= 2
--> 450         score_array = fn(y_true, y_pred)
    451         if mask is not None:
    452             # Cast the mask to floatX to avoid float64 upcastin
g in theano
```

```
<ipython-input-4-7c677a2183a9> in triplet_loss(y_true, y_pred, alpha)
     26         basic_loss = (pos_dist - neg_dist) + alpha
     27         # Step 4: Take the maximum of basic_loss and 0.0. Sum over
 the training examples.
---> 28         loss = tf.reduce_sum(tf.maximum(basic_loss, 0.0), axis=0)
     29         ### END CODE HERE ###
     30

/opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.p
y in reduce_sum(input_tensor, axis, keep_dims, name, reduction_indices)
   1249           _ReductionDims(input_tensor, axis, reduction_indices),
   1250           keep_dims,
-> 1251           name=name)
   1252
   1253

/opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/gen_math_o
ps.py in _sum(input, reduction_indices, keep_dims, name)
   2526   result = _op_def_lib.apply_op("Sum", input=input,
   2527                                 reduction_indices=reduction_ind
ices,
-> 2528                                 keep_dims=keep_dims, name=name)
   2529   return result
   2530

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/op_d
ef_library.py in apply_op(self, op_type_name, name, **keywords)
    765           op = g.create_op(op_type_name, inputs, output_types, na
me=scope,
    766                            input_types=input_types, attrs=attr_pr
otos,
--> 767                            op_def=op_def)
    768         if output_structure:
    769           outputs = op.outputs

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/ops.
py in create_op(self, op_type, inputs, dtypes, input_types, name, attr
s, op_def, compute_shapes, compute_device)
   2506                     original_op=self._default_original_op, op_d
ef=op_def)
   2507     if compute_shapes:
-> 2508       set_shapes_for_outputs(ret)
   2509     self._add_op(ret)
   2510     self._record_op_seen_by_control_dependencies(ret)

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/ops.
py in set_shapes_for_outputs(op)
   1871         shape_func = _call_cpp_shape_fn_and_require_op
   1872
-> 1873   shapes = shape_func(op)
   1874   if shapes is None:
   1875     raise RuntimeError(

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/ops.
py in call_with_requiring(op)
   1821
   1822   def call_with_requiring(op):
```

```
-> 1823         return call_cpp_shape_fn(op, require_shape_fn=True)
   1824
   1825   _call_cpp_shape_fn_and_require_op = call_with_requiring

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/comm
on_shapes.py in call_cpp_shape_fn(op, input_tensors_needed, input_tenso
rs_as_shapes_needed, debug_python_shape_fn, require_shape_fn)
    608     res = _call_cpp_shape_fn_impl(op, input_tensors_needed,
    609                                   input_tensors_as_shapes_neede
d,
--> 610                                   debug_python_shape_fn, requir
e_shape_fn)
    611   if not isinstance(res, dict):
    612     # Handles the case where _call_cpp_shape_fn_impl calls un
known_shape(op).

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/comm
on_shapes.py in _call_cpp_shape_fn_impl(op, input_tensors_needed, input
_tensors_as_shapes_needed, debug_python_shape_fn, require_shape_fn)
    674         missing_shape_fn = True
    675       else:
--> 676         raise ValueError(err.message)
    677
    678   if missing_shape_fn:

ValueError: Invalid reduction dimension 0 for input with 0 dimensions.
 for 'loss/lambda_1_loss/Sum_2' (op: 'Sum') with input shapes: [], [] a
nd with computed input tensors: input[1] = <0>.
```
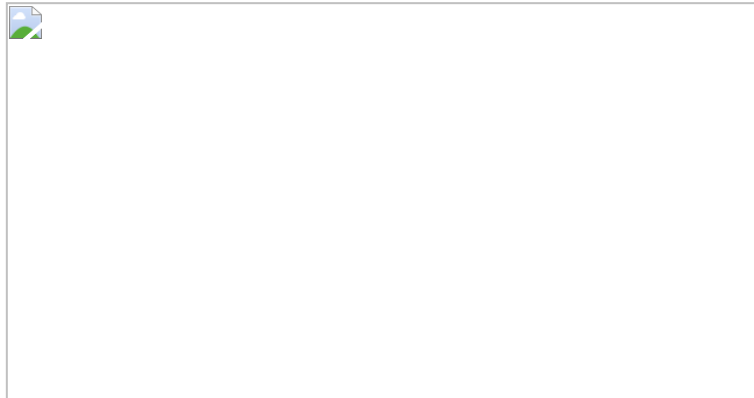
Here are some examples of distances between the encodings between three individuals:

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

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

3 - Applying the model

You are building a system for an office building where the building manager would like to offer facial recognition to allow the employees to enter the building.

You'd like to build a **Face verification** system that gives access to the list of people who live or work there. To get admitted, each person has to swipe an ID card (identification card) to identify themselves at the entrance. 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 who is allowed to enter the office. To generate the encoding we use `img_to_encoding(image_path, model)` , which 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 [7]: 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", FRmode
        l)
        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 between 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 it.

- As presented above, you should use the L2 distance np.linalg.norm (https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.norm.html).
- (Note: In this implementation, compare the L2 distance, not the square of the L2 distance, to the threshold 0.7.)

Hints

- `identity` is a string that is also a key in the `database` dictionary.
- `img_to_encoding` has two parameters: the `image_path` and `model`.

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

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

             Arguments:
             image_path -- path to an image
             identity -- string, name of the person you'd like to verify the iden
             tity. Has to be an employee who works in the office.
             database -- python dictionary mapping names of allowed people's name
             s (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, FRmodel)

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

             # 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 in!")
                 door_open = True
             else:
                 print("It's not " + str(identity) + ", please go away")
                 door_open = False

             ### END CODE HERE ###

             return dist, door_open
```
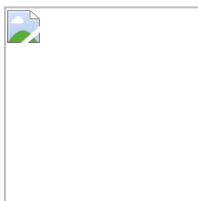
Younes is trying to enter the office and the camera takes a picture of him ("images/camera_0.jpg"). Let's run your verification algorithm on this picture:

```
In [9]: verify("images/camera_0.jpg", "younes", database, FRmodel)
```
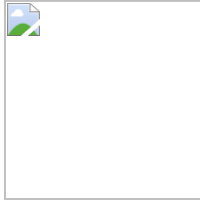
It's younes, welcome in!

Out[9]: (0.067291878, True)

Expected Output:

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

Benoit, who does not work in the office, stole Kian's ID card and tried to enter the office. The camera took a picture of Benoit ("images/camera_2.jpg). Let's run the verification algorithm to check if benoit can enter.


```
In [10]: verify("images/camera_2.jpg", "kian", database, FRmodel)
```

It's kian, welcome in!

Out[10]: (0.15565163, True)

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 office the next day and couldn't get in!

To solve this, 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 building, and the 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 one of the inputs.

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
2. Find the encoding from the database that has smallest distance with the target encoding.
   - 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 the 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 [12]:  # GRADED FUNCTION: who_is_it

          # GRADED FUNCTION: who_is_it

          def who_is_it(image_path, database, model):
              """
              Implements face recognition for the office by finding who is the per
          son 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_e
          ncoding() 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 curr
          ent db_enc from the database. (≈ 1 line)
                  dist = np.linalg.norm(encoding - db_enc)

                  # 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 [13]:  who_is_it("images/camera_0.jpg", database, FRmodel)

          it's younes, the distance is 0.0672919

Out[13]:  (0.067291878, '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.

Congratulations!

- Your face recognition system is working 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.

Ways to improve your facial recognition model

Although we won't implement it here, here are 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 increase 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.

Key points to 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)