Application of One-Shot Learning, Siamese Networks and Triplet Loss

- Introduction
- Problems
- One-Shot Classification and Siamese Network **Face Recognition** problem and **Humanoid Robot** with **Triplet loss** as the loss function. We will:
  - Implement the Triplet loss function
  - Use a pre-trained ConvNet to map images into 128-dimensional encodings.
  - Use these encodings to perform **face verification** and **face recognition**

Recall notation: m denotes the number of training examples.

Face Recognition - "is this the named person?" for example, at some airports, we can pass through custom by letting a system scan our passport and then verifying that we look like the person we claim to be. Biometrics, fingerprint scanners, and other systems, using our face to verify we are who we say we are, are becoming more common.

Face Recognition - "who is this person?". For example, the video lecture shows a face recognition video of Baidu employees entering their workplace. Also, Facebook uses face recognition to tag people in photos.

Classification vs One Shot Learning

- **Classical classification** - this approach does not work well for face recognition because:
  - Given only a few examples of a person's face, it is not enough to train a good classifier. Also, if a new person shows up, it is hard to retrain the face recognition model. Finally, the algorithm, in this image belonging to each of the 4 classes, has too many degrees of freedom. Accordingly, implementing the probability of the image belonging to each of the 4 classes is too much.

Two approaches which can address these issues:

- 1. Doing the training process, so that it learns a function of images for each of the two cases, Alice, Victor and Bob/Eve. The trained function outputs one for the element-wise difference of the two images if they are the same person, and zero if they are different people. Then, to recognize a person, we can compare their image with a picture we have on file, and if the output is close to one, then the people agree.
- 2. Learn a function that takes an another image as input and outputs how similar the two input images are. This is called a similarity function. Denote the similarity function by d, $d(image\ 1, image\ 2)$ = degree of difference between the two images. Then, if $d(image\ 1, image\ 2) \leq \tau$ for some threshold τ, we can predict the two people are the same, and if it is greater than τ, we can predict they are different.

Two applications where we would want this function to work: 1. face verification and 2. face recognition.

- **Face verification** - "is this the named person?" For example, at some airports, we can pass through custom by letting a system scan our passport and then verifying that we look like the person we claim to be. Biometrics, fingerprint scanners, and other systems, using our face to verify we are who we say we are, are becoming more common.
- **Face Recognition** - "who is this person?". For example, the video lecture shows a face recognition video of Baidu employees entering their workplace. Also, Facebook uses face recognition to tag people in photos.

Classification vs One Shot Learning

- **Classical classification** - this approach does not work well for face recognition because:
  - Given only a few examples of a person's face, it is not enough to train a good classifier. Also, if a new person shows up, it is hard to retrain the face recognition model. Finally, the algorithm, in this image belonging to each of the 4 classes, has too many degrees of freedom. Accordingly, implementing the probability of the image belonging to each of the 4 classes is too much.

Two approaches which can address these issues:

- 1. Doing the training process, so that it learns a function of images for each of the three cases, Alice, Victor and Bob/Eve. The trained function outputs one for the element-wise difference of the two images if they are the same person, and zero if they are different people. Then, to recognize a person, we can compare their image with a picture we have on file, and if the output is close to one, then the people agree.
- 2. Learn a function that takes an another image as input and outputs how similar the two input images are. This is called a similarity function. Denote the similarity function by d, $d(image\ 1, image\ 2)$ = degree of difference between the two images. Then, if $d(image\ 1, image\ 2) \leq \tau$ for some threshold τ, we can predict the two people are the same, and if it is greater than τ, we can predict they are different.

Two applications where we would want this function to work: 1. face verification and 2. face recognition.

Problems:

- Given one picture, we first require a lot of different images of each of the 3 people in the organization which might not be feasible. Imagine if we are doing this for an organization with thousands of employees.
- What if a new person joins or leaves the organization? We need to take the part of the underlying deep again and re-train the system. Imagine if we are doing this for an organization with thousands of employees.

Now let's understand how we can approach this problem using one-shot classification and triplet loss to solve both of the above issues.

Instead of directly classifying an inputted image to one of the 4 people in the organization, this network instead takes a pair of images as input and outputs the degree of difference between the two images. Then, we can use this function to recognize a person. Specifically, for **verification**, we can compare the inputted image to a stored image for the person we claim to be. Also, for **recognition**, we can compare the inputted image to images of all people we know.

Now let's understand how this can be implemented.

Face Recognition

- **Face Recognition** problem is continuing from last session a human face example and instead of an image, a robot and also our friend Alice. Alice has agreed to let her photo be used in our database to use as our identity verification in the robot's face recognition system. Alice's photo is the only image we have of her. Alice's photo is a 100 x 100 x 3 image. Alice's photo is the only image we have of her. Alice's photo is the only image we have of her. Alice's photo is the only image we have of her

Benoit Image

```
In [14]: !mkdir -p /content/drive/MyDrive/Images
```

```
In [15]: !cp -R /content/drive/MyDrive/Images/* /content/Images
```

```
In [16]: !ls -R /content/Images
```

```
/content/Images:
Benoit  Yousra

/content/Images/Benoit:
Benoit.jpeg

/content/Images/Yousra:
Yousra.jpeg
```

3.2 Facial Recognition

The **face verification** system is mostly working well. But since that got the ID card details, when he came back to the office the next day and another girl ...

To solve this, we'd like to change our **face verification** system into a **face recognition** system. This way, no one has to use an ID card anymore. Instead, the system will recognize who is in the picture, and let them in if they are a team member.

With Facenet's **face recognition**, system will take as input an image, and figure out if it is one of the authorized persons stored in our area. Unlike the previous **face verification** system, we will no longer get a person's name as one of the inputs.

We will now implement `who_is_it()`. We 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 be used to store the smallest distance encoding to the target encoding.
   - Loop over the database dictionary's names and encodings. To loop over, 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`.

Let's create a function for doing Facial Recognition

```
In [17]: def who_is_it(image_path, database, model):
    ^^^
```