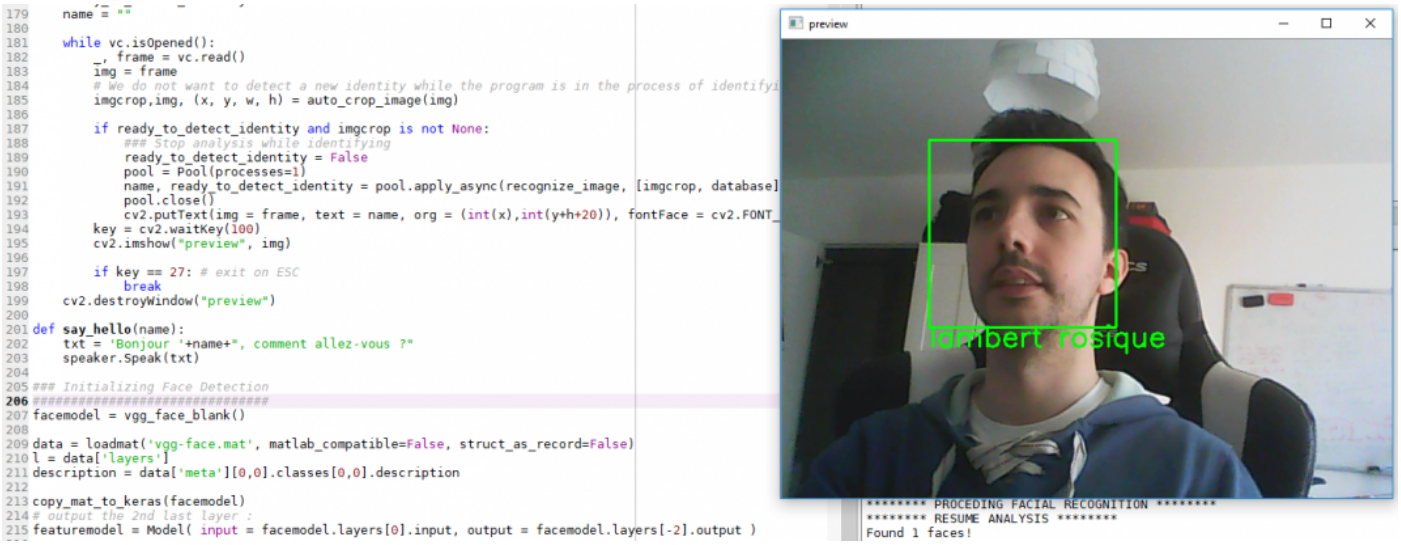



# TP : La Reconnaissance Faciale

Par **Lambert R.** - 14 mai 2019

Dans le focus précédent, nous avons vu en détail ce qu'était la **reconnaissance faciale** et comment elle fonctionnait.

Aujourd'hui, je vous propose de mettre tout ça en pratique : programmons notre première IA de reconnaissance faciale, capable de fonctionner avec *n'importe quel ordinateur* !

En bonus, nous verrons comment saluer la personne identifiée, quelles pistes d'amélioration sont envisageables, ainsi que beaucoup d'autres éléments intéressants.

**Lien vers le focus sur la Reconnaissance Faciale**

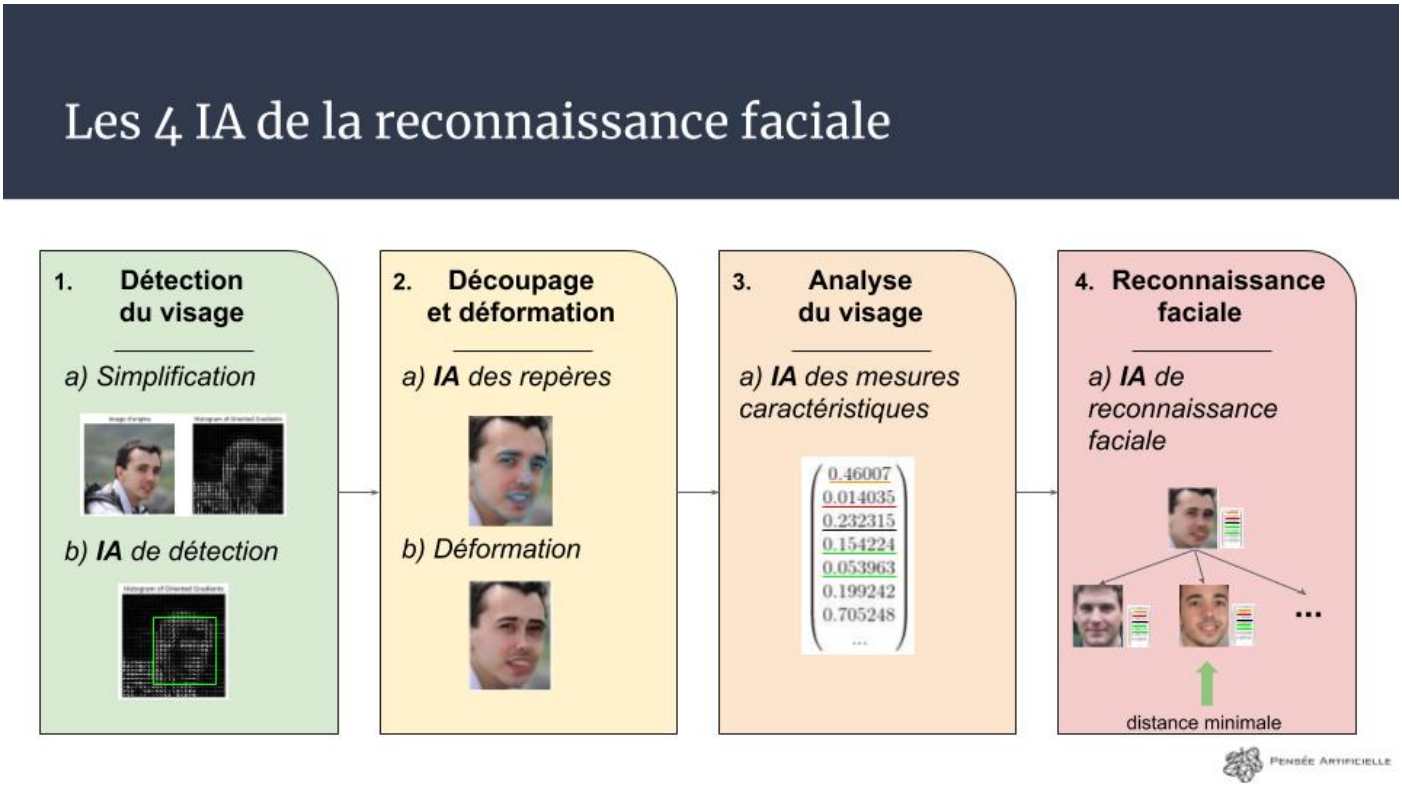
**Sommaire** [Masquer]

# I. Petits rappels et préparation de l'environnement

## La reconnaissance faciale : 4 IA qui œuvrent de concert

Même si tout est expliqué dans le focus en lien plus haut, la reconnaissance faciale n'est pas « une intelligence artificielle qui reconnait les gens » mais… 4 IA, différentes, qui s'enchaînent pour arriver à identifier quelqu'un (ou à vérifier une identité) :

- Tout d'abord, une IA trouve le visage dans l'image (avec l'algorithme Haar Cascade par exemple)
- Puis le visage est découpé et déformé pour être recentré et réaligné par rapport à l'image : c'est le Dlib Shape Predictor qui peut faire ça
- Ensuite une IA analyse l'image et en donne 128 nombres qui la représentent, via un réseau de neurones convolutifs : VGG Face




*Les 4 IA de la reconnaissance faciale*

Dans ce TP, nous allons mettre en place toute la chaîne ci-dessus à l'exception de la partie 2 (découpage et déformation) car on se placera toujours face à la caméra pour les reconnaissances, et je souhaite alléger les calculs de l'ordinateur (mais j'en parlerai en bonus en fin d'article).

## Ordinateur portable ou ordinateur fixe pour ce TP ?

Comme promis, ce TP a été pensé pour fonctionner sur un **ordinateur portable** classique, uniquement grâce au **processeur** (CPU). Les temps de prédiction pour reconnaître quelqu'un seront de l'ordre *de 2 à 10s*, donc si vous souhaitez avoir du « temps réel » je vous recommande fortement de passer par votre carte graphique (GPU) pour faire les calculs.

| Device | Speed of training, examples/sec |
|---|---|
| 2 x AMD Opteron 6168 | 440 |
| i7-7500U | 415 |
| GeForce 940MX | 1190 |
| GeForce 1070 | 6500 |

*Les entraînements (et les prédictions) sont beaucoup plus rapides via la carte graphique ! (Source)*

Comment ? C'est très simple : au lieu d'installer Tensorflow, il va falloir installer **Tensorflow-GPU** (il faudra avoir une carte graphique GeForce compatible). Le temps de prédiction sera d'environ **0.1 à 0.3s**.

## Récupération des sources

entraînés).

**Lien vers le projet Github de Pensée Artificielle**

D'autre part, il va falloir télécharger également le fichier suivant de 1Go. Il s'agit d'un modèle d'intelligence artificielle **déjà entraîné**, nommé VGG Face, capable de construire les vecteurs représentant les images (3e étape) :

**Lien pour télécharger VGG Face (1Go)**

## Python et dépendances

Pour ce TP, nous allons utiliser Python 3, car il s'agit d'un langage parfaitement adapté à nos besoins, qui dispose des outils pour traiter les images, la caméra et les intelligences artificielles.

**Lien pour télécharger Python 3**

*ATTENTION : N'installez surtout pas la version 3.7 de Python car elle n'est pas compatible avec Tensorflow, mais plutôt une version 3.5.X ou 3.6.X (pour ma part j'utilise la **3.6.5** mais vous pouvez utiliser la 3.6.8 par exemple) !*

Maintenant que vous avez Python, on va avoir besoin des librairies suivantes :

- **Numpy** : une librairie qui fournit de nombreuses fonctions mathématiques de calcul et une gestion des vecteurs et des matrices très poussée. La plupart des algorithmes d'intelligence artificielle prennent en entrée des matrices Numpy
- **OpenCV** : qui est dédiée au traitement d'images
- **SciPy** : réservée aux calculs scientifiques, dont le traitement du signal (on s'en servira surtout pour la 4e partie de recherche dans la base de données)

Ouvrez une invite de commande et saisissez les lignes suivantes (suivant votre installation de Python, vous devrez remplacer « pip » par « pip3 »).

MS DOS

```
pip install numpy
pip install opencv-python
pip install scipy
```

développé par Google). Il est donc parfaitement adapté aux projets « simples » qui ne veulent pas s'embarrasser de trop de paramètres techniques et cherchent un résultat rapide (c'est notre cas !).

*ATTENTION : **veillez à bien avoir installé tensorflow OU tensorflow-gpu au préalable** (mais pas les deux en même temps, et pour le GPU vous aurez besoin d'autres logiciels avant) ! N'hésitez pas à consulter notre article dédié au sujet…*

```
MS DOS
1 pip install tensorflow OU pip install tensorflow-gpu==1.10.0
2 pip install keras
```

Pour conclure, dans notre partie bonus, nous allons voir comment utiliser le synthétiseur vocal de Windows pour faire « parler » votre ordinateur et souhaiter la bienvenue aux gens reconnus par notre IA. Il faudra donc installer les deux dépendances suivantes :

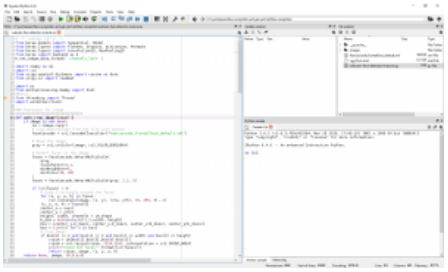
```
MS DOS
1 pip install pywin32
2 pip install pypiwin32
```

A noter que j'utilise pour coder **Spyder 3**, qui permet de voir ses variables, d'utiliser une console IPython et beaucoup d'autres choses encore. Je vous le recommande ou, à défaut, Jupyter Notebook (vous devrez copier/coller mon code dans ce cas) :

```
MS DOS
1 pip install spyder
2 OU
3 pip install jupyter
```

**Et voilà, nous sommes prêts !**

Normalement, votre environnement devrait ressembler à cette image. Attention à bien vous placer dans le répertoire face-recognition (renommé en reconnaissance-faciale sur Github) dans le cadre en haut à droite, car c'est le dossier affiché qui est utilisé par la console (en bas à droite) !



Si vous n'y êtes pas, vous aurez des erreurs au chargement des fichiers.

# II. TP Reconnaissance Faciale : bien le bonjour !

Tout est téléchargé et installé, nous pouvons commencer !

N'hésitez pas à remplacer mes photos du dossier **images** par les vôtres, en incluant une photo de vous, mais n'en mettez pas trop car l'algorithme va rapidement perdre en prédiction. A noter également que vous pouvez mettre une ou plusieurs photos d'une même personne sans que cela pose le moindre problème.

A noter qu'il est conseillé, pour vos essais, d'utiliser des photos de personnes connues (acteurs notamment), car elles sont souvent libres de droit et vous n'aurez aucun mal à en trouver davantage !

*Pour ce TP, j'ai choisi les acteurs d'Harry Potter + une photo générée par une IA sur le site thispersondoesnotexist. Remplacez librement ces photos par celles de votre choix, et idéalement ajoutez votre propre photo à la collection pour tester la caméra.*

## Imports

Commençons par les imports dans notre nouveau fichier « webcam-face-detection-tutorial.py » à la racine du dossier que vous avez téléchargé tout à l'heure.

Importons numpy, opencv et scipy :

Python

```
import numpy as np
import cv2
from scipy.spatial.distance import cosine as dcos
from scipy.io import loadmat
```

Puis Keras :

Python

```
from keras.models import Sequential, Model
from keras.layers import Flatten, Dropout, Activation, Permute
from keras.layers import Convolution2D, MaxPooling2D
from keras import backend as K
K.set_image_data_format( 'channels_last' )
```

La notation « import backend as K » est très souvent utilisée car elle permet de configurer plus finement notre framework. En particulier, j'ai fixé la manière qu'a Keras de voir les vecteurs : en indiquant channels_last (les channels ou canaux sont le nombre de « couleurs », 3 pour RGB ou 1 pour niveaux de gris), il s'attendra à recevoir le format (lignes, colonnes, nombre de canaux) au lieu de (nombre de canaux, lignes, colonnes).

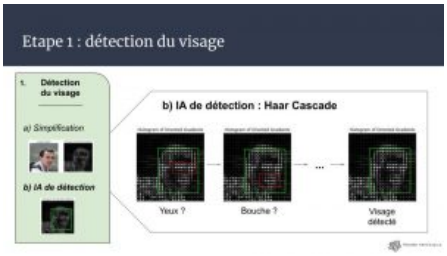
Enfin, on va importer quelques librairies utilitaires

- os pour aller lire le contenu des répertoires et charger les images
- multiprocessing pour faire les prédictions dans un thread séparé et ne pas ralentir la vidéo
- threading pour lancer l'audio (en bonus) séparément sans bloquer la vidéo ou avoir plusieurs voix en même temps
- win32com pour récupérer la voix de Windows (en bonus)

```python
import os
from multiprocessing.dummy import Pool
#Bonus :
from threading import Thread
import win32com.client
```

## 1) Détection des visages dans une image

Cette première étape est cruciale, et c'est celle qu'il manquait à Woodrow Bledsoe pour que sa première IA de reconnaissance faciale soit autonome ! (des humains s'occupaient de cette étape 1).




*IA de détection : trouver le visage en cascade*

D'après notre schéma, dans cette étape nous devons :

- convertir l'image en niveaux de gris
- appliquer l'algorithme HOG (Histogram of Oriented Gradients) pour simplifier l'image
- envoyer cette image à l'algorithme Haar Cascade qui s'occupera de trouver les visages

Bien souvent, j'insiste sur la nécessité de connaître les outils que l'on emploie. En particulier, dès que l'on télécharge un modèle entraîné, il faut se demander :

- Quelles étaient les données d'entrée ? Pour anticiper certaines failles de l'apprentissage, comme le fait que l'IA n'ait jamais vu de paysage enneigé ou sans humain
- Quelle est la préparation qui est faite sur chaque donnée avant d'être injectée dans l'IA ? Si l'IA a été entraînée à trouver des visages dans des images en niveaux de gris, il est hors de question de lui fournir des images en couleur, car elle ne s'en sortira pas aussi bien ! De même, si les images d'entrée étaient recadrées en 224×224 pixels et normalisées entre 0 et 1, alors il faudra faire pareil, sinon le réseau de neurones convolutifs (par exemple) crashera…

HOG ! Cela va nous permettre d'accélérer (un tout petit peu) les calculs. *Vous retrouverez un exemple de HOG en bonus à la fin.*

On commence donc par charger notre fichier haarcascade et par le configurer en indiquant comment il doit fusionner les visages qu'il identifie (en effet, il va trouver plusieurs fois le même visage en décalant simplement sa zone de quelques pixels) :

Python

```python
def auto_crop_image(image):
    if image is not None:
        im = image.copy()
        # Load HaarCascade from the file with OpenCV
        faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

        # Read the image
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Detect faces in the image
        faces = faceCascade.detectMultiScale(
            gray,
            scaleFactor=1.1,
            minNeighbors=5,
            minSize=(30, 30)
        )
        faces = faceCascade.detectMultiScale(gray, 1.2, 5)
```

Puis dans un second temps on va tracer des rectangles autour de tous les visages trouvés et on va découper l'image autour du premier.

Ici, si vous le souhaitez, vous pouvez découper tous les visages et les traiter en parallèle, rien ne vous en empêche (à part la puissance de votre ordinateur).

Python

```python
        if len(faces) > 0:
            # Draw a rectangle around the faces
            for (x, y, w, h) in faces:
                cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
            (x, y, w, h) = faces[0]
            center_x = x+w/2
            center_y = y+h/2
            height, width, channels = im.shape
            b_dim = min(max(w,h)*1.2,width, height)
            box = [center_x-b_dim/2, center_y-b_dim/2, center_x+b_dim/2, center_y+b_dim/2]
            box = [int(x) for x in box]
            # Crop Image
            if box[0] >= 0 and box[1] >= 0 and box[2] <= width and box[3] <= height:
                crpim = im[box[1]:box[3],box[0]:box[2]]
                crpim = cv2.resize(crpim, (224,224), interpolation = cv2.INTER_AREA)
                print("Found {0} faces!".format(len(faces)))
                return crpim, image, (x, y, w, h)
    return None, image, (0,0,0,0)
```

Grâce à cette méthode, on est donc en mesure de trouver tous les visages qui sont dans

## 2) Découpage et déformation

On vient à l'instant de découper le visage qui était dans notre image. **Haar Cascade** centrant ses zones autour des visages, ces derniers sont automatiquement centrés et bien alignés s'ils se tiennent bien face à la caméra. Nous n'avons donc rien de particulier à faire ici, et si vous voulez retrouver le **Dlib Shape Predictor** (qui déforme les visages), je vous redirige sur la sections bonus.

## 3) Analyse du visage

On attaque le cœur de la reconnaissance faciale qui va nécessiter beaucoup de code.

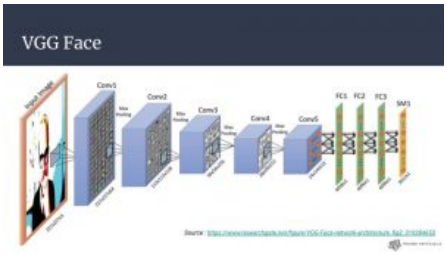
### Chargement du modèle VGG Face

VGG Face n'est vraiment pas évident à utiliser si l'on ne regarde pas des exemples l'utilisant déjà : parfois, il faut chercher des projets github qui utilisent un modèle pré-entraîné particulier pour comprendre ce qui est attendu en entrée et pouvoir procéder aux pré-traitements.

Ici, je me suis inspiré de ce qui était fait dans examples/cnn_vgg_face.m (sur le site de VGG Face donné précédemment) et j'ai regardé à quoi ressemblait le jeu de données d'entraînement, puis cherché des projets l'utilisant.

On se retrouve ainsi avec le code suivant qui peut être divisé en deux phases :

- Construction du modèle de réseau convolutif qui est identique à VGG
- Chargement des poids de toutes les variables du réseau, qui sont sauvegardées dans vgg-face.mat, et que l'on applique au modèle que l'on a réécrit




*Voici la structure de ce fameux réseau convolutif « VGG Face »*

Pourquoi avoir réécrit le modèle ? Il faut savoir que lorsqu'on sauvegarde son IA, il y a deux éléments que l'on peut enregistrer (séparément) : le modèle (i.e. comment est construite notre IA, combien de couches de neurones, avec quelles fonctions mathématiques, etc…) et les variables (les poids des différentes liaisons de neurones, les filtres de convolutions, etc…). Ici, le fichier que l'on a téléchargé est celui de ces fameux poids, donc on doit

## Création du modèle de CNN :

Python

```python
def convblock(cdim, nb, bits=3):
    L = []
    for k in range(1,bits+1):
        convname = 'conv'+str(nb)+'_'+str(k)
        L.append( Convolution2D(cdim, kernel_size=(3, 3), padding='same', activation='relu',
    L.append( MaxPooling2D((2, 2), strides=(2, 2)) )
    return L

def vgg_face_blank():
    withDO = True # no effect during evaluation but usefull for fine-tuning
    if True:
        mdl = Sequential()
        mdl.add( Permute((1,2,3), input_shape=(224,224,3)) )
        for l in convblock(64, 1, bits=2):
            mdl.add(l)
        for l in convblock(128, 2, bits=2):
            mdl.add(l)
        for l in convblock(256, 3, bits=3):
            mdl.add(l)
        for l in convblock(512, 4, bits=3):
            mdl.add(l)
        for l in convblock(512, 5, bits=3):
            mdl.add(l)
        mdl.add( Convolution2D(4096, kernel_size=(7, 7), activation='relu', name='fc6') )
        if withDO:
            mdl.add( Dropout(0.5) )
        mdl.add( Convolution2D(4096, kernel_size=(1, 1), activation='relu', name='fc7') )
        if withDO:
            mdl.add( Dropout(0.5) )
        mdl.add( Convolution2D(2622, kernel_size=(1, 1), activation='relu', name='fc8') )
        mdl.add( Flatten() )
        mdl.add( Activation('softmax') )

        return mdl

    else:
        raise ValueError('not implemented')
```

## Importation des poids :

Python

```python
def copy_mat_to_keras(kmodel):
    kerasnames = [lr.name for lr in kmodel.layers]
    prmt = (0,1,2,3)

    for i in range(l.shape[1]):
        matname = l[0,i][0,0].name[0]
        if matname in kerasnames:
            kindex = kerasnames.index(matname)
            l_weights = l[0,i][0,0].weights[0,0]
            l_bias = l[0,i][0,0].weights[0,1]
            f_l_weights = l_weights.transpose(prmt)
            assert (f_l_weights.shape == kmodel.layers[kindex].get_weights()[0].shape)
            assert (l_bias.shape[1] == 1)
            assert (l_bias[:,0].shape == kmodel.layers[kindex].get_weights()[1].shape)
            assert (len(kmodel.layers[kindex].get_weights()) == 2)
            kmodel.layers[kindex].set_weights([f_l_weights, l_bias[:,0]])
```

```python
# CNN model initialization
facemodel = vgg_face_blank()
# Load the pretrained weights into the model
data = loadmat('vgg-face.mat', matlab_compatible=False, struct_as_record=False)
l = data['layers']
description = data['meta'][0,0].classes[0,0].description

copy_mat_to_keras(facemodel)
# Final model that can get inputs and generate a prediction as an output
featuremodel = Model( input = facemodel.layers[0].input, output = facemodel.layers[-2].output
```

Pour avoir une prédiction sur une image il suffira alors d'appeler (je vous encourage à essayer de comprendre par vous-même qu'est-ce que l'on met de côté après la prédiction) :

Python

```python
featuremodel.predict(vector_image)[0,:]
```

# 4) Trouver le vecteur le plus proche dans la base de données

## Chargement de la base de données du TP

Pour accélérer les performances de notre IA, on va stocker dans notre base de données le vecteur de chaque image de la base, plutôt que chaque image (on réalise ainsi une seule fois la prédiction des images de la base).

Pour cela, c'est très simple :

- On parcourt le dossier /images
- Pour chaque image trouvée, on extrait le visage
- Puis on réalise une prédiction dessus (i.e. on génère un vecteur de 128 nombres grâce à notre CNN)
- Et on stocke ce vecteur dans un dictionnaire qui va représenter notre base de données !

Python

```python
def generate_database(folder_img = "images"):
    database = {}
    for the_file in os.listdir(folder_img):
        file_path = os.path.join(folder_img, the_file)
        try:
            if os.path.isfile(file_path):
                name = the_file.split(".")[0]
                img = cv2.imread(file_path)
                crpim, srcimg, (x, y, w, h) = auto_crop_image(img)
                vector_image = crpim[None,...]
                database[name] = featuremodel.predict(vector_image)[0,:]
        except Exception as e:
            print(e)
```

```
db = generate_database()
```

pour générer les vecteurs de toute notre base.

Par exemple, pour ma photo, on a dans la base de données un vecteur de 128 mesures caractéristiques :

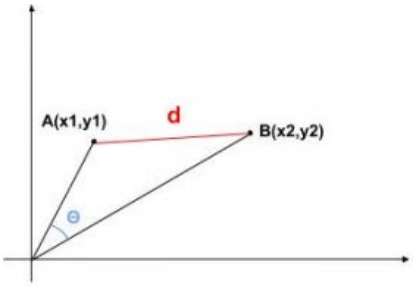
*In [2]: db[« lambert rosique »]*
*Out[2]: array(**[1.4528134, 0. , 2.7342448, …, 0. , 0.9508694, 1.7749509]**, dtype=float32)*

## Cherchons dans la base à qui appartient cette image…

Dernière étape de notre reconnaissance faciale, on dispose d'une base de données de vecteurs et l'on cherche celui qui est le plus proche du vecteur issu de la caméra.

Plusieurs méthodes sont possibles :

- Utiliser un algorithme de Machine Learning, qui va par exemple grouper les vecteurs dans l'espace (on parle de clustering) ou construire une frontière qui va les séparer les uns des autres (avec **Support Vector Machine** notamment)
- Ou alors, calculer soit la distance entre les vecteurs soit **l'angle entre les vecteurs** (solution que nous allons privilégier) et dire que deux vecteurs sont proches si leur angle est faible




*On mesure l'écart entre A et B soit par la distance d qui les sépare soit par l'angle teta que forment ces deux vecteurs (similarité cosinus)*

Pour calculer cet angle, on va utiliser la fonction « dcos » (alias la similarité cosinus) de scipy qui fonctionne très bien. Par contre, en termes de « prédictions », s'il y a trop d'images dans la base de données, l'imprécision sera grande et il vaudra mieux s'orienter sur une autre approche. Dans le cadre de la distance cosinus, je vous conseillerai donc (avec ce TP), de ne pas dépasser la vingtaine d'images.

Voici le code pour calculer la distance entre deux vecteurs :

Python

```
def find_closest(img, database, min_detection=2.5):
    imarr1 = np.asarray(img)
```

```
    umin = ""
    for key, value in database.items():
        fvec2 = value
        dcos_1_2 = dcos(fvec1, fvec2)
        if umin == "":
            dmin = dcos_1_2
            umin = key
        elif dcos_1_2 < dmin:
            dmin = dcos_1_2
            umin = key
    if dmin > min_detection:
        umin = ""
    return umin, dmin
```

## Lancement de la caméra et analyse en temps réel

Pour démarrer la caméra, nous allons utiliser cette commande d'OpenCV :

Python

```
cv2.VideoCapture(0)
```

Le « 0 » signifie que l'on prend le premier périphérique caméra que l'on trouve (celui par défaut donc). Si vous souhaitez utiliser une **vidéo déjà enregistrée** sur votre ordinateur, il suffit de remplacer le 0 par le chemin de votre fichier ! Et si vous voulez l'appliquer à une image, il faut aller directement à la partie « traitement de l'image extraite de la vidéo ».

On va construire une fonction qui :

- lance la caméra et récupère les frames
- pour chaque frame, détecte la présence de visage
- si l'ordinateur n'est pas en train d'analyser une image, alors on lui demande de trouver qui est sur la frame (dans un processus séparé pour ne pas figer l'image de la caméra le temps des calculs)
- une fois le traitement terminé, on affiche le nom trouvé par l'algorithme sous le cadre tracé autour du visage !
- et si vous appuyez sur la touche ECHAP vous pourrez interrompre le flux

Python

```
def webcam_face_recognizer(database):
    cv2.namedWindow("preview")
    vc = cv2.VideoCapture(0)
    ready_to_detect_identity = True
    name = ""

    while vc.isOpened():
        _, frame = vc.read()
        img = frame
        # Image analysis (start here with img loaded with your image)
```

```
            pool = Pool(processes=1)
            name, ready_to_detect_identity = pool.apply_async(recognize_image, [imgcrop, dat
            pool.close()
            cv2.putText(img = frame, text = name, org = (int(x),int(y+h+20)), fontFace = cv2
        key = cv2.waitKey(100)
        cv2.imshow("preview", img)

        if key == 27: # exit on ESC
            break
    cv2.destroyWindow("preview")
```

Il ne reste alors plus qu'à appeler cette méthode, et votre IA de reconnaissance faciale sera terminée !

Python

```
webcam_face_recognizer(db)
```

# III. Pistes d'améliorations et bonus

Votre IA de reconnaissance faciale est terminée et fonctionnelle, bravo ! Maintenant si vous êtes là, vous souhaitez approfondir un peu le sujet ou améliorer vos résultats…

## Amélioration des performances : 1 frame sur 10

Pour gagner en performances, une bonne idée est de ne pas analyser à chaque image qui est présent mais de se limiter à une image toutes les X images ou toutes les X millisecondes.

On a souvent recourt au tracking, en partant du principe qu'une personne ne va pas se « téléporter » entre deux images… Donc quelqu'un de reconnu n'a pas besoin de l'être immédiatement après, car a priori personne n'est passé devant entre temps, et il n'a pas pu disparaître.

Pour faire ça, on peut, par exemple, stocker l'emplacement des visages à un instant t et regarder à l'instant t+1 quels sont les emplacements les plus proches des emplacements précédents (généralement, quelqu'un ne bougera que de quelques pixels).

Python

```
    cpt = 0
    while vc.isOpened():
        _, frame = vc.read()
        cpt += 1
        if cpt >= 10:
            cpt = 0
            img = frame
```

## Amélioration des résultats : 3 frames consécutives

Il arrive parfois qu'une personne soit détectée dans le décor lors d'un changement de luminosité (les algorithmes ne sont ni parfaits ni optimaux). Pour ne pas tout changer, une bonne idée est d'obliger à détecter 3 fois d'affilée un visage dans l'image pour considérer qu'il y a une personne à l'écran.

Pour ça, on met simplement un compteur qui est incrémenté à chaque fois qu'il y a un visage dans l'image et réinitialisé s'il n'y en a pas. Si ce compteur vaut au moins 3, on lance l'analyse du visage de l'image (car le visage est là depuis au moins 3 images).

```python
        imgcrop,img = auto_crop_image(img)
        if ready_to_detect_identity:
            if imgcrop is not None:
                if previous_detection < nb_frames_with_face-1:
                    previous_detection += 1
                else:
                    previous_detection = 0
                    ready_to_detect_identity = False
                    pool = Pool(processes=1)
                    pool.apply_async(recognize_image, [imgcrop, database])
            else:
                previous_detection = 0
```

## Amélioration des résultats : 3 noms pour une personne

Il arrive souvent que l'IA me reconnaisse, puis le temps d'une frame me confonde avec quelqu'un d'autre avant de reconsidérer que c'est moi. Pour pallier à cette forte volatilité de notre algorithme, je propose la technique suivante :

- Analyser 3 images consécutives et réaliser 3 prédictions sur chacune d'elles
- Le nom de la personne sera celui qui sera ressorti au moins 2 fois sur les 3 (en cas d'égalité, recommencer). Bien sûr, si un nom sort deux fois d'affilée, pas besoin d'attendre la 3e fois !

```python
    names = []
    while vc.isOpened():
        _, frame = vc.read()
        img = frame
        # Image analysis (start here with img loaded with your image)
        # We do not want to detect a new identity while the program is in the process of ide
        imgcrop,img, (x, y, w, h) = auto_crop_image(img)

        if ready_to_detect_identity and imgcrop is not None:
            ### Stop analysis while identifying
```

```
                names = []
                cv2.putText(img = frame, text = name, org = (int(x),int(y+h+20)), fontFace =
            else:
                names.append(name)
                if len(names) >= 3:
                    names = []
        key = cv2.waitKey(100)
        cv2.imshow("preview", img)
        if key == 27: # exit on ESC
...
```

Ainsi qu'une petite fonction qui servira à lancer des fichiers audio dans la partie bonus :

Python

```
def recognize_image(img, database):
    print("******** PROCEDING FACIAL RECOGNITION ********")
    name, dmin = find_closest(img ,database)
    #Speech
    #t = Thread(target=say_hello, args=[name])
    #t.start()
    print("******** RESUME ANALYSIS ********")
    return name, True
```

## Faire parler Windows !

Trouver un nom, c'est bien, saluer cette personne, c'est encore mieux.

Pour démarrer le synthétiseur vocal, il suffit d'ajouter la ligne de code suivante :

Python

```
# Initialization of the voice
speaker = win32com.client.Dispatch("SAPI.SpVoice")
```

Ensuite on définit une fonction qui sera chargée de lire le texte

Python

```
def say_hello(name):
    txt = 'Bonjour '+name+", comment allez-vous ?"
    speaker.Speak(txt)
```

Et pour finir, lorsque l'on identifie une personne, on lui dit bonjour.

**Attention** : ce code doit être mis ailleurs si vous voulez utiliser l'une des améliorations précédentes !

**Attention** : en l'état, l'ordinateur va, pour chaque détection, empiler les demandes de salutation. Si la reconnaissance est plus rapide que le texte à lire, l'ordinateur risque de parler un moment… Evitez donc de lui faire dire bonjour trop souvent ! Vous pouvez par exemple mettre un décompte qui demande 20 identifications avant de parler…

```
    name, dmin = find_closest(img ,database)
    #Speech
    t = Thread(target=say_hello, args=[name])
    t.start()
    print("******** RESUME ANALYSIS ********")
    return name, True
```

## Faire parler son ordinateur, la deuxième solution

Nommé PyTTSx3, ce module propose d'utiliser les voix de l'ordinateur en mode offline.

Commençons par l'installer avec

Python
```
pip install pyttsx3
```

Pour écouter toutes les voix disponibles, il va falloir initialiser le module. On lui demande ensuite de prononcer un mot pour se faire une idée de l'accent, et il faudra noter l'ID de la voix choisie :

Python
```
def list_voices_windows():
    engine = pyttsx3.init();
    voices = engine.getProperty('voices')
    for voice in voices:
        print("id :",voice.id)
        engine.say('bonjour')
        engine.runAndWait()
```

Pour choisir une voix en particulier, il faut lancer cette commande (la méthode ci-dessus n'est donc utile que pour trouver la bonne clé à mettre)

Python
```
engine = pyttsx3.init();
# J'utilise Hortense car elle a un assez bon accent, mais rien ne vous empêche de choisir une
engine.setProperty('voice', "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens\TTS_M
```

Enfin, pour lire le texte, on remplace notre méthode say_hello du paragraphe d'avant par celle-ci :

Python
```
def say_hello(name):
    txt = 'Bonjour '+name+", comment allez-vous ?"
    engine.say(txt)
    engine.runAndWait()
```

## Histogram of Oriented Gradients HOG

Nous aurons besoin de deux librairies supplémentaires :

- scikit-image (skimage) qui possède une méthode HOG déjà programmée
- pillow (PIL) qui est la librairie de base pour manipuler des images (les charger notamment)

MS DOS
```
pip install scikit-image
pip install pillow
```

Puis le code repose sur la configuration de notre HOG (la taille des zones que l'on va considérer : 16×16, les directions de changement de couleur que l'on veut considérer : haut/bas/droite/gauche/diagonales)

Python
```
import cv2
from skimage.feature import hog
from PIL import Image

scale = 10
def generate_hog(filename):
    im = cv2.imread(filename)
    gr = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    image = 255-gr
    fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16), cells_per_block=(1,
    return hog_image

def save_hog(filename, hog_image):
    name=filename[:-4]+"__HOG.png"
    img = Image.fromarray(hog_image*scale).convert('L')
    img.save(name)

img_hog = generate_hog("images/lambert rosique.jpg")
save_hog("hog.png",img_hog)
```

## Dlib shape predictor : un exemple

Pour finir ce TP, voici une mise en place de la détection des 68 points clés du visage grâce à Dlib Shape Predictor, un algorithme avec un modèle pré-entraîné à disposition (à noter que des modèles existent également pour 5 points, 68 points et même davantage) !

*68 marqueurs placés par le dlib shape predictor (l'image provient de « this person does not exist », une IA qui génère des visages inventés)*

De plus, Dlib dispose de HOG, ce qui a le mérite de proposer une autre version du paragraphe précédent…

Enfin, veuillez noter qu'il y a toujours plusieurs manières

dépendre de la criticité de la détection dans vos choix (qui seront à faire aussi pour les phase 3 et 4 !)… En tout cas, vous trouverez plus d'informations sur dlib dans le tutoriel de PyImageSearch (dont le code utilisé ici provient).

Commençons par installer ces deux librairies :

MS DOS
```
pip install dlib
pip install imutils
```

Puis le code va être organisé de cette manière :

- Detector est le « hog » de dlib
- Predictor va charger un modèle déjà entraîné à placer les 68 marqueurs
- Ensuite, detector va trouver le visage dans l'image puis predictor va venir placer les marqueurs dans cette région
- Il ne nous restera plus qu'à afficher le rectangle, et les points !

Python
```
import dlib
import cv2
from imutils import face_utils

image = cv2.imread("images/thispersondoesnotexist.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

dots_size = 5

rects = detector(gray, 1)
# loop over the face detections
for (i, rect) in enumerate(rects):
    # determine the facial landmarks for the face region, then
    # convert the facial landmark (x, y)-coordinates to a NumPy
    # array
    shape = predictor(gray, rect)
    shape = face_utils.shape_to_np(shape)

    # convert dlib's rectangle to a OpenCV-style bounding box
    # [i.e., (x, y, w, h)], then draw the face bounding box
    (x, y, w, h) = face_utils.rect_to_bb(rect)
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

    # show the face number
    cv2.putText(image, "Face #{}".format(i + 1), (x - 10, y - 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

    # loop over the (x, y)-coordinates for the facial landmarks
    # and draw them on the image
    for (x, y) in shape:
        cv2.circle(image, (x, y), dots_size, (0, 255, 0), -1)
```

**Lambert R.**

*http://lambertrosique.fr*

Ingénieur d'Etudes et Data Scientist depuis plusieurs années, mes travaux et mon parcours scolaire (master en mathématiques fondamentales) m'ont amené aux abords de l'intelligence artificielle. Aujourd'hui j'écris des articles en data science, deep learning, big data et intelligence artificielle pour PenseeArtificielle.fr, dans le but de promouvoir et vulgariser les promesses d'avenir qu'offrent ces domaines de pointe.