Lab 5 - Reconnaissance faciale avec des bibliothèques Python prédéfinies

Introduction

Jusqu'à présent, nous avons utilisé l'apprentissage automatique pour résoudre des problèmes isolés qui n'ont qu'une seule étape : régression linéaire et polynomiale, détection de chiffres, codage automatique des chiffres et indication si une image contient un certain objet.

Tous ces problèmes peuvent être résolus en choisissant un algorithme d'apprentissage automatique, en introduisant des données et en obtenant le résultat.

La **reconnaissance faciale** est en réalité une série de plusieurs problèmes liés :

1. Tout d'abord, regardez une image et **trouvez tous les visages** qu'elle contient

2. Deuxièmement, **concentrez-vous sur chaque visage** et soyez capable de comprendre que même si un visage est tourné dans une direction étrange ou dans un mauvais éclairage, c'est toujours la même personne.

3. Troisièmement, être capable de choisir les **caractéristiques uniques** du visage que vous pouvez utiliser pour le distinguer des autres personnes, comme la taille des yeux, la longueur du visage, etc.

4. Enfin, **comparez les caractéristiques** uniques de ce visage à toutes les personnes que vous connaissez déjà pour déterminer le nom de la personne.

Le cerveau humain est programmé pour faire tout cela automatiquement.

Les ordinateurs ne sont pas capables de ce genre de généralisation de haut niveau (du moins pas encore…), nous devons donc leur apprendre à faire chaque étape de ce processus séparément.

Nous devons créer un pipeline dans lequel nous résolvons chaque étape de la reconnaissance faciale séparément et transmettons le résultat de l'étape actuelle à l'étape suivante. Autrement dit, nous allons enchaîner plusieurs algorithmes de machine learning :

5.1 Reconnaissance faciale — étape par étape

Abordons ce problème une étape à la fois. Pour chaque étape, nous découvrirons un algorithme d'apprentissage automatique différent. On ne va pas expliquer complètement chaque algorithme pour éviter que cela ne se transforme en livre, mais vous apprendrez les idées principales derrière chacun et vous apprendrez comment vous pouvez créer votre propre système de reconnaissance faciale en **Python** en utilisant `OpenFace` et `dlib`.

«`dlib` *is a general purpose cross-platform software library written in the programming language C++. Its design is heavily influenced by ideas from design by contract and component-based software engineering* »

5.1.1 Étape 1 : Trouver tous les visages

La première étape de notre pipeline est la détection des visages. Évidemment, nous devons localiser les visages sur une photo avant de pouvoir essayer de les distinguer !

Si vous avez utilisé une caméra au cours des 10 dernières années, vous avez probablement vu la détection des visages en action.

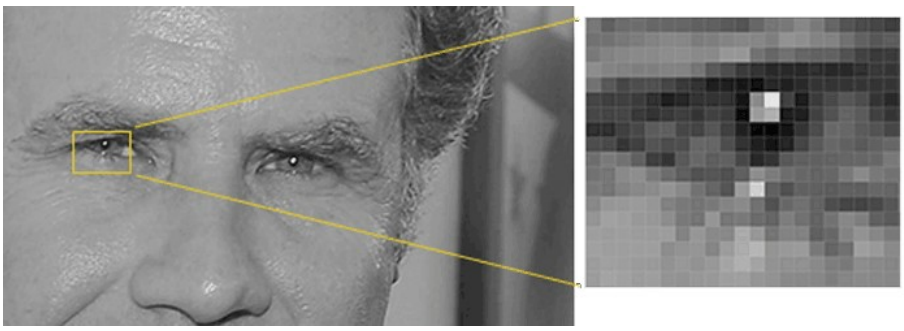
La détection des visages est une fonctionnalité intéressante pour les caméras. Lorsque l'appareil photo peut sélectionner automatiquement les visages, il peut s'assurer que tous les visages sont mis au point avant de prendre la photo. Mais nous l'utiliserons dans un but différent : trouver les zones de l'image que nous voulons transmettre à l'étape suivante de notre pipeline.

La détection des visages s'est généralisée au début des années 2000 lorsque Paul Viola et Michael Jones ont inventé un moyen de détecter les visages suffisamment rapide pour fonctionner sur des appareils photo bon marché. Cependant, des solutions beaucoup plus fiables existent maintenant. Nous allons utiliser une méthode inventée en 2005 appelée **Histogram of Oriented Gradients** - ou simplement **HOG** pour faire court.

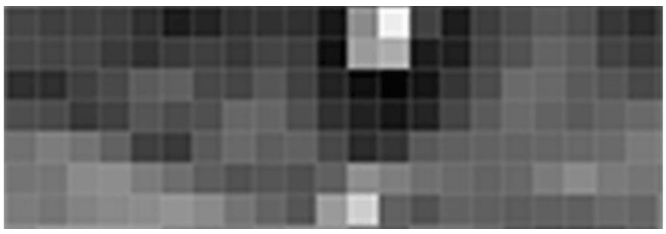
Pour trouver le nous commençons par faire nos images en noir et blanc.

Ensuite, nous examinerons chaque pixel de notre image un par un.

Pour chaque pixel, nous voulons regarder les pixels qui l'entourent directement :



Notre objectif est de déterminer à quel point le pixel actuel est sombre par rapport aux pixels qui l'entourent directement. Ensuite, nous voulons dessiner une flèche indiquant dans quelle direction l'image s'assombrit :



En ne regardant que ce pixel et les pixels qui le touchent, l'image s'assombrit vers le coin supérieur droit.

Si vous répétez ce processus pour chaque pixel de l'image, vous vous retrouvez avec chaque pixel remplacé par une flèche. Ces flèches sont appelées dégradés et elles montrent le flux du clair au foncé sur l'ensemble de l'image :

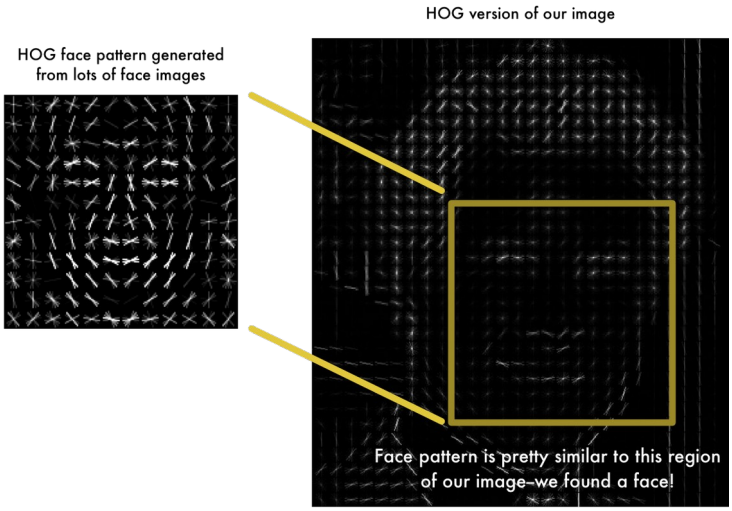
Cela peut sembler aléatoire, mais il y a une très bonne raison de remplacer les pixels par des dégradés. Si nous analysons directement les pixels, les images vraiment sombres et les images vraiment claires de la même personne auront des valeurs de pixels totalement différentes. Mais en ne considérant que la direction dans laquelle la luminosité change, les images vraiment sombres et les images vraiment lumineuses se retrouveront avec la même représentation exacte. Cela rend le problème beaucoup plus facile à résoudre!

Mais enregistrer le dégradé pour chaque pixel nous donne beaucoup trop de détails. Ce serait mieux si nous pouvions simplement voir le flux de base de luminosité/obscurité à un niveau supérieur afin que nous puissions voir le motif de base de l'image.

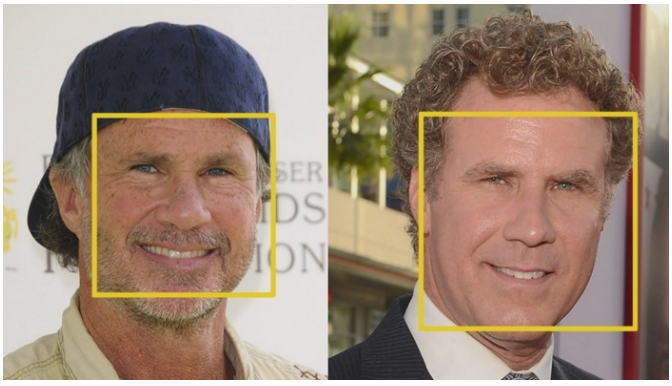
Pour ce faire, nous allons diviser l'image en petits carrés de 16x16 pixels chacun. Dans chaque carré, nous compterons combien de dégradés pointent dans chaque direction principale (combien pointent vers le haut, pointent vers le haut à droite, pointent vers la droite, etc…). Ensuite, nous remplacerons ce carré dans l'image par les directions des flèches qui étaient les plus fortes.

Le résultat final est que nous transformons l'image originale en une représentation très simple qui capture la structure de base d'un visage d'une manière simple :

Pour trouver des visages dans cette image HOG, tout ce que nous avons à faire est de trouver la partie de notre image qui ressemble le plus à un **motif HOG connu** qui a été extrait d'un tas d'autres visages d'entraînement :



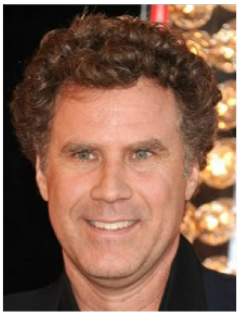
Pour trouver des visages dans cette image HOG, tout ce que nous avons à faire est de trouver la partie de notre image qui ressemble le plus à un motif HOG connu qui a été extrait d'un tas d'autres visages d'entraînement :

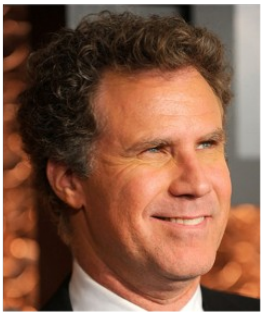


Si vous voulez essayer cette étape vous-même en utilisant **Python** et `dlib`, voici le code montrant comment générer et afficher des représentations HOG d'images.

5.1.2 Étape 2 : Pose et projection de visages

Nous avons isolé les visages à notre image. Mais maintenant, nous devons faire face au problème selon lequel les visages tournés dans des directions différentes semblent totalement différents pour un ordinateur :



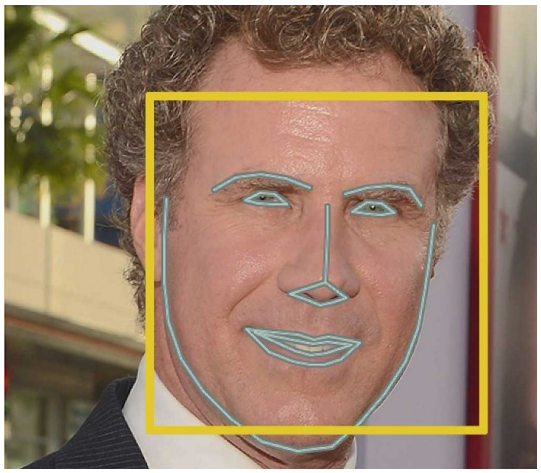


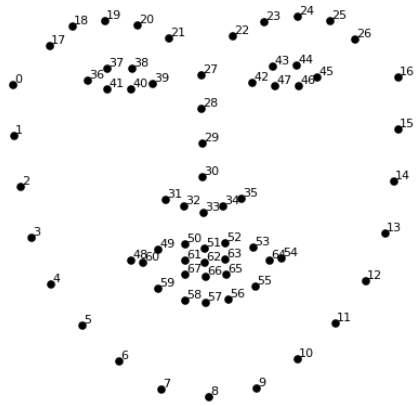
Les humains peuvent facilement reconnaître que les deux images sont de Will Ferrell, mais les ordinateurs verraient ces images comme deux personnes complètement différentes.

Pour tenir compte de cela, nous allons essayer de déformer chaque image de sorte que les yeux et les lèvres soient toujours au même emplacement dans l'image. Cela nous permettra de comparer beaucoup plus facilement les visages dans les prochaines étapes.

Pour ce faire, nous allons utiliser un algorithme appelé estimation de point de repère de visage. Il existe de nombreuses façons de le faire, mais nous allons utiliser l'approche inventée en 2014 par Vahid Kazemi et Joséphine Sullivan.

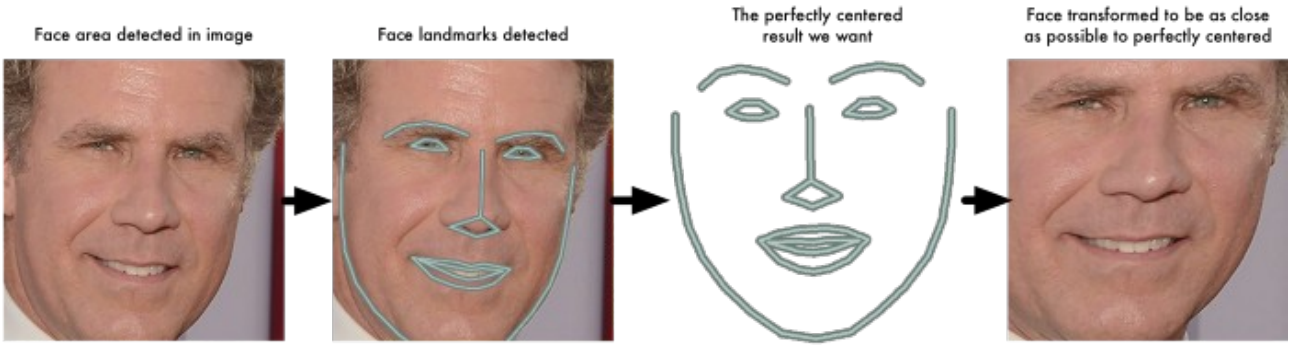
L'idée de base est que nous allons proposer 68 points spécifiques (appelés repères) qui existent sur chaque visage - le haut du menton, le bord extérieur de chaque œil, le bord intérieur de chaque sourcil, etc. Ensuite, nous entraînerons une machine algorithme d'apprentissage pour pouvoir retrouver ces 68 points spécifiques sur n'importe quel visage :





Voici le résultat de la localisation des 68 points de repère du visage sur notre image de test.

Maintenant que nous savons où se trouvent les yeux et la bouche, nous allons simplement faire pivoter, mettre à l'échelle et cisailler l'image afin que les yeux et la bouche soient centrés le mieux possible. Nous ne ferons pas de déformations 3D sophistiquées car cela introduirait des distorsions dans l'image. Nous n'allons utiliser que des transformations d'image de base comme la rotation et l'échelle qui préservent les lignes parallèles (appelées transformations affines) :



Maintenant, peu importe la façon dont le visage est tourné, nous sommes capables de centrer les yeux et la bouche à peu près dans la même position dans l'image. Cela rendra notre prochaine étape beaucoup plus précise.

Si vous voulez essayer cette étape vous-même en utilisant **Python** et `dlib`, voici le code pour trouver des repères de visage et voici le code pour transformer l'image en utilisant ces repères.

5.1.3 Étape 3 : Encodage des visages

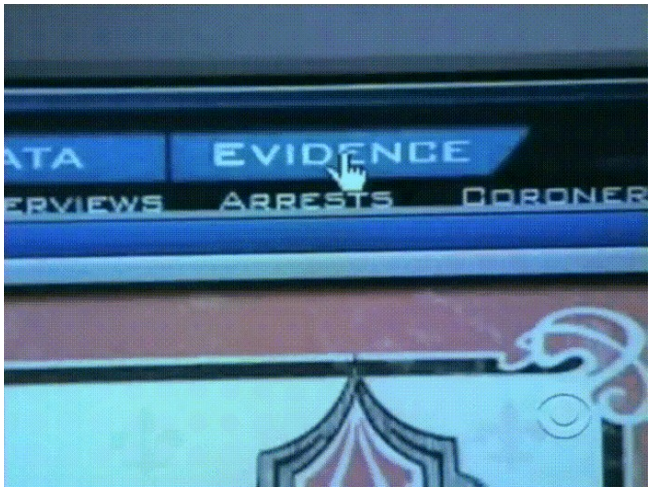
Nous en sommes maintenant au cœur du problème – comment distinguer les visages. C'est là que les choses deviennent vraiment intéressantes !

L'approche la plus simple de la reconnaissance faciale consiste à comparer directement le visage inconnu que nous avons trouvé à l'étape 2 avec toutes les photos que nous avons de personnes qui ont déjà été taguées.

Lorsque nous trouvons un visage précédemment tagué qui ressemble beaucoup à notre visage inconnu, il doit s'agir de la même personne , non?

Il y a en fait un énorme problème avec cette approche. Un site comme Facebook avec des milliards d'utilisateurs et un billion de photos ne peut pas parcourir tous les visages précédemment marqués pour les comparer à chaque image nouvellement téléchargée. Cela prendrait beaucoup trop de temps. Ils doivent être capables de reconnaître les visages en millisecondes, pas en heures.

Ce dont nous avons besoin, c'est d'un moyen d'extraire quelques mesures de base de chaque visage. Ensuite, nous pourrions mesurer notre visage inconnu de la même manière et trouver le visage connu avec les mesures les plus proches. Par exemple, nous pourrions mesurer la taille de chaque oreille, l'espacement entre les yeux, la longueur du nez, etc.



Le moyen le plus fiable de mesurer un visage

Quelles mesures devrions-nous collecter sur chaque visage pour construire notre base de données de visages connus ? Taille des oreilles ? Longueur du nez ? Couleur des yeux? Autre chose?

Il s'avère que les mesures qui semblent évidentes pour nous, les humains (comme la couleur des yeux) n'ont pas vraiment de sens pour un ordinateur qui examine les pixels individuels d'une image. Les chercheurs ont découvert que l'approche la plus précise consiste à laisser l'ordinateur calculer les mesures pour les collecter lui-même.

L'apprentissage en profondeur fait un meilleur travail que les humains pour déterminer quelles parties d'un visage sont importantes à mesurer.

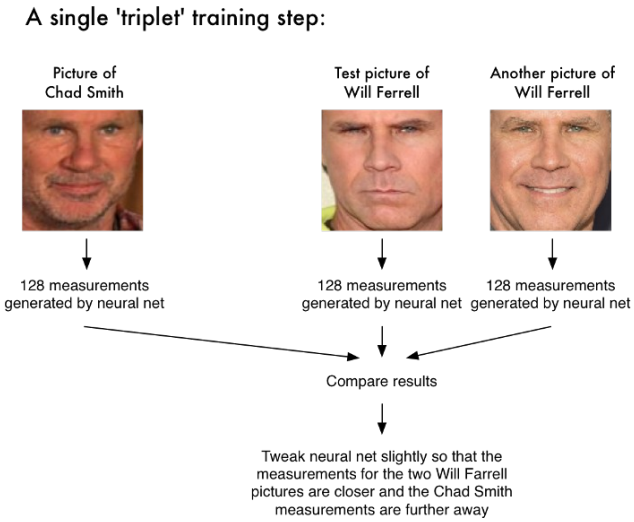
La solution est de former un **réseau de neurones à convolution profonde** (CNN). Mais au lieu d'entraîner le réseau à reconnaître les objets images comme nous l'avons fait la dernière fois, nous allons l'entraîner à générer **128 mesures pour chaque visage**.

Le processus de formation fonctionne en regardant 3 images de visage à la fois :

1. Chargez une image de visage d'entraînement d'une personne connue
2. Chargez une autre photo de la même personne connue
3. Chargez une photo d'une personne totalement différente

Ensuite, l'algorithme regarde les mesures qu'il génère actuellement pour chacune de ces 3 images.

Il modifie ensuite légèrement le réseau de neurones pour s'assurer que les mesures qu'il génère pour les images 1 et 2 sont légèrement plus proches tout en s'assurant que les mesures pour 2 et 3 sont légèrement plus éloignées :



Après avoir répété cette étape des millions de fois pour des millions d'images de milliers de personnes différentes, le réseau de neurones apprend à générer de manière fiable 128 mesures pour chaque personne. Dix images différentes de la même personne devraient donner à peu près les mêmes mesures.

Les personnes en apprentissage automatique appellent les 128 mesures de chaque face une intégration (**embedding**).

L'idée de réduire des **données brutes compliquées** comme une image en une **liste de nombres** générés par ordinateur revient beaucoup dans l'apprentissage automatique.

Encoder notre image de visage

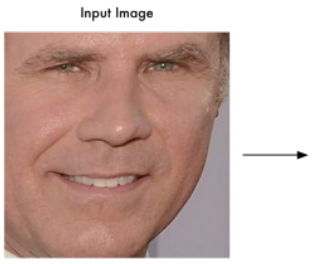
Ce processus de formation d'un réseau de neurones convolutifs pour produire des intégrations de visage nécessite beaucoup de données et de puissance de traitement. Même avec une carte GPU NVidia Telsa , il faut environ 24 heures de formation continue pour obtenir une bonne précision.

Mais une fois que le réseau a été formé, il peut générer des mesures pour n'importe quel visage, même ceux qu'il n'a jamais vu auparavant ! Cette étape n'est donc à faire qu'une seule fois. Heureusement pour nous, les gens d'**OpenFace** l'ont déjà fait et ils ont publié plusieurs réseaux formés que nous pouvons utiliser directement.

Donc, tout ce que nous devons faire nous-mêmes, c'est exécuter nos images de visage via leur réseau pré-entraîné pour obtenir les 128 mesures pour chaque visage.

Alors, quelles parties du visage ces 128 nombres mesurent-ils exactement ? Il s'avère que nous n'en avons aucune idée. Cela n'a pas vraiment d'importance pour nous. Tout ce qui nous importe, c'est que le réseau génère à peu près les mêmes nombres lorsque l'on regarde deux images différentes de la même personne.

Si vous souhaitez essayer cette étape vous-même, **OpenFace** fournit un script lua qui générera des intégrations de toutes les images dans un dossier et les écrira dans un fichier `csv`.



128 Measurements Generated from Image

0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.175216317717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.141315251588882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540404039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.125671759294778	-0.105685450136666	-0.127286553848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.123694673180568
0.046741496771574	0.006176188122481 1	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.008584329858422 3
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.006481129676103 6	0.21180312335491	-0.050584398210049
0.15245945751667	-0.165823280811 31	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.007277775555849 1	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.009367933496832 8	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.004863874055445 2	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.052281547337777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.009803973138332 4	0.037022035568953	0.11009479314089	0.116387886878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.004033768083900 2	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

Voici les mesures pour notre image de test.

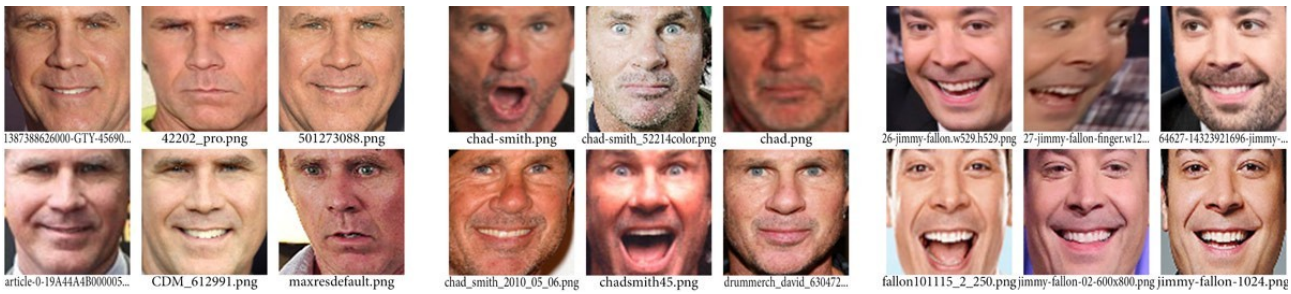
5.1.4 Étape 4 : Trouver le nom de la personne à partir de l'encodage

Cette dernière étape est en fait l'étape la plus simple de tout le processus. Tout ce que nous avons à faire est de trouver la personne dans notre base de données de personnes connues qui a **les mesures les plus proches** de notre image de test.

Vous pouvez le faire en utilisant n'importe quel **algorithme de classification d'apprentissage automatique** de base. Aucune astuce d'apprentissage en profondeur sophistiquée n'est nécessaire. Nous utiliserons un simple **classificateur SVM linéaire**, mais de nombreux algorithmes de classification pourraient fonctionner.

Tout ce que nous avons à faire est de former un classificateur qui peut prendre les mesures d'une nouvelle image de test et indiquer quelle personne connue est la correspondance la plus proche. L'exécution de ce classificateur prend des **millisecondes**. Le résultat du classificateur est le nom de la personne !

Essayons donc notre système. Tout d'abord, nous avons formé un classificateur avec les intégrations d'environ 20 images chacune de Will Ferrell, Chad Smith et Jimmy Falon :



Ensuite, nous pouvons exécuter le classificateur sur chaque image de la célèbre vidéo youtube de Will Ferrell et Chad Smith faisant semblant d'être l'un l'autre dans l'émission Jimmy Fallon :

5.2 Système de reconnaissance faciale avec Nvidia Jetson Nano et Python

5.2.1 Installation des bibliothèques Linux et Python requises pour la reconnaissance faciale

Une fois que vous avez terminé la configuration initiale de Linux/Ubuntu , vous devez installer plusieurs bibliothèques que nous utiliserons dans notre système de reconnaissance faciale.

Depuis le bureau Jetson Nano, ouvrez une fenêtre Terminal et exécutez les commandes suivantes. Chaque fois qu'il vous demande votre mot de passe, saisissez le même mot de passe que vous avez entré lors de la création de votre compte utilisateur :

Tout d'abord, nous utilisons **`apt`**, qui est l'outil d'installation de logiciel Linux standard que nous utiliserons pour installer les autres bibliothèques système dont nous avons besoin.

```
sudo apt install python-pip
```

Ensuite, nous installons quelques bibliothèques Linux dont nous avons besoin et qui ne sont pas pré-installées.

Nous installons **`dlib`** (version 19.15.0) (machine learning library) et **`face_recognition.`**

```
pip install dlib==19.15.0
pip install face_recognition
```

Cette commande **compilera** ces bibliothèques à partir du code source. Cela pourrait prendre **plus qu'une heure**.

Lorsque cela est enfin terminé, votre Jetson Nano est prêt à effectuer la reconnaissance faciale avec une **accélération GPU CUDA** complète.

Après l'installation du logiciel nous pouvons nous rendre dans le répertoire **`face_recognition/exemples`**.

```
benchmark.py
blink_detection.py
blur_faces_on_webcam.py
digital_makeup.py
face_distance.py
facerec_from_video_file.py
facerec_from_webcam_faster.py
facerec_from_webcam_multiprocessing.py
facerec_from_webcam.py
facerec_ipcamera_knn.py
face_recognition_knn.py
face_recognition_svm.py
facerec_on_raspberry_pi.py
facerec_on_raspberry_pi_Simplified_Chinese.py
find_faces_in_batches.py
find_faces_in_picture_cnn.py
find_faces_in_picture.py
find_facial_features_in_picture.py
identify_and_draw_boxes_on_faces.py
mqtt_eclipse.py
mqtt_loop.eclipse.py
mqtt_loop.py
mqtt_publish.py
mqtt_subscribe.py
recognize_faces_in_pictures.py
recvTS.py
sendTS.py
web_service_example.py
web_service_example_Simplified_Chinese.py
```

5.3 Premier exemple avec : `find_faces_in_picture.py` et `find_faces_in_picture_cnn.py`

5.3.1 `find_faces_in_picture.py`

Dans ce programme on cherche l'image correspondant au photo de la personne fourni dans la fonction :

```
image = face_recognition.load_image_file("biden.jpg")
```

Code complet :

```
from PIL import Image
import dlib
import face_recognition
import sys

# Load the jpg file into a numpy array
image = face_recognition.load_image_file("biden.jpg")

# Find all the faces in the image using the default HOG-based model.
# This method is fairly accurate, but not as accurate as the CNN model and not
GPU accelerated.
# See also: find_faces_in_picture_cnn.py

face_locations = face_recognition.face_locations(image)

print("I found {} face(s) in this photograph.".format(len(face_locations)))

for face_location in face_locations:

    # Print the location of each face in this image
    top, right, bottom, left = face_location
    print("A face is located at pixel location Top: {}, Left: {}, Bottom: {},
Right: {}".format(top, left, bottom, right))

    # You can access the actual face itself like this:
    face_image = image[top:bottom, left:right]
    pil_image = Image.fromarray(face_image)
    pil_image.show()
```

A faire :

1. Exécuter le programme ci-dessus

2. Remplacer le nom du fichier par `sys.argv[1]`

3. Lancer le programme avec son argument : `find_faces_in_picture.py two_people.jpg`

5.4 Deuxième exemple avec : `facerec_from_webcam_faster.py`

Dans cet exemple nous allons tester l'utilisation de la WebCam pour la reconnaissance faciale de personnes devant la WebCam.

Voici le code :

```
import dlib
import face_recognition
import cv2
import numpy as np

# This is a demo of running face recognition on live video from your webcam. It's a little more
complicated than the
# other example, but it includes some basic performance tweaks to make things run a lot faster:
#   1. Process each video frame at 1/4 resolution (though still display it at full resolution)
#   2. Only detect faces in every other frame of video.

# PLEASE NOTE: This example requires OpenCV (the `cv2` library) to be installed only to read from
your webcam.
```

```
# OpenCV is *not* required to use the face_recognition library. It's only required if you want to
run this
# specific demo. If you have trouble installing it, try any of the other demos that don't require it
instead.

# Get a reference to webcam #0 (the default one)
video_capture = cv2.VideoCapture(0)
video_capture.set(3,640)
video_capture.set(4,480)

# Load a sample picture and learn how to recognize it.
nicolas_image = face_recognition.load_image_file("nicolas.jpg")
nicolas_face_encoding = face_recognition.face_encodings(nicolas_image)[0]

# Load a second sample picture and learn how to recognize it.
maureen_image = face_recognition.load_image_file("maureen.jpg")
maureen_face_encoding = face_recognition.face_encodings(maureen_image)[0]

# Create arrays of known face encodings and their names
known_face_encodings = [
    nicolas_face_encoding,
    maureen_face_encoding
]
known_face_names = [
    "Nicolas",
    "Maureen"
]

# Initialize some variables
face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()

    # Resize frame of video to 1/4 size for faster face recognition processing
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # Convert the image from BGR color (which OpenCV uses) to RGB color (which face_recognition
uses)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Only process every other frame of video to save time

# Only process every other frame of video to save time
    if process_this_frame:
        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            name = "Unknown"

            # # If a match was found in known_face_encodings, just use the first one.
            # if True in matches:
            #     first_match_index = matches.index(True)
            #     name = known_face_names[first_match_index]

            # Or instead, use the known face with the smallest distance to the new face
            face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = known_face_names[best_match_index]

            face_names.append(name)

    process_this_frame = not process_this_frame


    # Display the results
    for (top, right, bottom, left), name in zip(face_locations, face_names):
        # Scale back up face locations since the frame we detected in was scaled to 1/4 size
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

        # Draw a box around the face
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

        # Draw a label with a name below the face
```

```
        cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

# here you can put the code to send the result - name to an IoT broker or server

    # Display the resulting image
    cv2.imshow('Video', frame)

    # Hit 'q' on the keyboard to quit!
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()
```

A faire :

1. Saisir votre image par l'application **Cheese** et l'enregistrer dans le même répertoire que votre programme (éventuellement vous pouvez créer un sous-répertoire pour y stocker l'ensemble des images à tester.

« **Cheese** est un logiciel qui vous permet de capturer des photos et des vidéos (son inclus) grâce à votre webcam avec éventuellement des effets spéciaux ».

2. Ajouter votre image dans le code :

```
# Load a sample picture and learn how to recognize it.
nicolas_image = face_recognition.load_image_file("nicolas.jpg")
nicolas_face_encoding = face_recognition.face_encodings(nicolas_image)[0]

# Load a second sample picture and learn how to recognize it.
maureen_image = face_recognition.load_image_file("maureen.jpg")
maureen_face_encoding = face_recognition.face_encodings(maureen_image)[0]

# Load a second sample picture and learn how to recognize it.
votre_image = face_recognition.load_image_file("votre.jpg")
votre_face_encoding = face_recognition.face_encodings(maureen_image)[0]

# Create arrays of known face encodings and their names
known_face_encodings = [
    nicolas_face_encoding,
    maureen_face_encoding,
    votre_face_encoding
]
known_face_names = [
    "Nicolas",
    "Maureen"
    "Votre_Image"
]
```

3. Préparer un code d'envoi de votre « label » sur un broker/serveur IoT (**MQTT**, **ThingSpeak**,..). Ce code doit être ajouté à l'endroit du commentaire :

```
# here you can put the code to send the result - name to an IoT broker or server
```

5.5 Premier projet - Reconnaissance faciale et envoi de données au broker MQTT (`mqtt.eclipseprojects.io`)

Le code suivant présente un exemple permettant d'envoyer (publish) et de recevoir (subscribe) un texte

```
import paho.mqtt.client as mqtt #import the client1
import time
def on_message(client, userdata, message):
    print("message received " ,str(message.payload.decode("utf-8")))
    print("message topic=",message.topic)
    print("message qos=",message.qos)
    print("message retain flag=",message.retain)

broker_address="emqx.io"
print("creating new instance")
client = mqtt.Client("jetson1") #create new instance
client.on_message=on_message #attach function to callback

print("connecting to broker")
client.connect("mqtt.eclipseprojects.io", 1883, 60)

print("Subscribing to topic","faces/name")
client.subscribe("faces/name")


client.loop_start() #start the loop
i=0
while True:
    print("Publishing message to topic","faces/name")
    client.publish("faces/name","bako"+str(i))
    i=i+1
    time.sleep(8) # wait

client.loop_stop() #stop the loop
```

Remarque:

Le broker MQTT **ne stocke pas des message** publiés il les seulement retransmet vers les postes suscrits au sujet (*topic*) donné.

A faire :

1. Installer le paquet **`paho`** :

```
pip3 install paho-mqtt
```

2. Installation d'une application client MQTT sur Android: - **MyMQTT**

3. Intégrer dans le programme **`facerec_from_webcam_faster.py`** le code nécessaire pour la publication d'un message – label de l'image reconnu sur le broker MQTT.

4. Tester l'ensemble avec l'application Android (ou un programme Python – subscriber sur votre ordinateur)

5.6 Deuxième projet - Reconnaissance faciale et envoi de données à ThingSpeak

5.6.1 Réception des données à partir de ThingSpeak

Les identifiants et les étiquettes/noms des visages détectés et reconnus peuvent être envoyés au serveur externe tel que **`ThingSpeak.com`**.

Dans notre cas, nous utilisons deux champs d'un canal un pour l'identifiant (**`field1`**)et un pour le nom du visage(**`field2`**). A noter que le nom du visage ne peut pas être présenté sur le tracé.

Cependant, nous pouvons écrire un programme simple qui lit les données des canaux **ThingSpeak**, y compris les chaînes de caractères dans le **`field2`**.

```
import urllib.request as urllib2
import json
import time

READ_API_KEY="20E9AQVFW7Z6..OM"  # votre clé de lecture
CHANNEL_ID=1538..4

print("libraries OK")

while True:
    TS = urllib2.urlopen("https://api.thingspeak.com/channels/%s/feeds/last.json?api_key=%s" \
                         % (CHANNEL_ID,READ_API_KEY))

    response = TS.read()
    data=json.loads(response)

    a = data['created_at']
    b = data['field1']
    c = data['field2']
    print(a,b,c)
    time.sleep(15)

    TS.close()
```

5.6.2 Envoi des données vers ThingSpeak

```
import urllib.request as urllib2
import json
import time

TSserver="https://api.thingspeak.com/update?"
Tswkey="api_key=YOX31M0EDKO0JATK&"  # votre clé d'ecriture

READ_API_KEY="20E9AQVFW7Z6XXOM"
CHANNEL_ID=1538804

print("libraries OK")

# sending to ThingSpeak, index (int) and name (char*)
def TSsend(ind,ind_name):
    TS=urllib2.urlopen(TSserver+TSwkey+"field1="+str(ind)+"&field2="+ind_name)
    print("\nYour message has successfully been sent!")
    time.sleep(15) # ThingSpeak.com free account limitation
    return b
i=0
while True:
   i=i+1
   iname="toto"
   print(i,iname)
   TSsend(i,iname)
   time.sleep(15)

TS.close()
```

Les programmes ci-dessus peuvent être exécutés sur votre Jetson Nano ou sur tout autre ordinateur.

5.6.3 Le programme de reconnaissance des visage et expédition vers TS

```
import face_recognition
import cv2
import numpy as np
from urllib2 import urlopen
import time

TSserver="https://api.thingspeak.com/update?"
TSwkey="api_key=YOX31M0EDKO0JATK&"

# This is a demo of running face recognition on live video from your webcam. It's a little more
complicated than the
# other example, but it includes some basic performance tweaks to make things run a lot faster:
#   1. Process each video frame at 1/4 resolution (though still display it at full resolution)
#   2. Only detect faces in every other frame of video.

# PLEASE NOTE: This example requires OpenCV (the `cv2` library) to be installed only to read from
your webcam.
# OpenCV is *not* required to use the face_recognition library. It's only required if you want to
run this
# specific demo. If you have trouble installing it, try any of the other demos that don't require it
instead.

# sending to ThingSpeak, index (int) and name (char*)
def TSsend(ind,ind_name):
#    b=urllib.request.urlopen('http://192.168.1.29:3000/update?
api_key=HEU64K3PGNWG36C4&field1=5.7&field2='+msg)
    b=urlopen(TSserver+TSwkey+"field1="+str(ind)+"&field2="+ind_name)
    print("\nYour message has successfully been sent!")
    time.sleep(15) # ThingSpeak.com free account limitation
    return b

# Get a reference to webcam #0 (the default one)
cap = cv2.VideoCapture(0)

def make_1080p():
    cap.set(3, 1920)
    cap.set(4, 1080)

def make_720p():
    cap.set(3, 1280)
    cap.set(4, 720)

def make_480p():
    cap.set(3, 640)
    cap.set(4, 480)

def change_res(width, height):
    cap.set(3, width)
    cap.set(4, height)

#make_720p()
#change_res(1280, 720)

make_480p()
change_res(640, 480)

# Load a sample picture and learn how to recognize it.
obama_image = face_recognition.load_image_file("obama.jpg")
obama_face_encoding = face_recognition.face_encodings(obama_image)[0]

# Load a second sample picture and learn how to recognize it.
biden_image = face_recognition.load_image_file("biden.jpg")
biden_face_encoding = face_recognition.face_encodings(biden_image)[0]

# Load a second sample picture and learn how to recognize it.
cathy_image = face_recognition.load_image_file("cathy.jpg")
cathy_face_encoding = face_recognition.face_encodings(cathy_image)[0]

# Load a second sample picture and learn how to recognize it.
bako_image = face_recognition.load_image_file("bako.jpg")
bako_face_encoding = face_recognition.face_encodings(bako_image)[0]

# Create arrays of known face encodings and their names
known_face_encodings = [
    obama_face_encoding,
    bako_face_encoding,
    cathy_face_encoding,
    biden_face_encoding
]
known_face_names = [
    "Barack Obama",
    "Bako",
    "Cathy",
    "Joe Biden"
]
```

```
# Initialize some variables
face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

while True:
    # Grab a single frame of video
    ret, frame = cap.read()

    # Resize frame of video to 1/4 size for faster face recognition processing
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # Convert the image from BGR color (which OpenCV uses) to RGB color (which face_recognition
uses)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Only process every other frame of video to save time
    if process_this_frame:
        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            name = "Unknown"

            # # If a match was found in known_face_encodings, just use the first one.
            # if True in matches:
            #     first_match_index = matches.index(True)
            #     name = known_face_names[first_match_index]

            # Or instead, use the known face with the smallest distance to the new face
            face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = known_face_names[best_match_index]

            face_names.append(name)

    process_this_frame = not process_this_frame

   # Display the results
    for (top, right, bottom, left), name in zip(face_locations, face_names):
        # Scale back up face locations since the frame we detected in was scaled to 1/4 size
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

        # Draw a box around the face
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

        # Draw a label with a name below the face
        cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)
        TSsend(best_match_index,name)

    # Display the resulting image
    cv2.imshow('Video', frame)

    # Hit 'q' on the keyboard to quit!
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release handle to the webcam
cap.release()
cv2.destroyAllWindows()
```

A faire:

1. Codez les programmes ci dessus sur votre Jetson Nano.
2. Saisissez avec la WEBCAM vos images de visage (les images restent sur la carte SD) dans le répertoire ou se trouve votre programme Python.
3. Modifiez le code pour intégrer l'encodage de vos images.
4. Modifiez les paramètres de votre canal ThingSpeak : lancer le programme de reconnaissance et d'envoi
5. Préparez le code lecteur à partir de ThingSpeak.
6. Lancez le lecteur.

5.6.4 Réception de données à partir de ThingSpeak sur une carte IoT

Dans cet exercice vous allez utiliser un kit IoT de **SmartComputerLab**. Ce kit intègre une carte avec un **SoC IoT - ESP32** et un écran **OLED**.

La carte ESP32 communique avec le serveur ThingSpeak par un lien WiFi. Vous pouvez utiliser votre smartphone avec un point d'accès soft , par exemple PhoneAP et un mot de passe, par exemple smartcomputerlab.

Sur la Jetson Nano vous trouvez IDE Arduino préinstallé prêt à programmer et charger votre carte ESP32.

Voici le code :

```
#include <WiFi.h>
#include "ThingSpeak.h" // always include thingspeak header file after other header files and custom
macros
#include <Wire.h>
#include "SSD1306Wire.h"

WiFiClient  client;

const char* ssid = "PhoneAP";
const char* pass = "smartcomputerlab";

// here put your channell number and read key
unsigned long ChannelNumber = 153..04;
const char *readAPIKey="20E9AQVFW..6XXOM";

int statusCode = 0;
int field[8] = {1,2,3,4,5,6,7,8};

SSD1306Wire display(0x3c, 12, 14);   // ADDRESS, SDA, SCL  -  SDA and SCL usually populate
automatically based on your board's pins_arduino.h

void display_SSD1306(int d1,String d2,String d3)
{
  char buff[64],buff1[32],buff2[32];
  d2.toCharArray(buff1,32);  d3.toCharArray(buff2,32);
  display.init();
  //display.flipScreenVertically();
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_16);
  memset(buff,0x00,64); sprintf(buff,"Face ID: %d",d1);
  display.drawString(0, 0, buff);
  memset(buff,0x00,64); sprintf(buff,"%s",buff1);
  display.drawString(0, 16, buff);
  display.setFont(ArialMT_Plain_10);
  memset(buff,0x00,64); sprintf(buff,"%s",buff2);
  display.drawString(0, 36, buff);
  display.drawString(20, 52, "SmartComputerLab");
  display.display();
}

void setup() {
  Serial.begin(9600);
  while(!Serial); delay(200);
  Wire.begin(12,14);
  Serial.println("Wire started");
  ThingSpeak.begin(client);  // Initialize ThingSpeak
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to the WiFi network\n");
}



void loop()
{
  statusCode = ThingSpeak.readMultipleFields(ChannelNumber,"20E9AQVFW7Z6XXOM");
  if(statusCode == 200)
    {
    int faceID = ThingSpeak.getFieldAsInt(field[0]); // Field 1
    String faceName = ThingSpeak.getFieldAsString(field[1]); // Field 2
    String createdAt = ThingSpeak.getCreatedAt();
    Serial.println("faceID: " + String(faceID));
    Serial.println("faceName: " + faceName);
    Serial.println("Created at: " + createdAt);
    createdAt.replace("T","   ");createdAt.replace("Z"," ");
    display_SSD1306(faceID,faceName,createdAt);
    }
```

```
  else
    {
    Serial.println("Problem reading channel. HTTP error code " + String(statusCode));
    }
  Serial.println();

  delay(15000); // no need to fetch too often
}
```

Table of Contents