2020

Rapport de projet du module de traitement d'images

MEISSA BIRIMA COULY MBAYE
ABDOURAHAMANE HASSANE WANKOYE

UCAD\ESP\DGI\M2GLSI

TABLE DES MATIERES

TABLE DES FIGURES

Introduction

Un système de reconnaissance faciale est une application logicielle visant à reconnaître une personne grâce à son visage de manière automatique. C'est un domaine de la vision par ordinateur consistant à reconnaître automatiquement une personne à partir d'une image de son visage. Il s'agit d'un sujet particulièrement étudié en vision par ordinateur, avec de très nombreuses publications et brevets, et des conférences spécialisées.

La reconnaissance de visage a de nombreuses applications en vidéo-surveillance, biométrie, robotique, indexation d'images et de vidéos, recherche d'images par le contenu, etc. Ces systèmes sont généralement utilisés à des fins de sécurité pour déverrouiller ordinateur/mobile/console, mais aussi en domotique. Ils sont appréciés car considérés comme peu invasifs, en comparaison avec les autres systèmes biométriques (empreintes digitales, reconnaissance de l'iris…). Le fonctionnement de ces systèmes se base sur une ou plusieurs caméras pour reconnaître l'utilisateur.

Ils peuvent également être utilisés afin de faciliter la vie de l'utilisateur, comme le font par exemple certains réseaux sociaux sur internet (Facebook, Google+) ou certaines applications mobiles (NameTag, FaceRec) pour identifier des visages sur des images. Ces systèmes se basent alors sur des photos/vidéos d'une ou plusieurs personnes.

Il faut toutefois veiller à ne pas confondre la détection de visage, qui consiste à repérer qu'un visage est présent sur une image, et la reconnaissance faciale, qui consiste à reconnaître quelqu'un depuis une image ou une vidéo. La reconnaissance de visage fait partie du domaine du traitement du signal.

Les systèmes de reconnaissance faciale sont de plus en plus présents au quotidien. C'est la raison pour laquelle, parmi les projets proposés, nous avons opté pour l'implémentation d'un logiciel de reconnaissance faciale permettant de détecter le visage d’une personne.

I. Présentation du projet

Étant donné que les systèmes de reconnaissance faciale sont omniprésents dans notre quotidien (smartphones, appareil photo, etc.), nous avons opté pour le projet consistant à réaliser un tel système.

En effet, la réalisation de ce projet consiste à concevoir un logiciel de reconnaissance faciale permettant de détecter le visage d'une personne à partir des images d'une vidéo surveillance pour des raisons de sécurité ou dans le cadre d'une recherche de personne déclarée disparue par exemple.

Afin de mener à bien ce projet, plusieurs outils sont disponibles comportant chacun ses caractéristiques ou spécificités.

II. Présentation des outils utilisés

II.1 OpenCV

Figure 1 : Logo OpenCV

OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. La société de robotique Willow Garage et la société ItSeez se sont succédé au support de cette bibliothèque. Depuis 2016 et le rachat d'ItSeez par Intel, le support est de nouveau assuré par Intel.

Cette bibliothèque est distribuée sous licence BSD.

La bibliothèque OpenCV met à disposition de nombreuses fonctionnalités très diversifiées permettant de créer des programmes en partant des données brutes pour aller jusqu'à la création d'interfaces graphiques basiques.

La raison principale pour laquelle nous avons opté pour cet outil est sa faculté à pouvoir effectuer diverses opérations sur les images et vidéos mais également la disponibilité de certains algorithmes classiques d'apprentissage.

Traitement d'images	Traitement vidéo
- Lecture, écriture et affichage d'images - Calcul de l'histogramme des niveaux de gris ou d'histogrammes couleurs - Lissage, filtrage - Seuillage d'image (méthode d'Otsu, seuillage adaptatif) - Segmentation (composantes connexes, GrabCut) ; - Morphologie mathématique	- Lecture, écriture et affichage d'une vidéo (depuis un fichier ou une caméra) - Détection de droites, de segment et de cercles par Transformée de Hough - Détection de visages par la méthode de Viola et Jones - Cascade de classifieurs boostés - Triangulation de Delaunay - Diagramme de Voronoi - Enveloppe convexe - Etc.

Figure 2 : Quelques fonctionnalités de traitements d'image et vidéo d'OpenCV

Algorithmes disponibles
- K-means - AdaBoost et divers algorithmes de boosting - Réseau de neurones artificiels - Séparateur à vaste marge - Estimateur (statistique) - Les arbres de décision et les forêts aléatoires

Figure 3 : Algorithmes d'apprentissage disponible sous OpenCV

II.2 Python

Figure 4 : Logo Python

Python est un langage de programmation interprété, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

Le langage Python est placé sous une licence libre proche de la licence BSD et fonctionne sur la plupart des plates-formes informatiques, des smartphones aux ordinateurs centraux9, de Windows à Unix avec notamment GNU/Linux en passant par MacOs, ou encore Android, iOS, et peut aussi être traduit en Java ou .NET. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

Il est également apprécié par certains pédagogues qui y trouvent un langage où la syntaxe, clairement séparée des mécanismes de bas niveau, permet une initiation aisée aux concepts de base de la programmation.

Python est un langage qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées.

Cependant, Python est monté plus en popularité ces dernières années grâce à l'intelligence artificielle et ses multiples branches (Machine Learning, Deep Learning, computer vision, etc.).

III.Mise en œuvre du projet

III.1 Installation de Python et des dépendances

Pour ce projet, nous allons utiliser Python 3, car il s'agit d'un langage parfaitement adapté à nos besoins et aussi parce que c'est obligatoire, qui dispose d'outils pour le traitement d'images, de vidéos (caméra) ainsi que l'intelligence artificielle et ses différentes branches.

Afin d'installer Python 3, il suffit de se rendre sur le site officielle de Python et de télécharger ce dernier.

Lien : https://www.python.org/downloads/

A la fin de l'installation, vous pouvez vérifier que Python est bien disponible en ligne de commande en affichant par exemple sa version comme l'illustre la figure suivante :

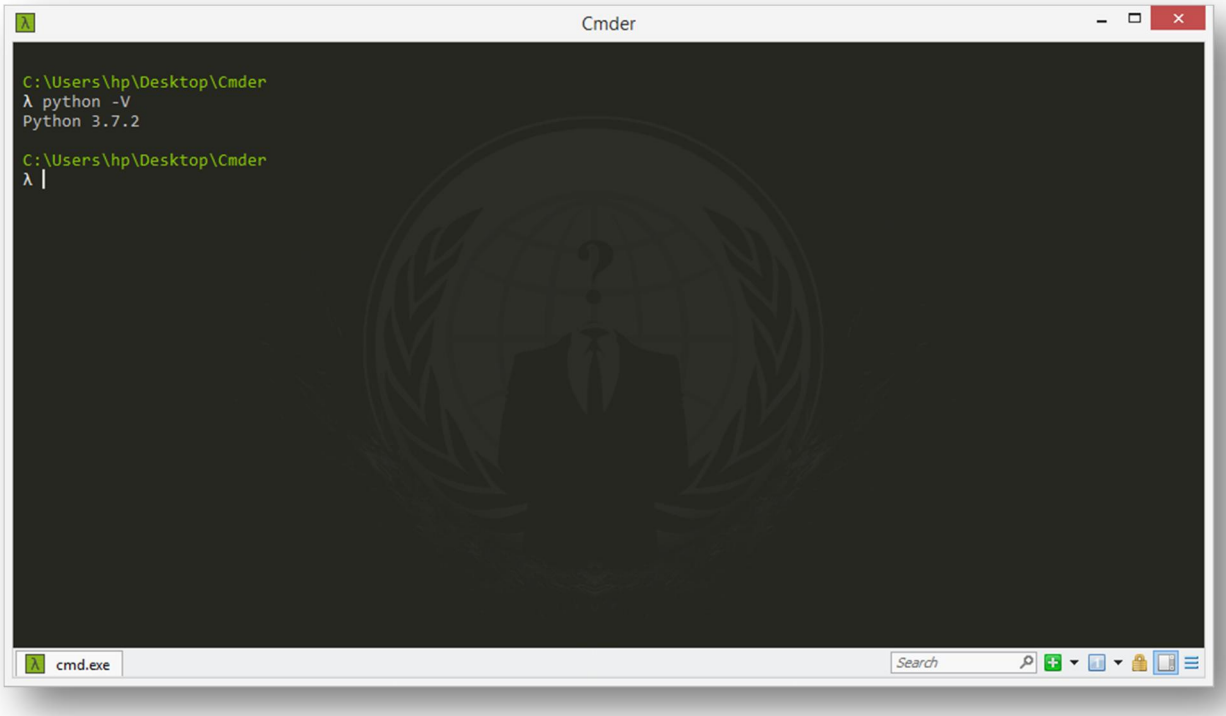

Figure 5 : Vérification de la disponibilité de Python

Maintenant que nous avons Python, on va avoir besoin des librairies suivantes :

- **OpenCV :** qui est dédiée au traitement d'images et vidéos

- **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

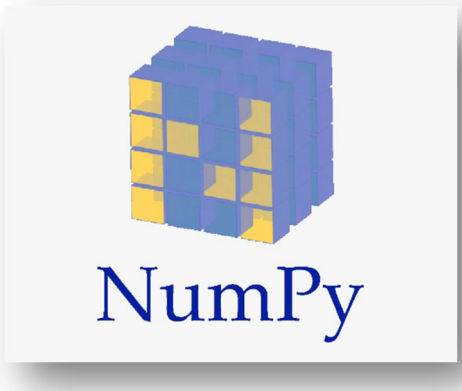

Figure 7 : Logo Numpy

- **Scikit-learn :** est une bibliothèque libre Python destinée à l'apprentissage automatique également appelé Machine Learning.

Figure 6 : Logo scikit-learn

- **Pickle :** module (interne de python) implémente des protocoles binaires pour sérialiser et dé-sérialiser une structure d'objet Python.
- **Imutils :** une série de fonctions pratiques pour rendre les fonctions de traitement d'images de base telles que la translation, la rotation, le redimensionnement, squelettisation, et l'affichage des images Matplotlib plus facile avec OpenCV.

Afin d'installer ces dépendances nécessaires à notre projet, nous allons un utilitaire de Python appelé « **pip** ». « pip » est ce qu'on appelle un gestionnaire de dépendances (ou packages) et va nous permettre de télécharger et d'installer facilement des packages.

Pour utiliser ce gestionnaire, il suffit de saisir au niveau de la console la commande suivante :

pip install <nom_du_package>

On peut alors procéder comme suit afin d'installer tous les dépendances nécessaires :

pip install imutils opencv-python numpy scikit-learn

III.2 Fonctionnement

Pour construire notre système de reconnaissance des visages, nous allons d'abord effectuer la détection des visages, extraire les encastrements de chaque visage grâce à un apprentissage approfondi, former un modèle de reconnaissance des visages sur les encastrements, puis enfin reconnaître les visages dans les images et les flux vidéo avec OpenCV.

- **Comment fonctionne la reconnaissance faciale d'OpenCV ?**

Bien que nous ayons utilisé OpenCV pour faciliter la reconnaissance des visages, OpenCV lui-même n'était pas responsable de l'identification des visages.

Dans le didacticiel d'aujourd'hui, nous apprendrons comment appliquer ensemble l'apprentissage en profondeur et OpenCV (sans autre bibliothèque que scikit-learn) pour:

1. Détecter les visages
2. Calculer des incorporations de visage à 128 jours pour quantifier un visage
3. Former une machine à vecteur de support (SVM) sur le dessus des plongements
4. Reconnaître les visages dans les images et les flux vidéo

Toutes ces tâches seront accomplies avec OpenCV, ce qui nous permettra d'obtenir un pipeline de reconnaissance faciale OpenCV « pur ».

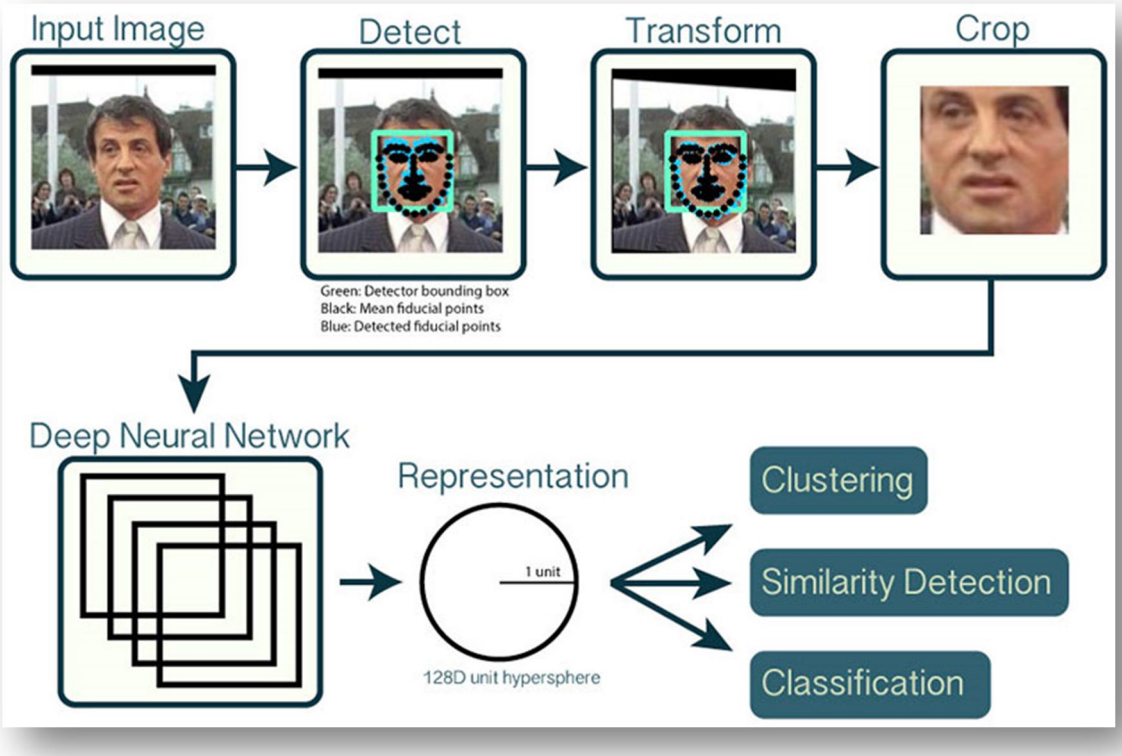

Figure 8 : Schéma récapitulatif du fonctionnement d'OpenCV

Afin de construire notre pipeline de reconnaissance faciale OpenCV, nous appliquerons le deep learning en deux étapes clés:

1. Pour appliquer la *détection de visage*, qui détecte la *présence* et l'emplacement d'un visage dans une image, mais ne l'identifie pas;
2. Pour extraire les vecteurs de caractéristiques à 128 j (appelés « plongements ») qui *quantifient* chaque face dans une image.

III.3 Structure et architecture du projet

```
$ tree --dirsfirst
.
├── dataset
│   ├── adrian [6 images]
│   ├── trisha [6 images]
│   └── unknown [6 images]
├── images
│   ├── adrian.jpg
│   ├── patrick_bateman.jpg
│   └── trisha_adrian.jpg
├── face_detection_model
│   ├── deploy.prototxt
│   └── res10_300x300_ssd_iter_140000.caffemodel
├── output
│   ├── embeddings.pickle
│   ├── le.pickle
│   └── recognizer.pickle
├── extract_embeddings.py
├── openface_nn4.small2.v1.t7
├── train_model.py
├── recognize.py
└── recognize_video.py
```

Notre projet a quatre répertoires dans le dossier racine:

- jeu de données (dataset) : contient nos images de visage organisées en sous-dossiers par nom.
- images : Contient trois images de test que nous utiliserons pour vérifier le fonctionnement de notre modèle.
- face_detection_model : contient un modèle d'apprentissage en profondeur Caffe pré-formé fourni par OpenCV pour *détecter les* visages. Ce modèle *détecte* et *localise les* visages dans une image.
- output : contient mes fichiers de cornichons de sortie. Si vous travaillez avec votre propre jeu de données, vous pouvez également stocker vos fichiers de sortie ici. Les fichiers de sortie incluent:

- embeddings.pickle: un fichier d'enregistrements faciaux sérialisé. Les incorporations ont été calculées pour chaque face du jeu de données et sont stockées dans ce fichier.
- le.pickle : notre encodeur d'étiquettes. Contient les étiquettes de nom des personnes que notre modèle peut reconnaître.
- recognizer.pickle : Notre modèle de machine à vecteur de support linéaire (SVM). Il s'agit d'un modèle d'apprentissage automatique plutôt que d'un modèle d'apprentissage en profondeur et il est chargé de *reconnaître* réellement *les* visages.

Résumons les cinq fichiers du répertoire racine :

- extract_embeddings.py : Nous examinerons ce fichier à l' **étape 1** qui est responsable de l'utilisation d'un extracteur de fonctionnalités d'apprentissage en profondeur pour générer un vecteur 128-D décrivant un visage. Tous les visages de notre ensemble de données seront passés à travers le réseau neuronal pour générer des plongements.
- openface_nn4 .small2 .v1 .t7 : Un modèle d'apprentissage en profondeur Torch qui produit les plongements faciaux 128-D. Nous utiliserons ce modèle d'apprentissage en profondeur dans les **étapes # 1, # 2 et # 3**ainsi que dans la section **Bonus**.
- train_model .py : Notre modèle SVM linéaire sera formé par ce script à l'étape **2**. Nous allons *détecter les* visages, *extraire les* plongements et *adapter* notre modèle SVM aux données des plongements.
- reconnaître.py : à l'étape **3** et nous allons reconnaître les visages dans les images. Nous allons détecter les visages, extraire les plongements et *interroger* notre modèle SVM pour déterminer qui se trouve dans une image. Nous allons dessiner des boîtes autour des visages et annoter chaque boîte avec un nom.
- reconnaissent_vidéo.py : Notre section **Bonus** décrit comment reconnaître qui se trouve dans les images d'un flux vidéo comme nous l'avons fait à l'étape # **3** sur les images statiques.

Passons à la première étape !

IV. Implémentation

IV.1 Etape 1 : Extraction des incorporations du jeu de données de face

On dispose du code suivant au niveau du fichier **extract_embeddings.py** :

```
6  from imutils import paths
7  import numpy as np
8  import argparse
9  import imutils
10 import pickle
11 import cv2
12 import os
13
14 # construct the argument parser and parse the arguments
15 ap = argparse.ArgumentParser()
16 ap.add_argument("-i", "--dataset", required=True,
17     help="path to input directory of faces + images")
18 ap.add_argument("-e", "--embeddings", required=True,
19     help="path to output serialized db of facial embeddings")
20 ap.add_argument("-d", "--detector", required=True,
21     help="path to OpenCV's deep learning face detector")
22 ap.add_argument("-m", "--embedding-model", required=True,
23     help="path to OpenCV's deep learning face embedding model")
24 ap.add_argument("-c", "--confidence", type=float, default=0.5,
25     help="minimum probability to filter weak detections")
26 args = vars(ap.parse_args())
27
```

Figure 9 : Extrait de code du fichier

Nous importons nos packages requis sur les **lignes 6-12,** Ensuite, nous traitons nos arguments de ligne de commande.

Maintenant que nous avons importé nos packages et nos arguments de ligne de commande analysés, on peut charger le détecteur de visage et l'embarquer à partir du disque ensuite nous prenons nos chemins d'image et effectuons des initialisations :

```
# load our serialized face detector from disk
print("[INFO] loading face detector...")
protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
modelPath = os.path.sep.join([args["detector"],
    "res10_300x300_ssd_iter_140000.caffemodel"])
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# load our serialized face embedding model from disk
print("[INFO] loading face recognizer...")
embedder = cv2.dnn.readNetFromTorch(args["embedding_model"])

# grab the paths to the input images in our dataset
print("[INFO] quantifying faces...")
imagePaths = list(paths.list_images(args["dataset"]))

# initialize our lists of extracted facial embeddings and
# corresponding people names
knownEmbeddings = []
knownNames = []

# initialize the total number of faces processed
total = 0
```

Figure 10 : Initialisations des paramètres

Commençons à boucler sur les chemins de l'image - cette boucle sera chargée d'extraire les plongements des visages trouvés dans chaque image .De là, nous détectons les visages dans l'image en passant l'image Blob (construit) à travers le réseau de détecteurs.

Nous continuons ce processus de bouclage sur les images, de détection des visages et d'extraction des incorporations de visages pour chaque image de notre ensemble de données.

Tout ce qui reste à la fin de la boucle est de vider les données sur le disque :

```python
for (i, imagePath) in enumerate(imagePaths):
    # extract the person name from the image path
    print("[INFO] processing image {}/{}".format(i + 1,
        len(imagePaths)))
    name = imagePath.split(os.path.sep)[-2]

    # load the image, resize it to have a width of 600 pixels (while
    # maintaining the aspect ratio), and then grab the image
    # dimensions
    image = cv2.imread(imagePath)
    image = imutils.resize(image, width=600)
    (h, w) = image.shape[:2]

    # construct a blob from the image
    imageBlob = cv2.dnn.blobFromImage(
        cv2.resize(image, (300, 300)), 1.0, (300, 300),
        (104.0, 177.0, 123.0), swapRB=False, crop=False)

    # apply OpenCV's deep learning-based face detector to localize
    # faces in the input image
    detector.setInput(imageBlob)
    detections = detector.forward()

    # ensure at least one face was found
    if len(detections) > 0:
        # we're making the assumption that each image has only ONE
        # face, so find the bounding box with the largest probability
        i = np.argmax(detections[0, 0, :, 2])
        confidence = detections[0, 0, i, 2]
```

Figure 11 : Bouclage, détection et extraction des visages

```python
        if confidence > args["confidence"]:
            # compute the (x, y)-coordinates of the bounding box for
            # the face
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # extract the face ROI and grab the ROI dimensions
            face = image[startY:endY, startX:endX]
            (fH, fW) = face.shape[:2]

            # ensure the face width and height are sufficiently large
            if fW < 20 or fH < 20:
                continue

            # construct a blob for the face ROI, then pass the blob
            # through our face embedding model to obtain the 128-d
            # quantification of the face
            faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
                (96, 96), (0, 0, 0), swapRB=True, crop=False)
            embedder.setInput(faceBlob)
            vec = embedder.forward()

            # add the name of the person + corresponding face
            # embedding to their respective lists
            knownNames.append(name)
            knownEmbeddings.append(vec.flatten())
            total += 1

# dump the facial embeddings + names to disk
print("[INFO] serializing {} encodings...".format(total))
data = {"embeddings": knownEmbeddings, "names": knownNames}
f = open(args["embeddings"], "wb")
f.write(pickle.dumps(data))
```

Figure 12 : Suite de la détection de visages

IV.2 Etape 2 : Formation du modèle de reconnaissance faciale

À ce stade, nous avons une BD qui contient les visages - mais comment pouvons-nous réellement reconnaître une personne sur la base de ces intégrations?

La réponse est que nous devons former un modèle d'apprentissage machine « standard » (tel qu'un SVM, un classificateur k-NN, une forêt aléatoire, etc.)

Dans le fichier **train_model.py**

```
6  from sklearn.preprocessing import LabelEncoder
7  from sklearn.svm import SVC
8  import argparse
9  import pickle
10
11 # construct the argument parser and parse the arguments
12 ap = argparse.ArgumentParser()
13 ap.add_argument("-e", "--embeddings", required=True,
14     help="path to serialized db of facial embeddings")
15 ap.add_argument("-r", "--recognizer", required=True,
16     help="path to output model trained to recognize faces")
17 ap.add_argument("-l", "--le", required=True,
18     help="path to output label encoder")
19 args = vars(ap.parse_args())
20
21 # load the face embeddings
22 print("[INFO] loading face embeddings...")
23 data = pickle.loads(open(args["embeddings"], "rb").read())
24
25 # encode the labels
26 print("[INFO] encoding labels...")
27 le = LabelEncoder()
28 labels = le.fit_transform(data["names"])
29
30 # train the model used to accept the 128-d embeddings of the face and
31 # then produce the actual face recognition
32 print("[INFO] training model...")
33 recognizer = SVC(C=1.0, kernel="linear", probability=True)
34 recognizer.fit(data["embeddings"], labels)
35
36 # write the actual face recognition model to disk
37 f = open(args["recognizer"], "wb")
38 f.write(pickle.dumps(recognizer))
39 f.close()
40
41 # write the label encoder to disk
42 f = open(args["le"], "wb")
```

Figure 13 : Extrait de code du fichier d'entrainement du modèle

IV.3 Etape 3 : Reconnaissance des visages avec OpenCV

Nous sommes maintenant prêts à effectuer la reconnaissance faciale avec OpenCV !

```python
# import the necessary packages
import numpy as np
import argparse
import imutils
import pickle
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-d", "--detector", required=True,
    help="path to OpenCV's deep learning face detector")
ap.add_argument("-m", "--embedding-model", required=True,
    help="path to OpenCV's deep learning face embedding model")
ap.add_argument("-r", "--recognizer", required=True,
    help="path to model trained to recognize faces")
ap.add_argument("-l", "--le", required=True,
    help="path to label encoder")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# load our serialized face detector from disk
print("[INFO] loading face detector...")
protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
modelPath = os.path.sep.join([args["detector"],
    "res10_300x300_ssd_iter_140000.caffemodel"])
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# load our serialized face embedding model from disk
print("[INFO] loading face recognizer...")
embedder = cv2.dnn.readNetFromTorch(args["embedding_model"])
```

Figure 14 : Extrait de code de l'entrée du projet

```
        # compute the (x, y)-coordinates of the bounding box for the
        # face
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        # extract the face ROI
        face = image[startY:endY, startX:endX]
        (fH, fW) = face.shape[:2]

        # ensure the face width and height are sufficiently large
        if fW < 20 or fH < 20:
            continue

        # construct a blob for the face ROI, then pass the blob
        # through our face embedding model to obtain the 128-d
        # quantification of the face
        faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255, (96, 96),
            (0, 0, 0), swapRB=True, crop=False)
        embedder.setInput(faceBlob)
        vec = embedder.forward()

        # perform classification to recognize the face
        preds = recognizer.predict_proba(vec)[0]
        j = np.argmax(preds)
        proba = preds[j]
        name = le.classes_[j]

        # draw the bounding box of the face along with the associated
        # probability
        text = "{}: {:.2f}%".format(name, proba * 100)
        y = startY - 10 if startY - 10 > 10 else startY + 10
        cv2.rectangle(image, (startX, startY), (endX, endY),
            (0, 0, 255), 2)
        cv2.putText(image, text, (startX, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)

# show the output image
cv2.imshow("Image", image)
```

Figure 15 : Suite de l'extrait de code de l'entrée du programme

IV.4 Exécution et résultat

Après l'implémentation des trois (3) premières étapes, nous passons aux premiers tests de notre programme.

```
c:\python
λ python opencv-face/recognize.py --detector opencv-face/face_detection_model --embedding-model opencv-face/openface_nn4
.small2.v1.t7 --recognizer opencv-face/output/recognizer.pickle --le opencv-face/output/le.pickle --image opencv-face/im
ages/densel_wha.jpg
[INFO] loading face detector...
[INFO] loading face recognizer...
```

Figure 16 : Exécution en ligne du programme

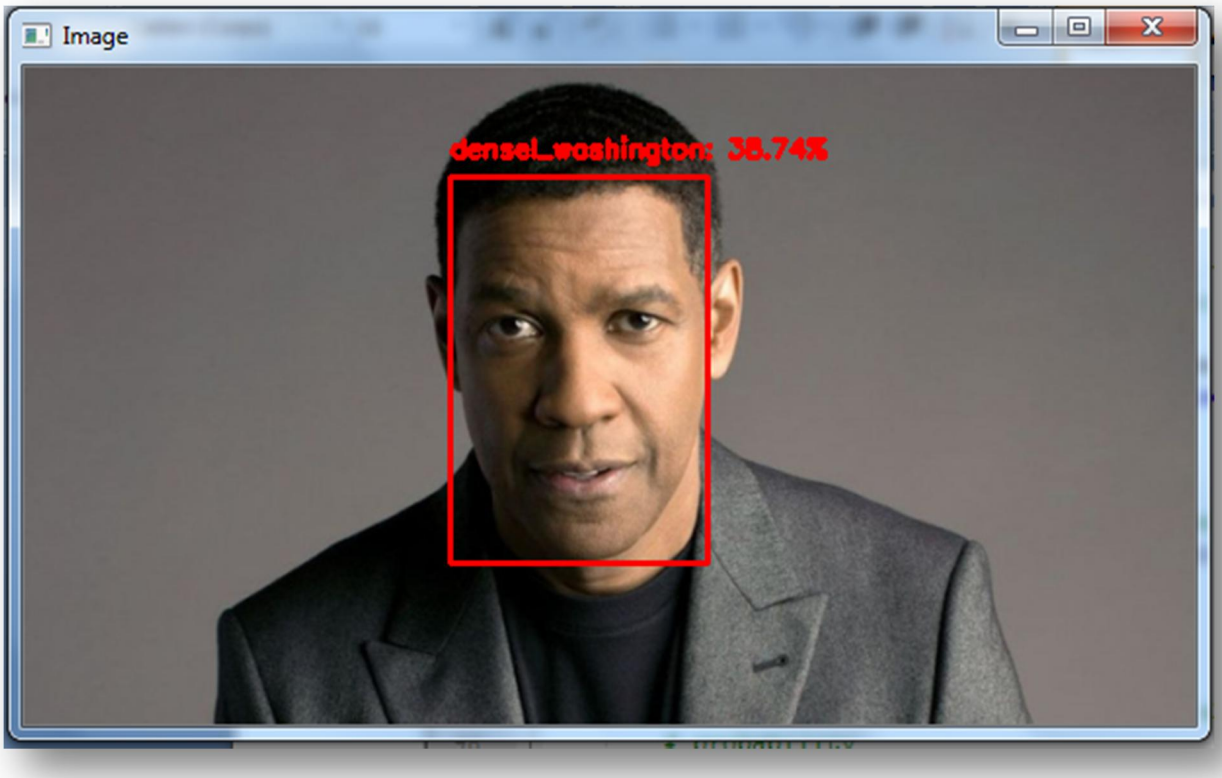

Figure 17 : Résultat de l'exécution de notre premier test

IV.5 Etape 4 : Reconnaître les visages dans les flux vidéo

Le pipeline lui-même est presque identique à la reconnaissance des visages dans les images, avec seulement quelques mises à jour

Le code se trouve dans le fichier recognize_video.py :

*

```python
# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import pickle
import time
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--detector", required=True,
    help="path to OpenCV's deep learning face detector")
ap.add_argument("-m", "--embedding-model", required=True,
    help="path to OpenCV's deep learning face embedding model")
ap.add_argument("-r", "--recognizer", required=True,
    help="path to model trained to recognize faces")
ap.add_argument("-l", "--le", required=True,
    help="path to label encoder")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# load our serialized face detector from disk
print("[INFO] loading face detector...")
protoPath = os.path.sep.join([args["detector"], "deploy.prototxt"])
modelPath = os.path.sep.join([args["detector"],
    "res10_300x300_ssd_iter_140000.caffemodel"])
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# load our serialized face embedding model from disk
print("[INFO] loading face recognizer...")
embedder = cv2.dnn.readNetFromTorch(args["embedding_model"])
```

Figure 18 : Extrait du contenu du fichier

```
                (96, 96), (0, 0, 0), swapRB=True, crop=False)
            embedder.setInput(faceBlob)
            vec = embedder.forward()

            # perform classification to recognize the face
            preds = recognizer.predict_proba(vec)[0]
            j = np.argmax(preds)
            proba = preds[j]
            name = le.classes_[j]

            # draw the bounding box of the face along with the
            # associated probability
            text = "{}: {:.2f}%".format(name, proba * 100)
            y = startY - 10 if startY - 10 > 10 else startY + 10
            cv2.rectangle(frame, (startX, startY), (endX, endY),
                (0, 0, 255), 2)
            cv2.putText(frame, text, (startX, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)

    # update the FPS counter
    fps.update()

    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# stop the timer and display FPS information
fps.stop()
print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

Figure 19 : Suite de l'extrait de code de la reconnaissance dans un flux vidéo

IV.6 Exécution et résultat

La dernière étape qui consistait à reconnaitre des visages au niveau d'un flux vidéo implémentée, nous procédons à un test final.

```
c:\python
λ python opencv-face/recognize_video.py --detector opencv-face/face_detection_model --embedding-model opencv-face/openfa
ce_nn4.small2.v1.t7 --recognizer opencv-face/output/recognizer.pickle --le opencv-face/output/le.pickle
[INFO] loading face detector...
[INFO] loading face recognizer...
[INFO] starting video stream...
```

Figure 20 : Exécution du programme

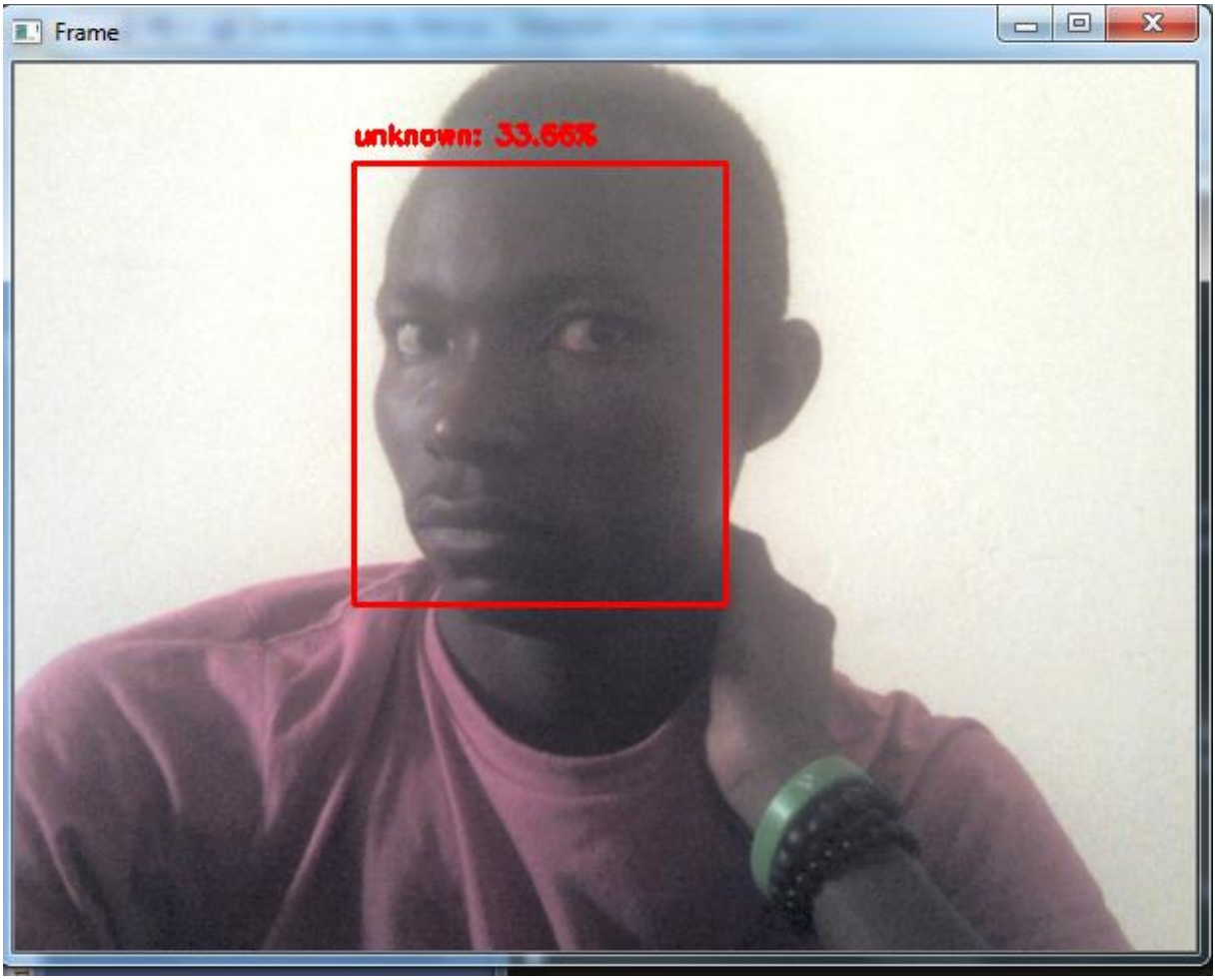

Figure 21 : Résultat de l'exécution

Comme l'illustre la figure ci-dessus, le programme a bien détecté le visage de la personne puis effectué sa reconnaissance. Etant donné, que la personne que nous cherchons à identifier ne correspond pas à celle présente sur la capture (webcam), notre programme nous l'indique avec le message « Unknown » avec le pourcentage de correspondance.

Conclusion

Dans le cadre du cours de traitement d'images, notre réflexion fut soumise à un projet consistant à mettre en place un programme de reconnaissance faciale permettant par exemple, pour des raisons de sécurité de reconnaitre une personne dans une vidéo surveillance.

Pour se faire, nous avons utilisé le langage Python et différents packages tels que OpenCV pour le traitement d'images et vidéos, scikit learn pour notre modèle deep learning entre autre.

Ce fut un projet intéressant, et nous a permis de mettre en pratique les compétences acquises au cours du module, mais également de découvrir d'autres domaines et sujets d'actualités tels que l'intelligence artificielle et la vision par ordinateur ou computer vision.