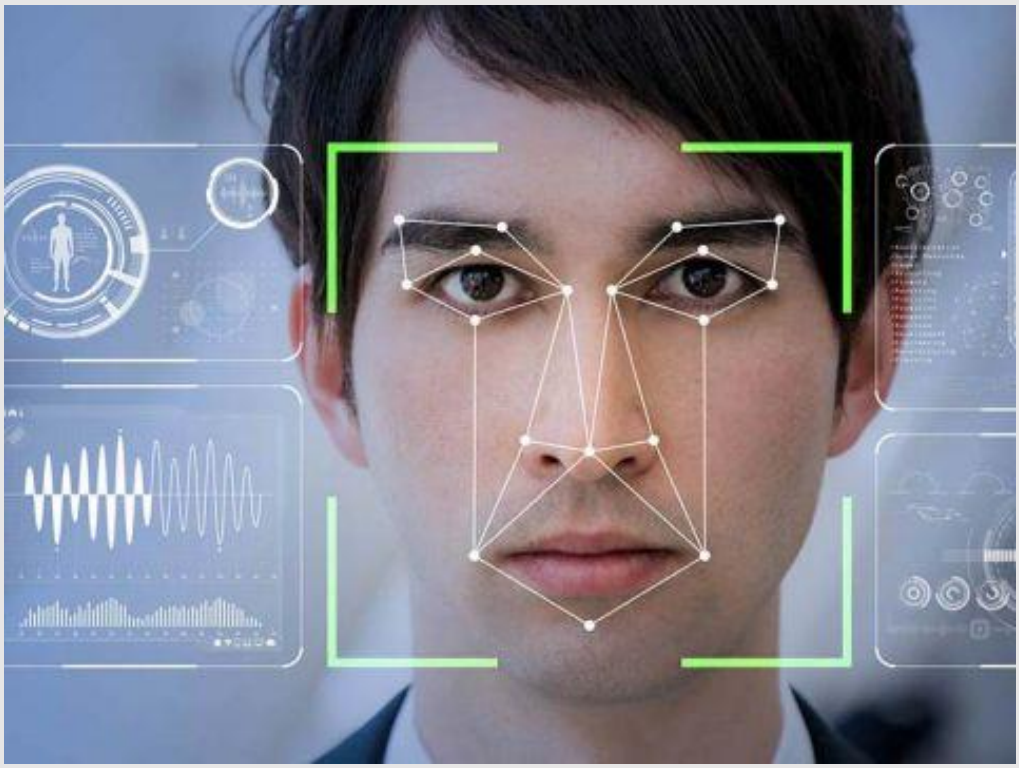
BỘ GIÁO DỤC VÀ ĐÀO TẠO TRƯỜNG ĐẠI HỌC NÔNG LÂM TP HCM

KHOA CÔNG NGHỆ THÔNG TIN

# PROJECT GIỮA KÌ

# NGHIÊN CỨU XÂY DỰNG HỆ THỐNG ĐIỂM DANH BẰNG PHƯƠNG PHÁP NHẬN DIỆN KHUÔN MẶT SỬ DỤNG MÃ NGUỒN MỞ OPENCV




Giảng viên hướng dẫn:
TS. Nguyễn Văn Dũ.

Sinh viên thực hiện:
Nguyễn Võ Công Toàn – 18130247
Ngô Thị Tuyết Mai – 18130136
Nguyễn Nga Anh – 18130012
Lê Thị Cẩm Thu – 18139299


Tp. Hồ Chí Minh, ngày 20 tháng 01 năm 2021

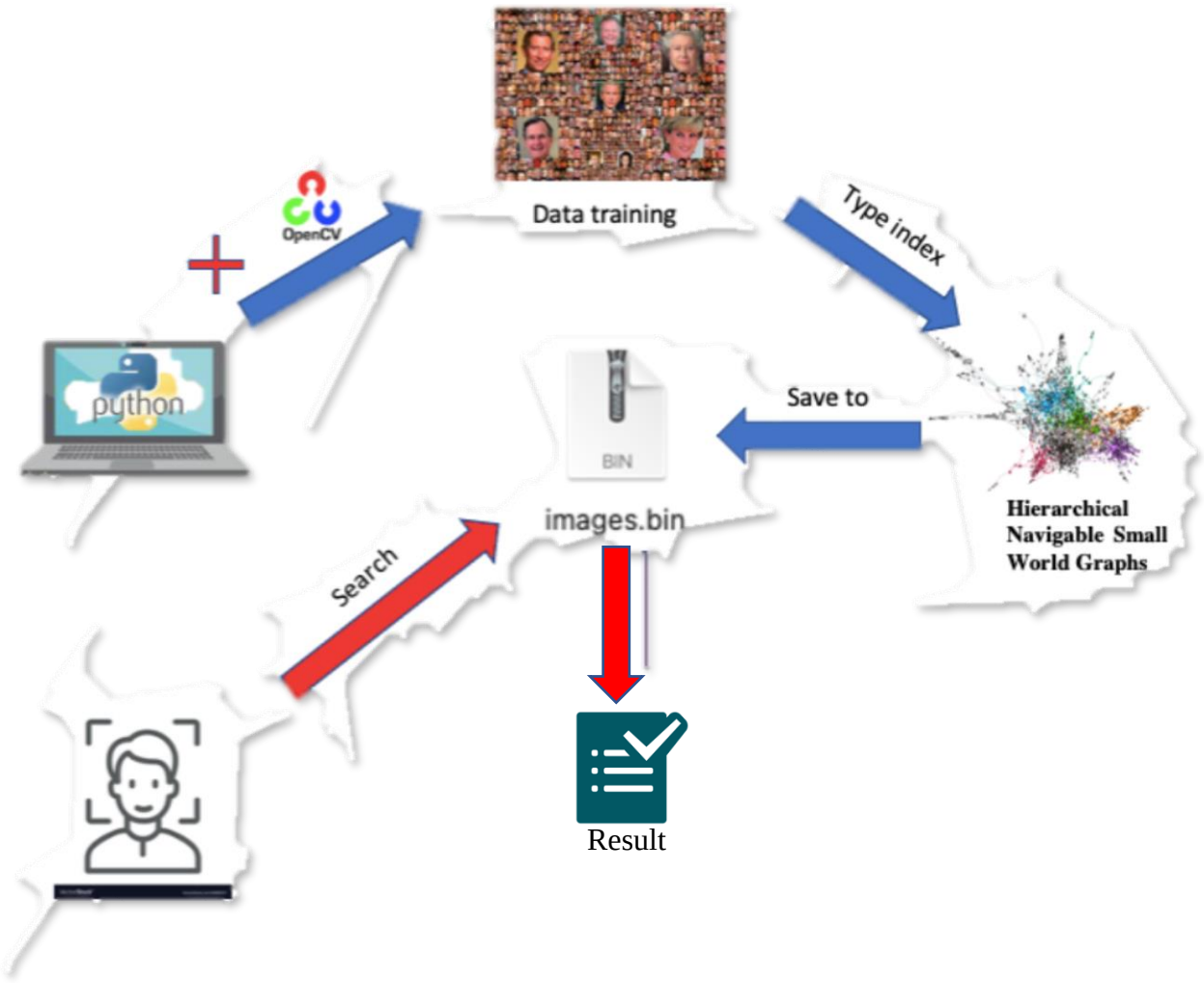
# Mục lục

## Đặt vấn đề:

Trong xu thế của cuộc cách mạng công nghiệp 4.0 đã kéo theo sự phát triển nhanh của các ngành khoa học và công nghệ. Trí tuệ nhân tạo (AI – Artificial Intelligence) là một lĩnh vực phát triển vượt bậc và tạo ra nhiều ứng dụng phục vụ cuộc sống con người. Trên thế giới và ở Việt Nam đã có nhiều công trình nghiên cứu các hệ thống thông minh ứng dụng trí tuệ nhân tạo như bệnh viện thông minh, giao thông thông minh, nhà thông minh, xe tự lái,… Trong đó, ứng dụng nhận dạng khuôn mặt cũng được ứng dụng với nhiều mục đích như là điểm danh chấm công, phát hiện tội phạm, phân biệt tuổi tác, trong đó cũng không thể thiếu mục đích điểm danh chấm công. Chính vì sự cần thiết và mức độ ứng dụng phù hợp cho hệ thống điểm danh lớp học nên em quyết định chọn đề tài này.

## Chương 1: Phân tích hệ thống ứng dụng.

**1. Sơ đồ hệ thống.**



# Sơ đồ của hệ thống

**2. Tất cả thư viện dùng cho chương trình:**

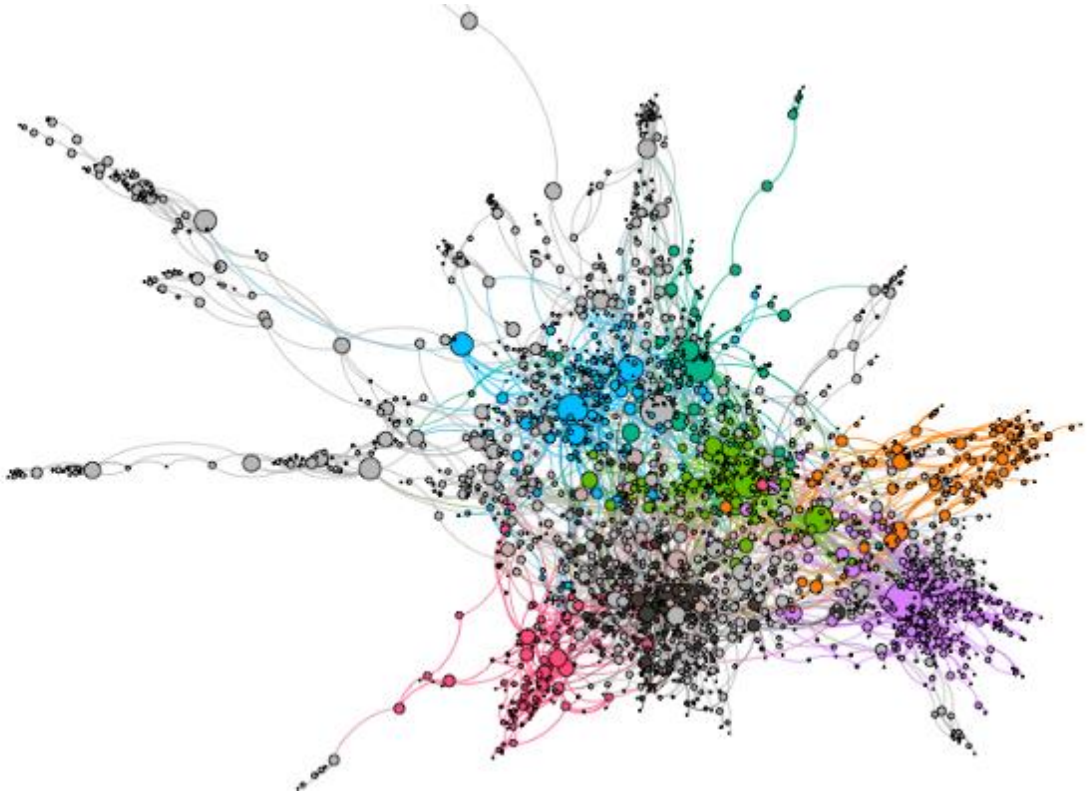
- opencv-python==4.2.0.34
  - OpenCV (Open Source Computer Vision Library) là một thư viện mã nguồn mở về lĩnh vực computer vision và machine learning. OpenCV được xây dựng để cung cấp cơ sở hạ tầng chung cho các ứng dụng thị giác máy tính và để đẩy nhanh việc sử dụng nhận thức của máy trong các sản phẩm thương mại.
- opencv-contrib-python==4.2.0.34
- Keras:
  - **Keras là** một open source cho Neural Network được viết bởi ngôn ngữ **Python**. ... **Keras** có thể sử dụng chung với các thư viện nổi tiếng như Tensorflow, CNTK, Theano
- Tensorflow:
  - TensorFlow chính là thư viện mã nguồn mở cho machine learning nổi tiếng nhất thế giới, được phát triển bởi các nhà nghiên cứu từ Google. Việc hỗ trợ mạnh mẽ các phép toán học để tính toán trong machine learning và deep learning đã giúp việc tiếp cận các bài toán trở nên đơn giản, nhanh chóng và tiện lợi hơn nhiều.
- os:
  - Để truy cập hệ thống trên máy tính.
- numpy:
  - NumPy là gói cơ bản cho tính toán mảng với Python.
- Pillow==7.1.2:
  - Là một fork từ thư viện **PIL** của **Python** được sử dụng **để** xử lý hình ảnh
- h5py==2.10.0:
  - Hỗ trợ đọc ghi dữ liệu vào numpy
- imutils==0.5.3:
  - Xử lý thao tác với file (hình ảnh)
  - Dùng để căn chỉnh
- hnswlib
  - Đây là một thư viện fast approximate nearest neighbor search dựa trên thuật toán HNSW
  - Nhẹ, nhanh ít dung lượng
  - Có supỏt C++, python và R
  - Hỗ trợ thêm và xóa phần tử mà không phải giải phóng bộ nhớ (annoy không làm được)
  - Có thể tự định nghĩa hàm tìm kiếm khoảng cách riêng (C++)
- Dlib:
  - là một bộ toolkit viết bằng C++ chứa các thuật toán machine learning, deep learning xử lý rất nhiều vấn đề trong computer vision, nhưng điều chúng ta biết nhiền nhất đến dlib là sử dụng nó

cho bài toán nhận dạng khuôn mặt với độ chính xác và truy vấn và độ chính xác không cần quá cao.

- Cách cài đặt dlib cho Mac và Ubuntu các bạn có thể tham khảo tại đây:
  - How to install dlib from source on macOS or Ubuntu

- face-recognition:
  - Thư viện nhận dạng khuôn mặt được viết bằng Python cực kỳ đơn giản sử dụng deep learning.
- tqdm:
  - Là một tiện ích của python có chức năng đa xử lý để hiển thị thanh tiến trình.
- Matplotlib:
  - Là thư viện vẽ đồ thị rất mạnh của python.
  - Module được dùng nhiều nhất trong Matplotlib là Pyplot
- mtcnn:
  - MTCNN là viết tắt của Multi-task Cascaded Convolutional Networks. Nó là bao gồm 3 mạng CNN xếp chồng và đồng thời hoạt động khi detect khuôn mặt. Mỗi mạng có cấu trúc khác nhau và đảm nhiệm vai trò khác nhau trong task. Đầu ra của MTCNN là vị trí khuôn mặt và các điểm trên mặt như: mắt, mũi, miệng…

**3. Thuật toán tìm kiếm của chương trình - Hierarchical Navigable Small World Graphs(HNSW):**



Một phương pháp tiếp cận mới để tìm kiếm K-nearest neighbor dựa trên navigable small world graphs. Để có thể tìm kiếm được một đối tượng nào đó trong dataset thì ta phải phân cấp và cấu trúc nó theo dạng graph. Rất nhiều graph trong thế giới thực có tính chất phân cụm rất cao và các node của đồ thị đó(đối tượng) có xu hướng gần nhau mà ta sẽ gọi là small-world graph. Để tìm kiếm ta sẽ bắt đầu tại một số điểm ngẫu nhiên và lặp đi lặp lại cách tìm kiếm đó ở từng small-world graph khác nhau , đi qua mỗi đồ thị thuật toán sẽ kiểm tra khoảng cách của truy vấn hàng xóm của node hiện tại và chọn node đó làm node cơ sở tiếp theo, từ đó sẽ giúp thu nhỏ khoảng cách lại. Thuật toán sẽ dừng tìm kiếm K hàng xóm tốt nhất khi gặp một số điều kiện được đáp ứng với thuật toán.

```
SEARCH-LAYER(q, ep, ef, lc)

Input: query element q, enter points ep, number of nearest to q elements to return
ef, layer number lc

Output: ef closest neighbors to q

1 v ← ep // set of visited elements

2 C ← ep // set of candidates

3 W ← ep // dynamic list of found nearest neighbors

4 while |C| > 0

5 c ← extract nearest element from C to q

6 f ← get furthest element from W to q

7 if distance(c, q) > distance(f, q)

8 break // all elements in W are evaluated

9 for each e ∈ neighbourhood(c) at layer lc // update C and W

10 if e ∉ v

11 v ← v ⋃ e

12 f ← get furthest element from W to q

13 if distance(e, q) < distance(f, q) or |W| < ef

14 C ← C ⋃ e

15 W ← W ⋃ e

16 if |W| > ef

17 remove furthest element from W to q

18 return W
```

Ta xây dựng thuật toán dự trên cách chèn liên tiếp các phần tử lưu trữ vào cấu trúc đồ thị đã từ ban đầu. Với mỗi phần từ được chèn vào tương ứng với một layer gọi là $l$ là một số nguyên được chọn ngẫu nhiên với phân phối xác xuất rời rạc theo cấp số nhân. Theo thuật toán ở trên thì phase đầu tiên trong quá trình bắt đầu từ top layer bằng cách sử dụng giải thuật tham lam đi qua graph theo trình tự để tìm ra $ef$ láng giềng gần

nhất đến phần tử được chèn là $q$ trong layer. Sau đó giải thuật tiếp tục search layer tiếp theo bằng $ef$ láng giềng gần nhất vừa tìm được, và quá trình lặp đi lặp lại như vậy cho đến khi tìm được đủ số láng giềng gần nhất cần tìm.

❖ Ưu điểm

- Ta có thể điều chỉnh tham số để trade off accuracy và tốc độ truy vấn
- Hộ trợ truy vấn với batchsize lớn
- Thuật toán có độ phức tạp thời gian đa giác và có thể vượt trội hơn thuật toán đối thủ trên nhiều bộ dữ liệu trong thế giới thực.

❖ Nhược điểm

- Hoạt động kém khi điểm similarity ở khoảng cách xa so với điểm cần truy vấn.
- Cần nhiều bộ nhớ (RAM)

# Chương 2: Triển khai ứng dụng

## 1. Giai đoạn 1: Thu thập dữ liệu.

### 1.1. Một số thư viện cần thiết trong giai đoạn thu thập dữ liệu:

```
import cv2
import numpy as np
import os
import dlib
from imutils import face_utils
from imutils.face_utils import FaceAligner
```

### 1.2. Bước 1: Khởi tạo:

```
detector = dlib.get_frontal_face_detector()

shape_predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

face_aligner = FaceAligner(shape_predictor, desiredFaceWidth=IMAGE_SIZE)

video_capture = cv2.VideoCapture(0)
```

- Khởi tạo face object detection với function get_frontal_face_detector
- Detect face landmarks và trả về các key points trên khuôn mặt, ở đây dlib sẽ trả về 62 points
- Align các khuôn mặt bằng cách sử dụng phép biến đổi affine bằng function FaceAligner bằng cách dựa vào 62 điểm đã phát hiện trên khuôn mặt và căn chỉnh nó lại cho chính xác
- Sau đó dùng opencv khởi tạo hàm VideoCapture() để capture video, tham số '0' là dùng cho những thiết bị camera cắm ngoài vào.

### 1.3. Bước 2: Nhận dữ liệu và chuyển đổi:

```
while number_of_images < MAX_NUMBER_OF_IMAGES:

            ret, frame = video_capture.read()

            frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            faces = detector(frame_gray)

            if len(faces) == 1:

                face = faces[0]

                (x, y, w, h) = face_utils.rect_to_bb(face)

                cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)

                face_img = frame_gray[y-50:y + h+100, x-50:x + w+100]

                face_aligned = face_aligner.align(frame, frame_gray, face)

                cv2.imwrite(os.path.join(directory,
str(mssv+str(number_of_images)+'.jpg')), face_aligned)
```

- Đầu tiên ta sẽ chạy vòng lặp để nhận số lượng hình ảnh để training mà mình mong muốn.
- Camera sẽ chụp liên tục.
- Tiếp theo ta sẽ chuyển kênh màu RGB sang thang đo độ xám.
- Ta sẽ thông qua hàm face_utils.rect_to_bb(face) trả về face đã được cut từ các tọa độ lấy ở trên, các tham số 50, 100 chỉ là để căn chỉnh thêm cho khuôn mặt( có thể thêm vào hoặc không).
- Ta sẽ dùng hàm FaceAligner đã khởi tạo ở trên để thực hiện việc thực hiện căn chỉnh khuôn mặt ảnh xám đã được cắt ra, hàm này sẽ nhận vào cả khung hình màu, khung hình xám, khuôn mặt.
- Căn chỉnh xong thì ta sẽ ghi nó vào thư mục của mỗi người.

## 2. Giai đoạn 2: Đánh index:

### 2.1. Một số thư viện cần thiết trong giai đoạn thu thập dữ liệu;

```
import cv2
import numpy as np
import os
import dlib
from imutils import face_utils
from imutils.face_utils import FaceAligner
```

### 2.2. Bước 1: Khởi tạo:

```
p = hnswlib.Index(space = 'l2', dim = DIM)

p.init_index(max_elements = NUM_ELEMENTS, ef_construction = EF_CONSTRUCTION,
M = M)

imagePaths = list(paths.list_images('images'))
```

- Giai đoạn này khá quan trọng. Ta sẽ tiếp hành dùng thư viện thuật toán hnsw để khơi tạo không gian mà mỗi điểm trong không gian là 1 không gian 128 chiều.
- p.init_index(max_elements = NUM_ELEMENTS, ef_construction = EF_CONSTRUCTION, M = M) khởi tạo các index rỗng vào trong space, với max_elements là số lượng phần tử tối đa có thể được lưu trữ trong cấu trúc, ef_construction là thời gian khởi tạo/đánh đổi accuracy vs speed khi search, M là số lượng node kết nối với nhau để tạo nên small world graphs trong graph.
- detector = MTCNN() Sử dụng MTCNN để kiểm tra và trích xuất face một lần nữa trước khi đánh index.

### 2.3. Bước2: Tiến hành căn chỉnh và đánh index

```
def image_encoding(coodirnate, img):
    x, y, w, h = [v for v in coodirnate]
    x2, y2 = x + w, y + h
    face = img[y:y+h, x:x+w]
    img_emb = face_recognition.face_encodings(face)
    return img_emb
```

- Ta sử dụng thư viện face_recognition để encode image ra vector 128 chiều thì mới đánh index được và ghi mỗi image đó tương ứng là 1 điểm không gian 128 trong không gian index đã tạo trước đó, function image_encoding sẽ encode face images ra thành vector 128 chiều cho chúng ta.

```
for i, imagePath in tqdm(enumerate(imagePaths)):
    coodirnate, img = check_image_path(imagePath)
    if img.shape == (IMAGE_SIZE, IMAGE_SIZE, DIMENTIONAL):
        img_emb  = face_recognition.face_encodings(img)
        if len(img_emb) == 0:
            pass
        else:
            p.add_items(np.expand_dims(img_emb[0], axis = 0), i)
    else:
        if len(coodirnate) == 0:
            pass
        else:
            img_emb = image_encoding(tuple(coodirnate), img)
            if len(img_emb) == 0:
                pass
            else:
                p.add_items(np.expand_dims(img_emb[0], axis = 0), i)



index_path='images.bin'
print("Saving index to '%s'" % index_path)
p.save_index("images.bin")
del p
```

- Tạo vòng lặp và tiến hành kiểm tra qua từng image xem đã được căn chỉnh chưa,

còn nếu chưa được căn chỉnh thì tiến hành căn chỉnh lại và

tiến hành encode ra vector 128 chiều và tiến hành đánh index vào không gian index,

- p.add_items(np.expand_dims(img_emb[0], axis = 0), i) expand dimension sau đó add vào space
- p.save_index("images.bin") space sẽ được lưu dưới dạng file là "images.bin" có thể coi nó là một database và chúng ta search trên nó.

## 3. Giai đoạn 3: Nhận và tìm kiếm vơi Hnswlib:

### 3.1. Bước 1: Một số thư viện cần thiết cho quá trình tìm kiếm khuôn mặt:

```
import cv2
import numpy as np
import imutils
import face_recognition
from imutils import paths
import matplotlib.pyplot as plt
import os
from collections import Counter
import time
import hnswlib
```

3.2. Bước 2: Tiến hành tìm kiếm

```
output_name = []

known_face_names = []

video_capture = cv2.VideoCapture(0)

p = hnswlib.Index(space='l2', dim=DIM)  # the space can be changed - keeps the data, alters the distance function.

p.load_index("images.bin", max_elements = NUM_ELEMENTS)

imagePaths = list(paths.list_images('images'))
```

- p.load_index("images.bin", max_elements = NUM_ELEMENTS) Load index từ file images.bin , max_elements Đặt lại số lượng phần tử tối đa trong cấu trúc file.

```
for i, imagePath in enumerate(imagePaths):

    name = imagePath.split(os.path.sep)[-2]

    known_face_names.append(name)
```

- Lấy tên từ các thư mục đã được đánh index từ trước và append nó vào trong một mảng. Ở giai đoạn đánh index thì index của các image trong mỗi thư mục bằng với index của tên từ mục khi lấy ra listPath trong thư mục images.

```
def append_names(frame):

        small_frame = cv2.resize(frame, (0, 0), fx=FX, fy=FY)

        rgb_small_frame = small_frame[:, :, ::-1]



        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:

            labels, distances = p.knn_query(np.expand_dims(face_encoding, axis
= 0), k = 1)

            known_face_encoding = p.get_items([labels])

            name = "unknown"

            if distances < 0.13:

                name = known_face_names[labels[0][0]]

            face_names.append(name)

        return face_names
```

- rgb_small_frame = small_frame[:, :, ::-1] Chuyển về kênh màu RGB
- face_locations = face_recognition.face_locations(rgb_small_frame) Trả về tọa độ của face có trong frame
- face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations) Kết hợp giữa cut face từ frame với tọa độ cho trước và encode nó ra thành vector 128 chiều. Ta sẽ đối chiếu vector 128 chiều này vào không gian index và dùng dlib để tìm kiếm lân cận xem gần index nào thì nó sẽ thuộc nhóm index đó.
- labels, distances = p.knn_query(np.expand_dims(face_encoding, axis = 0), k = 1) Với face_encoding vừa lấy ra được từ frame ta sẽ thực hiện query nó trong DB
- if distances < 0.13: Nếu euclidean distance của face vừa trích xuất ra với face trong DB mà nhỏ hơn 0.13 thì xuất ra tên đã tìm được nếu ko trả về "unknown"

Source code