1. **Mô tả mô hình**

Mô hình sử dụng flask api và module deepface và thuật toán MTCNN để phát hiện khuôn mặt: https://github.com/quangminh159/so\_sanh\_video\_deep\_face

Định dạng thư mục của task

so\_sanh\_video\_deep\_face/│

├── app.py - File chính Flask (mã ứng dụng Flask)

├── compare.py - File chứa các hàm so sánh khuôn mặt

├── static/

│ ├── uploads/ - Thư mục lưu ảnh và video tải lên

│ └── faces/ - Thư mục lưu khuôn mặt đã phát hiện

├── templates/

│ ├── index.html - Mẫu trang chủ (upload ảnh/video)

│ └── results.html - Mẫu trang kết quả so sánh khuôn mặt

└── requirements.txt - File chứa các thư viện cần cài đặt

**Phần xử lý để có thể phát hiện ra khuôn mặt và so sánh khuôn mặt compare.py:**

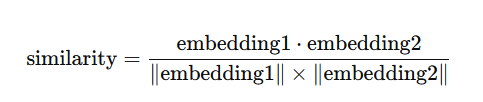
1. **Tính độ tương đồng giữa 2 vector embedding**

def cosine\_similarity(embedding1, embedding2):

    """tính độ tương đồng cosine giữa hai vector embedding"""

    e1, e2 = np.array(embedding1), np.array(embedding2)

    return np.dot(e1, e2) / (np.linalg.norm(e1) \* np.linalg.norm(e2))

Công thức tính độ tương đồng: 

Kết quả đầu ra sẽ cho kết quả từ -1 to 1

1. **Hàm nhận diện khuôn mặt trong ảnh**

def detect\_faces(image):

    """nhận diện khuôn mặt trong ảnh"""

    if image is None:

        print("⚠ Lỗi: Ảnh đầu vào bị rỗng!")

        return []

    detector = MTCNN()

    rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

    results = detector.detect\_faces(rgb\_image)

    faces = []

    for res in results:

        x, y, width, height = res['box']

        x, y = max(0, x), max(0, y)

        face = image[y:y + height, x:x + width]

        faces.append(face)

    return faces

* Hàm kiểm tra ảnh đầu vào (if image is None): Hàm kiểm tra nếu ảnh (image) là None (nghĩa là ảnh không hợp lệ hoặc không tồn tại). Nếu đúng hàm sẽ in ra một thông báo lỗi và trả về một danh sách rỗng.
* Gọi ra hàm phát hiện khuôn mặt (detector=MTCNN()): khởi tạo bộ phát hiện khuôn mặt MTCNN (Multi-task Cascaded Convolutional Networks). MTCNN là một mô hình học sâu được sử dụng phổ biến để nhận diện khuôn mặt trong ảnh
* Chuyển đổi màu sắc của ảnh đầu vào ( rgb\_image=cv2.cvtColor(image.COLOR\_BRG2RGB)): chuyển ảnh về không gian màu RGB phù hợp với MTCNN
* (results=dectector.detect\_face(rgb)image)): sử dụng mô hình MTCNN để phát hiện các khuôn mặt trong ảnh. Kết quả trả về là một danh sách các đối tượng chứa thông tin về các khuôn mặt, bao gồm vị trí (hộp chứa khuôn mặt) và các đặc điểm khác
* Hàm duyệt qua từng kết quả khuôn mặt (res), và lấy tọa độ của hộp chứa khuôn mặt (x, y, width, height)
* x, y = max(0, x), max(0, y) đảm bảo rằng các giá trị tọa độ không bị âm (đảm bảo không cắt ảnh ngoài phạm vi hợp lệ)
* face = image[y:y + height, x:x + width] sử dụng các tọa độ để cắt khuôn mặt từ ảnh gốc
* Cuối cùng, hàm trả về một danh sách faces chứa các ảnh của khuôn mặt đã được cắt từ ảnh gốc

1. **Hàm trích xuất embedding từ danh sách khuôn mặt:**

def get\_embeddings(faces, model\_name="Facenet512"):

    """trích xuất embedding từ danh sách khuôn mặt"""

    embeddings = []

    for i, face in enumerate(faces):

        face\_rgb = cv2.cvtColor(face, cv2.COLOR\_BGR2RGB)

        try:

            embedding\_result = DeepFace.represent(img\_path=face\_rgb, model\_name=model\_name, enforce\_detection=False)

            if embedding\_result:

                embeddings.append(list(embedding\_result[0]["embedding"]))

        except Exception as e:

            print(f"⚠ Lỗi khi trích xuất embedding khuôn mặt {i + 1}: {str(e)}")

    return embeddings

- Khởi tạo danh sách embeddings: embeddings = [] khởi tạo một danh sách rỗng, nơi sẽ lưu trữ các embedding của mỗi khuôn mặt.

* Duyệt qua danh sách các khuôn mặt: Hàm sử dụng vòng lặp for i, face in enumerate(faces) để duyệt qua tất cả các khuôn mặt trong danh sách faces. i là chỉ số của khuôn mặt, và face là ảnh của khuôn mặt hiện tại.
* Chuyển đổi màu sắc ảnh khuôn mặt: face\_rgb = cv2.cvtColor(face, cv2.COLOR\_BGR2RGB) chuyển đổi ảnh khuôn mặt từ không gian màu BGR (màu mặc định trong OpenCV) sang không gian màu RGB (phù hợp với DeepFace).
* Trích xuất embedding từ khuôn mặt:

embedding\_result=DeepFace.represent(img\_path=face\_rgb,model\_name=model\_name, enforce\_detection=False) sử dụng DeepFace để trích xuất embedding từ ảnh khuôn mặt. Hàm represent sẽ trả về một danh sách các embedding cho khuôn mặt, và chúng ta lấy embedding đầu tiên từ danh sách (do mỗi khuôn mặt chỉ có một embedding duy nhất).model\_name là tên của mô hình được sử dụng (mặc định là "Facenet512"). Đây là mô hình học sâu được sử dụng để trích xuất embedding khuôn mặt.enforce\_detection=False chỉ định rằng không cần kiểm tra khuôn mặt trong ảnh trước khi trích xuất embedding

* Xử lý lỗi:Nếu có lỗi trong quá trình trích xuất embedding (ví dụ: ảnh khuôn mặt không hợp lệ hoặc không thể xử lý), một ngoại lệ sẽ được ném ra. Hàm sẽ bắt và in thông báo lỗi liên quan đến khuôn mặt cụ thể thông qua( try-except)
* Thêm embedding vào danh sách: Nếu quá trình trích xuất thành công, embedding\_result[0]["embedding"] chứa embedding của khuôn mặt và được thêm vào danh sách embeddings
* Trả về kết quả: Cuối cùng, hàm trả về danh sách embeddings, chứa tất cả các embedding đã được trích xuất từ các khuôn mặt

1. **Lưu ảnh khuôn mặt đã nhận diện:**

def save\_detected\_faces(faces, base\_filename, folder="static/faces"):

    """lưu ảnh khuôn mặt đã nhận diện"""

    os.makedirs(folder, exist\_ok=True)

    saved\_faces = []

    for i, face in enumerate(faces):

        face\_filename = f"{folder}/{base\_filename}\_face\_{i}.jpg"

        cv2.imwrite(face\_filename, face)

        saved\_faces.append(face\_filename)

    return saved\_faces

* os.makedirs(folder, exist\_ok=True) sẽ tạo thư mục folder nếu thư mục này chưa tồn tại. Tham số exist\_ok=True giúp tránh lỗi nếu thư mục đã có sẵn
* saved\_faces là một danh sách rỗng, dùng để lưu trữ tên các file ảnh đã lưu
* for i, face in enumerate(faces) duyệt qua danh sách các khuôn mặt faces. Mỗi khuôn mặt sẽ được lưu dưới dạng một ảnh riêng biệt.
* face\_filename = f"{folder}/{base\_filename}\_face\_{i}.jpg" tạo tên file cho mỗi ảnh khuôn mặt. Tên file sẽ được tạo dựa trên base\_filename và thêm chỉ số khuôn mặt (i)
* cv2.imwrite(face\_filename, face) lưu ảnh khuôn mặt vào file với tên face\_filename. Hàm cv2.imwrite sẽ ghi ảnh khuôn mặt vào file hệ thống.
* saved\_faces.append(face\_filename) thêm tên file của ảnh khuôn mặt vào danh sách saved\_faces
* Return saved\_faces trả về lại danh sách chứa tất cả các tên file và các khuôn mặt đã được lưu

1. **Hàm so sánh khuôn mặt từ 2 ảnh khác nhau**

def compare\_faces(image1\_path, image2\_path, faces\_folder):

    image1 = cv2.imread(image1\_path)

    image2 = cv2.imread(image2\_path)

    faces1 = detect\_faces(image1)

    faces2 = detect\_faces(image2)

faces1\_saved = save\_detected\_faces(faces1, "image1", faces\_folder)

    faces2\_saved = save\_detected\_faces(faces2, "image2", faces\_folder)

    embeddings1 = get\_embeddings(faces1)

    embeddings2 = get\_embeddings(faces2)

    results = []

    for i, emb1 in enumerate(embeddings1):

        for j, emb2 in enumerate(embeddings2):

            similarity = cosine\_similarity(emb1, emb2)

            match = similarity > 0.6

            results.append({

                "face1\_index": i + 1,

                "face2\_index": j + 1,

                "similarity": similarity,

                "match": match,

                "face1\_img": faces1\_saved[i],

                "face2\_img": faces2\_saved[j]

            })

    return results, faces1\_saved, faces2\_saved

* cv2.imread(image1\_path) và cv2.imread(image2\_path) sẽ đọc hai ảnh từ đường dẫn image1\_path và image2\_path và lưu vào các biến image1 và image2
* detect\_faces(image1) và detect\_faces(image2) gọi hàm detect\_faces đã định nghĩa trước đó để phát hiện khuôn mặt trong hai ảnh
* save\_detected\_faces(faces1, "image1", faces\_folder) và save\_detected\_faces(faces2, "image2", faces\_folder) sẽ lưu các khuôn mặt từ faces1 và faces2 vào thư mục faces\_folder. Các khuôn mặt này sẽ được lưu với tên tương ứng
* Các đường dẫn lưu ảnh khuôn mặt được trả về và lưu vào faces1\_saved và faces2\_saved
* get\_embeddings(faces1) và get\_embeddings(faces2) sẽ trích xuất các embedding từ các khuôn mặt trong faces1 và faces2. Các embedding này sẽ là các vector đặc trưng giúp so sánh sự tương đồng giữa các khuôn mặt
* for i, emb1 in enumerate(embeddings1) và for j, emb2 in enumerate(embeddings2) duyệt qua tất cả các embedding trong embeddings1 và embeddings2
* cosine\_similarity(emb1, emb2) tính toán độ tương đồng giữa hai embedding. Độ tương đồng này đo lường sự giống nhau giữa các khuôn mặt, với giá trị trong khoảng từ -1 đến 1.
* match = similarity > 0.6 kiểm tra nếu độ tương đồng lớn hơn 0.6 thì coi là "khớp" (match), nghĩa là các khuôn mặt này có thể là của cùng một người.
* Mỗi kết quả so sánh (bao gồm các chỉ số khuôn mặt, độ tương đồng, kết quả khớp và đường dẫn đến ảnh) được lưu vào danh sách results
* Hàm trả về ba giá trị:

results: Một danh sách chứa các kết quả so sánh giữa các khuôn mặt từ hai ảnh.

faces1\_saved: Danh sách các tên file của các khuôn mặt đã lưu từ ảnh 1

faces2\_saved: Danh sách các tên file của các khuôn mặt đã lưu từ ảnh 2

1. **Trích xuất khuôn mặt từ video**

def extract\_faces\_from\_video(video\_path, frame\_interval=30):

    """Trích xuất khuôn mặt từ video"""

    cap = cv2.VideoCapture(video\_path)

    faces\_list = []

    frame\_count = 0

    while cap.isOpened():

        ret, frame = cap.read()

        if not ret:

            break

        if frame\_count % frame\_interval == 0:

            faces = detect\_faces(frame)

            faces\_list.extend(faces)

        frame\_count += 1

    cap.release()

    return faces\_list

* cv2.VideoCapture(video\_path) được sử dụng để mở video từ đường dẫn video\_path
* cap là đối tượng dùng để đọc video, cung cấp các khung hình theo từng frame
* faces\_list: Danh sách để lưu các khuôn mặt được trích xuất
* frame\_count: Đếm số lượng khung hình đã đọc
* cap.isOpened() kiểm tra xem video có đang mở không
* cap.read() đọc một khung hình từ video
* Nếu không đọc được (ret == False), có nghĩa là video đã hết, và vòng lặp sẽ dừng
* Chỉ xử lý các khung hình có chỉ số chia hết cho frame\_interval (mặc định là 30)
* Điều này giúp giảm tải xử lý và tránh việc phát hiện khuôn mặt trên quá nhiều khung hình liên tiếp
* detect\_faces(frame) gọi hàm nhận diện khuôn mặt trên khung hình hiện tại.
* faces\_list.extend(faces) thêm các khuôn mặt tìm thấy vào danh sách faces\_list
* Giữ chỉ số khung hình để đảm bảo chỉ xử lý sau mỗi frame\_interval khung hình
* cap.release() đóng video, giải phóng tài nguyên của OpenCV
* return faces\_list: Trả về danh sách chứa tất cả các khuôn mặt đã được phát hiện từ video

1. **So sánh giữa ảnh và video**

def compare\_faces\_in\_video(video\_path, image\_path, faces\_folder):

    image = cv2.imread(image\_path)

    faces\_img = detect\_faces(image)

    faces\_img\_saved = save\_detected\_faces(faces\_img, "image\_target", faces\_folder)

    video\_faces = extract\_faces\_from\_video(video\_path)

    faces\_vid\_saved = save\_detected\_faces(video\_faces, "video", faces\_folder)

    embeddings\_img = get\_embeddings(faces\_img)

    embeddings\_vid = get\_embeddings(video\_faces)

    results = []

    for i, emb1 in enumerate(embeddings\_img):

        for j, emb2 in enumerate(embeddings\_vid):

            similarity = cosine\_similarity(emb1, emb2)

            match = similarity > 0.6

            results.append({

                "face1\_index": i + 1,

                "face2\_index": j + 1,

                "similarity": similarity,

                "match": match,

                "face1\_img": faces\_img\_saved[i],

                "face2\_img": faces\_vid\_saved[j]

            })

    return results, faces\_img\_saved, faces\_vid\_saved

* Đọc ảnh từ image\_path bằng OpenCV
* Nhận diện khuôn mặt trong ảnh bằng detect\_faces(image)
* Lưu khuôn mặt đã nhận diện vào thư mục faces\_folder với tên "image\_target"
* Lưu đường dẫn của ảnh khuôn mặt vào danh sách faces\_img\_saved
* Trích xuất khuôn mặt từ video bằng hàm extract\_faces\_from\_video(video\_path)
* Lưu các khuôn mặt từ video vào thư mục faces\_folder với tiền tố "video"
* Lưu đường dẫn của ảnh khuôn mặt từ video vào danh sách faces\_vid\_saved
* Trích xuất embedding của khuôn mặt từ ảnh (embeddings\_img)
* Trích xuất embedding của khuôn mặt từ video (embeddings\_vid)
* Mỗi embedding là một vector đặc trưng giúp so sánh khuôn mặt
* Duyệt qua tất cả các embedding trong ảnh (embeddings\_img) và video (embeddings\_vid)
* Tính độ tương đồng giữa hai embedding bằng cosine\_similarity(emb1, emb2)
* Xác định khuôn mặt có khớp hay không, nếu similarity > 0.6 thì match = True
* Lưu kết quả vào danh sách results, bao gồm:

Chỉ số khuôn mặt trong ảnh (face1\_index)

Chỉ số khuôn mặt trong video (face2\_index)

Độ tương đồng (similarity)

Kết quả so khớp (match)

Đường dẫn đến ảnh khuôn mặt từ ảnh (face1\_img)

Đường dẫn đến ảnh khuôn mặt từ video (face2\_img)

* return results, faces\_img\_saved, faces\_vid\_saved

+ results: Danh sách kết quả so sánh khuôn mặt giữa ảnh và video.

+ faces\_img\_saved: Danh sách các ảnh khuôn mặt từ ảnh đầu vào.

+ faces\_vid\_saved: Danh sách các ảnh khuôn mặt từ video.

1. **So sánh giữa video và video**

def compare\_faces\_between\_videos(video1\_path, video2\_path, faces\_folder):

    faces\_vid1 = extract\_faces\_from\_video(video1\_path)

    faces\_vid2 = extract\_faces\_from\_video(video2\_path)

    faces\_vid1\_saved = save\_detected\_faces(faces\_vid1, "video1", faces\_folder)

    faces\_vid2\_saved = save\_detected\_faces(faces\_vid2, "video2", faces\_folder)

    embeddings\_vid1 = get\_embeddings(faces\_vid1)

    embeddings\_vid2 = get\_embeddings(faces\_vid2)

    results = []

    for i, emb1 in enumerate(embeddings\_vid1):

        for j, emb2 in enumerate(embeddings\_vid2):

            similarity = cosine\_similarity(emb1, emb2)

            match = similarity > 0.6

            results.append({

                "face1\_index": i + 1,

                "face2\_index": j + 1,

                "similarity": similarity,

                "match": match,

                "face1\_img": faces\_vid1\_saved[i],

                "face2\_img": faces\_vid2\_saved[j]

            })

    return results, faces\_vid1\_saved, faces\_vid2\_saved

* Gọi hàmextract\_faces\_from\_video để lấy danh sách các khuôn mặt từ video thứ nhất (faces\_vid1) và video thứ hai (faces\_vid2)
* Mỗi danh sách chứa các ảnh khuôn mặt được cắt từ video
* Lưu các ảnh khuôn mặt từ video 1 vào thư mục faces\_folder với tiền tố "video1"
* Lưu các ảnh khuôn mặt từ video **2** vào thư mục faces\_folder với tiền tố "video2"
* Các ảnh này giúp hiển thị và kiểm tra trực quan kết quả
* Gọi hàmget\_embeddings để lấy vector đặc trưng (embedding) của từng khuôn mặt trong hai video
* Embedding là một vector số giúp mô hình nhận diện và so sánh khuôn mặt
* Duyệt qua tất cả embedding từ video 1 (embeddings\_vid1) và video 2 (embeddings\_vid2)
* Tính độ tương đồng giữa hai embedding bằng cosine\_similarity(emb1, emb2)
* Xác định xem khuôn mặt có khớp không: Nếu similarity > 0.6, ta coi hai khuôn mặt là cùng một người (match = True)
* Lưu kết quả vào danh sách results, bao gồm:

+ Chỉ số khuôn mặt trong video 1 (face1\_index)

+ Chỉ số khuôn mặt trong video 2 (face2\_index)

+ Độ tương đồng giữa hai khuôn mặt (similarity)

+ Kết quả so khớp (match)

+ Đường dẫn đến ảnh khuôn mặt trong video 1 (face1\_img)

+ Đường dẫn đến ảnh khuôn mặt trong video 2 (face2\_img)

* return results, faces\_vid1\_saved, faces\_vid2\_saved

+ results: Danh sách kết quả so sánh khuôn mặt giữa hai video.

+ faces\_vid1\_saved: Danh sách đường dẫn đến các ảnh khuôn mặt từ video 1

+ faces\_vid2\_saved: Danh sách đường dẫn đến các ảnh khuôn mặt từ video 2

**Phần xử lý chính để chạy ứng dụng app.py:**

1. Khai báo các biến

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = 'static/uploads'

FACES\_FOLDER = 'static/faces'

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

app.config['FACES\_FOLDER'] = FACES\_FOLDER

ALLOWED\_EXTENSIONS = {'png', 'jpg', 'jpeg', 'mp4', 'avi', 'mov'}

* Khởi tạo ứng dụng: flask:app=Flask(\_\_name\_\_)
* UPLOAD\_FOLDER: Thư mục để lưu trữ tệp hình ảnh/video do người dùng tải lên
* FACES\_FOLDER: Thư mục để lưu trữ ảnh khuôn mặt đã phát hiện
* **Cấu hình Flask** để lưu đường dẫn thư mục tải lên và thư mục chứa ảnh khuôn mặt
* Danh sách các định dạng tệp hợp lệ mà người dùng có thể tải lên:
* Hình ảnh: .png, .jpg, .jpeg
* Video: .mp4, .avi, .mov

def allowed\_file(filename):

    """kiểm tra file có đúng định dạng không"""

    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

* Kiểm tra file có đúng định dạng không

for folder in [UPLOAD\_FOLDER, FACES\_FOLDER]:

    if not os.path.exists(folder):

        os.makedirs(folder)

* Tạo thư mục nếu chưa tồn tại: Khi ứng dụng Flask khởi chạy, nó sẽ kiểm tra xem các thư mục UPLOAD\_FOLDER và FACES\_FOLDER có tồn tại hay không.
* Nếu chưa tồn tại, nó sẽ tạo thư mục bằng os.makedirs(folder)

1. Cho phép người dùng và so sánh các khuôn mặt với nhau

@app.route("/", methods=["GET", "POST"])

def index():

    if request.method == "POST":

        compare\_type = request.form.get("compare\_type")

        if compare\_type == "image\_image":

            img1 = request.files["image1"]

            img2 = request.files["image2"]

            if img1 and img2:

                path1 = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(img1.filename))

                path2 = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(img2.filename))

                img1.save(path1)

                img2.save(path2)

                results, faces1, faces2 = compare\_faces(path1, path2, app.config["FACES\_FOLDER"])

                return render\_template("results.html", results=results,

                                       file1=img1.filename, file2=img2.filename,

                                       detected\_faces1=faces1, detected\_faces2=faces2)

        elif compare\_type == "image\_video":

            img = request.files["image"]

            vid = request.files["video"]

            if img and vid:

                img\_path = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(img.filename))

                vid\_path = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(vid.filename))

                img.save(img\_path)

                vid.save(vid\_path)

                results, faces\_img, faces\_vid = compare\_faces\_in\_video(vid\_path, img\_path, app.config["FACES\_FOLDER"])

                return render\_template("results.html", results=results,

                                       file1=img.filename, file2=vid.filename,

                                       detected\_faces1=faces\_img, detected\_faces2=faces\_vid)

        elif compare\_type == "video\_video":

            vid1 = request.files["video1"]

            vid2 = request.files["video2"]

            if vid1 and vid2:

                path1 = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(vid1.filename))

                path2 = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(vid2.filename))

                vid1.save(path1)

                vid2.save(path2)

                results, faces\_vid1, faces\_vid2 = compare\_faces\_between\_videos(path1, path2, app.config["FACES\_FOLDER"])

                return render\_template("results.html", results=results,

                                       file1=vid1.filename, file2=vid2.filename,

                                       detected\_faces1=faces\_vid1, detected\_faces2=faces\_vid2)

    return render\_template("index.html")

* Cho phép cả **GET** (hiển thị trang upload) và **POST** (xử lý tệp tải lên)
* compare\_type = request.form.get("compare\_type")
* Lấy thông tin người dùng chọn loại so sánh khuôn mặt
* if compare\_type == "image\_image":

img1 = request.files["image1"]

img2 = request.files["image2"]

* Lấy hai ảnh từ form
* Lưu ảnh vào thư mục tạm
* path1 = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(img1.filename)) path2 = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(img2.filename)) img1.save(path1) img2.save(path2)
* So sánh khuôn mặt trong hai ảnh
* results, faces1, faces2 = compare\_faces(path1, path2, app.config[“FACES\_FOLDER”])
* kết quả sẽ hiển thị ra trang result:
* return render\_template("results.html", results=results,

file1=img1.filename, file2=img2.filename,

detected\_faces1=faces1, detected\_faces2=faces2)

* Lấy một ảnh và một video từ form
* elif compare\_type == "image\_video":

img = request.files["image"]

vid = request.files["video"]

* Lưu tệp tài nguyên:
* img\_path = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(img.filename))

vid\_path = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(vid.filename))

img.save(img\_path)

vid.save(vid\_path)

* So sánh khuôn mặt trong ảnh vào video và trả về kết quả
* results, faces\_img, faces\_vid = compare\_faces\_in\_video(vid\_path, img\_path, app.config["FACES\_FOLDER"])
* So sánh video với video
* Lấy 2 video từ form
* elif compare\_type == "video\_video":

vid1 = request.files["video1"]

vid2 = request.files["video2"]

* Lưu video tải lên:
* path1 = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(vid1.filename))

path2 = os.path.join(app.config["UPLOAD\_FOLDER"], secure\_filename(vid2.filename))

vid1.save(path1)

vid2.save(path2)

* So sánh khuôn mặt giữa 2 video
* results, faces\_vid1, faces\_vid2 = compare\_faces\_between\_videos(path1, path2, app.config["FACES\_FOLDER"])
* Trả về trang mặc định nếu không có POST
* return render\_template("index.html")

1. **Nhận xét mô hình**

Mô hình so sánh giữa nhiều khuôn mặt trong ảnh hoặc video sử dụng flask và mô hình deepface

**Kết quả đạt được qua task này:**

1. Tìm hiểu thêm thuật toán là MTCNN để phát hiện khuôn mặt và xác định các điểm đặc trưng của khuôn mặt.
2. Sử dụng flask api backend để xử lý tải ảnh và video, thực hiện nhận diện khuôn mặt và trả kết quả.
3. Sử dụng MTCNN face detection để xác định vị trí khuôn mặt trong ảnh và video.
4. Dùng embedding để biểu diễn đặc trưng của ảnh thành vector trong bài sử dụng vector có kích thước 512 để đại diện cho khuôn mặt.
5. Web UI và API client để người dùng có thể tải ảnh lên.

**Ưu điểm của mô hình:**

1. Hỗ trợ nhiều loại so sánh khuôn mặt

Cho phép so sánh giữa:

* Ảnh với ảnh
* Ảnh với video
* Video với video

Giúp ứng dụng linh hoạt hơn trong các tình huống thực tế.

1. Sử dụng thư viện DeepFace và MTCNN

* DeepFace giúp trích xuất đặc trưng khuôn mặt và so sánh với độ chính xác cao.
* MTCNN là một trong những mô hình nhận diện khuôn mặt hiệu quả, giúp cắt khuôn mặt chính xác.

1. Quản lý tệp tải lên tốt

* Lưu ảnh và video vào thư mục riêng (static/uploads) để tránh ghi đè dữ liệu.
* Tạo thư mục nếu chưa tồn tại (os.makedirs(folder)) để đảm bảo tính ổn định của hệ thống.

1. Trả kết quả rõ ràng

* Lưu khuôn mặt đã nhận diện và hiển thị lại cho người dùng.
* Hiển thị mức độ tương đồng giữa các khuôn mặt bằng cosine similarity.

**Nhược điểm của mô hình**

1. Không kiểm soát số lượng khuôn mặt được phát hiện

* Nếu có nhiều khuôn mặt trong ảnh/video, mô hình sẽ so sánh tất cả với nhau

1. Hiệu suất chưa tối ưu với video

* Mỗi frame cách nhau 30 frames (frame\_interval=30), nhưng vẫn có thể mất nhiều thời gian để xử lý video dài.
* Lọc khuôn mặt bị trùng lặp trong video và trong ảnh