Nhận diện mặt và tên người trong ảnh

Giới thiệu

Công cụ này sẽ nhận diện khuôn mặt của người bằng mô hình MTCNN, và sẽ so sánh khuôn mặt người đó so với cơ sở dữ liệu khuôn mặt để đoán ra tên người đó. Công cụ còn có chức năng thêm khuôn mặt mới vào cơ sở dữ liệu vào tạo ra biến thể của ảnh đó bằng các phương pháp xử lý ảnh để có thể phần nào nhận diện được người đó ở nhiều môi trường khác nhau.

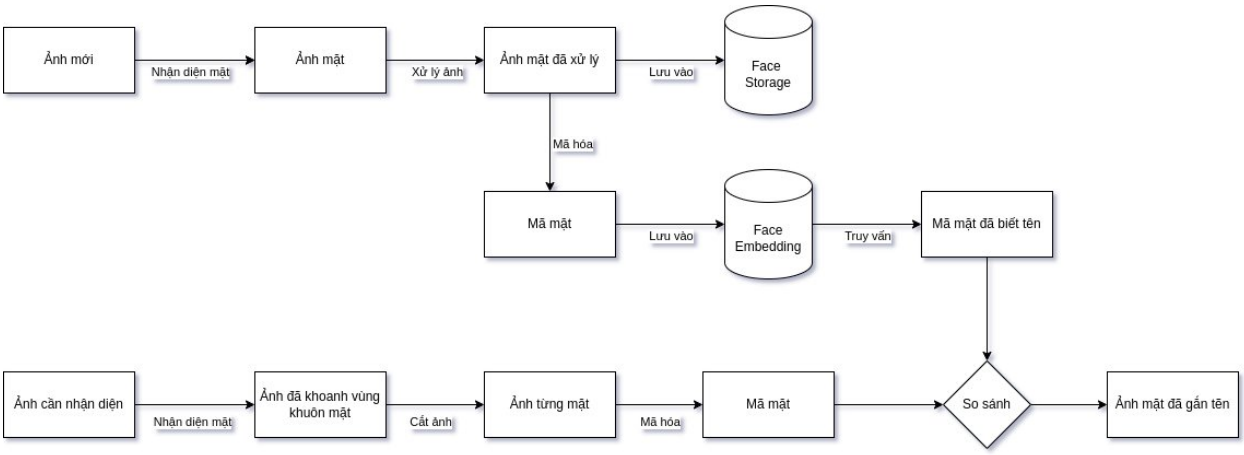
Luồng hoạt động

Thêm ảnh mặt mới

- Nạp ảnh mặt và tên mặt
- Xử lý ảnh mặt cho ra các biến thể của ảnh
- Mã hóa các ảnh
- Lưu mã ảnh và tên vào dataset
- Lưu ảnh mặt và tên vào bộ nhớ để có thể nạp lại khi khởi động app

Nhận diện mặt

- Nạp ảnh
- Nhận diện và khoanh vùng các khuôn mặt có trong ảnh
- Cắt mặt ra khỏi ảnh để cho ra các ảnh mặt
- Mã hóa các ảnh mặt
- So sánh mã mặt với mã có trong dataset để cho ra tên mặt
- Vẽ khung và tên mặt vào ảnh gốc



Giải thuật

Nhận diện khuôn mặt bằng MTCNN

```python
detector = MTCNN()


def detect_faces(image):  3 usages
    """
    Detects faces in an image using MTCNN.
    :param image: Input image in BGR format.
    :return:
    """
    detections = detector.detect_faces(image)
    if not detections:
        return None

    boxes = []
    for detection in detections:
        boxes.append(detection['box'])

    return boxes
```

Giải thích mô hình MTCNN

Mô hình này là kết hợp của 3 mô hình nhỏ theo từng bước:

- PNet: Đoán ra các vùng khuôn mặt tiềm năng
- RNet: Tăng cường dự đoán vùng khuôn mặt và lọc các vùng không phù hợp
- ONet: Trả ra vùng khuôn mặt dự đoán cuối cùng và nhận diện các đường nét trên khuôn mặt

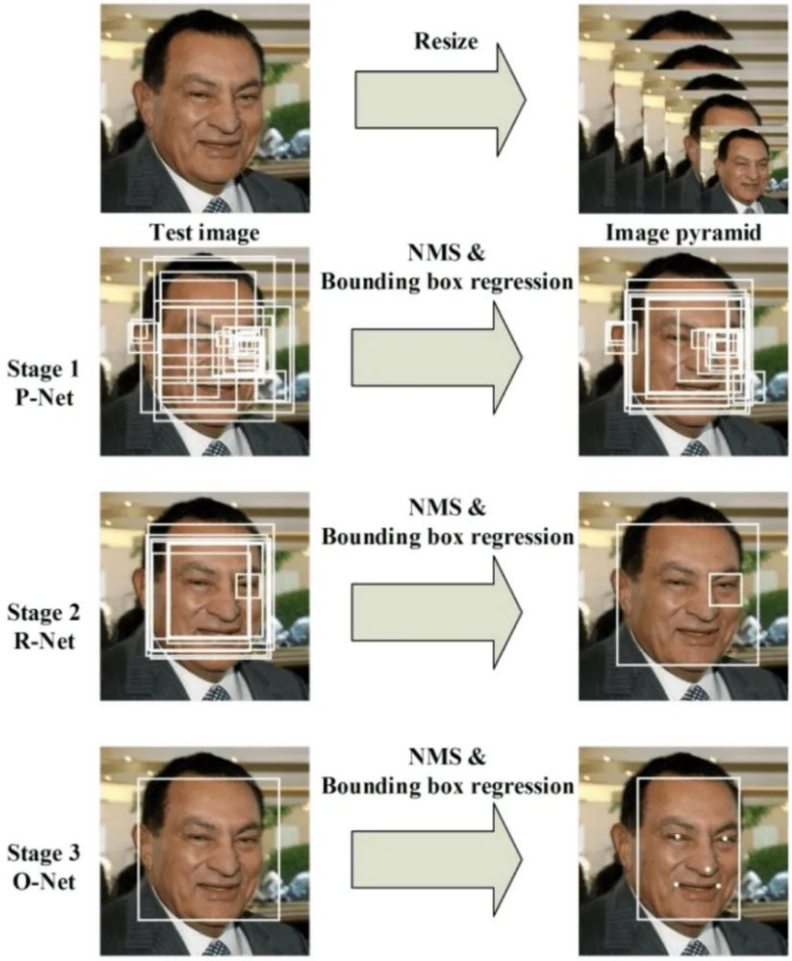


Figure 1 Luồng xử lý trên mô hình MTCNN (Nguồn: [How MTCNN Detects Faces: A Simple Guide to Powerful Technology])

Cắt mặt ra khỏi ảnh

```python
def crop_face(image):  2 usages
    """
    Detects and tightly crops the face from an image.
    """
    detections = detect_faces(image)

    if detections:
        x, y, w, h = detections[0]
        face = image[y:y + h, x:x + w]

        # Ensure tight cropping with no extra borders
        face = remove_black_background(face)

        return face
    return None


def remove_black_background(image):  2 usages
    """
    Removes black background and trims edges.
    """
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    _, mask = cv2.threshold(gray, thresh: 1, maxval: 255, cv2.THRESH_BINARY)

    # Find contours and bounding box of the non-black area
    coords = cv2.findNonZero(mask)
    x, y, w, h = cv2.boundingRect(coords)

    # Crop to bounding box
    cropped = image[y:y + h, x:x + w]
    return cropped
```

Xử lý ảnh, Xuất ra biến thể ảnh

```python
def apply_transformations(image, num_variations=10):  2 usages
    """
    Generates multiple variations of the face image.
    """
    processed_images = []
    h, w = image.shape[:2]

    for _ in range(num_variations):
        transformed = image.copy()

        # Random rotation (-30 to +30 degrees)
        angle = random.randint(-30,  b: 30)
        matrix = cv2.getRotationMatrix2D( center: (w // 2, h // 2), angle,  scale: 1)
        transformed = cv2.warpAffine(transformed, matrix,  dsize: (w, h))

        # Random flip
        if random.choice([True, False]):
            transformed = cv2.flip(transformed,  flipCode: 1)

        # Random brightness/contrast
        alpha = random.uniform( a: 0.7,  b: 1.3)
        beta = random.randint(-40,  b: 40)
        transformed = cv2.convertScaleAbs(transformed, alpha=alpha, beta=beta)

        # Gaussian blur
        if random.choice([True, False]):
            kernel_size = random.choice([(3, 3), (5, 5)])
            transformed = cv2.GaussianBlur(transformed, kernel_size,  sigmaX: 0)

        # Remove any remaining black borders
        transformed = remove_black_background(transformed)

        processed_images.append(transformed)

    return processed_images
```

Mã hóa mặt

```python
def encode_face(image):  3 usages
    """
    Encode a face image into a vector.
    """
    encodings = DeepFace.represent(image, model_name="Dlib", detector_backend="skip", max_faces=1)
    if len(encodings) == 0:
        return None
    return encodings[0]['embedding']
```

So sánh khoảng cách giữa mã mặt gốc với mã có trong dataset

```python
def face_distance(image):  1 usage
    """
    Calculate the Euclidean distance between the given face image and known faces.
    """
    encoding = encode_face(image)
    if encoding is None:
        return None

    distances = {}
    for name, encodings in KNOWN_FACES.items():
        min_distance = float('inf')
        for known_encoding in encodings:
            distance = np.linalg.norm(np.array(encoding) - np.array(known_encoding))
            if distance < min_distance:
                min_distance = distance
        distances[name] = min_distance

    return distances
```

Đoán tên mặt

```python
def guess_face(image):  2 usages
    """
    Guess the name of the person in the image.
    """
    distances = face_distance(image)
    if distances is None or len(distances) == 0:
        return None

    closest_name = min(distances, key=distances.get)
    closest_name_distance = distances[closest_name]
    return closest_name, closest_name_distance
```

Nạp ảnh mặt mới

```python
def add_new_face(name, image):  2 usages
    """
    Add a new face to the dataset.
    """
    face_image = crop_face(image)
    if face_image is None:
        return False

    face_variations = apply_transformations(face_image)
    face_variations.append(face_image)

    if not os.path.exists(f"{DIR}/{name}"):
        os.makedirs(f"{DIR}/{name}")

    for i, img in enumerate(face_variations):
        image_name = f"{name}_{i}.jpg"
        cv2.imwrite( filename: f"{DIR}/{name}/{image_name}", img)

    face_encodings = [encode_face(face) for face in face_variations]
    KNOWN_FACES[name] = face_encodings
    return True
```

Kết quả

Xử lý ảnh

Nhận diện