

Simple Achievement of Face Swapping

Group 7

XIE Suiwei 2030026168

MA Yishu 2030026107

MA Sijia 2030026105

Part I. Face Swapping Implementation in Pictures

1. Extract Facial Landmarks

The landmark of a face image is a set of points that mark the key points of a specific position of a face. These key points usually include the eyes, eyebrows, nose, mouth, chin and other important areas of the face. They are used to represent the shape and structure of the face and provide quantitative information about facial features.

1.1 Process Flow

- i) Initialize the facial landmark detector: The code initializes the facial landmark detector using a pretrained model.
- ii) Convert the image to grayscale
- iii) Perform face detection: The detector is applied to the grayscale image to detect faces. It returns a list of face objects, each representing a detected face region.
- iv) Iterate over detected faces: For each detected face, the code performs the following steps:
 - a. Draw a rectangle around the face: Using the face object's coordinates, a rectangle is drawn around the face region on the original image.
 - b. Predict facial landmarks: The predictor model is applied to the face region to predict the facial landmarks. The landmarks represent specific points on the face, such as the corners of the eyes, nose, mouth, etc.
 - c. Iterate over the predicted landmarks: For each predicted landmark, the code performs the following steps:
 - i. Retrieve the coordinates of the landmark.
 - ii. Draw a circle at the landmark position on the original image.

1.2 Pseudo-Code

Algorithm of read_im_and_landmarks()

Input:

original image

Output:

image with landmark

- ```
1 detector = initialize_facial_landmark_detector()
2 image = load_image("image.jpg")
3 gray = convert_to_grayscale(image)
```

```

4 faces = detect_faces(gray)
5 for face in faces:
6 draw_rectangle(image, face)
7 landmarks = predict_facial_landmarks(gray, face)
8 for landmark in landmarks:
9 x,y = get_landmark_coordinates(landmark)
10 draw_circle(image, x, y)
11 display_image(image)
12 Return

```

---

### 1.3 Result



## 2. Align Faces with a Procrustes Analysis

The two faces have their own landmarks. Now we need to align face B with a face so that the result can look more real. But because A and B's face size, rotation, and so on are not quite the same. To transform the faces, here we use Procrustes analysis to find a transformation matrix from B to A. Procrustes analysis is to normalize the landmarks that we obtained from the previous steps and get consistent processing results. Procrustes analysis considers three transformations, translation, global scaling, and rotation. We believe that the graph obtained by combining these three transformations is equivalent. The goal of Procrustes analysis is to obtain an equivalent graph after such transformation of the image, and this equivalent graph is closest to a reference graph.

### 2.1 Process Flow

- i) Translation can be done by calculating the average value, and then the

landmarks are normalized.

ii) Global scaling calculates the variance of the landmarks, and then the difference is normalized.

iii) Rotation matrix can be obtained by estimating an orthogonal matrix  $R$  so that the point transformed by the  $R$  matrix is closest to the reference figure.  $R$  matrix calculation can first find the covariance matrix of the two figures, and then SVD decomposition,  $R=UV^T$

## 2.2 Pseudo-Code

---

### Algorithm of calculate\_affine\_matrix()

---

**Input:**

landmarks1, landmarks2

**Output:**

affined\_matrix

```
1 # Calculate the mean of the set of points
2 c1 = calculate_mean(landmarks1)
3 c2 = calculate_mean(landmarks2)
4
5 # Centralized set of points
6 landmarks = landmarks1 - c1
7 landmarks2 = landmarks1 - c1
8
9 # Calculate the standard deviation of the set of points
10 s1 = calculate_std(landmarks1)
11 s2 = calculate_std(landmarks2)
12
13 # Normalized point set
14 landmarks = landmarks1 / s1
15 landmarks2 = landmarks2 / s1
16
17 # Calculate the eigenvectors and eigenvalues between point sets
18 U, S, Vt = svd(transpose(landmarks1) * landmarks2)
19
20 # Computed rotation matrix
21 R = transpose(U * Vt)
22
23 # Construct an affine matrix
24 affine_matrix = stack([stack([(s2 / s1) * R, transpose(c2) - (s2 / s1) * R * transpose(c1)]),
25 [0, 0, 1]])
25 Return
```

---

## 3. Color Correction

In this step, we try to eliminate the difference of the skin color and lighting between the two images. Specifically, we want to blend the face of im2 to im1, so the

following function try to change the color of im2 to match the color of im1.

### 3.1 Process Flow

- i) Divide im2 by a gaussian blur of im2
- ii) Multiply by a gaussian blur of im1

\* It's important to set a suitable pupillary distance of Gaussian Filter, because when the distance is too small, the facial features from the first image will show up in the second, and when the distance is too large, the kernel strays outside of the face area for pixels being overlaid, and discolouration occurs. So, we apply pupillary distance = 0.6.

### 3.2 Pseudo-Code

---

#### Algorithm of correct\_colours()

---

**Input:**

image1, image2, landmarks1

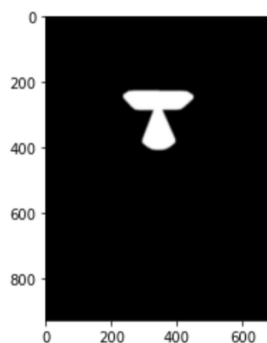
**Output:**

image

```
1 blur_amount = COLOUR_CORRECT_BLUR_FRAC * np.linalg.norm(
2 np.mean(landmarks1[LEFT_EYE_POINTS], axis = 0) -
 np.mean(landmarks1[RIGHT_EYE_POINTS], axis = 0))
3 blur_amount = int(blur_amount)
4 if blur_amount % 2 == 0 then:
5 blur_amount += 1
6 im1_blur = cv2.GaussianBlur(im1, (blur_amount, blur_amount), 0)
7 im2_blur = cv2.GaussianBlur(im2, (blur_amount, blur_amount), 0)
8
9 # Avoid divide-by-zero errors
10 im2_blur += (128 * (im2_blur <= 1.0)).astype(im2_blur.dtype)
11
12 # Divide im2 by a gaussian blur of im2 and then multiply by a gaussian blur of
 im1
13 Return (im2.astype(np.float64) * im1_blur.astype(np.float64) /
 im2_blur.astype(np.float64))
14 Return
```

---

## 4. Blending Features



We want to blend the facial features of im2 to the face of im1. So, we are going to

select features of faces by using a mask shown above. The white region (value = 1) is the area where the im2 should be selected, and the black region (value = 0) is the area where the im1 should be selected.

#### 4.1 Process Flow

- i) Input the coordinates of the points
- ii) Get the convex hull of the points by using cv2.convexHull() fill the convex with colour = 1 with cv2.fillConvexPoly()
- iii) Use Gaussian filter to feather the discontinuities.

#### 4.2 Pseudo-Code

---

**Algorithm of get\_face\_mask()**

---

**Input:**

image, landmarks

**Output:**

image

```
1 im = numpy.zeros(im.shape[:2], dtype = numpy.float64)
2 for group in OVERLAY_POINTS:
3 draw_convex_hull(im, landmarks[group], color = 1)
4 landmarks = predict_facial_landmarsk(gray, face)
5 im = numpy.array([im, im, im].transpose((1, 2, 0)))
6 im = (cv2.GaussianBlur(im, (FEATHER_AMOUNT, FEATHER_AMOUNT), 0) > 0) *
 1.0
7 im = (v2.GaussianBlur(im, (FEATHER_AMOUNT, FEATHER_AMOUNT), 0))
8 Return
```

---

---

**Algorithm of draw\_convex\_hull()**

---

**Input:**

image, points, color(=1)

**Output:**

NONE

```
1 points = cv2. convexHull(points)
2 cv2. fillConvexPoly(im, points, color = color)
3 Return
```

---

## 5. Testing

### 5.1 Initialization

```

FACE_POINTS = list(range(17, 68))
MOUTH_POINTS = list(range(48, 61))
RIGHT_BROW_POINTS = list(range(17, 22))
LEFT_BROW_POINTS = list(range(22, 27))
RIGHT_EYE_POINTS = list(range(36, 42))
LEFT_EYE_POINTS = list(range(42, 48))
NOSE_POINTS = list(range(27, 35))
JAW_POINTS = list(range(0, 17))

Points used to line up the images.
ALIGN_POINTS = (LEFT_BROW_POINTS + RIGHT_EYE_POINTS + LEFT_EYE_POINTS + RIGHT_BROW_POINTS
 + NOSE_POINTS + MOUTH_POINTS)

Points from the second image to overlay on the first. The convex hull of each element will be overlaid.
OVERLAY_POINTS = [LEFT_EYE_POINTS + RIGHT_EYE_POINTS + LEFT_BROW_POINTS + RIGHT_BROW_POINTS,
 NOSE_POINTS + MOUTH_POINTS,]

Amount of blur to use during colour correction, as a fraction of the
pupillary distance.
COLOUR_CORRECT_BLUR_FRAC = 0.6

```

Set the coordinates of the features in a face.

## 5.1 Testing

```

Read the two images and get the landmark of each
im1, landmarks1 = read_im_and_landmarks(sys.argv[1])
im2, landmarks2 = read_im_and_landmarks(sys.argv[2])

Compute the transformation matrix
M = transformation_from_points(landmarks1[ALIGN_POINTS], landmarks2[ALIGN_POINTS])

Get the mask for image2
mask = get_face_mask(im2, landmarks2)

Transform the mask for image 2 into image 1's coordinates
warped_mask = warp_im(mask, M, im1.shape)

Combine the two masks into one by using element-wise maximum method.
combined_mask = np.max([get_face_mask(im1, landmarks1), warped_mask],
 axis=0)

Let the features from image 1 are covered, and that the features from image 2 are shown
warped_im2 = warp_im(im2, M, im1.shape) # Features of image 2

Colour correction
warped_corrected_im2 = correct_colours(im1, warped_im2, landmarks1)

The mask is applied to give the final image
output_im = im1 * (1.0 - combined_mask) + warped_corrected_im2 * combined_mask

cv2.imwrite('output34.jpg', output_im)

```

Step 1. Read the two images and get the landmark of each

Step 2. Compute the transformation matrix

Step 3. Get the mask for image2

Step 4. Transform the mask for image 2 into image 1's coordinates


Step 5. Combine the two masks into one by using element-wise maximum method

Step 6. Let the features from image 1 are covered, and that the features from image 2 are shown.

Step 7. Colour correction based on image 1.

Step 8. The mask is applied to give the final image.

## 5.2 Results

|          |                                                                                    |                                                                                     |
|----------|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| Original |   |   |
| Result   |  |  |

## Part II. Face Swapping Implementation in Videos

### 1. Video Face Location

#### 1.1 Process Flow

- i) Set parameters to read and write the video.
- ii) Initialize the facial landmark detector: `dlib.get_frontal_face_detector()`
- iii) Read each frame of the video
- iv) Perform face locate operation the same as part 1 until we traverse all frames, merge each frame to a video.

## 1.2 Pseudo-Code

---

### Algorithm of read\_im\_and\_landmarks()

---

**Input:**

original video

**Output:**

Video with landmark

```
12 while True:
13 ret,img=capture.read()
14 if not ret:
15 break
16 do Facial Landmarks Extraction
17 out.write(img)
18 if cv2.waitKey(1) & 0xff == ord('q'):
19 break
20 cv2.imshow('face detection landmark',img)
21
22 capture.release()
23 out.release()
24 cv2.destroyAllWindows()

25 Return
```

---

## 1.3 Result



## 1.4 Problem

The image on the right shows that when people in the video lower the heads, the detector cannot detect the face. So it will cause trouble in the face swapping later. It means we have strict rules to choose the original video.

## 2. Video Face Swapping

### 2.1 Process Flow

- i) Set parameters to read and write the video. Initialize the facial landmark detector: The code initializes the facial landmark detector using a pretrained



- model.
- ii) Read each frame of the video.
- iii) Perform face swapping operation the same as operation 2 until we traverse all frames, save each frame as one image.
- iv) Traverse each frame and save as video, which has the same fps and size as original video.

## 2.2 Pseudo-Code

### 2.2 Pseudo-Code

---

#### Algorithm of read\_im\_and\_landmarks()

---

##### Input:

original video

##### Output:

Face Swapping Video

```

26 capture = cv2.VideoCapture("junface.mp4")
27 fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
28 out = cv2.VideoWriter('jun_face_change_video.mp4', fourcc, 29.44, (960,720))
29 detector = initialize_facial_landmark_detector()
30 predictor=dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
31 count = 0
32 while True:
33 if count==58: # set the frame amount we want to save
34 break
35 ret,img=capture.read()
36 if not ret:
37 break
38 do image face-swapping operations
39 if cv2.waitKey(1) & 0xff == ord('q'):
40 break
41 cv2.imwrite('./jun_video/output'+str(count)+'.jpg', output_im)
42 count+=1
43 end while
44 for i in range(count):
45 out.write(cv2.imread("./jun_video/output"+str(i)+".jpg"))
46 end for
47 capture.release()
48 out.release()
49 cv2.destroyAllWindows()

50 Return

```

---

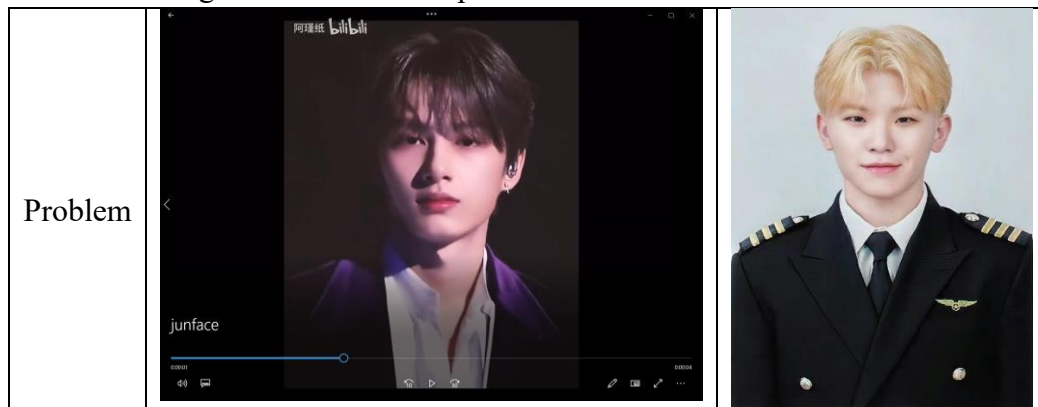
## 2.3 Result



## 2.4 Problems & Solutions

2.4.1 In the process of video face swapping, we originally intended to use the video writer function in the loop to directly write the result of each frame after face-swapping, but it caused the problem of the screen turning to flash or failing to open the file. We think it may be a change in the frame format after face-swapping, but we can't fix the problem in a short time. So we decided to read the frames and then save the face-swapping result frames in jpg format. Then use a loop to read each image and save it as video format.

2.4.2 We can observe that the image on the right has the eyebrow part a little bit untrue, it is because the original image and the swapping face we use below. The eyebrow in the original video is dark and the swapping face eyebrow part is bright. So it will give us the result that our face swapping function may not do well in the illumination change. We can further optimize it.



## Reference

- [1] K.P. Kaiser. (2015, August 6). The (Mostly) Newbie's Guide to Automatically Swapping Faces in Video. Kpkaiser. <https://www.kpkaiser.com/programming/the-mostly-newbies-guide-to-automatically-swapping-faces-in-video/>
- [2] Liangwe Liang. (2021, July 14). AI-Change-face-in-the-video. Github. <https://github.com/Liangwe/AI-Change-face-in-the-video/blob/9390c2bc65664ca3d341a6431fa91c28544d34bd/README.md?plain=1#L1>
- [3] Matthew Earl. (2015, July 28). Switching Eds: Face swapping with Python, dlib, and OpenCV. Github. <https://matthewearl.github.io/2015/07/28/switching-eds-with-python/>
- [4] Matt Zheng. (2018, Jan 3). Face\_Swapping. Github. [https://github.com/mattzheng/Face\\_Swapping](https://github.com/mattzheng/Face_Swapping)
- [5] Fakerth. (2022, July 13). 基于特征点检测的人脸融合技术. CSDN blog. [https://blog.csdn.net/weixin\\_43912621/article/details/125170648?spm=1001.2101.3001.6650.1&utm\\_medium=distribute.pc\\_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-125170648-blog-54982632.235%5Ev35%5Epc\\_relevant\\_increate\\_t0\\_download\\_v2\\_base&depth\\_1-utm\\_source=distribute.pc\\_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-125170648-blog-54982632.235%5Ev35%5Epc\\_relevant\\_increate\\_t0\\_download\\_v2\\_base&utm\\_relevant\\_index=2](https://blog.csdn.net/weixin_43912621/article/details/125170648?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-125170648-blog-54982632.235%5Ev35%5Epc_relevant_increate_t0_download_v2_base&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-125170648-blog-54982632.235%5Ev35%5Epc_relevant_increate_t0_download_v2_base&utm_relevant_index=2)