

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH



NHẬP MÔN THỊ GIÁC MÁY TÍNH
BÁO CÁO ĐỒ ÁN CUỐI KỲ

Giảng viên hướng dẫn: NGUYỄN VINH TIỆP

Lớp: CS231.N21

Sinh viên thực hiện:

Nguyễn Ngọc Thúc	MSSV: 21521506
Phạm Thanh Lâm	MSSV: 21520055
Lê Thị Như Ý	MSSV: 21522818
Nguyễn Thị Thùy	MSSV: 21521514

TP. Hồ Chí Minh – 26/06/2023

MỤC LỤC

Chương 1. TỔNG QUAN ĐỀ TÀI	2
Chương 2. MÔ TẢ BÀI TOÁN	3
Chương 3. THỰC HIỆN	4
Phần 1: Công việc chung của tất cả thành viên	4
Phần 2: Công việc của từng thành viên	4
1. Smile with the star filter	4
1.1. Cách filter hoạt động	5
1.2. Thực hiện filter	5
2. Cat in opening mouth filter	9
2.1. Giới thiệu filter	9
2.2. Quá trình thực hiện	10
3. Heart filter	16
3.1 Giới thiệu filter	16
3.2 Quy trình thực hiện	17
4. Dog filter snapchat	24
4.1. Giới thiệu Filter	24
4.2. Quy trình thực hiện	25
Chương 4. TỔNG KẾT	33

Chương 1. TỔNG QUAN ĐỀ TÀI

Trong cuộc sống hiện đại ngày nay, việc sử dụng các nền tảng mạng xã hội ngày càng trở nên phổ biến và quen thuộc. Theo đó, các ứng dụng mạng xã hội như Facebook, Instagram, Twitter... và đặc biệt là Tiktok dần dần đi vào cuộc sống của mỗi người. Cũng từ đó, các bộ filter độc lạ và nổi bật trở thành một công cụ không thể thiếu, giúp cho mọi người thể hiện cá tính riêng của mình trên thế giới ảo. Thậm chí còn có một số ứng dụng dùng filter làm mũi nhọn chiến lược cho nền tảng của mình như Snapchat, Snow, B612... điều này cho thấy nhu cầu to lớn và tính ứng dụng rộng rãi của chúng.

Nắm bắt được điều đó, nhóm chúng tôi quyết định chọn thực hiện **đề tài “Filter Instagram”** nhằm tạo ra những filter mới lạ, mang đậm tính cách của các thành viên trong nhóm. Thông qua đó, các thành viên có cơ hội để vận dụng các kiến thức đã học trong môn “Nhập môn Thị giác máy tính” đồng thời tìm hiểu sâu hơn và mở rộng các kiến thức của mình trong lĩnh vực Thị giác máy tính.

Đề tài của chúng tôi gồm có 4 bộ filter của mỗi thành viên, mỗi người sẽ tự sáng tạo filter theo phong cách của mình, sau đó tìm hiểu các công nghệ có liên quan và thực hiện chúng.

Mục tiêu của chúng tôi là vận dụng được kiến thức đã học và kết hợp thêm những công nghệ mới... để tạo ra những filter mang phong cách cá nhân nhưng đồng thời cũng phải đẹp và tự nhiên, dễ sử dụng.

Chương 2. MÔ TẢ BÀI TOÁN

Bài toán cần giải quyết: tạo filter realtime trên webcam máy tính.

Input: Các frame ảnh liên tục được lấy về từ webcam



Output: Các frame ảnh đã được gắn filter



Vấn đề chung của bài toán là filter-realtime. Do đó, bài toán sẽ được chia thành 2 vấn đề con là “realtime” và “filter”. “Filter” là khác nhau ở từng thành viên còn “realtime” sẽ sử dụng một mẫu chung.

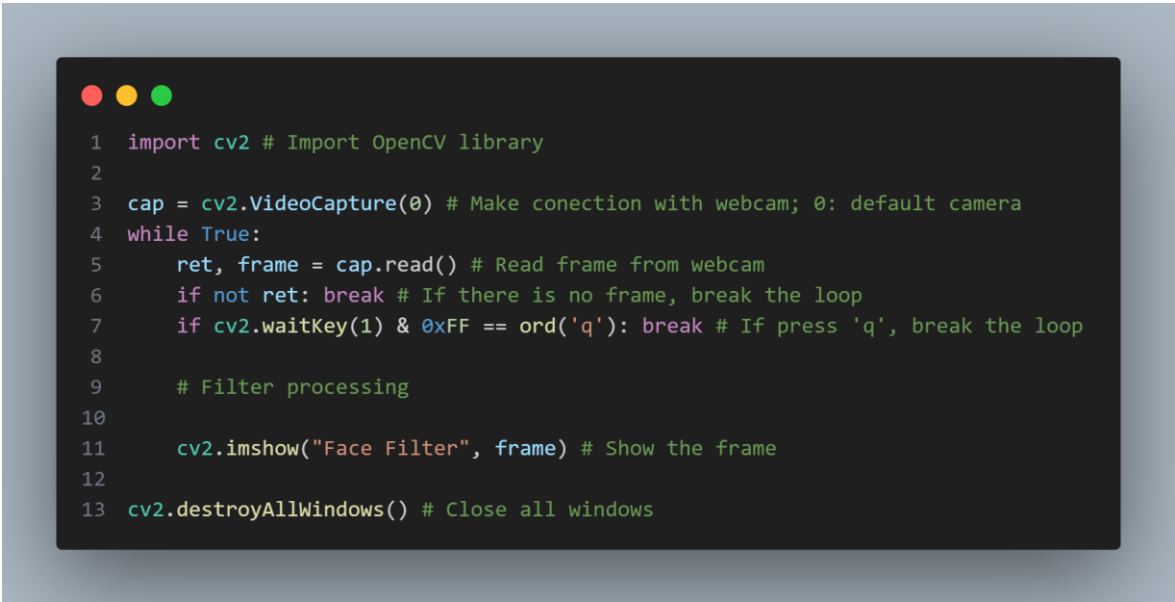
Chương 3. THỰC HIỆN

Phần 1: Công việc chung của tất cả thành viên

Giải quyết vấn đề con thứ nhất: realtime

Để có thể áp filter realtime thì có yêu cầu là filter phải đủ nhẹ và phải có một module làm nhiệm vụ liên tục lấy frame ảnh từ webcam sau đó xử lý và xuất frame ảnh đã được áp filter cho người dùng.

Source code của module này như sau:



```
1 import cv2 # Import OpenCV library
2
3 cap = cv2.VideoCapture(0) # Make connection with webcam; 0: default camera
4 while True:
5     ret, frame = cap.read() # Read frame from webcam
6     if not ret: break # If there is no frame, break the loop
7     if cv2.waitKey(1) & 0xFF == ord('q'): break # If press 'q', break the loop
8
9     # Filter processing
10
11     cv2.imshow("Face Filter", frame) # Show the frame
12
13 cv2.destroyAllWindows() # Close all windows
```

Biến ‘ret’ là một cờ hiệu cho biết việc đọc thành công hay không.

Phần 2: Công việc của từng thành viên

Khi đã có module xử lý vấn đề realtime thì công việc của từng thành viên là giải quyết vấn đề thứ 2: Tạo filter. 4 filter tương ứng với 4 thành viên lần lượt là: “Smile with the star filter”, “Cat in opening mouth filter”, “Hearts filter”, “Dog filter snapchat”. Cụ thể như sau:

1. Smile with the star filter

Thành viên thực hiện: Nguyễn Thị Thùy - 21521514

1.1. Cách filter hoạt động

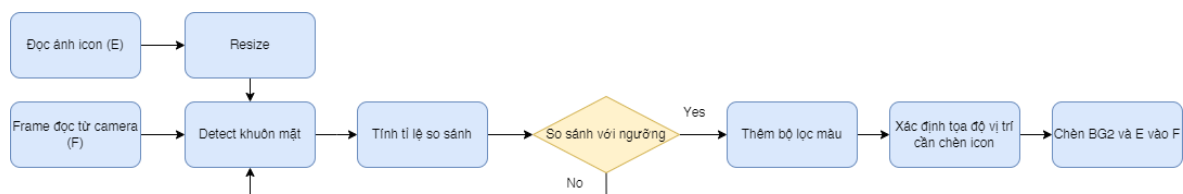
#ảnh minh họa

Filter sẽ nhận biết khi người dùng cười sẽ thêm icon vào vị trí đã xác định và thay đổi màu sắc.

Khi thực hiện filter này, chúng tôi gặp phải một vài thách thức như sau:

- Xác định một ngưỡng so sánh có thể ổn định nhất để nhận biết người dùng cười.
- Xác định bộ lọc màu phù hợp với màu sắc mà chúng tôi mong muốn.

1.2. Thực hiện filter



Pipeline bài toán

1.2.1. Facial landmark detection

Các thao tác để tạo facial landmark detection:

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

Chúng tôi sử dụng pre-train model là tệp “shape_predictor_68_face_landmarks.dat” để dự đoán các điểm đặc trưng trên khuôn mặt, bao gồm 68 điểm đặc trưng trên khuôn mặt.

Đầu tiên chúng ta cần khai báo thư viện OpenCV và dlib.

Sử dụng hàm `dlib.get_frontal_face_detector()` để tạo một bộ phát hiện khuôn mặt.

Sử dụng hàm `dlib.shape_predictor()` để tạo một bộ dự đoán các điểm đặc trưng trên khuôn mặt.

```
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = detector(frame)
```

Biến `gray_frame` sẽ chứa những frame được lấy ra từ camera thực đã chuyển đổi sang ảnh grayscale.

Biến `faces` chứa kết quả phát hiện khuôn mặt từ frame.

```
landmarks = predictor(gray_frame, face)
```

Biến `landmarks` chứa kết quả khuôn mặt đã được dự đoán các điểm đặc trưng.

1.2.2. Tạo bộ lọc màu

#ảnh minh họa

Các thao tác tạo bộ lọc màu:

```
def LookupTable(x, y):
    spline = UnivariateSpline(x, y)
    return spline(range(256))
def Summer(img):
    increaseLookupTable = LookupTable([0, 64, 128, 256], [0, 80, 160, 256])
    decreaseLookupTable = LookupTable([0, 64, 128, 256], [0, 50, 100, 256])
    blue_channel, green_channel, red_channel = cv2.split(img)
    red_channel = cv2.LUT(red_channel, increaseLookupTable).astype(np.uint8)
    blue_channel = cv2.LUT(blue_channel, decreaseLookupTable).astype(np.uint8)
    sum = cv2.merge((blue_channel, green_channel, red_channel ))
    return sum
```

Hàm `LookupTable(x,y)` tạo một đường cong spline từ các điểm dữ liệu (x,y), sử dụng hàm `UnivariateSpline()` trong thư viện `scipy.interpolate`. trả về đối tượng `spline`.

Hàm `Summer()` nhận đầu vào là một tấm ảnh và thực hiện tăng cường độ tương phản trên ảnh này. Đầu tiên, hàm này sử dụng hàm `cv2.split()` để tách các kênh màu của ảnh thành các ma trận riêng biệt: `blue_channel`, `green_channel` và `red_channel`.

Sau đó, hàm này sử dụng hai bảng tra cứu `increaseLookupTable` và `decreaseLookupTable` để áp dụng các phép biến đổi tăng cường độ tương phản cho hai kênh màu là kênh màu đỏ và màu xanh dương. Cụ thể, kênh màu đỏ được tăng cường độ tương phản bằng cách sử dụng bảng tra cứu `increaseLookupTable`, trong khi kênh màu xanh dương được tăng cường độ tương phản bằng cách sử dụng bảng tra cứu `decreaseLookupTable`.

Cuối cùng hàm `cv2.merge()` được sử dụng để ghép các kênh màu đã được tăng cường độ tương phản thành một ảnh mới.

1.2.3. Tính giá trị so sánh

Các thao tác tính giá trị so sánh:

```
#MOUTH IS SMILING
left_lip = (landmarks.part(48).x, landmarks.part(48).y)
right_lip = (landmarks.part(54).x, landmarks.part(54).y)
center_lip = (landmarks.part(66).x, landmarks.part(66).y)
lips_width = int(hypot(left_lip[0] - right_lip[0],
                       left_lip[1] - right_lip[1]) * 1.7)
lips_height = int(lips_width * 0.77)

# Calculate jaw width
left_jaw = (landmarks.part(3).x, landmarks.part(3).y)
right_jaw = (landmarks.part(13).x, landmarks.part(13).y)
jaw_width = int(hypot(left_jaw[0] - right_jaw[0],
                      left_jaw[1] - right_jaw[1]) * 1.7)
# Calculate the ratio of lips and jaw widths
ratio = lips_width/jaw_width
```

Đầu tiên ta xác định tọa độ các điểm đặc trưng trên khuôn mặt mà ta cần xét đến là các điểm: 48, 54 và 66.

Sau đó ta tính kích thước của độ rộng và chiều cao của miệng bằng cách dùng hàm `hypot()` là hàm dùng để tính độ dài của hai tọa độ.

Biến `lips_width` dùng để lưu kết quả độ rộng của miệng sau khi tính toán.

Biến `lips_height` dùng để lưu kết quả chiều cao của miệng sau khi được tính toán.

Để xác định được giá trị so sánh để nhận biết khi nào người dùng cười, ta cần xác định thêm một đại lượng nữa đó là `jaw_width` là độ rộng của hàm.

Ta xác định tọa độ của hai điểm 3 và 13. Sau đó tính độ dài của hai điểm này và lưu vào biến `jaw_width`.

Và biến `ratio` là giá trị so sánh với ngưỡng và nó được xác định bằng công thức:

$$\text{ratio} = \text{lips_width} / \text{jaw_width}$$

Công thức này được xác định dựa trên sự thay đổi của độ rộng miệng khi người dùng cười và độ rộng hàm thì không thay đổi.

1.2.4. Xác định ngưỡng so sánh

```
if ratio > 0.43 :
```

Để xác định ra ngưỡng so sánh là 0.43 chúng tôi dựa trên các giá trị thực tế là độ rộng của miệng so với độ rộng hàm 0.5. Do đó chúng tôi sẽ thử nghiệm với các ngưỡng gần bằng 0.5 để tìm ra ngưỡng tốt nhất có thể.

Các ngưỡng chúng tôi thử nghiệm là: 0.4, 0.43, 0.46, 0.5. Và mức độ cười mà chúng tôi mong muốn để filter nhận biết và thay đổi các đặc điểm là khi người dùng cười mỉm như ảnh trở lên.

#ảnh minh họa

Kết quả ghi nhận được từ các ngưỡng trên như sau:

0.4	Phải cười khá to thì filter mới nhận biết và khá chậm chạp
0.43	Nhận biết với đúng mức độ yêu cầu và ổn định
0.46	Với mức độ cười yêu cầu thì filter khá chậm chạp
0.5	Gần như không nhận biết được với mức độ cười như yêu cầu

1.2.5. Chèn icon

Các bước chèn icon vào vị trí xác định:

```
# Adding the new nose
nose_im = cv2.resize(star_image, (nose_width, nose_height))
nose_im_gray = cv2.cvtColor(nose_im, cv2.COLOR_BGR2GRAY)
_, nose_mask = cv2.threshold(nose_im_gray, 25, 255, cv2.THRESH_BINARY_INV)
nose_area = frame[top_left[1]: top_left[1] + nose_height,
                  top_left[0]: top_left[0] + nose_width] #[cao,rộng]
nose_area_no_nose = cv2.bitwise_and(nose_area, nose_area, mask=nose_mask)
final_nose = cv2.add(nose_area_no_nose, nose_im)
frame[top_left[1]: top_left[1] + nose_height,
      top_left[0]: top_left[0] + nose_width] = final_nose
```

Hình ảnh icon cần chèn là star_image.

Đầu tiên, biến `nose_im` lưu kết quả resize ảnh icon để có kích thước phù hợp với vùng mũi (`nose_width`, `nose_height`).

Biến `nose_im_gray` lưu kết quả chuyển đổi ảnh `nose_im` thành ảnh grayscale.

Biến `nose_mask` lưu kết quả tạo mặt nạ bằng các áp dụng ngưỡng cho ảnh grayscale. Mặt nạ này dùng để loại bỏ phần nền của hình ảnh khi được vẽ lên khuôn mặt.

Biến `nose_area` lưu kết quả khu vực mà icon sẽ được gắn lên khuôn mặt.

Biến `nose_area_no_nose` lưu kết quả khi đã loại bỏ phần nền của khu vực mũi cần chèn icon.

Biến `final_nose` lưu kết quả ghép `nose_area_no_nose` và hình ảnh icon.

Cuối cùng hiển thị `final_nose` lên frame.

2. Cat in opening mouth filter

Thành viên thực hiện: Lê Thị Như Ý - 21522818

2.1. Giới thiệu filter



Ban đầu, filter sẽ thay đổi background của người dùng.

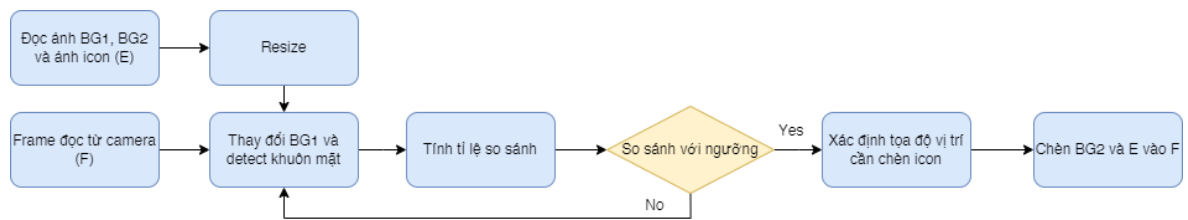
Khi người dùng mở miệng, filter sẽ thay đổi background một lần nữa và chèn thêm icon vui nhộn vào.

Đối với filter này, khi thực hiện chúng tôi có một số thách thức như sau:

- Cần xác định một ngưỡng so sánh phù hợp để nhận diện việc mở miệng của người dùng.

- Chỉnh sửa kích thước icon theo các kích thước của khuôn mặt người dùng một cách cân đối nhất có thể.

2.2. Quá trình thực hiện



Pipeline bài toán

2.2.1. Facial landmark detection



Thao tác thực hiện facial landmark detection:

```

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
  
```

Gọi hàm `get_frontal_face_detector()` của thư viện `dlib` để detect khuôn mặt.

- Khi gọi hàm `get_frontal_face_detector()` không cần bất kỳ tham số nào, khi gọi hàm sẽ trả về the pre-trained HOG + Linear SVM face detector trong thư viện `Dlib`.

Dùng `shape_predictor_68_face_landmarks.dat` là một pre-trained model của thư viện `Dlib` để tạo 68 điểm neo trên khuôn mặt.

```
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = detector(frame)
```

Đổi từ không gian màu BGR(Blue-Green-Red) sang Grayscale.

- Trong thư viện OpenCV, mặc định khi load ảnh từ file hoặc video sẽ lưu bằng không gian màu BGR.

```
landmarks = predictor(gray_frame, face)
```

Các tham số:

- detector: lưu hàm detect khuôn mặt
- predictor: lưu model tạo facial landmark
- gray_frame: kiểu numpy.array lưu kết quả trả về
- faces: lưu kết quả detect khuôn mặt
- landmarks: lưu kết quả 68 điểm neo đã facial landmark

2.2.2. Tách và thay đổi background



Thao tác thực hiện tách và thay đổi background:

```
from cvzone.SelfiSegmentationModule import SelfiSegmentation
```

Dùng CVzone là một package để xử lý hình ảnh, nó dùng thư viện OpenCV và MediaPipe.

Gọi module SelfiSegmentation để có thể xóa background và thay đổi background mới cùng lúc.

```
segmentor = SelfiSegmentation()
```

```
segmentated_img = segmentor.removeBG(frame, background, threshold=0.9)
```

Các tham số:

- segmentor: modele xóa và đổi background
- segmentated_img: lưu kết quả trả về khi đã xóa và đổi background mới
- frame: được lấy ra từ real camera (đối tượng cần xóa và đổi background)
- background: file ảnh đã được xử lý để dùng thay đổi background cho frame
- threshold: ngưỡng ta có thể thay đổi tùy theo mức độ thay đổi background ta mong muốn (ví dụ với threshold = 1.0 thì background mới sẽ che luôn cả người trên frame)

2.2.3. Tính ratio để so sánh

Các thao tác tính ratio để so sánh:

```
left_lip = (landmarks.part(48).x, landmarks.part(48).y)
right_lip = (landmarks.part(54).x, landmarks.part(54).y)
center_lip = (landmarks.part(66).x, landmarks.part(66).y)
lips_width = int(hypot(left_lip[0] - right_lip[0],
                        left_lip[1] - right_lip[1]) * 1.2)
lips_height = int(lips_width * 0.9)
lip49 = (landmarks.part(49).x, landmarks.part(49).y)
lip59 = (landmarks.part(59).x, landmarks.part(59).y)
A = int(hypot(lip49[0] - lip59[0],
              lip49[1] - lip59[1]) * 1.7)
lip53 = (landmarks.part(53).x, landmarks.part(53).y)
lip55 = (landmarks.part(55).x, landmarks.part(55).y)
B = int(hypot(lip53[0] - lip55[0],
              lip53[1] - lip55[1]) * 1.7)
ratio = (A+B)/(2.0*lips_width)
```

Gọi tọa độ (x,y) của các điểm neo bằng landmarks.part().x hoặc landmarks.part().y

Xác định tọa độ các điểm cần tính gồm có: 48, 54, 49, 59, 53, 55.

Dùng hàm hypot() để tính độ dài từ tọa độ đã xác định.

Các tham số:

- left_lip: lưu tọa độ điểm 48

- right_lip: lưu tọa độ điểm 54
- lip_width: lưu kích thước chiều rộng của miệng
- lip_height: lưu kích thước chiều cao của miệng
- lip49, lip59, lip53, lip55: lần lượt lưu các tọa độ của điểm 49, 59, 53, 55
- A: lưu độ dài của đoạn 49-59
- B: lưu độ dài của đoạn 53-55
- ratio: lưu ngưỡng so sánh đã được tính toán

Để lập nên công thức tìm ra ratio:

Ban đầu, chúng tôi sử dụng cách nhận biết người dùng mở miệng bằng cách dùng tỉ lệ giữa độ cao của miệng (lip_height) thay đổi so với độ rộng của miệng (lip_weight) không thay khi mở miệng.

$$\text{ratio} = \text{lip_height} / \text{lip_weight}$$

Chúng tôi nhận thấy với cách tính ratio này, khi người dùng mở miệng quá nhỏ thì khó nhận biết được và tùy khẩu hình miệng của người dùng mà cách tính ratio này không được chính xác nữa.



Mở miệng quá nhỏ khó thể nhận biết được

Do đó, chúng tôi quyết định sử dụng công thức:

$$\text{ratio} = (A+B) / (2.0 * \text{lip_width})$$

Với công thức này, độ cao của miệng sẽ được xác định bằng độ dài của 2 đoạn A và B, điều này giúp khắc phục hạn chế của công thức trước đó.

2.2.4. Xác định ngưỡng so sánh

```
if ratio > 0.5 :
```

Ở đây, chúng tôi xác định ngưỡng so sánh = 0.5. Để tìm ra con số này chúng ta đã thực hiện:

Đầu tiên, chúng tôi nhận thấy tổng độ dài của hai đoạn A và B gần bằng chiều rộng của miệng (lip_width) nên ratio sẽ nằm quanh con số 0.5 là tốt nhất.

Tiếp theo, chúng tôi thử nghiệm lần lượt các ngưỡng: 0.4, 0.45, 0.5, 0.55, 0.6 với các mức độ độ lớn khi mở miệng như sau:



Và mức độ chúng tôi mong muốn là 3 tức là khi người dùng mở miệng có độ lớn từ mức độ 3 trở lên thì filter mới xử lý các thay đổi.

Bảng ghi nhận hoạt động của các ngưỡng khác nhau

Ngưỡng	Ghi nhận
0.4	mức 1 đã chập chờn, thay đổi liên tục
0.45	mức 2 đã thay đổi
0.5	mức 3 sẽ thay đổi và ổn định
0.55	mức 3 sẽ thay đổi và chập chờn
0.6	mức 5 mới thay đổi một cách ổn định

2.2.5. Chèn icon

Các thao tác chèn icon vào frame:


```

lip_im = cv2.resize(cat_image, (lips_width, lips_height))
lip_im_gray = cv2.cvtColor(lip_im, cv2.COLOR_BGR2GRAY)
_, lip_mask = cv2.threshold(lip_im_gray, 25, 255, cv2.THRESH_BINARY_INV)
lip_area = frame[top_leftlip[1]: top_leftlip[1] + lips_height,
                 top_leftlip[0]: top_leftlip[0] + lips_width]
lip_area_no_lip = cv2.bitwise_and(lip_area, lip_area, mask=lip_mask)
final_lip = cv2.add(lip_area_no_lip, lip_im)
frame[top_leftlip[1]: top_leftlip[1] + lips_height,
      top_leftlip[0]: top_leftlip[0] + lips_width] = final_lip

```

Thay đổi kích thước của ảnh được chọn làm icon theo kích thước của miệng bằng cách dùng hàm `resize()`.

Tạo ảnh grayscale bằng cách dùng hàm `cvtColor()`.

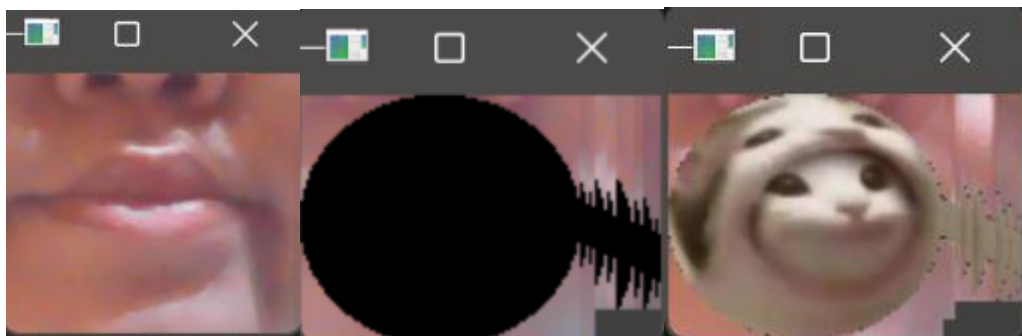
Tạo mask bằng ảnh binary dùng hàm `threshold()`.

Xác định vùng thêm icon.

Dùng hàm `bitwise_and` để bôi đen vùng icon trên `lip_area`.

Dùng hàm `add()` để thêm ảnh icon vào vùng đã bôi đen.

Cuối cùng cho vùng frame với tọa độ giống `lip_area` thành `final_lip`.



lip_area

lip_area_no_lip

final_lip

Các tham số:

- `lip_im`: lưu kết quả ảnh icon đã resize theo kích thước của miệng.
- `lip_im_gray`: lưu kết quả chuyển đổi sang grayscale của `lip_im`.
- `lip_mask`: lưu kết quả chuyển đổi sang binary của `lip_im_gray`.
- `lip_area`: lưu kết quả vùng cần thao tác trên frame.
- `lip_area_no_lip`: bôi đen (black out) vùng có icon trên `lip_area`.
- `final_lip`: lưu kết quả khi cộng `lip_area_no_lip` và `lip_im`.

3. Heart filter

Thành viên thực hiện: Phạm Thanh Lâm - 21520055

3.1 Giới thiệu filter

Đầu tiên, filter sẽ thực hiện thay đổi background trên từng frame ảnh của webcam bằng ảnh background mà người dùng đưa vào.

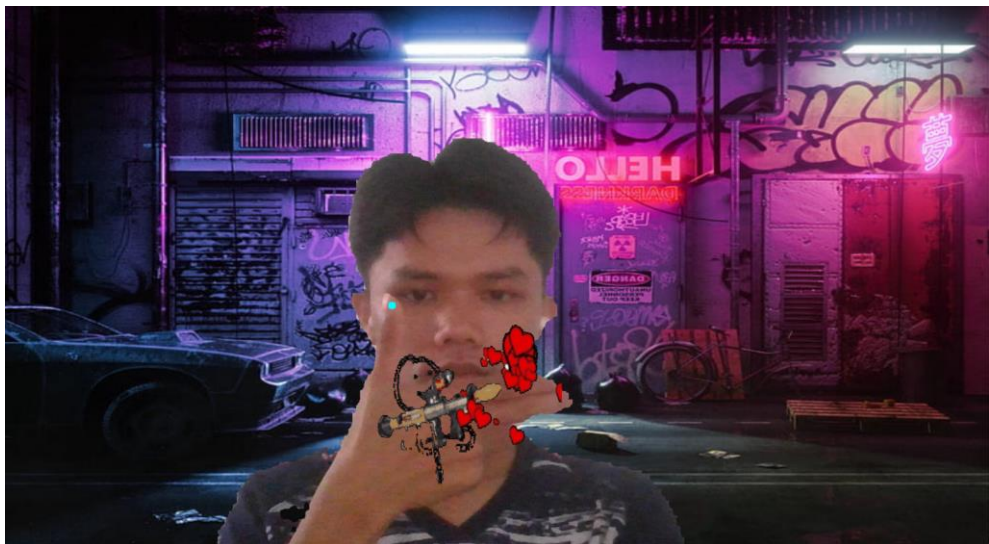
Khi người dùng đưa tay lên webcam, filter sẽ phát hiện bàn tay của người dùng và thực hiện một số thao tác tạo hiệu ứng khác nhau ứng với trường hợp là cả 2 bàn tay được phát hiện hoặc một trong 2 bàn tay.



Filter khi phát hiện 2 bàn tay



Filter khi phát hiện bàn tay phải



Filter khi phát hiện bàn tay trái

3.2 Quy trình thực hiện

Bước 1: Tìm ảnh background và các hiệu ứng để thực hiện tạo filter.

Background:



Bộ hiệu ứng gồm có 3 gif:



IU



Heart



Shot-heart

Bước 2: Thực hiện thay đổi background

- Đọc ảnh background từ tệp mà người dùng cung cấp và điều chỉnh kích thước để cùng kích thước với ảnh đầu vào từ webcam
- Sử dụng đối tượng “segmentor” để xóa nền của ảnh đầu vào bằng cách gọi hàm ‘removeBG(image, background_image, threshold = 0.9)

Trong đó:

- + image: là ảnh đầu vào, từng frame từ webcam
 - + background_image: là ảnh nền
 - + threshold: là ngưỡng xóa nền
- Trả về ảnh đã xử lý và ảnh nền được thay đổi

Bước 3: Tạo filter

Tạo đối tượng `selfie_segmentation` để thực hiện việc phân đoạn chính mình trong hình ảnh.

Khởi tạo đối tượng `detector` để phát hiện các bàn tay trong khung hình.

Nếu chỉ có 1 tay được phát hiện trong khung hình:

- Lấy khung hình của hiệu ứng filter từ chuỗi hình ảnh `gif0` và `gif1`
- Thay đổi kích thước khung hình của hiệu ứng filter để phù hợp với kích thước khung hình đầu vào
- Áp dụng hiệu ứng filter 0 và 1 lên khung hình bằng cách gọi hàm `sticker1(frame, gif0_frame, gif1_frame, hands)`
- Trong đó:
 - `frame`: là khung hình đầu vào
 - `gif0_frame`: là khung hình hiệu ứng filter 0
 - `gif1_frame`: là khung hình hiệu ứng filter 1
 - `hands`: là danh sách các tay được phát hiện
- Hàm `sticker1` hoạt động như sau:

```
if len(hands) == 2:  
    return frame  
hand = hands[0]
```

- Kiểm tra số lượng tay được phát hiện trong danh sách `hands`. Nếu có hai tay được phát hiện, không thực hiện bất kỳ hiệu ứng filter nào và trả về khung hình gốc.
- Nếu chỉ có một tay được phát hiện, lấy tay đó từ danh sách `hands`.

```
thumb_tip = hand["lmList"][4][:2]  
pinky_mcp = hand["lmList"][17][:2]  
index_finger_tip = hand["lmList"][8][:2]  
distance = math.sqrt((pinky_mcp[0] - thumb_tip[0]) ** 2 + (pinky_mcp[1] - thumb_tip[1]) ** 2)  
  
# Vẽ hình tròn tại thumb tip  
cv2.circle(frame, thumb_tip, 5, (255, 255, 0), -1)  
effect0_copy = cv2.resize(effect0, (int(distance), int(distance)))  
effect1_copy = cv2.resize(effect1, (int(distance), int(distance)))
```

- Xác định các điểm quan trọng trên tay:

- Thump tip: Đầu ngón cái.
- Pinky MCP: Điểm gốc ngón út.
- Index finger tip: Đầu ngón trỏ.
- Tính toán khoảng cách giữa pinky_mcp và thump_tip để xác định kích thước của hiệu ứng filter.
- Vẽ hình tròn tại thump_tip để trực quan hóa vị trí của ngón cái trên khung hình.
- Tạo bản sao của hiệu ứng filter 0 và hiệu ứng filter 1 với kích thước phù hợp dựa trên khoảng cách tính được.
- Dựa vào vị trí của thump_tip và index_finger_tip, xác định các điểm góc tạo thành một hình chữ nhật xác định vùng áp dụng hiệu ứng filter trên khung hình và hiệu ứng nào sẽ áp dụng với trường hợp hướng các đầu ngón tay trở về bên phải hay trái

```
# Tạo matrix transform từ 4 điểm của sticker và frame
pts_sticker = np.float32([[0, 0], [effect0_copy.shape[1], 0], [0, effect0_copy.shape[0]],
                          [effect0_copy.shape[1], effect0_copy.shape[0]]])
pts_frame = np.float32([[x_min, y_min], [x_max, y_min], [x_min, y_max], [x_max, y_max]])
matrix = cv2.getPerspectiveTransform(pts_sticker, pts_frame)

# Áp dụng matrix transform để đưa sticker vào frame
sticker_warped = cv2.warpPerspective(effect0_copy, matrix, (frame.shape[1], frame.shape[0]))

# Tạo mask từ sticker_warped để áp dụng blend
sticker_mask = cv2.cvtColor(sticker_warped, cv2.COLOR_BGR2GRAY)
_, sticker_mask = cv2.threshold(sticker_mask, 1, 255, cv2.THRESH_BINARY)

# Áp dụng blend
foreground = cv2.bitwise_and(sticker_warped, sticker_warped, mask=sticker_mask)
background = cv2.bitwise_and(frame, frame, mask=cv2.bitwise_not(sticker_mask))
frame = cv2.add(foreground, background)
```

- Tạo ma trận biến đổi (matrix) từ tọa độ của hiệu ứng filter và khung hình.
- Áp dụng ma trận biến đổi để đưa hiệu ứng filter vào khung hình.
- Tạo mask từ hiệu ứng filter đã được biến đổi để áp dụng blend.
- Áp dụng blend giữa hiệu ứng filter và khung hình ban đầu bằng cách sử dụng toán tử bitwise_and và toán tử bitwise_not.
- Trả về khung hình sau khi đã áp dụng hiệu ứng filter.

Nếu cả 2 bàn tay được phát hiện:

- Lấy khung hình của hiệu ứng filter 2 từ chuỗi hình ảnh gif2.

- Thay đổi kích thước khung hình của hiệu ứng filter 2 để phù hợp với kích thước khung hình đầu vào.
- Áp dụng hiệu ứng filter 2 lên khung hình bằng cách gọi hàm `sticker2(frame, gif2_frame, hands)`.
- Trong đó:
 - `frame`: là khung hình đầu vào
 - `gif2_frame`: là khung hình của hiệu ứng filter 2
 - `hands`: là danh sách các tay được phát hiện.
- Hàm `sticker2` hoạt động như sau:

```
if len(hands) != 2:
    return frame
hand_left = hands[0]
hand_right = hands[1]
```

- Kiểm tra số lượng tay được phát hiện trong danh sách `hands`. Nếu không có hai tay được phát hiện, không thực hiện bất kỳ hiệu ứng filter nào và trả về khung hình gốc.
- Nếu có hai tay được phát hiện, lấy thông tin của tay trái từ `hand_left` và tay phải từ `hand_right`.

```
pinky_mcp_left = hand_left["lmList"][17][:2]
pinky_mcp_right = hand_right["lmList"][17][:2]

# Vẽ hình tròn tại pinky tip trái và phải
cv2.circle(frame, pinky_mcp_left, 5, (255, 0, 255), -1)
cv2.circle(frame, pinky_mcp_right, 5, (255, 0, 255), -1)

# Tính khoảng cách giữa hai tọa độ pinky tip
distance = math.sqrt((pinky_mcp_right[0] - pinky_mcp_left[0]) ** 2 + (pinky_mcp_right[1] - pinky_mcp_left[1]) ** 2)
```

- Xác định các điểm quan trọng trên tay trái và tay phải:
- Pinky MCP left: Điểm gốc ngón út tay trái.
- Pinky MCP right: Điểm gốc ngón út tay phải.
- Tính toán khoảng cách giữa `pinky_mcp_left` và `pinky_mcp_right` để xác định kích thước của hiệu ứng filter.
- Vẽ hình tròn tại `pinky_mcp_left` và `pinky_mcp_right` để trực quan hóa vị trí của ngón út trên khung hình.

```

effect_copy = cv2.resize(effect, (int(distance), int(distance)))

center_x = int((pinky_mcp_left[0] + pinky_mcp_right[0]) / 2)
center_y = int((pinky_mcp_left[1] + pinky_mcp_right[1]) / 2)

x_min = int(center_x - distance / 2)
y_min = int(center_y - distance / 2)
x_max = int(x_min + distance)
y_max = int(y_min + distance)

```

- Tạo bản sao của hiệu ứng filter 2 với kích thước phù hợp dựa trên khoảng cách tính được.
- Dựa vào vị trí của pinky_mcp_left và pinky_mcp_right, xác định điểm giữa giữa hai tay để tạo một vùng áp dụng hiệu ứng filter trên khung hình.

```

# Tạo matrix transform từ 4 điểm của sticker và frame
pts_sticker = np.float32([[0, 0], [effect_copy.shape[1], 0], [0, effect_copy.shape[0]],
                          [effect_copy.shape[1], effect_copy.shape[0]]])
pts_frame = np.float32([[x_min, y_min], [x_max, y_min], [x_min, y_max], [x_max, y_max]])
matrix = cv2.getPerspectiveTransform(pts_sticker, pts_frame)

# Áp dụng matrix transform để đưa sticker vào frame
sticker_warped = cv2.warpPerspective(effect_copy, matrix, (frame.shape[1], frame.shape[0]))

# Tạo mask từ sticker_warped để áp dụng blend
sticker_mask = cv2.cvtColor(sticker_warped, cv2.COLOR_BGR2GRAY)
_, sticker_mask = cv2.threshold(sticker_mask, 1, 255, cv2.THRESH_BINARY)

# Áp dụng blend
foreground = cv2.bitwise_and(sticker_warped, sticker_warped, mask=sticker_mask)
background = cv2.bitwise_and(frame, frame, mask=cv2.bitwise_not(sticker_mask))
frame = cv2.add(foreground, background)

```

- Tạo ma trận biến đổi (matrix) từ tọa độ của hiệu ứng filter và khung hình.
- Áp dụng ma trận biến đổi để đưa hiệu ứng filter vào khung hình.
- Tạo mask từ hiệu ứng filter đã được biến đổi để áp dụng blend.
- Áp dụng blend giữa hiệu ứng filter và khung hình ban đầu bằng cách sử dụng toán tử bitwise_and và toán tử bitwise_not.
- Trả về khung hình sau khi đã áp dụng hiệu ứng filter.

Hiển thị khung hình gốc nếu không có tay nào được phát hiện.

Lặp lại các bước trên cho từng khung hình trong video hoặc cho đến khi người dùng nhấn phím 'q' để thoát.

4. Dog filter snapchat

4.1. Giới thiệu Filter

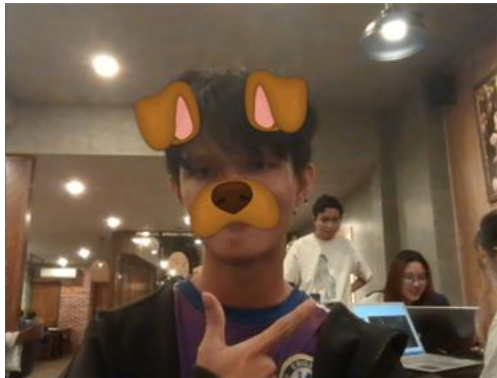
Thành viên thực hiện: Nguyễn Ngọc Thức - 21521506

Cá tính: đơn giản, tiện dụng

Ý tưởng: được lấy cảm hứng từ bộ filter cùng tên của Snapchat, filter này được tạo ra là một phiên bản “mộc mạc” hơn, tuy đơn giản nhưng phù hợp với gương mặt nam giới hơn bản gốc.



Filter gốc



Filter của chúng tôi

Filter sẽ gắn tai và mũi trong mọi trường hợp. Khi mở miệng ra thì sẽ hiển thị thêm lưỡi dưới dạng gif cho đến khi nào khép miệng lại và gif lưỡi chạy xong thì thôi.

4.2. Quy trình thực hiện

Bước 1: Tìm source icon lấy landmark và gán điểm dữ liệu

Các icon được chọn để tạo filter là 1 bộ 2 ảnh và 1 gif 4 kênh màu gồm có:



Dog_ears.png



Dog_nose.png



Dog_tongue.gif

Bộ filter này sẽ sử dụng thư viện Mediapipe được cung cấp miễn phí bởi Google để nhận diện các điểm trên gương mặt, nó sẽ phát hiện được 468 điểm trên mỗi gương mặt. Tuy nhiên, chúng tôi chỉ sử dụng 75 điểm có liên quan tới mắt, mũi, miệng để sử dụng.

Bảng mapping từ bộ 468 sang bộ 75 landmark như sau:

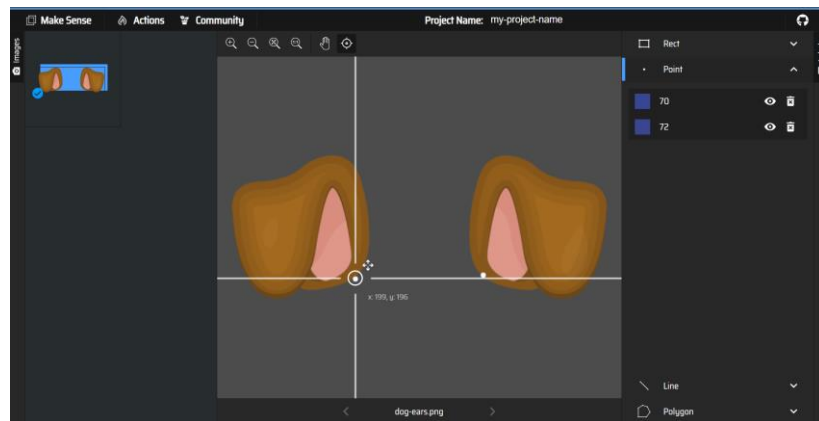
```
1 selected_keypoint_indices = [127, 93, 58, 136, 150, 149, 176, 148, 152, 377, 400, 378, 379, 365, 288, 323, 356, 70, 63, 105, 66, 55,  
2 285, 296, 334, 293, 300, 168, 6, 195, 4, 64, 60, 94, 290, 439, 33, 160, 158, 173, 153, 144, 39, 385,  
3 387, 466, 373, 380, 61, 40, 39, 0, 269, 270, 291, 321, 405, 17, 181, 91, 78, 51, 13, 311, 306, 402, 14,  
4 178, 162, 54, 67, 10, 297, 284, 389]
```

75 landmarks này được hiển thị như sau:

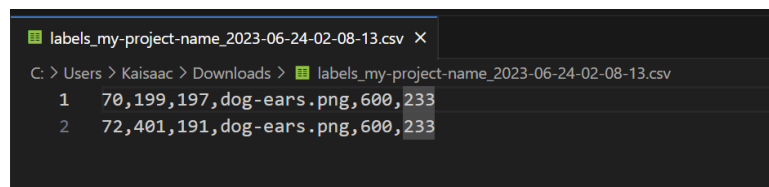


Ta sẽ gắn tai vào Landmark 70, 72, mũi vào 31, 35, lưỡi vào vị trí 59, 55.

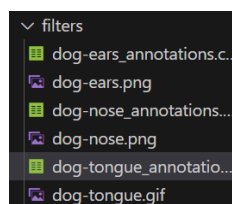
Gán landmark tương ứng với tọa độ trên từng filter bằng công cụ makesense.ai:



Sau khi gán xong, ta sẽ được file csv như sau:



Các giá trị lần lượt là landmark, 2 số tiếp theo là tọa độ của pixel tương ứng với landmark đó, tên file ảnh, kích thước của file ảnh đó. Làm tương tự với ảnh mũi, lưỡi.



Ta được folder như trên ảnh, bước đầu tiên đến đây là hoàn tất.

Bước 2: Tạo cấu trúc dữ liệu chứa các file filter trong 2 trường hợp: có lưỡi và không có lưỡi.

```

1 filters = [
2   {
3     'animated': False, # animated is boolean use if filter dont have tongue
4     'data': {
5       'alpha': alpha of image,
6       'points': {first_landmark: (x of first_landmark, y of first_landmark), second_landmark: (x of second_landmark, y of second_landmark)}
7     }
8   }
9   {
10    'animated': True, # if animated is True, datas is list of data
11    'datas': [ # datas is list of data
12      {
13        'alpha': alpha of image,
14        'points': {first_landmark: (x of first_landmark, y of first_landmark), second_landmark: (x of second_landmark, y of second_landmark)}
15      }
16    ]
17  }
18 ]
19
20 ]

```

Cấu trúc dữ liệu này cho phép phân biệt giữa 2 trường hợp có lưỡi (animated) và không có lưỡi (not animated)

Nếu có lưỡi thì filter được load nhiều lần, tương ứng với từng frame của dog_tongue.gif.

Việc tạo ra một cấu trúc dữ liệu để lưu tất cả phiên bản và thông tin của các file filter sẽ tạo nên sự dễ dàng và thuận tiện hơn cho việc xử lý trên từng frame ảnh. Đồng thời cũng dễ dàng mở rộng chức năng như: gắn nhiều thành phần hơn vào một filter hay xử lý nhiều filter. Tuy nhiên, lưu trữ như vậy chỉ phù hợp để xử lý số lượng filter nhỏ vì nó sử dụng nhiều tài nguyên lưu trữ.

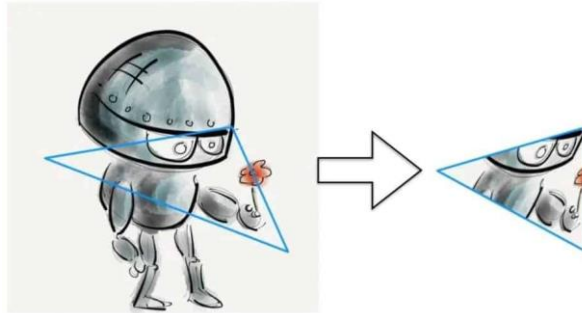
Bước 3: Gắn filter vào từng frame

Đây là bước quan trọng nhất, chúng tôi sẽ gắn filter vào khuôn mặt với các điểm landmark đã được detect trong từng frame ảnh. Tuy nhiên, nếu chỉ đơn giản gắn các ảnh filter vào frame ảnh thì sẽ bị “giả trân” do không tạo được hiệu ứng nghiêng khi nghiêng đầu mà chỉ có thể gắn với khuôn mặt nhìn trực diện. Cộng với việc filter và frame không có sự chuyển tiếp mượt mà. Làm cho filter trông có vẻ không ổn.

Từ những vấn đề trên, chúng tôi đã sử dụng hai kỹ thuật là Affine Transform và Gaussian Blur để xử lý.

Affine Transform là một phép biến đổi tuyến tính được sử dụng để thay đổi hình dạng, vị trí và tỷ lệ của một đối tượng trong không gian hai chiều. Phép biến đổi

affine bao gồm một tập hợp các phép biến đổi căn bản như dịch chuyển (translation), co giãn (scaling), quay (rotation) và cắt (shearing).



Ta lần lượt áp dụng Affine Transform cho từng frame và kênh alpha của frame đó.

```
1 trans_flt = cv2.warpAffine(img, tform, (frame.shape[1], frame.shape[0]))  
2 trans_alpha = cv2.warpAffine(img_alpha, tform, (frame.shape[1], frame.shape[0]))
```

Kết quả như sau:



Trước khi áp dụng Affine Transform



Sau khi áp dụng Affine Transform

Có thể thấy, tỷ lệ, độ biến dạng to nhỏ của filter được thay đổi cho phù hợp với góc nghiêng của khuôn mặt.



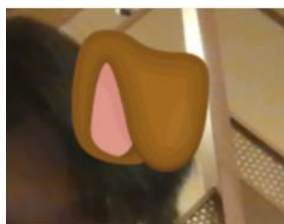
Khi nghiêng đầu xuống thì tai to hơn mũi

Sau đó sử dụng Gaussian Blur trên kênh alpha của frame để làm giảm sự khác biệt giữa filter và frame ảnh:

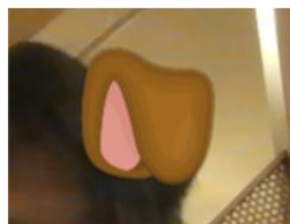
```
1 mask1 = cv2.GaussianBlur(mask1, (3, 3), 10)
2 mask2 = (255.0, 255.0, 255.0) - mask1
```

mask1 là kênh alpha của filter, mask2 là kênh alpha của frame

So sánh sự khác biệt giữa dùng và không dùng Gaussian Blur như sau:



Trước



Sau

Có thể thấy sau khi áp dụng kỹ thuật Blur trên ảnh mask thì khi sử dụng alpha blending để ghép frame và filter vào thì sẽ có sự chuyển tiếp nhẹ nhàng hơn, làm cho kết quả trong thật hơn.

Hai kĩ thuật trên sẽ được xử lý cho cả 3 thành phần của filter gồm tai, mũi, lưỡi (nếu có). Phần lưỡi chỉ xuất hiện khi mở miệng, vậy nên ta cần một thuật toán đơn giản để nhận biết khi nào thì miệng đang mở:

```
1 def mouth_is_open(points):
2     upper_lip=np.array(points[62])
3     lower_lip=np.array(points[66])
4     middle_nose=np.array(points[33])
5     return np.linalg.norm(upper_lip-lower_lip) >= 0.5*np.linalg.norm(upper_lip-middle_nose)
```

Hàm nhận biết miệng có đang mở không

Hàm này sẽ tính khoảng cách euclid giữa môi trên và môi dưới (1); môi trên và giữa mũi (2). Nếu (1) \geq 0.5 (2) thì trả về True.



Khi mở miệng, hàm trả về True, khi đó dog_tongue.gif cũng xuất hiện.

Bước 4: Lồng filter vào module lấy frame ảnh từ webcam đã thực hiện ở phần “công việc chung”


```

1 cap = cv2.VideoCapture(0)
2 filters = lb.load_filter()
3 i=0
4 f=False
5 while True:
6     ret, frame = cap.read()
7     if not ret: break
8     if cv2.waitKey(1) & 0xFF == ord('q'): break
9
10    # frame = load_frame()
11    points = lb.getLandmarks(frame)
12
13    if not points or (len(points) != 75): continue
14
15    for filter in filters:
16        if filter['animated'] == False:
17            frame = lb.apply_filter(frame,filter['data'],points)
18        else:
19            if lb.mouth_is_open(points) or f:
20                frame = lb.apply_filter(frame,filter['datas'][i],points)
21                f=True
22                i += 1
23                if i+1==len(filter['datas']):
24                    f=False
25                    i=0
26
27    cv2.imshow("Face Filter", frame)
28
29    cv2.destroyAllWindows()

```

Tổng hợp các phần đã làm được ở các bước trên và ta được kết quả là filter được ghép vào từng frame ảnh được lấy realtime từ webcam.

Chương 4. TỔNG KẾT

Trong suốt thời gian thực hiện **đề tài Filter Instagram** nhóm đã vận dụng các kiến thức của môn học đã được dạy, kết hợp thêm nhiều kỹ thuật mới để hoàn thành đề tài của mình. Với đề án này chúng tôi đã thực hiện bốn filter với những chức năng khác nhau, giúp người dùng làm đẹp và có những bức ảnh sinh động hơn bằng cách thêm icon, thay đổi background, màu ảnh và giúp người dùng tương tác với filter. Tuy nhiên so sánh với các filter của nhiều ứng dụng chụp ảnh nổi tiếng hiện nay thì sản phẩm của chúng tôi vẫn còn một số hạn chế. Chúng tôi hi vọng rằng, những sản phẩm đã làm được trong đề án này có thể được hoàn thiện hơn trong tương lai và có thể được phát hành, sử dụng một cách rộng rãi.

TÀI LIỆU THAM KHẢO

[Warp one triangle to another using OpenCV \(C++ / Python \) | LearnOpenCV #mediapipe/examples/face_landmarker/python at main · googlesamples/mediapipe · GitHub](#)

[Thi giác máy tính với OpenCV-Python Bài 2, Phần 2: Thao tác cơ bản với video \(imc.org.vn\)](#)

[Real-Time Background Replacement using OpenCV and CVzone \(analyticsvidhya.com\)](#)

[Không gian màu - Color space | MMLab UIT](#)