YOU ARE BEAUTIFUL

OUR PYTHON PROJECT PAPER

TEAM: URBEAUTIFUL

Tiffany Yen, Luhan Shen, Yueping Li, Xiaoyue Shen

Table of Contents

You Are Beautiful

Team: URBeautiful

Abstract

‘You Are Beautiful’ is an interesting project combining both applicability in real life and deep research into various python knowledge. The project required a combination of different programming skills to create a simplified Meitu app and B612. This combination enables users not only to add stickers and do makeups on the photos, but also automatically beautifies peoples’ faces through the users’ camera in real time.

Reason for Choosing This Topic

According to Zhicha Big Data, Beauty & Filter Camera users have exceeded 100 million people since 2017. People who are on amazing holidays, dining at fancy restaurants and cafes, or even just socializing with friends all want to use their phones to capture their special moments. Today, it is rare to see a picture that is not decorated with stylish and/or funny stickers and edited through beauty apps to make the users’ face handsome or beautiful.

Living in an era with widespread electronics and advanced technology, it is commonplace to see people holding up their phones and taking selfies everywhere. There is no doubt that people always have the desire to record photos where they are beautiful and charming. Since our group is made up of all females, we agreed that a project relating to beauty and photography would be interesting for all of us. Thus, the idea of a simplified beauty cam application emerged.

Introduction to Libraries

(1) face_recognition

This is a python library to recognize and manipulate the faces. The model has an accuracy of 99.38% on the Labeled Faces in the Wild benchmark.

Basic functions: Locate the faces in the pictures; Get locations and Outlines each user’s facial features.

(2) PIL(Python Imaging Library)

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

Basic functions: Image archives; Image display; Image processing

(3) OpenCV:

Open Source Computer Vision Library was released under a BSD license and hence it's free for both academic and commercial use. OpenCV was designed for computational efficiency and with a strong focus in real-time applications.

Recognition of Face and Facial Features

(1) Prepare the library

```
import face_recognition
```

(2) Load the file into a numpy array

```
image = face_recognition.load_image_file("picture.jpg")
```

(3) Get locations and outlines of the facial features and store the values into a dictionary.

```
face_landmarks_list = face_recognition.face_landmarks(image)
```

In this list, each face is stored as an element. The elements are then typed as dictionaries where values represent locations and key names are facial features such as 'chin', 'left_eyebrow'.

Makeup

(1) Prepare the library

Using Image module and ImageDraw module of PIL to process the pictures.

```
from PIL import Image, ImageDraw
```

(2) Use for-loop to beautify each face recognized

```
for face_landmarks in face_landmarks_list:
    pil_image = Image.fromarray(image)
    d = ImageDraw.Draw(pil_image, 'RGBA')
```

Create d as the object being drawn.

RGBA means 'red', 'green', 'blue' and 'alpha'.

(3) Drawing eyebrows

```
d.polygon(face_landmarks['left_eyebrow'], fill=(68, 54, 39, 128))
d.polygon(face_landmarks['right_eyebrow'], fill=(68, 54, 39, 128))
d.line(face_landmarks['left_eyebrow'], fill=(68, 54, 39, 150), width=5)
d.line(face_landmarks['right_eyebrow'], fill=(68, 54, 39, 150), width=5)
```

d.polygon(**xy, options**) : The polygon outline consists of straight lines between the given coordinates, plus a straight line between the first and last coordinates, therefore drawing a rough outline of eyebrows. The fill option gives the colors to use for the interior. This function fills the inside of the polygon with colors.

d.line(**xy, options**) : It draws a line between the coordinates in the xy list. The **width** option gives the line width. This function draws the upper outline of eyebrows.

(4) Glossing the lips

```
d.polygon(face_landmarks['top_lip'], fill=(150, 0, 0, 128))
d.polygon(face_landmarks['bottom_lip'], fill=(150, 0, 0, 128))
d.line(face_landmarks['top_lip'], fill=(150, 0, 0, 64), width=8)
d.line(face_landmarks['bottom_lip'], fill=(150, 0, 0, 64), width=8)
```

This step applies lip gloss on the lips. By changing the fill options, we can alter the colors of lipstick and the thinness and thickness of the application, meeting different needs for make-ups.

(5) Sparkling the eyes

```
d.polygon(face_landmarks['left_eye'], fill=(255, 255, 255, 30))
d.polygon(face_landmarks['right_eye'], fill=(255, 255, 255, 30))
```

Through the polygon function, the eyes are sparkled. Users have the option to switch the colors of their eyes if they want. Notice that this code cannot separate the white and black parts of eyes, thus changing the fill options would probably make eyes look weird.

(6) Applying eyeliners

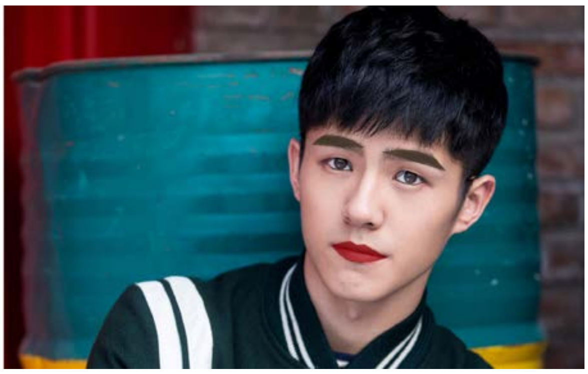
```
d.line(face_landmarks['left_eye'] + [face_landmarks['left_eye'][0]], fill=(0, 0, 0, 110), width=6)
d.line(face_landmarks['right_eye'] + [face_landmarks['right_eye'][0]], fill=(0, 0, 0, 110), width=6)
```

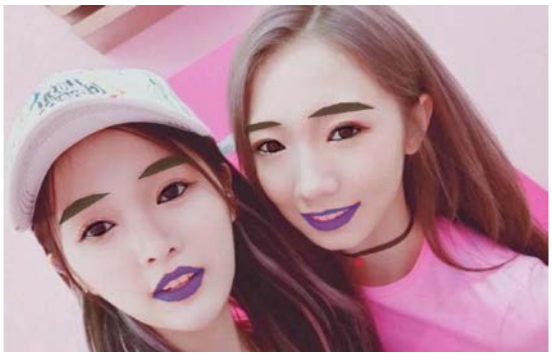
The code above helps draw in black eyeliners around the eyes. If users want to try brown eyeliner or smoky eyes, switching the numbers in fill options and width will work.

(7) display and save the images processed

```
pil_image.show()
pil_image.save('pictures/beautiful.jpg')
```

The result is shown below.





Makeup on People in Motion

Apart from makeups on faces appearing in the still pictures, we also manage to beautify people's faces in motion—specifically in video and in real time.

(1) Prepare the library

```
import face_recognition
from PIL import Image, ImageDraw
import cv2
import numpy as np
```

In this case, we would need face_recognition to detect and recognize faces and features and PIL to do concrete makeups just like what we need in static pictures. cv2 and numpy are used to process the videos.

(2) Split and read the video

```
def makeup(videofile):
    cap = cv2.VideoCapture(videofile)
    while(1):
        ret,frame = cap.read()
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        face_landmarks_list = face_recognition.face_landmarks(image)
        cv2.namedWindow("face", 0)
```

Before processing the faces, we will first use while-loop to read each frame of the video， turn them into RGB mode and recognize the faces. Then, we will create a window named “face” to finally show the pictures processed.

(3) Apply the makeups

```
pil_image = Image.fromarray(image)
d = ImageDraw.Draw(pil_image, 'RGBA')
for face_landmarks in face_landmarks_list:
    d.polygon(face_landmarks['left_eyebrow'], fill=(68, 54, 39, 180))
    d.polygon(face_landmarks['right_eyebrow'], fill=(68, 54, 39, 180))
    d.line(face_landmarks['left_eyebrow'], fill=(68, 54, 39, 150), width=3)
    d.line(face_landmarks['right_eyebrow'], fill=(68, 54, 39, 150), width=3)

    d.polygon(face_landmarks['top_lip'], fill=(150, 0, 0, 150))
    d.polygon(face_landmarks['bottom_lip'], fill=(150, 0, 0, 150))
    d.line(face_landmarks['top_lip'], fill=(150, 0, 0, 80), width=5)
    d.line(face_landmarks['bottom_lip'], fill=(150, 0, 0, 80), width=5)

    d.polygon(face_landmarks['left_eye'], fill=(255, 255, 255, 10))
    d.polygon(face_landmarks['right_eye'], fill=(255, 255, 255, 10))

    d.line(face_landmarks['left_eye'] + [face_landmarks['left_eye'][0]],fill=(0, 0, 0, 80),width=4)
    d.line(face_landmarks['right_eye'] + [face_landmarks['right_eye'][0]],fill=(0, 0, 0, 80),width=4)
```

This part is the same as the process in the static pictures.

(4) Present the outcome

```
        img = cv2.cvtColor(np.asarray(pil_image), cv2.COLOR_RGB2BGR)
        cv2.imshow("face", img)
        cv2.waitKey(1)
    cap.release()
```

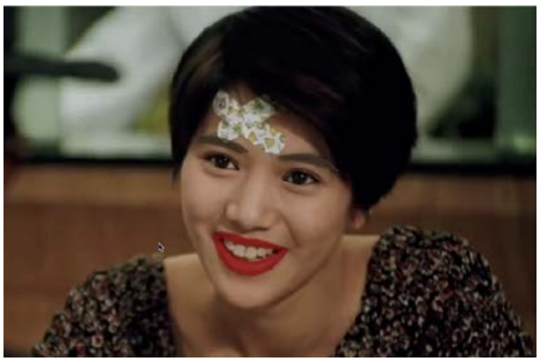
After switching the arrays into matrixes which are readable in openCV, the frame processed shows in the “face” window.

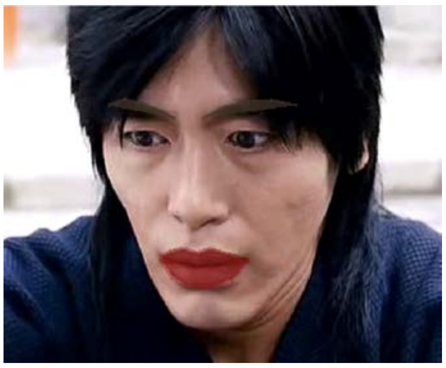
(5) Execute the function

```
makeup(0)
makeup('video.mp4')
```

The makeup function we defined can not only work on the video stored in computers, but also work in the real-time camera shooting.

The screenshots of the processed video are shown below





Stickers

Part1: Slogan style Sticker:

People who want their faces to be made into fashionable emoticon always favor the use of stickers such as

.

(1) Prepare the library

```python
import face_recognition
import cv2
from PIL import Image
```

To make slogan-style stickers for pictures, we would need face_recognition to detect and recognize the location of the face before pasting stickers at the most suitable location. We also need PIL to open and read two pictures and resize the slogan to make sure it fits into the picture nicely.

(2) Find face locations in the picture

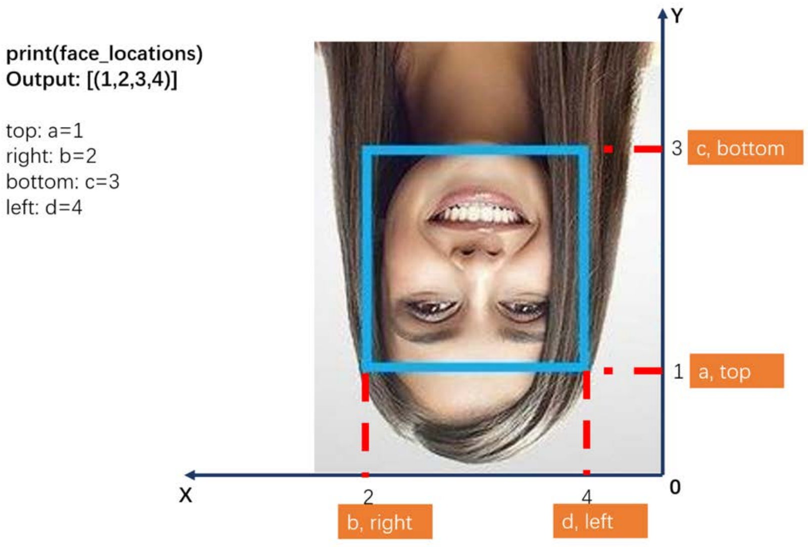
```python
image_file = "nan.jpg"
image = face_recognition.load_image_file(image_file)
```

Open picture as background and use face_recognition to find faces in the background picture.

```python
face_locations = face_recognition.face_locations(image)
print('face_locations:',face_locations)
```

Use face_locations in face_recognition function to find the actual location of the faces in the background picture, print the exact location in the form of [(a,b,c,d)]to make the coordinates clear. (The following picture will explain the exact meaning of a,b,c,d clearly.)

```
a=int(face_locations[0][0])
b=int(face_locations[0][1])
c=int(face_locations[0][2])
d=int(face_locations[0][3])
```



Take (502,915,965,451) as an example, define a, b, c, d as the first/second/third/fourth value in the list face_location. This can be easily seen in the picture above— name the real left-top point as(d, a) , real left-bottom point as (d, c) , right-top point as(b, a) and right-bottom point as (b,c). Once the coordinates are labelled, the location of the face detected should be easily detected by the face_recognition.

(3) Read, Open and Detect the size of pictures:

```
im1_path = r'C:\Users\Lucia\Documents\交大\Python\week11-code\nan.jpg'
im2_path = r'C:\Users\Lucia\Documents\交大\Python\week11-code\lengmo.jpg'
im1 = Image.open(im1_path)
im2 = Image.open(im2_path)
```

Use PIL to read the two pictures, im1_path as background picture and im2_path as slogan-style picture.

```
width1, height1 = im1.size
width2, height2 = im2.size
print('im1 size',im1.size, width1, height1)
print('im2 size',im2.size, width2, height2)
```

Use im.size to detect the actual size(width and height)of the background and the slogan picture, then print them out for further use.

(4) Resize the slogan to suit better with the background:

```
resizedIm= im2.resize((((c-a)//2),(c-a)))
resizedIm.save(r'C:\Users\Lucia\Desktop\afterresizing.jpg')
im3_path=r'C:\Users\Lucia\Desktop\afterresizing.jpg'
im3 = Image.open(im3_path)
```

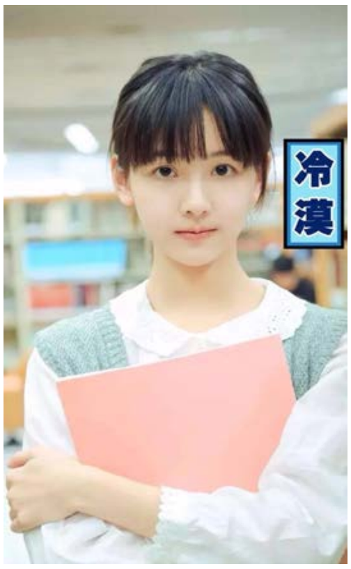
Resize the slogan-style sticker so that it has the same height of the detected face's height and the width of the sticker is half of the height of the face.

Paste the stickers to the faces:

```
im1.paste(im3,(c,a)) #设置贴纸的左上角粘在人脸
im1.save(r'C:\Users\Lucia\Desktop\结果.jpg')
im1.show()
```

Paste the sticker to the top-right point of the detected face and save the changed picture in the form of 结果.jpg at the computer's desktop. Show the changed picture at the same time.

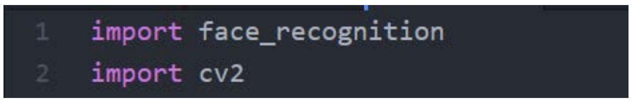
The result is shown below:



Part2: Make-up style Sticker:

People are also interested in using beautiful stickers such as flower clown or fashionable hats to make their pictures lovelier. Some popular stickers are and

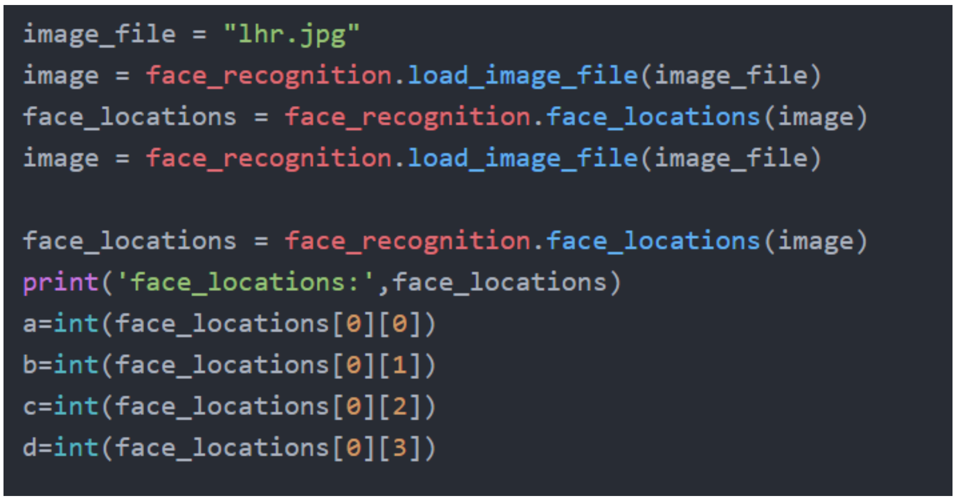
(1) Prepare the library:



```python
import face_recognition
import cv2
```

To insert different styles of stickers, we will use face_recognition to recognize and detect faces in the background pictures and use OpenCV to change sticker pictures into a gray image by applying the Fixed threshold binarization, bitwise, and bitwise_not command to make the transparent layer of the sticker invisible when added to the background pictures.

(2) Recognize face location



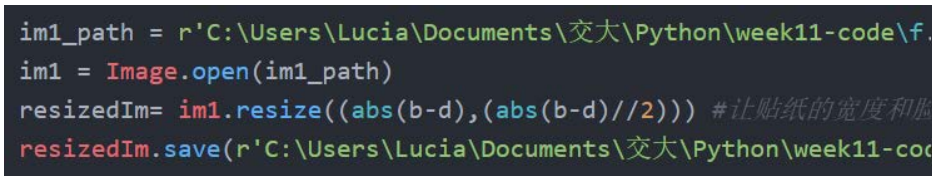
```python
image_file = "lhr.jpg"
image = face_recognition.load_image_file(image_file)
face_locations = face_recognition.face_locations(image)
image = face_recognition.load_image_file(image_file)

face_locations = face_recognition.face_locations(image)
print('face_locations:',face_locations)
a=int(face_locations[0][0])
b=int(face_locations[0][1])
c=int(face_locations[0][2])
d=int(face_locations[0][3])
```

Name the background picture as image_file .

Use face_recognition to load and detect the exact face_location of the picture . Print the locations[(a,b,c,d)] out to make figures visible and clear. Then name a ,b ,c, d as the first/second/third/fourth value in the list face_location

(3) Resize the sticker:



```python
im1_path = r'C:\Users\Lucia\Documents\交大\Python\week11-code\f
im1 = Image.open(im1_path)
resizedIm= im1.resize((abs(b-d),(abs(b-d)//2))) #让贴纸的宽度和
resizedIm.save(r'C:\Users\Lucia\Documents\交大\Python\week11-co
```

Resize the sticker so it has the same width as the face detected in the background picture and the height of the sticker is half of its width. Then save the sticker.

(4) Sticker Matting and Pasting

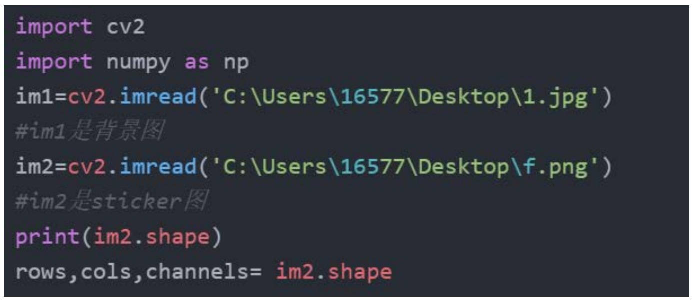
Since sticker pictures always have an opaque background, we need to eliminate the the areas that we don't want to appear in final picture. To do this, we need to:

1) use OpenCV to change the sticker image into gray image

2)use OpenCV to do Fixed threshold binarization

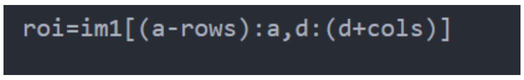
3)use bitwise_not and bitwise_and command to paste sticker to the background picture.

4.1 Preparation of the sticker:



```
import cv2
import numpy as np
im1=cv2.imread('C:\Users\16577\Desktop\1.jpg')
#im1是背景图
im2=cv2.imread('C:\Users\16577\Desktop\f.png')
#im2是sticker图
print(im2.shape)
rows,cols,channels= im2.shape
```

Open and read the background picture and sticker picture and name them im1 and im2; print the shape of sticker picture. Sticker picture is shown below:

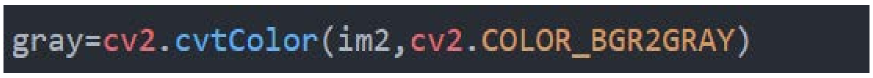


```
roi=im1[(a-rows):a,d:(d+cols)]
```

Cut part of the background picture as roi which has the same weight and length as the sticker picture, and add the position*((a-rows):a,d:(d+cols))*, choose a position that is suitable for flower clowns or other head decorations .

The result is shown below:

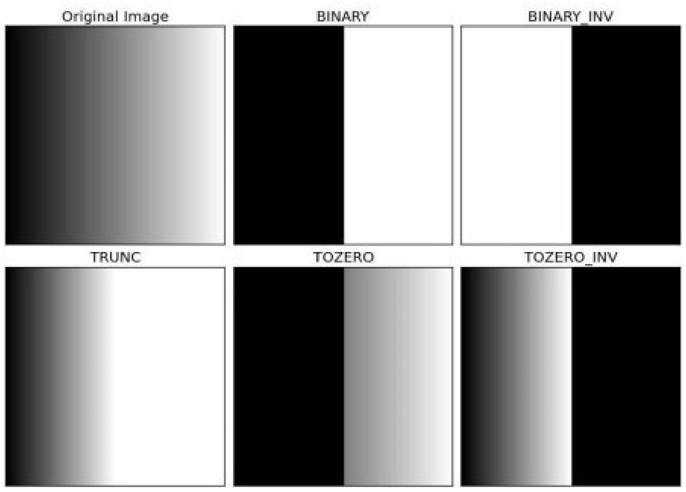
4.2 Change Sticker to Gray image:

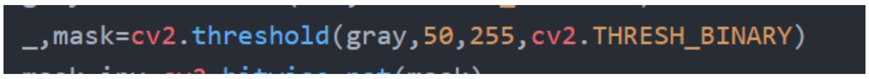


```
gray=cv2.cvtColor(im2,cv2.COLOR_BGR2GRAY)
```

Change the sticker image into a gray image, cv2.cvtColor(im2,cv2.COLOR_BGR2GRAY) changes im2 to gray color edition, which eliminates all colors except gray.

4.3 Fixed threshold binarization

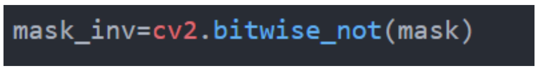




```
_,mask=cv2.threshold(gray,50,255,cv2.THRESH_BINARY)
```

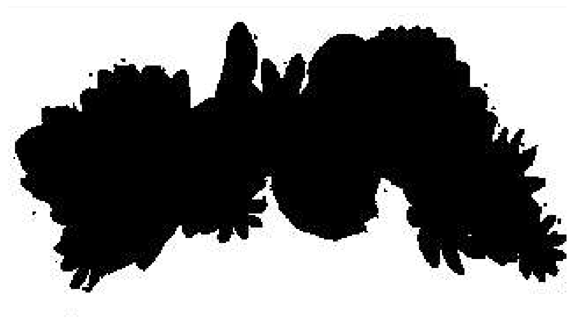
We use *Fixed threshold binarization* to change the gray image into two color (because we do cv2.THRESH_BINARY which contains black and white as can be seen above) and create a **mask** image. The mask image the flower part appears as white (1) and the background part of the sticker picture appears as black (0). It all depends on the threshold, and we now set the threshold as 50 and 255.

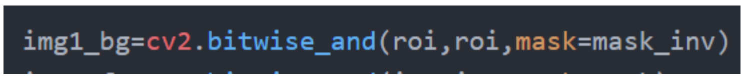
4.4 Bitwise_not and Bitwise_and command



```
mask_inv=cv2.bitwise_not(mask)
```

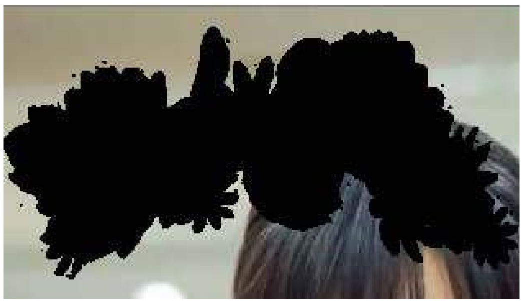
We then create another **mask_inv** which has the opposite color as the mask image, which means flower is black(0) and the background is white(1).
The result is shown below:

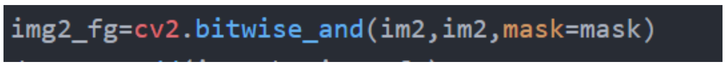




```
img1_bg=cv2.bitwise_and(roi,roi,mask=mask_inv)
```

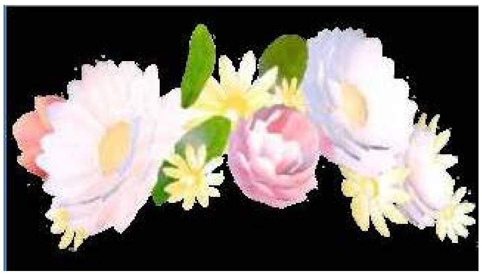
Then we add roi and mask_inv together. The result is where the flower part is black(so everything in the im1 under the mask_inv black all gone) and the background is white(so everything in the roi under the mask_inv white is still there) the result of img1_bg is shown below:



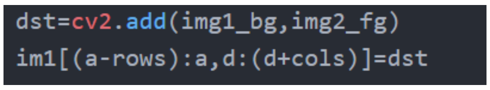


```
img2_fg=cv2.bitwise_and(im2,im2,mask=mask)
```

Then we add im2 and mask together, and the result is that the flower part is white (so everything in the im2 under the mask white is still there) and the background is black (so everything in the im2 under the mask Black is gone). The result of img2_fg is shown below:



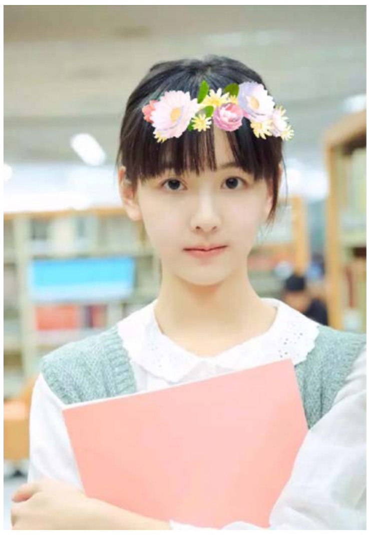
4.5 Combine pictures



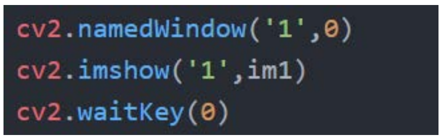
```
dst=cv2.add(img1_bg,img2_fg)
im1[(a-rows):a,d:(d+cols)]=dst
```

When we combine the img1_bg and img2_fg together, the black part in img1_bg is covered by the white part in img2_fg, so the sticker successfully paste on the background picture(roi) while the visible background of the sticker is now invisible .

The result is shown below:



(5) Save and show the result



```
cv2.namedWindow('1',0)
cv2.imshow('1',im1)
cv2.waitKey(0)
```

Create a new window to show the result of our makeup-slogan.

Weaknesses

(1) Lack of enough accuracy.

In the makeup part, from face recognition, feature locations to drawing are not accurate enough, thus the effects greatly depend on the quality of original pictures and videos.

In the sticker part, on account of the limitation of library "face_recognitinon", some data such as the specific width of face and the height of forehead is not accurate enough for adjusting the size of the stickers.

(2) Not perfect outcome

Since what we use are simple functions in some libraries, the outcome fails to catch up with several beautifying Apps. The output may not be satisfactory sometimes.

Conclusion

It is a precious experience for us to put what we have learned about python into practical use and to step out of our comfort zone for deeper insight into programming. Though facing great difficulties during our process, owing to the fact that we haven't learnt anything about image processing, we ought to teach ourselves by browsing Internet and ask our teacher for suggestions. We spent large amounts of time discussing and trying to figure out how to use libraries, such as Opencv and PIL. Not only do we have a basic idea of programming and methods of processing images with python, but we also have a sense of satisfaction after finishing the project.