FASTER FACE CHANGING TECH

[1] *Cheng-You Lee* (李承祐), [2] *Chan-Min Hsu* (許展銘), [3] *Chiou-Shann Fuh* (傅楸善)

[1] Graduate Institute of Communication Engineering,
[2] Graduate Institute of Biomedical Electronics and Bioinformatics,
[3] Department of Computer Science and Information Engineering,
National Taiwan University, Taiwan,
r07942099@ntu.edu.tw r07945010@ntu.edu.tw fuh@csie.ntu.edu.tw

ABSTRACT

Nowadays, "Face Recognition" was introduced to our daily life, it has been popular for people who are interested in computer vision. Among different applications, "Face Changing" is one of the hottest topics, but most of such technologies are not as good as we thought. Also, difficulty to implement is another issue that results in unpredictable outcomes. In this project, our goal is to improve the similarity and generalizability of face changing by adjusting existing algorithms, making the whole process much easier to handle. As an ultimate goal, we hope to get a better output compared with original image.

First, we have observed that the original algorithms only allow similar face angle to get somewhat good result. As a result, we try to focus on this point. However, we encounter some difficulties. Second, although the original two pictures have similar face angle, it still has some combination problems. These problems are hair and glass problem which will lead to strange results showed in Figure 3.

1. INTRODUCTION

While better hardware device such as GPU comes out recent years, Deep learning technology and its applications has overwhelmed every field of science, including computer vision. "Face-Swapping" is one of the most popular topics in such area.

So, what is "Face-Changing" mean? In short, we can treat it as AI-powered computer vison technology whose job is changing one's face to other's appearance. For example, an interesting technology called "Deep Fakes" debuted a few years ago is a successful application based on latest deep learning method "Generative Adversarial Network". It can easily swap the source face image in the video with any target face such as celebrities or politicians. Unfortunately, this technology needs large effort to implement and requires strong device for running. Hence, in order to popularize the face-changing technology and make it easy to implement, we introduced a faster and simpler method based on existed algorithm Vahid Kazemi and Josephine Sullivan (2014) published [1] to overcome the issues mentioned above.

The original algorithm used cascade of regressors to extract our face features like eyes, nose, and mouth. After reasonable iterations, we can get the feature map as our first input for the model. But, without any adjustment and refinement, the original result became a tragedy since most of swapped(changed) faces are unreal and distorted.

In this project, we will change face to some better degree. The original algorithm has some problems. One is that if the face angles in two images are different, the result will become inferior. Another is that if someone's hair is so long to cover his face, the result will become mixed with hair and face. We want to solve these issues.

However, the first part of rotating two images has a great difficulty for us. We just propose a thought which uses TP-GAN but needs to recover the face angle.

In general, we built a faster model to achieve the better performance by adding more adjustment. Though there are still some problems existing, performance itself has shown that, without thoroughgoing examining, people are hardly to distinguish it from the face in real world.

2. METHOD AND STEPS

2.0. Rotate Two Images into the Same Angle

If we want to implement such face changing technology, angle alignment is a must-do in order to reach high similarity.

To do so, there are plenty of ways which can finish the job. We will introduce one of them in this topic. A two-dimension rotation matrix:

$$M(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Let the angle between target image and source image be □, we can then multiply source image matrix by *M(*□*)* and get the new source image with the same face angle as target.

However, the result is not ideal. We start to search some methods for rotating face angles and find a machine learning method named TP-GAN[3]. TP-GAN is a Generative Adversarial Network method which can recover a frontal face image of the same person from a single face image under any poses.

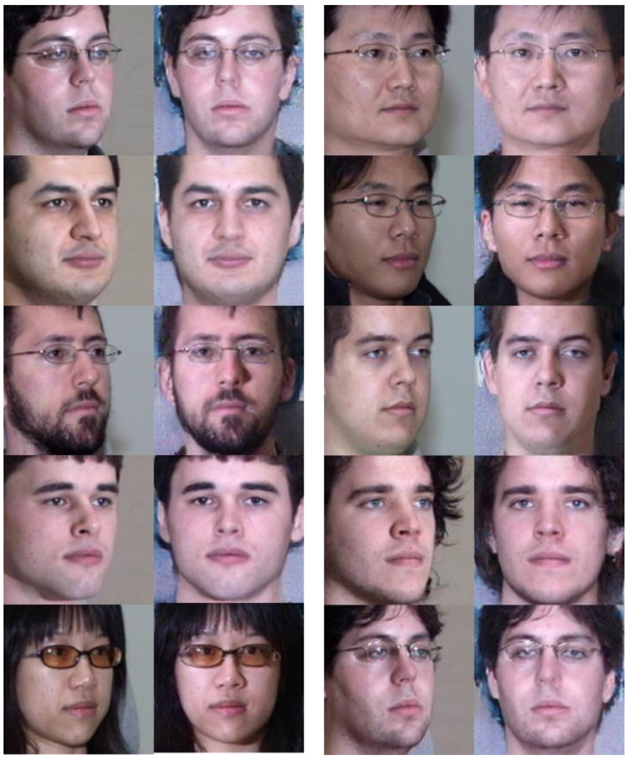


Figure 1 TP-GAN can rotate the face angles into frontal face.

By this technic, we can first rotate two face into frontal ideally, and then avoid face angles problems. Nevertheless, if we want to do face changing, we need to reserve origin face angle. Which means if we adopt TP-GAN, we need to find a method to recover original face angle. Recovering original face angle is also a difficult work which need GAN and consume execution time. Our goal is doing faster face changing, so we finally do not consider adopting this method. As a result, our algorithm exists limitation to the face angle problem. If we want to use this algorithm, we have to let the face angles of two images be close to.

Besides TP-GAN, if There exists a model which can arbitrary rotate face angle to a special angle, the problem will be solved clearly. However, as for today, nobody has found or created such a miraculous model.

2.1. Extract Face Features

First, we will use the existed algorithms Vahid Kazemi and Josephine Sullivan (2014) published [1] as basis. The idea in the paper is to estimate the facial landmarks in efficient method which uses cascade of regressors. To begin with, let $x_i \in R^2$ be the *x,y*-coordinates of the *i*th facial landmark in an image *I*. Then we put all the p facial landmarks into the vector S where its size is 68*2 ($p = 68$ with their original coordinates *x,y*). We take $\hat{S}^{(t)}$ to represent the current estimate of ***S***, and r_t (.,.) as regressor factor. After each prediction of face features, an update vector from image will be added into current estimate $\hat{S}^{(t)}$ to improve the estimate:

$$\hat{S}^{(t+1)} = \hat{S}^{(t)} + r_t(I, \hat{S}^{(t)})$$

This is the basic regression algorithm in our re-implementation.

The important part of the cascade is that the regressor predicts based on features such as pixel intensity values. We choose pixel intensity, because we can retain the geometric invariance during the process. When cascade proceeds, the prediction can be more certain about whether semantic location on the face is indexed or not.

As for each regressor, we train it with gradient tree boosting algorithm [1].

Second, we need to let the model to learn each regressor in the cascade. Assume we have training data (I_1,S_1,...,I_n,S_n) where each I_i is a face image S_i its shape vector. To learn the initial regression function r_0 in the cascade, we use the data to create a triplet of face image, initial shape estimate, target update step, that is, ($I_{\pi i}$,$\hat{S}_i^{(0)}$,$\Delta S_i^{(0)}$) where

$$\pi_i \in \{1,\dots,n\}$$
$$\hat{S}_i^{(0)} \in \{S_1,\dots,S_n\}$$
$$\Delta S_i^{(0)} = S_{\pi i} - \hat{S}_i^{(0)}$$

From this data we can learn the regression function r_0 using gradient tree boosting with a sum of square error loss. The

set of triplets is then updated and become the input of the next training data, ($I_{\pi i}$,$\hat{S}_i^{(1)}$,$\Delta S_i^{(1)}$).

For the next regressor r_1 in cascade, we set

$$\hat{S}_i^{(t+1)} = \hat{S}_i^{(t)} + r_t(I_{\pi i}, \hat{S}_i^{(t)})$$
$$\Delta S_i^{(t+1)} = S_{\pi i} - \hat{S}_i^{(t+1)}$$

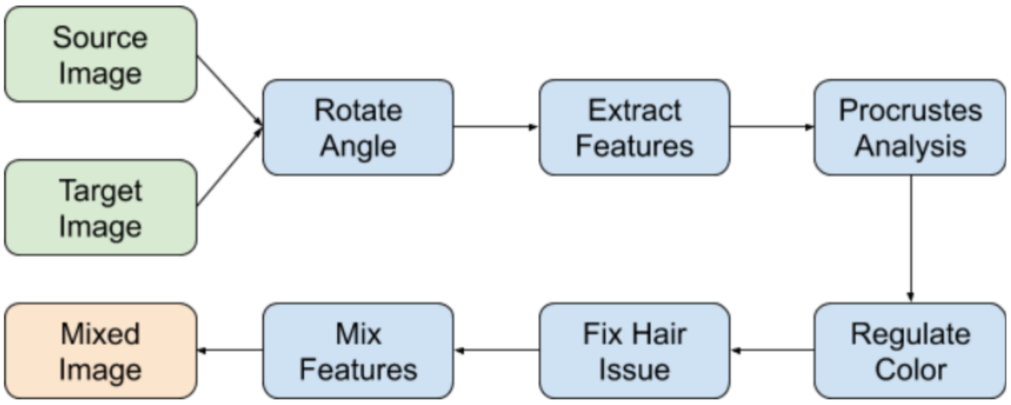


Figure 2 Flowchart of this algorithms.

The whole work is iterated until a cascade of T regressors $r_0, r_1, \ldots, r_{T-1}$ are learnt with sufficient level of accuracy.
With the help of Python library “dlib”, a modern C++ toolkit containing machine learning algorithms and tools, we can easily compute the algorithm mentioned above. Function *get_landmarks()* turns an image into a 2D array, and we use the detector to extract the face edge since we need the face edge as input for predictor to train. When the procedure is done, we can get an output of 68*2 matrix containing 68 different face feature positions.

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(PREDICTOR_PATH)

def get_landmarks(im):
    rects = detector(im)

    return matrix([[p.x, p.y] for p in predictor(im, rects)])
```

figure shows the pseudo code described above.

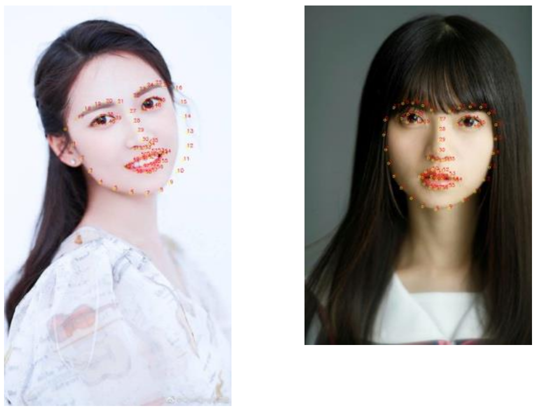


Figure 3 The result of dlib. This process has some problems. The main problem is that landmarks are few in face.

2.2. Procrustes Analysis

Thus at this point we have our two landmark matrices, each row having coordinates to a particular facial feature. We are now going to work out how to rotate, translate, and scale the points of the first vector such that they fit as closely as possible to the points in the second vector, the idea being that the same transformation can be used to overlay the second image over the first. To put it more mathematically, we seek T, s, and R such that

$$\sum_{i=1}^{68} \left|\left| sRp_i^T + T - \right.\right.$$

is minimized, where R is an orthogonal 2x2 matrix; s is a scalar; T is a 2-vector; and p_i and q_iq_i are the rows of the landmark matrices calculated above. It turns out that this sort of problem can be solved with an Ordinary Procrustes Analysis [2].
Function transformation_from_points:

1. Convert the input matrices into floats. This is required for the operations to follow.
2. Subtract the centroid from each of the point sets. Once an optimal scaling and rotation has been found for the resulting point sets, the centroids c1 and c2 can be used to find the full solution.
3. Similarly, divide each point set by its standard deviation. This removes the scaling component of the problem.
4. Calculate the rotation portion using the Singular Value Decomposition. See Wikipedia article on the Orthogonal Procrustes Problem for details of how this works.
5. Return the complete transformation as an affine transformation matrix.

The result can then be plugged into OpenCV’s cv2.warpAffin function to map the second image onto the first which in the function warp_im().
After doing Procrustes analysis, we can get a pair of same face angles and scales pictures.

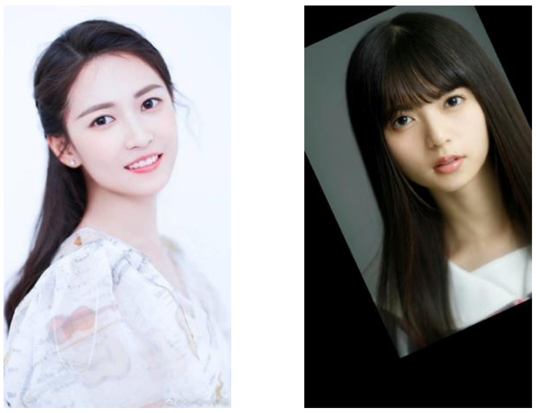


Figure 4 After Procrustes Analysis.

2.3. Regulate the Color of Source Image

Since the color and contrast will be different often, we must strive to solve this kind of problem. We attempt to change the colouring of image 2 to match that of image 1. . We first divide the original pixels of image 2 by the Gaussian Blur of it, called it image 2_blur. And then multiply image 2 by the value of image 1's Gaussian Blur. Finally, we can get the regulated version of source image 2. This procedure is based on the idea of scaling monitor R, G, and B.

This part of function in code is correct_colours().With this approach differences in lighting between the two images can be accounted for, to some degree. For example, if image 1 is lighted from one side but image 2 has uniform lighting then the colour corrected image 2 will appear darker on the unlit side as well.

However, the algorithm is a fairly crude solution and having some problems. One is that the forehead in the faces sometimes will be very different. For example, most of women have long hair which cover forehead. As a result, the output image will also have long hair which covers forehead and becomes so strange. Another is the problem of freckle. If face A has freckle and face B does not, the output image will become discontinuous. We need to solve the above problems to make a better result.

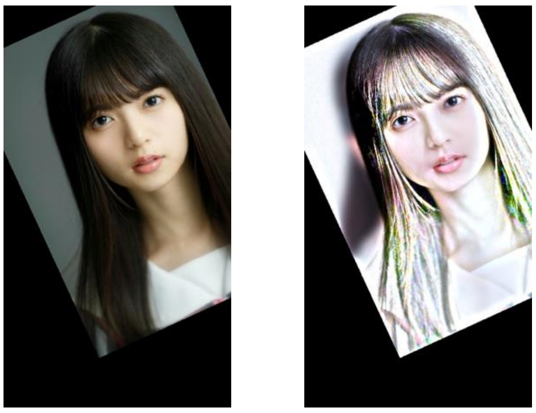


Figure 5 After regulating the Color of Source Image.

2.4. Clear Hair Problem

In Figure 3 we can find that the face changing will include hair part. It is because during 2.3 "Regulate the Color of Source Image" part, the algorithms will include brows. However, some hair especial women hairs will cover their brow. As a result, we should remove them to make the combination looks better.

We apply three method below:

1. Removing hair by pixel value

First, we assume that the hair color is nearly identical everywhere. Then using [4] to detect hair zone. Averaging pixel value of hair zone and setting a threshold. If some pixel value is between the (averaging pixel value + threshold) and (averaging pixel value - threshold), we see it as hair part. Finally, we remove it and use near pixel value belonging brow to recover the missing part. This method is a little overkill and waste execute time.

2. Directly remove hair from brow

If we only focus on the brow part, we can find that hair often only cover a little part of brow. As a result, we can depart the brow zone into some regions and calculate the averaging pixel values. Using this algorithm, we can find the location of hair covering brow and remove them. After removing the hair, we use near zone to recover the missing part. This method needs little execute time and can get a quite good result.

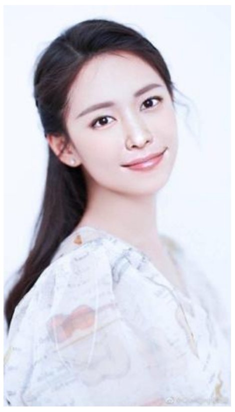
3.Ignoring brow part

We can just ignore brow part of face changing. It will lead to good result whether the picture has hair cover problem or not. However, if we ignore the brow, the final image will sometimes lack of spirit and become another person.

The funny result shows that brow is an important part for a person.

(a)

(b)



(c)

Figure 6 Face changing without brows, it will lead the combination become another person. (a) Original (source) image (Chen Yu Qi). (b) Target image (Aragaki Yui). (c) Final image we mix.

2.5. Mix the Feature to Target Image

Finally, we will create a mask that decides which pixel will be chosen as a part of the mixed image.

The full process is to compute two masks corresponding to source and target image. To achieve such mask, we use a function defined as *get_face_mask ()*, which calculate two convex hull: eye area and nose-mouth area, from the image and its 68*2 matrix. After that, we compute the Gaussian blur of two convex hulls in order to eliminate any discontinuous regions. By repeating the process, we can get both masks of source and target image in the end.

With these two masks, the last part of mixing feature is to combine them together by calculating the element-wise maximum value, so we can ensure the characteristic of source image shown on the target image.

Every matrix value of combined-mask is between 0 and 1. Hence, we can directly decide which image element should be chosen in certain region. For instance, we keep the target image as the same if the mask value is 0 and replace by the source image if the value is 1. For those values among 0 and 1, we simply take average of two images as final result.

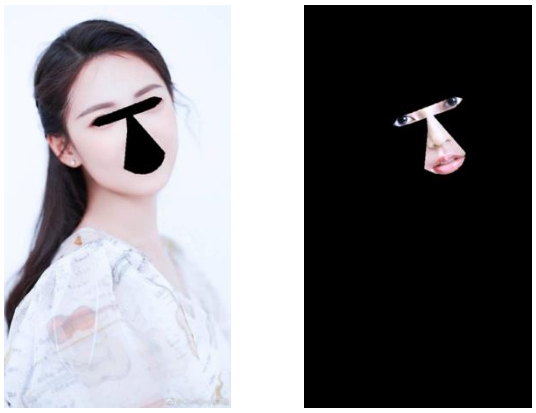
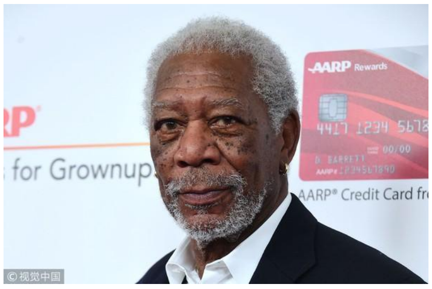


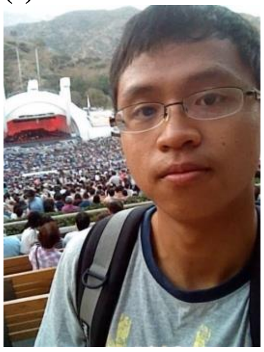
Figure 7 Using mask to determine which part of the face will be changed.

3. ORIGIN RESULT

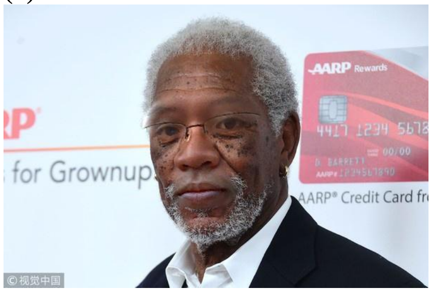
We will show the origin result first. Actually, the origin algorithm already has quite good result at some special situations. For example, if two faces have nearly face angles and face colours, the result will be good. The result can not only change face clearly but also confuses human eyes. However, if two faces have quite different face angles or colours, the result will be terrible and horrible.



(a)

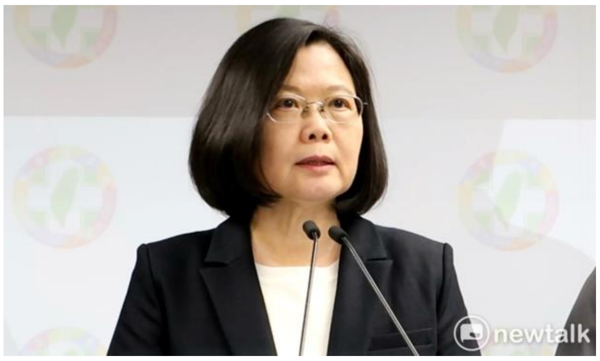


(b)

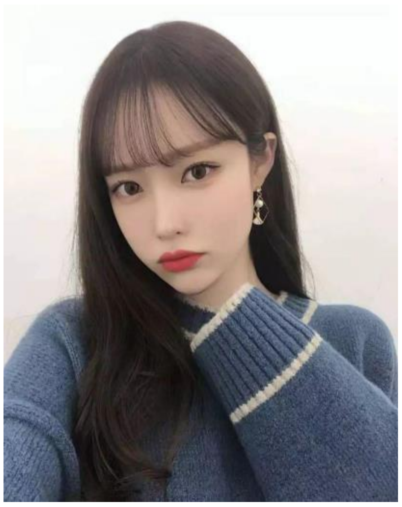


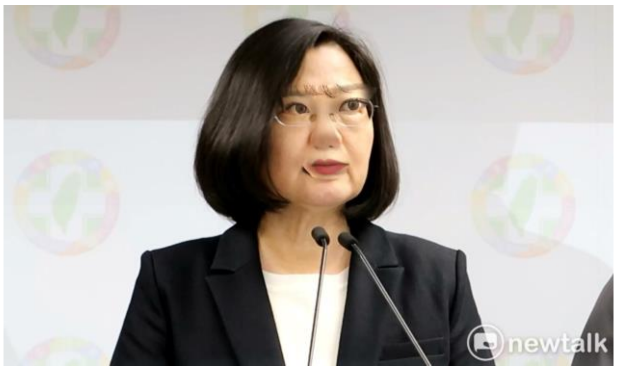
(c)

Figure 8 Somewhat good result of the algorithm, but it still has some problems. (a) Original (source) image (our classmate). (b) Target image (Morgan Freeman). (c) Final image we mix.



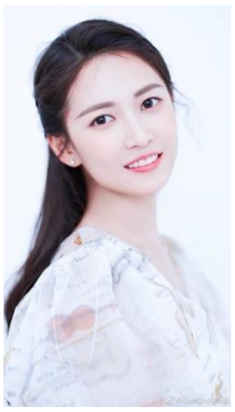
(a)

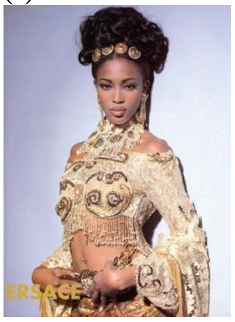

(b)



(c)

Figure 9 Somewhat bad result of the algorithm, it has obvious hair problem. (a) Original (source) image (The face of Ing-Wen Tsai which has no hair covering brows). (b) Target image (a beautiful face with hair covering brows). (c) Final image we mix.


(a)



(b)

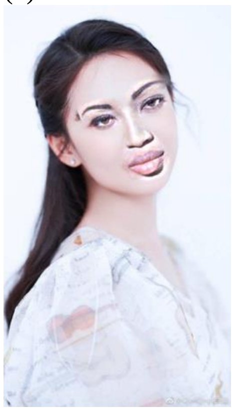

(c)

Figure 10 Somewhat bad result of the algorithm, the two face have quite different colours . (a) Original (source) image (Chen Yu Qi). (b) Target image (Naomi Campbell). (c) Final image we mix.

4. EXPERIMENT

In this section, we will discuss some parameters which can affect the quality of results.

First part is whether maintaining brows part or not. As we have discussed in section 2.4, if the target face has problem of hair covering brows. A good approach to avoid this problem is to ignore the brows during merging. We will show the effect below.

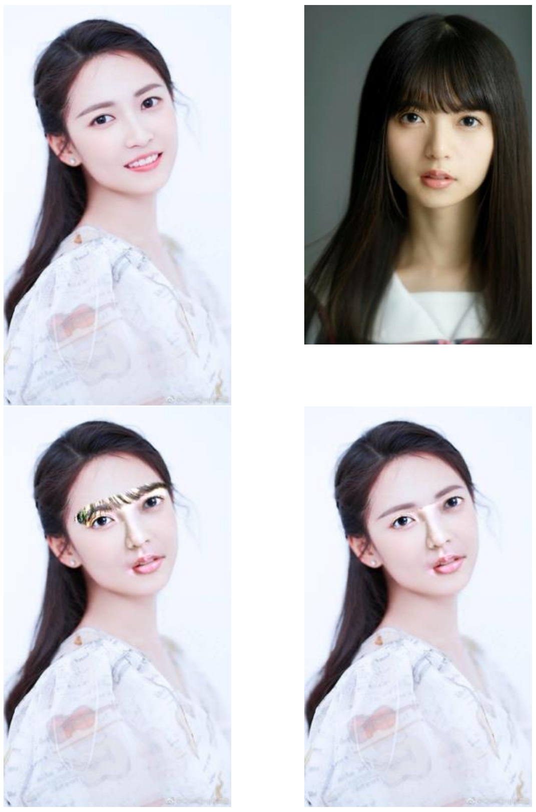


Figure 11 By ignoring brows, the result will be good.

However, if we look at the merging picture clearly, we will find discontinuity exists. The key to solve this problem is to add the parameter FEATHER_AMOUNT which helps hide any remaining discontinuities. Nevertheless, if FEATHER_AMOUNT becomes larger, it may cover the hair part. As a result, we should consider which is a good value for each situation.

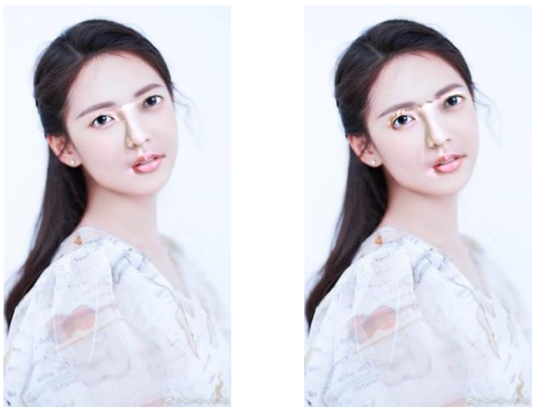
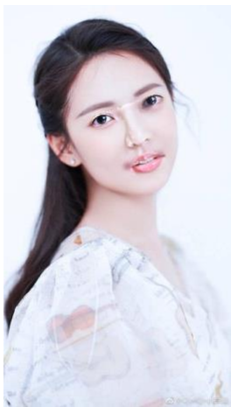


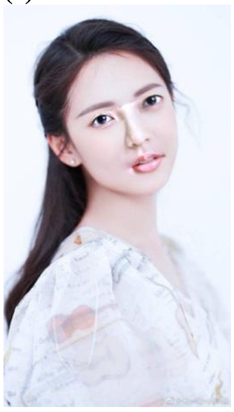
Figure 12 FEATHER_AMOUNT of left picture is 11. FEATHER_AMOUNT of right picture is 21. Right picture covers the hair part.

Besides FEATHER_AMOUNT, the parameter COLOUR_CORRECT_BLUR_FRAC also plays important role in the final result.

During Section 2.3 Regulating the Color of Source Image, an appropriate size Gaussian kernel is key. Too small and facial features from the first image will show up in the second. Too large and kernel strays outside of the face area for pixels being overlaid, and discolouration occurs. Let us show the difference between different COLOUR_CORRECT_BLUR_FRAC below.

(a)


(b)


(c)

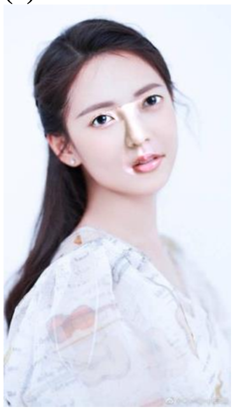
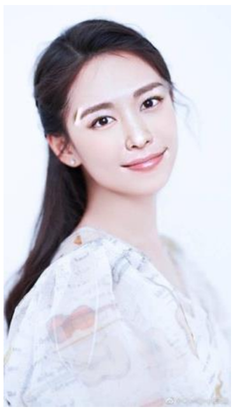

(d)

Figure 13 By using different COLOUR_CORRECT_BLUR_FRAC, the result will be quite different. (a) COLOUR_CORRECT_BLUR_FRAC = 0.1 (b) COLOUR_CORRECT_BLUR_FRAC = 0.3 (c) COLOUR_CORRECT_BLUR_FRAC = 0.6 (d) COLOUR_CORRECT_BLUR_FRAC = 0.9

What is more, the way to determine which zone should be mixed also affects the result. First, let us show two pictures.


(a)

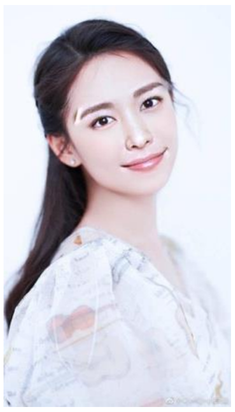
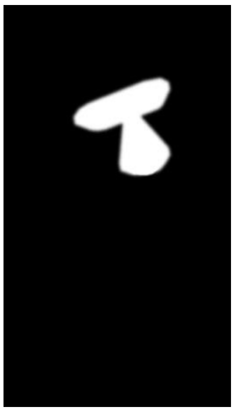

(b)

Figure 14 Different masks method also affect result.


(a)

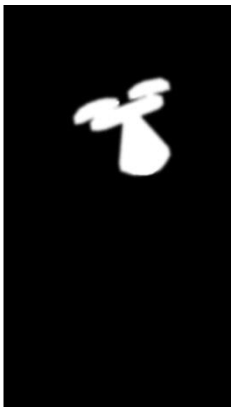

(b)

Figure 15 Different masks.

The white zone in Figure 15 represents where target face should appear, and the black zone in Figure 15 represents where original face should remain. Figure14 use masks in Figure 15.
Using b mask can get better result.

5. CONCLUSION

The original algorithm emphasizes using few codes (about 200 rows) and little executing time (about 3 seconds) to change faces. At first, we were amazed because it can get good result within only little executing time. After we start to research it more, we finally found some problems of this algorithm and decided to solve.
However, we cannot get the balance of executing time and performance. If we want better performance, we have to introduce many machine learning methods which will cost much executing time.
What we have contributed is that we discussed the probability of face changing model. We have proposed some ways to improve the results and demonstrated the tuned results.
Finally, we can get a conclusion. If we want a really good face changing technique, machine learning technique is important.

6. REFERENCES

[1] Kazemi, Vahid, and Josephine Sullivan. "One millisecond face alignment with an ensemble of regression trees." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.
[2] Colin Goodall. Procrustes Methods in the Statistical Analysis of Shape. Journal of the Royal Statistical Society. Series B (Methodological), Vol. 53, No. 2(1991), 285-339.
[3]Huang, Rui and Zhang, Shu and Li, Tianyu and He, Ran. Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis". ICCV17.
[4] Hair detection, segmentation, and hairstyle classification in the wild
U.R. Muhammad*, M. Svanera*, R. Leonardi, and S. Benini* *Image and Vision Computing*, 2018.