Prediction about Children's Faces

Final Project Report team 7
105061584 Ming-Chen Lu 105061519 Ya-Ling Hsieh 105061522 Hsuan Ou-Yang

Abstract

In this project, we present a method to generate composite children face from two given adults' images. Key point detection, Delaunay triangulation are first implemented to collect the facial features from each adult image. It's worth nothing that we not only morph two images but also give different weight in specific face features such as nose, eyes and mouth to produce a morphed adult image. In the age transformation, we blend the morphed image into base baby images obtained from the Internet as contour of a face. From the result, we can produce children's face images successfully without ambiguity.

1. Introduction

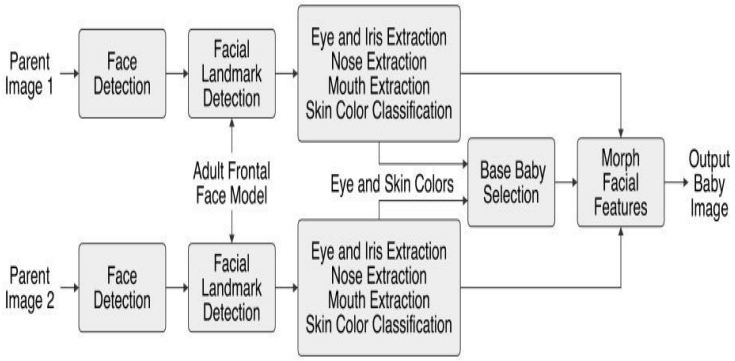
Sometimes parents are interested in the unborn child's face and want to know the face features are from father or mother. Furthermore, some people want to know if they have get married with celebrities, what their children look like.

Therefore, we introduce a method to predict what children will look like accords to our choice, i.e., if we want our child's eyes look like father and nose is inherited from mother, we can tune weights depending on ourselves.

2.1 Review of previous work

We refer to Baby Face Generator (2015,Divel,Shunhavanich) [1] for our project.

Flowchart of Baby Face Generator



In Baby Face Generator, they use face detector in OpenCV [2] to find face and marking facial landmarks through open source facial landmark detector [3].

They use these facial landmarks and Canny edge detector [4] to localize organs on the face such like a nose, eyes, a mouth and so on. It only extracted eyes, nose, mouth parts and morph these parts into baby faces. However, the results of morph babies seem to similar to original babies since it only changed eyes, nose and mouth. Thus, in our project, we morph whole face into baby's face, and the result is better because we can also see the parts of eyebrow and face shape.

2.2. Say why your method is better than previous work

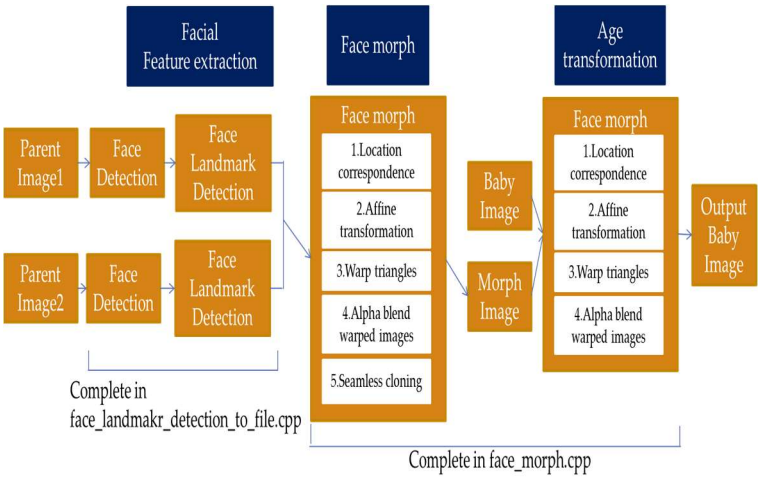
Facial Feature Extraction

We use the face detector and marking facial landmarks through function in Dlib C++ library [5] which is faster than previous method. Also, we mark more landmark points around nose for better morphing later and we use these points to construct Delaunay triangular, which is a more previous and efficient way to localize organs on the face such like a nose, eyes, a mouth and so on.

Morph Facial Features

In our project, we morph the morphed image from parents into baby's face instead of only facial features, and the result is better because we can also see the parts of eyebrow and face shape.
The main contributions of our work are first we mark more landmark points around nose for better morphing. Second we can change the facial features ratio. Third, since we can tune facial features, if we set nose totally from father or mother, the nose part would be strange because the color of nose would be different to skin color. After our algorithm, we can make nose part suitable for morph image. Fourth, we morph the morphed image from parents into baby's face instead of only facial features.

Flowchart of our method



3.1. Technical part:

Summary of the technical solution

There are three stages in this project:

1st stage: Facial feature detection

We detect face from input image and find eighty-four points on face in this stage. Then we resize input image to particular size (300*300) and find coordinate of feature points again. Finally, we construct Delaunay triangular with these points through Voronoi algorithm.

2nd stage: Face morph

In order to morph images without disconnection between two images, we find the location of pixel in our morphed image first. Then, we use Affine Transformation matrix to perform triangle correspondences before warping triangles. Finally, we give weights in different face feature region to produce morphed images.

3rd stage: Age transformation

After producing morphed images in the previous stage, we collect several baby photos to be our generated baby image base and blend with the morphed

image. Next, we perform the same procedure as the second stage.
Note that all the weightings can be tuned by users, and the default value is 0.5 since the intensity of transformed image should include two parents, the morphed image and the baby image.

3.2.1 Facial Feature Detection

A. Face detection

We use the function in Dlib C++ library for finding face and feature points in this part.
First, you can download frontal facial training data from the Internet [6] which is offered by Dlib developer.
Then, we use the function, face detector, which is in Dlib to detect face and find feature points from input image (Fig.1). The face detector is made using the classic Histogram of Oriented Gradients (HOG) feature combined with a linear classifier, an image pyramid, and sliding window detection scheme. The result is shown in Fig.2.

After detecting face of each image, we resize original input image (Fig.1) into image with same size which is almost full of face (Fig.3) since different input image might be different size. The same size of images would be convenient for us to morph in later procedure.

Finally, we detect feature points again since (x, y) coordinate would change after resizing. The result is shown in fig.4 (only some feature points is drawn on it).
Now, there are totally 84 feature points with correct (x, y) coordinate:
no.1~no.17 for recording jaw,
no.18~no.22 for recording left eyebrow,
no.23~no.27 for right eyebrow,
no.28~no.31 for nose bridge,
no.31~no.36 for lower nose,
no.37~no.42 for left eye,
no.43~no.48 for right eye,
no.49~no.60 for outer lip,
no.61~no.68 for inner lip and the other points for localization we use in later procedure.

Fig.1 Fig.2

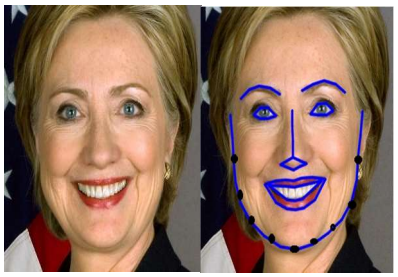


fig.3 fig.4

B. Delaunay triangular

We use 84 feature points detected in 3.2.1.A to construct Delaunay triangular through Voronoi diagram.

Voronoi diagram:

Given a set of points in a plane, a Voronoi diagram partitions the space such that the boundary lines are equidistant from neighboring points. Fig.5. shows the Voronoi diagram of our input image calculated from the points shown as black dots. You will notice that every boundary line passes through the center of two points. If you connect the points in neighboring Voronoi regions, you get a Delaunay triangulation.

The Delaunay triangulation derived from input image is shown as Fig.6 and

we record three vertex of each triangular.

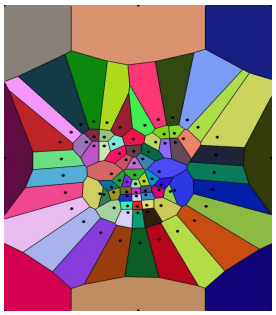


Fig 5.

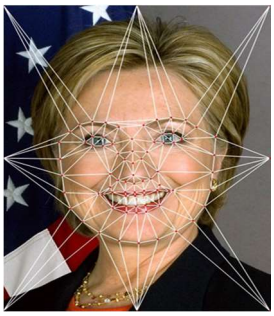


Fig.6

3.2.2 Face morphing

A. Location Correspondence

In order to morph image without disconnection between two images or ambiguous images, we need to establish location correspondence of features between the two images. In other words, for every pixel (x1, y1) in image 1, we need to find its corresponding pixel (x2, y2) in image 2.

The easiest way to deal with this problem is to calculating the location (xm, ym) of the pixel in the morphed image. It is given by the following equation:

$$x_m = (1-\beta)x_1 + \beta x_2$$
$$y_m = (1-\beta)y_1 + \beta y_2$$
$$\beta = 0.5$$

We use the equation to find the location of pixel in our morphed image. We always set parameter beta to be o.5 to make sure the location of the pixel in the morphed image between image 1 and image 2.

B. Affine Transformation

So far, we already have a list of triangles by Delaunay Triangulation in section 3.2.1.B, then we can perform triangle correspondence.

There are many coordinate transformations for the mapping between two triangles, we choose Affine Transform here to implement triangle correspondence.

Suppose we have two triangles ABC and DEF shown in Fig.7. An affine transformation is a linear mapping from one triangle to another. For every pixel P within triangle ABC, assume the position of Q is a linear combination of A, B, and C vectors. The transformation is given by the following equations

$$P = \lambda_1 A + \lambda_2 B + \lambda_3 C$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1,\ \lambda_i \geq 0$$

$$Q = T(P) = \lambda_1 D + \lambda_2 E + \lambda_3 F$$

Fig.7

In OpenCV, this can be done using function ‘‘get Affine Transform ’’. It can calculate affine transforms for every pair of triangles.

Therefore, we pick a triangle in image 1 and the corresponding triangle in the morphed image, then calculate the affine transform matrix that maps the three corners of the triangle in image 1 to the three corners of the corresponding triangle in the morphed image. Finally, repeat the process of image 2 and the morphed image.

C. Warp Triangles

Each triangle uses the affine transform calculated in the previous step to transform all pixels inside the triangle to the morphed image. In OpenCV this is achieved by using the

function ‘’warp Affine’’. After repeating this procedure for all triangles in image 1 and image 2, we will obtain warped images shown in Fig8.

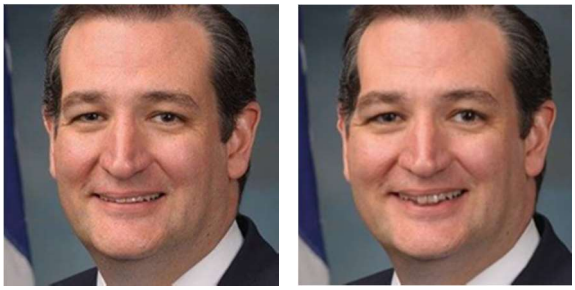


Fig.8 (a)

Fig.8 (b)

You can see carefully that the mouth of image Fig.8 (b) becomes larger. By contrast, the mouth of image Fig.8 (a) becomes smaller.

D. Alpha blend warped images

In the previous step we obtained warped image 1 and image 2. These two images can be blended using equation

$$M(x_m, y_m) = (1-a)I(x_1, y_1) + a$$
$$(0 \le \alpha \le 1)$$

Alpha is a weighting which controls the ratio of intensity (RGB values) from image 1 and image 2. This procedure is performed pixel by pixel, and each of the color components RGB are dealt with individually.

And most of all we are able to tune alpha in different face features easily. In other words, if we want the nose of morphed image is similar to image 1, we set alpha smaller and vice versa. Using this technique, we can generate specific morphed image with different combination of face features easily.

However, we set alpha to be 0.5 except for eyes, nose and mouth region where we can tune. The Fig.9 is our result of morphed image, and the parts of eyes, nose and mouth are all from Hillary.

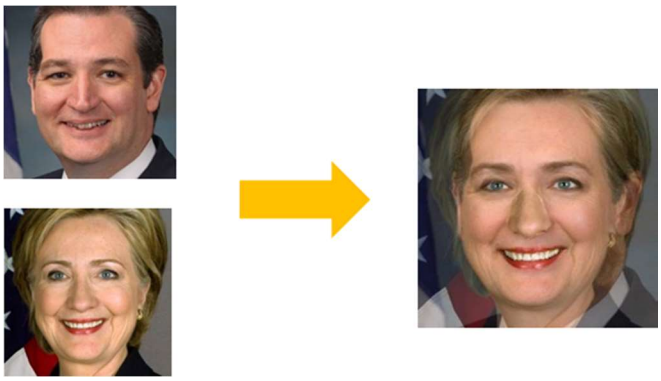


Fig.9

E. Seamless Cloning

Unfortunately, in the previous blending result, there is a unnatural phenomena around the nose because of intensity(RGB values) difference. As a consequence, we deal with it using Seamless cloning in OpenCV. Firstly, we need to define unnatural region (here is nose) as source image and define the image which alpha is 0.5 all of the face feature region. Secondly, we also produce a white mask the same size of source image. Finally, we call the function ‘’seamlessClone’’ to implement this step and make the nose’s color corresponding to around region. The result is shown in Fig.10 (b).

Fig.10 (a) Fig.10 (b) Fig.10 (c)

Fig.10 (a) The morphed image(b)The result of Seamless cloning (c)The result of smooth boundary

F. Smooth Boundary

Although we solve the intensity (RGB values) difference problem to make the morphed image looks natural, there are some unsmooth transition along the nose. As a result, we come up with a method to solve this problem. First, we detect the mask which has the same size as nose. Next, give both side of nose boundary weighting gamma to calculate the intensity pixel by pixel using equation

$$M(x_m, y_m) = (1-\gamma)I(x_{out}, y_{ou}$$
$$(0 \le \gamma \le 1)$$

The weight gamma will increase when the location close to the center of nose. After that, we eventually generate a natural image shown in Fig.10(c).

3.2.3 Age Transformation

A. Base Baby Selection

In order to perform age transformation, it is important to select base babies who have similar skin and eyes color to morphed images. Therefore, we collect several base babies around one years to three years old first. Then, doing Facial feature detection according to section 3.2.1.

B. Location Correspondence

After obtaining morphed images and perform Detai refinement, it is time to turn adult's faces into bace children' faces. Similarly, we perform location correspondence in order to align two images well the same as previous stage. After that we obtain the location (xg, yg) of pixels of Generated baby image.

C. Alpha blend warped images

Before performing Alpha blending, the morphed image and base baby image are treated by Affine Transformation and warping in advance. Next, combining two images using this equation

$$G(x_g, y_g) = (1-a)I(x_m, y_m) + aI(x_b, y_b)$$
$$(0 \le \alpha \le 1)$$

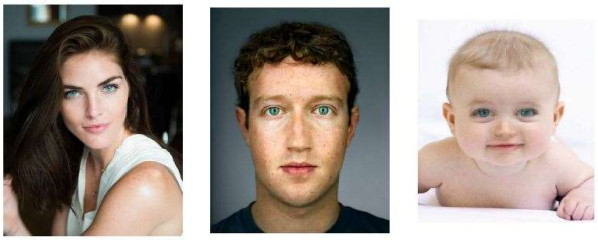
The equation is the same as previous stage, note that all the weightings can be tuned by users, and the default value is 0.5 because we want both characteristic from morphed image and base baby image. Fig.11 is an example and its result.

Fig.11 (a) Fig.11 (b) Fig.11 (c)

Fig.11 (a) Morphed image (b) Base baby image (c) Generated baby image

4. Experiments: results

We first compare our result to Baby Face Generator [1] and the results shown in Fig12. Fig12 shows the same base baby image morphed by two groups' parents. While the results of Baby Face Generator in Fig12-c and f looks very similar, we can only find little difference in eyes. And the results of our proposed method in Fig12-i and l looks very different. We can even find the eyes of Fig12-i are from Fig12-h and the eyes, nose and mouth of Fig12-l are from Fig12-j.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Fig.12
(a)Image 1(b) Image 2
(c) Morphed image by Baby Face Generator
(d)Image 1(e) Image 2
(f) Morphed image by Baby Face Generator
(g)Image 1(h) Image 2
(i) Morphed image by our proposed method
(j)Image 1(k) Image 2
(l)Morphed image by our proposed method

Our proposed method was tested using a combination of face features with different weighting and the results shown in Fig.13 and Fig.14.

(a) (b)

(c) (d)

(e) (f)

(g) (h)

Fig.13
(a) Image 1
(b) Image 2
(c)Morphed image with eyes, mouth and nose are from image 1.
(d) Generated baby image of (c).
(e)Morphed image with eyes, mouth and nose are from image 2.
(f)Generated baby image of (e).
(g)Morphed image with eyes and mouth from image 1, nose is from image 2.
(h) Generated baby image of (g).

(a) (b) (c)

(d) (e) (f)

Fig.14

(a) Image 1

(b) Image 2

(c) Morphed image with eyes from image 1 and other features' alpha are 0.5.

(d) Generated baby image of (c).

(e)Morphed image with eyes from image 2 and other features' alpha are 0.5.

(f) Generated baby image of (e).

Other results:

Fig15.

Example results of generated baby image. The right column shows the generated baby image of the parent images in the left and middle columns.

5: Conclusions

Although there are several papers and online websites already doing face morphing even age transformation, few people devote themselves to discussing the detail of face morphing. As a consequence, in this project, we consider most of detail in face morphing and succeed in generating natural composite children face images.

We successfully produce better results in face morphing than previous work because we deliberately choose specific landmark points around nose. Besides, it is easy for users to tune the facial features ratio in mouth, eyes and nose regions. It is worth mentioning that detail refinement works very well around nose region by using Seamless cloning and Smooth boundary. Finally, the algorithm of age transformation enable to combine all of the information of adult image and base baby image not just some face feature regions.

6: References

[1] S.Divel and P.Shunhavanich, “Baby Face Generator,“in Department of Electrical Engineering,”Stanford University,Stanford, CA 94305

[2] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in Computer Vision and Pattern Recognition, 2001. CVPR 2001.Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1. IEEE, 2001, pp. I–511.

[3] M. Uˇ riˇ c´ aˇr, V. Franc, and V. Hlav´ aˇc, “Detector of facial landmarks learned by the structured output SVM,” in VISAPP ’12: Proceedings of the 7th International Conference on Computer Vision Theory and Applications, G. Csurka and J. Braz, Eds., vol. 1. Portugal: SciTePress — Science and Technology Publications, February 2012, pp. 547–556.

[4] J. Canny, “A computational approach to edge detection, ” Pattern Analysis and Machine Intelligence, IEEE Transactions on, no.6, pp.679–698, 1986.

[5] Davis King,dlib C++ Library.

[6] "http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2"