# Face Swap Realization

Xiaowen Zhang, Haitian Sun, Yulong Cao
University of Michigan
500 S State St, Ann Arbor, MI 48109
face-swap@umich.edu

## Abstract

*The project aims at devising an algorithm for face swapping and an implementation that follows. This proposal presents the objectives of this project, an introduction to the algorithms used in the project, the implementation plan, and the expected output of this project.*

## 1. Introduction

There're many face-swap applications that have implemented different methods of face swaps and thus it makes it interesting to implement a face swap algorithm that is able to detect and mark face features in an image, transform one face and align it with another, and blend the two faces to create the final output.

The goal of this project is to implement an efficient algorithm to swap two faces in two images. We will implement everything by ourselves. We won't use any computer vision packages except for some training algorithms (e.g. SVM).

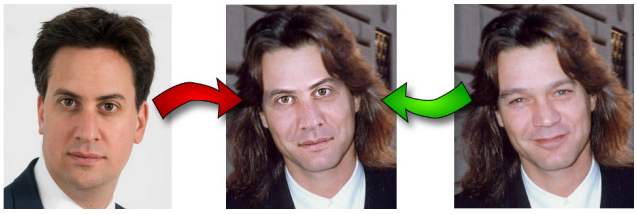

Figure 1. Example of face swapping [1]

## 2. Approach

In this section, we will discuss the approach we use to achieve our goal. Firstly, we detect the rough position of the face in the photo. We assume that there is exactly one face in the photo and the face is the dominant object in the photo (i.e. The height of the face is no smaller than 1/3 of height of the photo). Then, we detect the positions of eyes and mouth on the rough face detected in the first step. Based on the position of mouth and eyes, we find the exact template of face in the photo. Finally, we transpose the template of face from another photo and merge it with the current photo. We will elaborate all steps in the following sections.

### 2.1. Rough Face Detection

#### 2.1.1 Training Face Classifier

Firstly, we trained a classifier for face detection. We use "Large-scale CelebFaces Attributes (CelebA) Dataset" as our training data. Each photo in the dataset has one face with eyes, nose, and mouth annotated (Figure 2). Faces on all photos are aligned such that eyes are in the same height, so we don't need to rotate the photos to get front faces. Here is an example of annotated data.

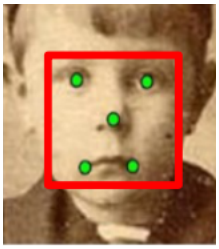

Figure 2. Example of annotated data [3]

The green points on the photo are facial landmarks, i.e. left and right eyes, nose, and left and right corners of mouth. The red rectangle is the tight boundary of the face. The left boundary is set as a distance away from left eye. The distance is calculated by a fixed ratio times the distance between two eyes. Similarly, we find the right boundary. The upper and lower boundaries are also calculated as a fixed ratio of distance from eyes and mouth.

Given the cropped faces as training data, we can extract features and train the classifier. We resize the face to 19 * 19. Then, we extract HoG features with cell size of 3. We reshape the features into a row vector as the input of training algorithm. Finally, we use SVM to train the

classifier of faces.

Besides data for faces, we use "CBCL Face Dataset"[2] as dataset for non-face data. The total training set has half face data and hald non-face data.

#### 2.1.2 Face Detection with Sliding Window

Now that we have classifier that can judge if the input is a tightly bounded face, we will introduce a method that circle faces from photos.

The basic idea is using a sliding window. There are two more factors we need to take care of. The first factor is the size of the window. The second factor is the ratio of height over width of the window. Since the tightly bounded faces are resized to 19 * 19 when training the classifier, the ratio of the window should also be variable. Intuitively, this handles the case that people's face have different shapes.

In real scenario, there is a high probability of false positive. Many windows are falsely classified as face. To avoid such situation, we choose the window with highest classification score as the rough boundary of the face.

### 2.2. Eye and Mouth Detection

#### 2.2.1 Training Eye and Mouth Classifier

We use the similar methods to get training data for eye and mouth. We simply set a fixed threshold to get eyes and mouth from the annotated data [3].

To get data for non-eye and non-mouth, we randomly crop the original photos. However, we won't add it to training data if it's too close to the real mouth/eye.

#### 2.2.2 Eye and Mouth Detection with Sliding Window

Given the rough face detected in the previous step as the potential area where a face can appear, we will enlarge the potential area and detect eyes and mouth in the enlarged area.

Since the probability of false positive of eye and mouth detection is much higher than that of rough face detection, we cannot simply choose the window with the highest classification score. So, we use a similar method as blob detection.

Firstly, we apply the sliding window and calculate the classification score for each window. Secondly, we set a threshold for the classification score. Thirdly, We do non-maximum suppression to find several potential eyes and mouths. Note that it is possible more than two eyes and one mouth are detected. We will discuss how to finally determine the real eyes and mouth in the following sections.

### 2.3. Face Template Extraction

This section describes how to extract the face template from the roughly detected face and the detected eyes and mouth(s).

#### 2.3.1 Eyes and Mouth Selection

Since the eyes detection and mouth detection potentially could give more or less eyes or mouths than needed, first, an error handling is applied, which counts the number of detected eyes and mouth(s) and retry detecting them if there are less than two eyes or less than one mouth, with a lower detection threshold.

Second, among all the combination of two eyes and one mouth in the detected $N$ eyes and $M$ mouth(s), a combination that satisfies the following conditions is chosen to the candidate eyes and mouth.

$$norm(\mathbf{eye}_a - \mathbf{eye}_b) < \alpha \cdot norm(\frac{\mathbf{eye}_a + \mathbf{eye}_b}{2} - \mathbf{mouth}_c) \tag{1}$$

$$dist(\mathbf{mouth}_c, PB(\mathbf{eye}_a, \mathbf{eye}_b)) = \min_{\forall i \in \{1,2,...M\}, j,k \in \{1,2,...N\}} dist(\mathbf{mouth}_i, PB(\mathbf{eye}_j, \mathbf{eye}_k)) \tag{2}$$

where equation (1) represents the constraint that the distance between two eyes has to be smaller than a certain ratio times the distance between the middle of two eyes and the mouth, and where equation (2) means the distance between the mouth and the perpendicular bisector should be minimized among all combinations.

#### 2.3.2 Mask Generation

After having two eyes and one mouth, we calculated the incline angle of the face based on the coordinates of the eyes, which is then used to calculate the coordinates of the mask as shown below.

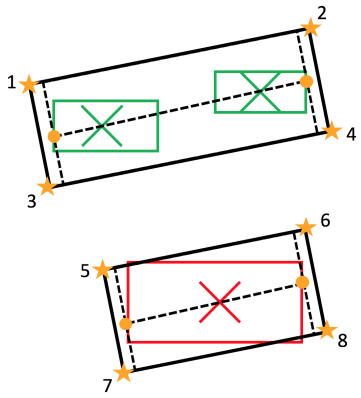


Figure 3. Mask calculation diagram

The green crosses and boxes indicate the locations and detected sizes of eyes, whereas the red ones are for the mouth. The dashed lines are connection lines and extension lines, the latter of which have a predefined length. The orange dots indicate the locations of intersections, and the orange stars indicate the locations of eight mask points calculated. Most of the dimensions of the mask have a predefined ration relative to the image size.

## 2.4. Face Swapping

In this part, given the positions of landmarks of faces in two image, we will swap one face with the other one. In order to achieve that, we generally need two steps: face registration (transform one face to the same position of the other one), face replacement (change the face in original image naturally).

### 2.4.1 Face registration

To swap face, we want to change eyes, nose and mouth. Therefore, we decide to choose the part including all of them using the landmark positions of left eye, right eye, left corner and right corner of the mouth.

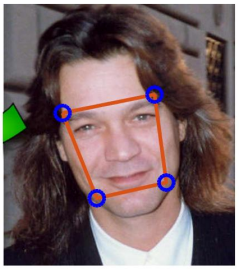

Figure 4. Face swapped component

Also, since we are using matlab to manipulate the image, which means the image is represented as matrix, we need to transform the swap component into a straight rectangle to avoid rounding problem in the transformation. After transforming both faces into rectangles, we reshape the second face to the same size of the first one and we finished preparation for face replacement.

Figure 5. Example of transformed faces

(Note: here we can see, sometimes the face component will include part outside of face. We will handle these in the step of face replacement.)

### 2.4.2 Face replacement

In this part we will replace the transformed face naturally. In order to achieve that, we use a guassian mask with decaying boundary.

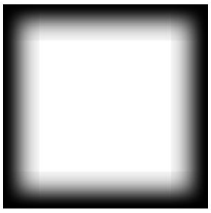

Figure 6. Guassian mask for face replacement

Instead of using default guassian filter provided by matlab, this mask designed by us can avoid faces combination (which is not swapping face) and obvious boundaries due to replacement of a rectangle area of the image.

Besides of the guassian mask, we also normalized the face component in order to decrease the influence of the light effects.

Figure 7. Example of face swapping with and without normalization

## 3. Implementation

In the rough face detection, eye detection, and mouth detection part, we pre-process training data ourselves. We use the HoG feature extractor in the vl-feat toolbox to extract features. We use the Matlab built-in SVM training and predicting functions. We implement sliding windows and non-maximum suppression by ourselves.

In the face swapping, we simulate the transformation with featured point and calculate the final position of the landmarks after transformation. Using Guassian function, we implement a Guassian mask that quickly and smoothly transit one face to the other. Applying the normalization to the swapped face, we reduce the effect of different light environment of images.

## 4. Experiments

### 4.1. Accuracy of Classifiers

#### 4.1.1 Rough Face Classifier

The training dataset and testing dataset have half face data and half non-face data. The accuracy of rough face classifier is 98%.

#### 4.1.2 Eye Classifier

The training dataset and testing dataset have half eye data (left and right eye together) and half non-eye data. The accuracy of eye classifier is 93%.

#### 4.1.3 Mouth Classifier

The training dataset and testing dataset have half mouth data and half non-mouth data. The accuracy of eye classifier is 96%.

### 4.2. Sliding Window and Non-maximum Suppression

#### 4.2.1 Rough Face Detection

As we can see in the result (Figure 3), the red rectangle shows the rough position of the face. It's not a tight boundary of the face, but it gives enough information for the detection of eyes and mouth detection.

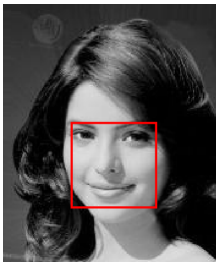

Figure 8. Example of rough face detection

#### 4.2.2 Eye Detection

This example shows the eyes detected on the face (Figure 4). The red rectangles show eyes detected whose scores pass the threshold. The blue crosses ('x') represent the centers of eyes after non-maximum suppression.

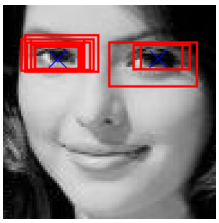

Figure 9. Example of eye detection

#### 4.2.3 Mouth Detection

This example shows the mouths detected on the face (Figure 5). The red rectangles show mouths detected whose scores pass the threshold. The green crosses ('x') represent the centers of eyes after non-maximum suppression.

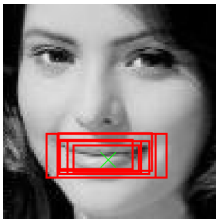

Figure 10. Example of mouth detection

### 4.3. Face Template Extraction

#### 4.3.1 Accuracy of Eyes and Mouth Selection

The following example shows the selection result (bottom) from a set of 3 detected eyes (upper left) and 3 detected mouths(upper right). The ratio $\alpha$ is adjusted to be 0.8.

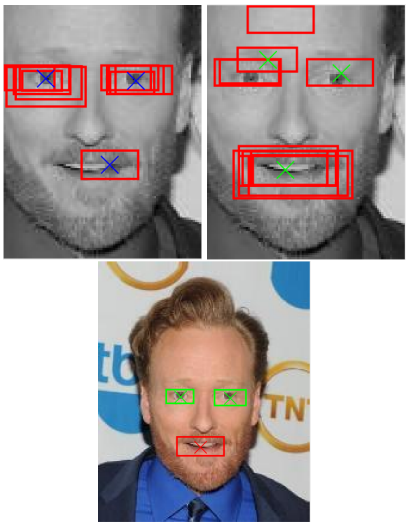

Figure 11. Example of eyes and mouth selection

It is noticed that eyes and mouths are detected around similar positions, which is understandable since they all have curly features on the sides and have a somewhat similar shading feature.

### 4.4. Mask Generation

The following example shows calculated mask points as described in section 2.3.2, where the cyan crosses indicate the locations of the mask points.

Figure 12. Example of calculated mask points

### 4.5. Face replacement

#### 4.5.1 Replace with irregular area

The following figure demonstrates an attempt to replace the exact face area marked by detected facial landmarks.

Figure 13. Example of replace with irregular area

We can see there is a clear black line that pollutes the result. This is due to the rounding value of transformation and can hardly be reduced, even with median filter. Compared to the replacement of a rectangle area, the latter is more reasonable. Therefore, the final swapping algorithm uses only rectangle areas.

## 5. Conclusion

As can be seen below (the two images above are the source images and the bottom one is an output image with the upper left image serving as a face frame), our face swap method successfully achieved the final outcome: swap two faces in two images. Our implementation is able to detect the facial landmarks needed for swapping, and it is able to blend two images without obvious artifacts.

Figure 14. Example of a final output

However, our algorithm does take a long time (around 1 minute) to detect the face and facial features (eyes and mouth), as a result of not implementing cascaded detection classifier.

## References


[1] M. Earl. Switching eds: Face swapping with python, dlib, and opencv. 2015.

[2] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[3] X. W. Ziwei Liu, Ping Luo and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.