CMSC 733 Project II: FaceSwap

Naitri Rajyaguru
email: nrajyagu@umd.edu

Angelos Mavrogiannis
email: angelosm@umd.edu

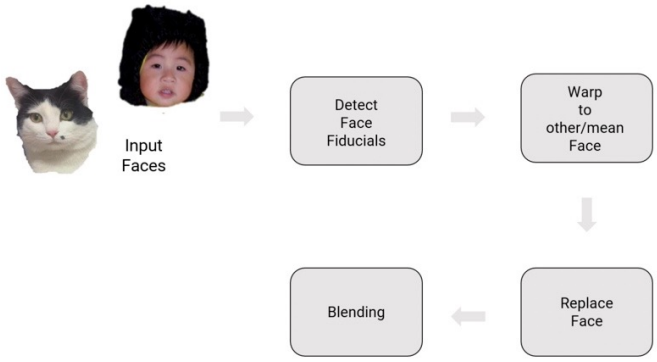
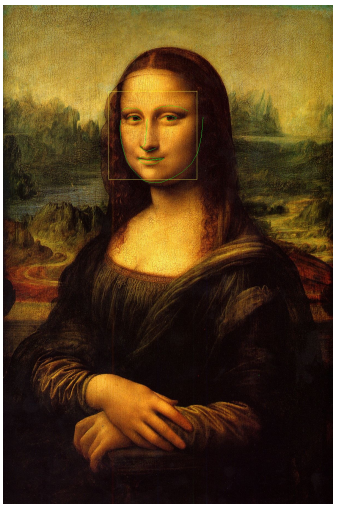


Fig. 1: The pipeline of the general faceswapping approach includes detecting face fiducials (facial landmarks), warping an image to an intermediate one, replacing faces, and finally blending the output so that it looks natural.

(a) Face detection src image



(b) Face detection dst image

***Abstract*—This project consists of a set of different methods that perform face swapping. Phase I follows a traditional approach and explores face warping via Delaunay Triangulation and Thin Plate Spline, while Phase II follows a deep learning approach that uses a network to obtain face fiducials. After the face replacement in each method, we need to blend the face in the new image so that it looks natural, and we need to make adjustments so that our pipeline works with faces in videos.**

I. Phase I: Traditional Approach

Our initial logic is based on identifying important features on a face, storing their locations and attempting to extract the locations of the corresponding features on the other face. We can then orchestrate an approach that leverages this one-to-one feature correspondence and employs different mathematical tricks to replace the two faces and make the replacement look as natural as possible. The overall approach can be seen in Fig.1.

A. Facial Landmarks Detection

We begin the implementation of the traditional methods by obtaining the facial landmarks of the two faces that we want to swap with each other. To detect the facial landmarks, we first need to localize the face in the image. For the face localization we use a pre-trained Histogram of Gradients (HOG) and Linear Support Vector Machine (SVM) object detector provided by the function *get_frontal_face_detector* from the *dlib* library. This provides us with a bounding box that is centered on the face of the input image. Having acquired this bounding box, we can now use a facial landmark detector focused on this rectangular area. Again we are using a built-in detector from the *dlib* library using the function *shape_predictor*. This is a pre-trained detector based on the implementation in [1], that attempts to extract the coordinates of 68 predefined key facial structures on a face.

B. Face Warping using Triangulation

The first classical method we implemented for the faceswap is based on the Delaunay Triangulation, which is the dual of the Voronoi diagram and consists of the sites of a Voronoi diagram, connected to each other with lines. We compute a triangulation for every face using the extracted facial landmark points as corners and assume that each triangle is planar. The intuition behind this is that we need to move to 3D from the 2D face representations. Using this simple planar assumption we establish an affine relationship between the triangles that consist of corresponding facial landmarks. We use the Delaunay Triangulation due to some of its nice properties, including the fact that it avoids returning triangles with very acute angles, maximizing the smallest angle in every triangle. Revisiting our initial plan, which was to establish correspondences between the facial landmarks of two faces, we expand this idea and establish correspondence between the triangles of the two face triangulations. Since a total of 68 facial landmark points were used, we will have a one-to-one correspondence between the triangles of the two triangulations. However, during implementation we noticed that despite the one-to-one correspondence of the facial landmarks, the triangles produced were not consistent in the two images. As a

workaround, we applied the Delaunay Triangulation on the points of the convex hull (using the function *convexHull* from *opencv*) of the extracted landmarks. We found out that this adaptation improves the correspondence between the triangles of the source and destination images. We compute the triangulations using the function $Subdiv2D.getTriangleList()$ from the *opencv* library. After that, we need to establish specific point correspondences between the triangles of the two triangulations. One way to do this is through the use of barycentric triangle coordinates, which formulate a general and elegant mathematical description of the relationship of a point with a triangle. More specifically, we first compute the forward barycentric transformation:

$$\begin{bmatrix} \mathcal{B}_{\alpha,x} & \mathcal{B}_{b,x} & \mathcal{B}_{c,x} \\ \mathcal{B}_{\alpha,y} & \mathcal{B}_{b,y} & \mathcal{B}_{c,y} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \mathcal{B}_{\Delta} B = p \quad (1)$$

for each homogeneous point $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$ of the triangulation in our destination image. a, b, c represent the corners of the current triangle and $b = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}^T$ is the barycentric coordinate of the point $p = \begin{bmatrix} x & y & 1 \end{bmatrix}^T$. Hence, we can compute the barycentric coordinate of a point in our triangulation by inverting $\mathcal{B}_{\Delta}$:

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \mathcal{B}_{\Delta}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2)$$

The barycentric coordinates provide us with a convenient and efficient way to reason whether a point lies inside a triangle of our triangulation. Specifically, if $B = \alpha \in [0, 1], \beta \in [0, 1], \gamma \in [0, 1], \alpha + \beta + \gamma \in (0, 1]$, then the corresponding point lies inside the triangle. This formulation works in theory but introduced some holes in the triangles. To remove the holes and account for points that were left out initially, we relaxed the bounds of the intervals for α, β, γ, adding an $\epsilon = 0.1$ term and shifting the range from $(0, 1]$ to $(-\epsilon, 1 + \epsilon)$. Now that we have computed the barycentric coordinates of a point p from our destination image triangulation, we can locate the corresponding point in the source image triangulation in the following way. Using the same barycentric coordinate vector $B = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}^T$, we compute the forward transformation:

$$p\prime = \mathcal{A}_{\Delta} B \quad (3)$$

and hence compute the following matrix:

$$\mathcal{A}_{\Delta} = \begin{bmatrix} \mathcal{A}_{\alpha,x} & \mathcal{A}_{b,x} & \mathcal{A}_{c,x} \\ \mathcal{A}_{\alpha,y} & \mathcal{A}_{b,y} & \mathcal{A}_{c,y} \\ 1 & 1 & 1 \end{bmatrix} \quad (4)$$

Then, $p\prime$ is going to give us the coordinates $\begin{bmatrix} x_A & y_A & z_A \end{bmatrix}^T$ of the corresponding point in the source image, which we divide by z_A to acquire the homogeneous representation. Finally, we use a 2D interpolation method, and specifically the function *interpolate.interp2d* from the *scipy* library to copy the pixel intensity value of the source image at (x_A, y_A) back to the target location in our intermediate or output image. We use a cubic spline interpolation method for swapping faces from different videos and a linear spline interpolation method for swapping faces in a single video, as the cubic method led to errors due to the dimensions of the input array at some video frames.

Fig. 3: Rambo on Test1.mp4 with blending using triangulation

Fig. 4: Scarlett on Test3.mp4 with blending using triangulation

Fig. 5: Output on Test2.mp4 with blending using triangulation

Fig. 6: Failure case when swapping two faces in Test2.mp4. This happens due to the different orientation and the poses of the faces not being focused towards the lens of the camera.

C. Face Warping using Thin Plate Spline

Triangulation assumes performing an affine transformation on each triangle and hence it is not very efficient to warp as the human face has a very complex shape. On the other hand, Thin Plate Spline can arbitrarily model very complex shapes. Using Thin Plate Spline, we want to get a smooth function that maps pixel co-ordinates from the source image to destination image. The smooth function is given by:

(a) Mask dst image (b) Face warp without blending (c) Face warp using tps

$$f_{x'}(x,y) = a_1 + a_x x + a_y y + \sum_{i=1}^{N} w_i U\left(\|(x_i,y_i) - (x,y)\|\right)$$

$$f_{y'}(x,y) = a_1 + a_x x + a_y y + \sum_{i=1}^{N} w_i U\left(\|(x_i,y_i) - (x,y)\|\right)$$

where U is

$$U(r) = r^2 \log\left(r^2\right)$$

$$L = \begin{bmatrix} K & P \\ P^T & O \end{bmatrix} \times \begin{bmatrix} w \\ a \end{bmatrix} = \begin{bmatrix} v \\ o \end{bmatrix}$$

where K represents $K_{ij} = U\left(\text{ distance } \left((x_i,y_i),(x_j,y_j)\right)\right.$ and ith of P is $(1, x_i, y_i)$.

The top row of the left side $\begin{bmatrix} K & P \end{bmatrix} \times [v,o]^T$ states that the function $f_{x'}$ and $f_{y'}$ with all control points is substituted. The second row of $\begin{bmatrix} P^T & O \end{bmatrix}$represents additional constraints to the system. The algorithm for face warping using Thin Plate Spline is explained in a step-wise manner as follows:

1) Use the landmark points obtained for both source and destination images for getting bounding boxes of the faces.
2) Get the estimated parameters of the source image using the equations mentioned above.
3) Get all the locations of the face in the mask formed by the destination image.
4) Now transform all the source landmark points using the estimated parameters and locations obtained by the mask.
5) Warp the source image to the destination image.
6) Blend the images as discussed in Section E.

D. Replace Face

Replacing face using both triangulation and TPS is similar as once we have a mask for the destination image we replace the pixels for the same locations from the source image. The results for TPS are shown below but as it is observed, it does not look natural and has different edges.

Fig. 8: Test3.mp4 without blending using TPS

Fig. 9: Test3.mp4 with blending using TPS

Fig. 10: Test2.mp4 with blending using TPS

E. Blending

For blending we use the Poisson Blending in-built function cv2.seamlessClone(). Here, we compute a mask of the face in the source image and calculate the location of the center of the source image in the destination image. This function supports two different modes of cloning: normal cloning and mixed cloning. Normal cloning preserves the texture of the source image in the output image, whereas mixed cloning yields an output texture which is the result of a combination of the texture of the source and destination images, introducing bias towards highlighting the most dominant texture and hence producing an output image that has smoothness discontinuities. Therefore, we choose the normal cloning method which proved to produce better and smoother output images. The results after blending are shown below:

Fig. 11: Rambo on Test1.mp4 without blending using TPS

Fig. 12: Rambo on Test1.mp4 with blending using TPS

We present results for the post-blended faceswap using triangulation for Test1.mp4 and Rambo.jpg in Fig. 3, for Test3.mp4 and Scarlett.jpg in Fig. 4, and for Test2.mp4 in Fig. 5.

II. Phase II: Deep Learning Approach

In the deep learning approach, we use PRNet [2] which stands for Position Map Regression Network. This network simultaneously reconstructs the 3D facial structure and provides dense alignment. To achieve this, this approach utilizes a 2D representation called UV position map which records the 3D shape of an entire face in the UV space and then trains a CNN to regress it from a single 2D image. The implementation code we use here has been provided by the authors of the paper. It also involves using dlib model to detect faces but a CNN detector works with higher accuracy. The network architecture is shown in Fig. 13.

III. Analysis

1) Dlib is not very accurate in providing face detection if the face orientation is not as required and hence in those cases, we have to skip those frames. On the other hand, PRNet is way superior in terms of face detection.

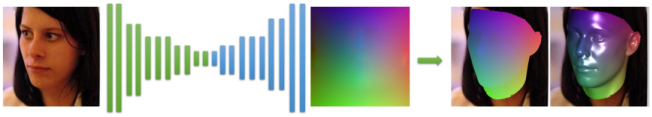


Fig. 13: PRNet architecture

Fig. 14: Test1.mp4 using PRNet

Fig. 15: Test3.mp4 using PRNet

2) Thin Plate Spline performs significantly better in image-to-image swapping but has certain disadvantages when used for face swapping in videos since little shifts in pixel locations of the images substantially decrease its performance.
3) The Delaunay Triangulation has a notable disadvantage compared to PRNet and TPS in terms of efficiency since it is based on the computationally expensive computation of the barycentric coordinates.
4) We present a failure case of the faceswap using triangulation in Fig. 6. Due to the different face orientation and the continuous changes in the poses of the people in the Test2.mp4 video, the *dlib* face detector is unable to detect the faces in multiple frames. This also happens when attempting to swap Ben Affleck's face with Scarlett Johansson's face as Ben's face constantly changes its orientation (Fig. 4). Furthermore, when the the orientation of the faces is not centered towards the lens of the camera, the output image for the faceswap looks unnatural and the correspondence of the facial landmarks and triangles.

REFERENCES

[1] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.
[2] Yao Feng, Fan Wu, Xiaohu Shao, Yanfeng Wang, and Xi Zhou. Joint 3d face reconstruction and dense alignment with position map regression network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.