CMSC733: Project 2 -FaceSwap

Chayan Kumar Patodi
UID : 116327428
Email: ckp1804@terpmail.umd.edu

Saket Seshadri Gudimetla Hanumath
UID : 116332293
Email: saketsgh@terpmail.umd.edu

***Abstract*—The purpose of this project is implement a pipeline to swap two faces present in images or videos. It is similar to the widely used Snapchat's swap filter. For implementing the said face swap both traditional and Deep Learning based approaches have been employed. Traditional methods involves the use of Delaunay Triangulation and Thin Plate Splines while the Deep Learning model uses Position Map Regression Network(PRNet) to achieve the same.**

I. Phase 1 : Traditional Approach

In this section we present the flow of how the traditional approaches work. To begin with, we generate our own data-set to work with. We use dlib from Python to detect the face features. Figure 1 shows the overview of the general pipeline which is followed in both the approaches which will be discussed in the latter sections. The pipeline is divided into 4 major steps viz. detection of facial landmarks(key points that are common to most human faces), warping source to destination image, overlaying the source face on destination face and blending to ensure a smooth and seamless output.

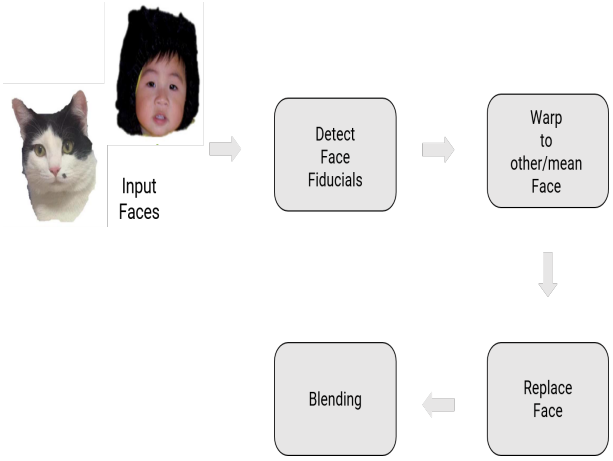

Fig. 1. Overview of the face replacement pipeline.

A. Facial Landmarks detection

The very first and important step in the traditional approach is to find the facial landmarks or key-points on the face, so that we have one-to-one correspondences between the two faces. Facial landmarks are used to represent regions of the face such as Eyes, Eyebrows, Nose, Mouth, and Jawline as seen in figure 2. Our goal is to detect important facial structures on the face using shape prediction methods. One of the major reasons to use facial landmarks instead of using all the points on the face is to reduce computational complexity. Detecting facial landmarks is a two step process - localise the face in the image and then detect the key facial structures on the face ROI. We can use OpenCV's built-in Haar cascades, or might apply HOG + linear SVM object detector. For this project we have used **dlib** library built into python. The pre-trained facial landmark detector inside the dlib library is used to estimate the location of the 68 important landmarks. Outputs of dlib can be seen in figure 3.

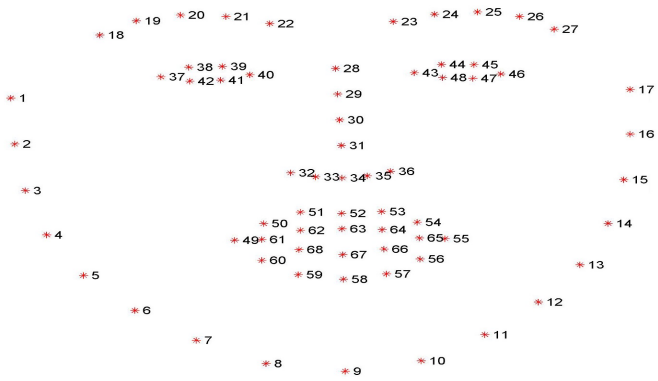

Fig. 2. Keypoints generated by Dlib

Fig. 3. Facial Landmarks detected(read from top to bottom) in Data-1, 2, 3

B. Face Warping using Triangulation

After obtaining facial landmarks faces must be warped in 3D. Since there is no 3D information of the faces available small assumptions can be made about the 2D images. The faces are divided into triangles (with the facial landmarks as the vertices) and the assumption is made that content inside the triangle is planar. Thus, it means the warping/transformation between the two faces is **affine** in nature. Delaunay Triangulation is used to perform the triangulation in the faces. It ensures that the smallest angle in each triangle is maximised which means there can not be triangles in a face with very big angles. This fixes the correspondence issue between the two faces. It also, in turn ensures consistency with the image boundary such that texture regions won't fade into the background while warping. This part is implemented through ***cv2.Subdiv2D.getTriangleList()*** function[9]. The output of this step can be seen in figure 4.

Once a correspondence has been established between the two faces warping can be done. For this part inverse warping is performed as it does not leave any holes after the warp in the source image[10]. This warping is performed using Barycentric Coordinates. The process of warping is explained as follows-

$$\begin{bmatrix} B_{a,x} & B_{b,x} & B_{c,x} \\ B_{a,y} & B_{b,y} & B_{c,y} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1}$$

The task is to find out which points lie inside a particular triangle of the destination face using the equation (1), find their respective Barycentric coordinates and then find out their corresponding locations in the source face. Here, (α, β, γ) are the Barycentric Coordinates. The matrix on the left contains the coordinates of the three vertices of each Triangle present in the destination face. (x, y, z) are the coordinates of the points that lie inside the triangle considered.

The condition is that for each point within the vicinity of a triangle in destination face(bounding rectangle in our case) the Barycentric Coordinates are constrained to certain values viz- $\alpha \in [0, 1]$, $\beta \in [0, 1]$ and $(\alpha + \beta + \gamma) \in [0, 1]$. However, introducing a factor $|\delta| = 0.1$ to Barycentric constraint condition gave better results.

Once the inside points and their Barycentric coordinates of a triangle in the destination image are obtained, the corresponding points in the source image are computed using the same (α, β, γ) although with a different set of triangle vertices(those of the corresponding source triangle). This can be understood from the equation (2)-

$$\begin{bmatrix} x_{\mathcal{A}} \\ y_{\mathcal{A}} \\ z_{\mathcal{A}} \end{bmatrix} = \mathcal{A}_{\Delta} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \tag{2}$$

where,

$$\mathcal{A}_{\Delta} = \begin{bmatrix} \mathcal{A}_{a,x} & \mathcal{A}_{b,x} & \mathcal{A}_{c,x} \\ \mathcal{A}_{a,y} & \mathcal{A}_{b,y} & \mathcal{A}_{c,y} \\ 1 & 1 & 1 \end{bmatrix} \tag{3}$$

Once the corresponding coordinates(warped source points) are obtained they are converted back to Cartesian from Homogeneous format and then the ***scipy.interpolate.interp2d()*** function is used to interpolate the intensities of the source face, which are then simply replaced from their corresponding counterparts in the destination face.

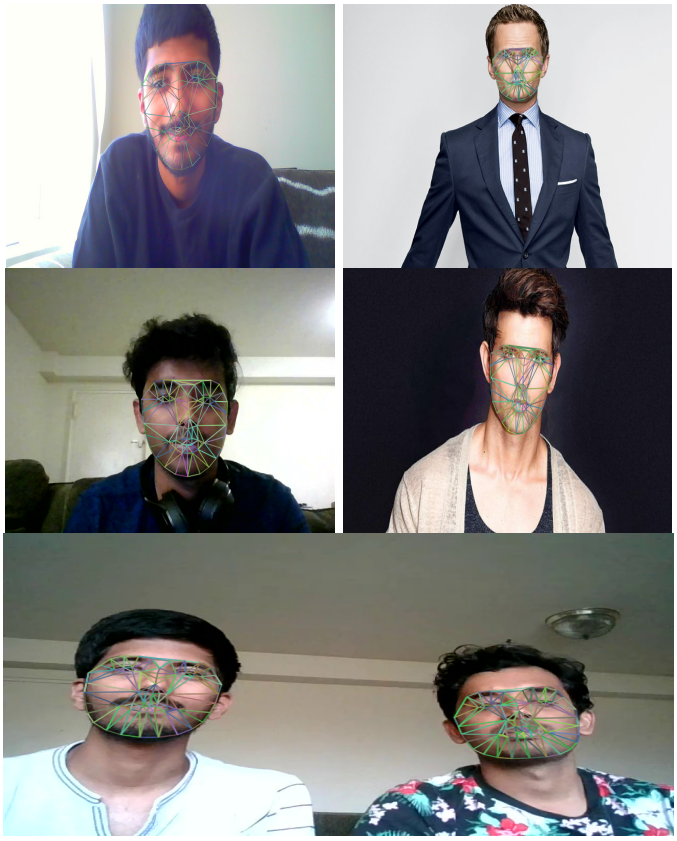

Fig. 4. Triangles in Data(from top to bottom)-1, 2, 3

The faces are warped to match the orientation of the other face. One such example is shown in Figure5 below. The warped faces are used to swap the faces and to generate the output, as seen in Figure6.

Fig. 5. Left: Original faces. Right: Faces warped to each other using Barycentric Coordinates.

Fig. 6. Face Swap using Triangulation, without Blending(read from Left to Right) in Data-1, 2, 3 respectively

C. Face Warping using Thin Plate Splines

As we discussed in the section above, triangulation assumes that we are doing affine transformation on each triangle. This might not be the best method to warp since the human face can be very complex and can have different shapes. A better way to do the transformation is by using Thin Plate Splines (TPS) which can model arbitrarily complex shapes to some extent. In layman terms, given two images, the goal is to deform an image(face) so it matches the second one(another face) and this is done using tps. Thin plate splines is a technique that provides a smooth interpolation between a set of control point [1].

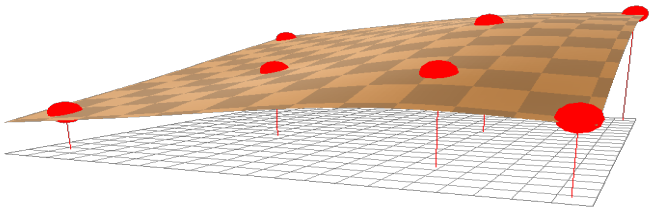

Fig. 7. A Thin Plate Splines that passes through a set of control points.[1]

A thin plate spline is given by the following equation 4:

$$f(x,y) = a_1 + (a_x)x + (a_y)y + \sum_{i=1}^{p} w_i U\left(||(x_i,y_i) - (x,y)||_1\right) \tag{4}$$

where,

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

In a nutshell , we are performing inverse warping, i.e., finding parameters of a Thin Plate Spline which will map from B to A. Warping using a TPS is performed in two steps [2]. The steps are given as follows:

Step1: Estimate Parameters of TPS, using this equation 5:

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \\ a_x \\ a_y \\ a_1 \end{bmatrix} = \left(\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} + \lambda I(p+3, p+3) \right)^{-1} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_p \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

Lambda is the regularization parameter. The regularization parameter , a positive scalar, controls the amount of smoothing; the limiting case of lambda equals to zero reduces to exact interpolation. As demonstrated in equation 5, we can solve for the TPS coefficients in the regularized case [6].

Step2: Use the estimated parameters for both x and y directions from Step 1, and transform all pixels in image B by the TPS model. Now, read back the pixel value from image A directly and replace the face by taking all the pixels from face A, warp them to fit face B. The pixels will not look natural as the lighting and edges look different. A sample output of face replacement using tps is shown below in figure 8:

Fig. 8. Face Swap using TPS, without Blending

D. Blending

This step involves making sure the face blends in seamlessly, and there are no discrepancies/artifacts occured in swapping the faces because of the edges or illumination. For this step, we use the inbuilt function cv2.seamlessClone from OpenCV library. We used ”normal” clone and ”mixed” clone interchangeably as sometimes we got good output using ”normal” and sometimes using ”mixed”. The output after passing in the swapped image and the mask, are given in figure 9 and 10:

Fig. 9. Face Swap using TPS, with Blending

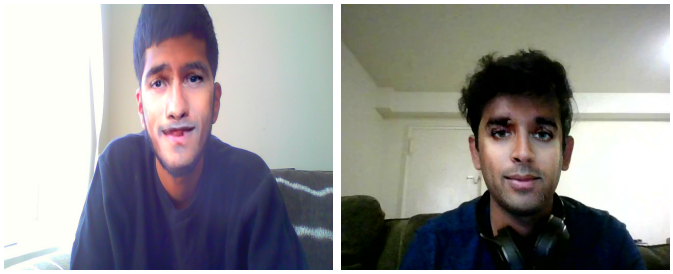

Fig. 10. Face Swap using Triangulation, with Blending

The final outputs for the Test Set are given in the Section ”Results on the Test Set” below.

II. Phase 2 : Deep Learning Approach

A. Facial Landmarks detection using PRNet

PRNet(Position Map Regression Network) method[8] simultaneously reconstructs the 3D facial structure and provides dense alignment. To achieve this, a 2D representation called UV position map is used which records the 3D shape of a complete face in UV space, then a simple Convolutional Neural Network regresses it from a single 2D image. PRNet like dlib also returns facial landmarks but, in addition it also provides depth and meshing.
This method does not rely on any prior face model, and can reconstruct full facial geometry along with semantic meaning. It is also very light-weighted and thus processing time on images is very less.
The CNN architecure used in PRNet can be seen in figure 11.

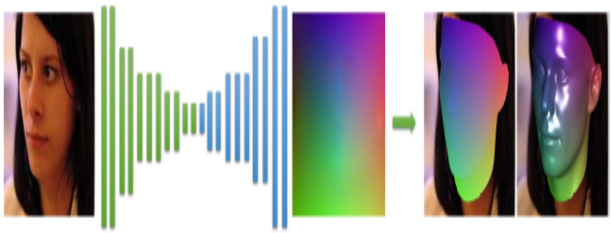

Fig. 11. The architecture of PRN. The Green rectangles represent the residual blocks, and the blue ones represent the transposed convolutional layers.

B. Warping and Blending

The warping and blending of faces is performed in two ways- the traditional way and using PRNet’s implementation of face swap.

1) Triangulation and TPS: First, the traditional pipeline was used with the only difference being that in this section PRNet is used to predict the facial landmarks instead of dlib. PRNet uses dlib to detect faces but then it uses its CNN to predict the facial landmarks. It can be seen in figure 12.

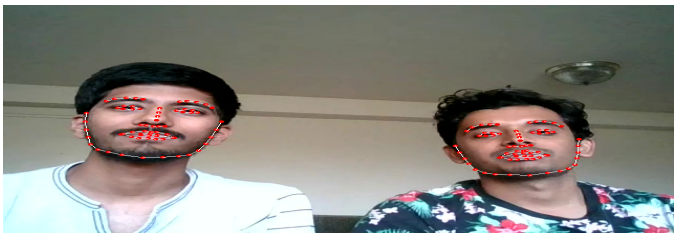

Fig. 12. Facial Landmarks predicted by PRNet

Hence, the landmarks are given to the two traditional methods which like before generate swapped faces. Output of this stage can be seen in figure 13 and 14.

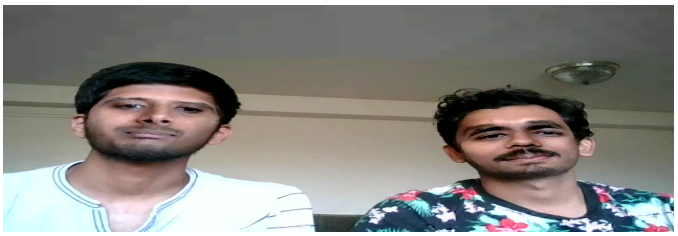

Fig. 13. PRNet+TPS

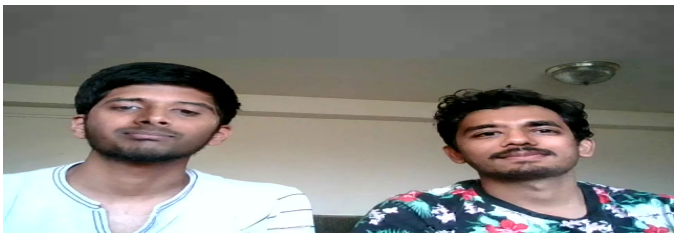

Fig. 14. PRNet+Triangulation

2) PRNet's Face Swap implementation: The PRNet's implementation replaces the texture of one with another, then warps it to original pose and uses Poisson editing to blend the two images. The outputs of this stage can be seen in figure 15.

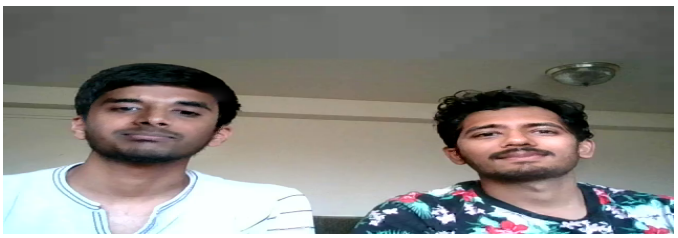

Fig. 15. PRNet's Face Swap

III. Results on Test Set

Fig. 16. Results on video Test1.mp4, using triangulation.

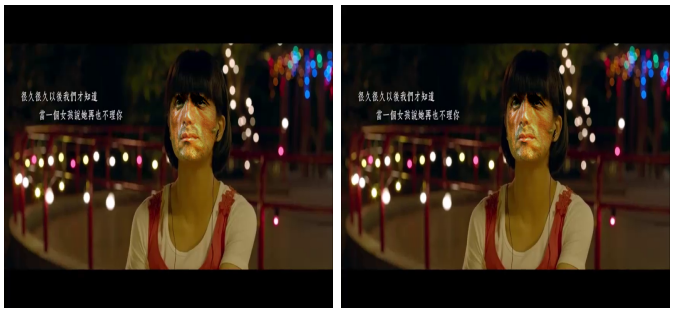

Fig. 17. Results on video Test1.mp4, using tps

Fig. 18. Results on video Test1.mp4, using PRNet

Fig. 19. Results on video Test2.mp4, using triangulation.

Fig. 20. Results on video Test2.mp4,using tps

Fig. 21. Results on video Test2.mp4, using PRNet

Fig. 22. Results on video Test3.mp4, using triangulation.

Fig. 23. Results on video Test3.mp4, using TPS

Fig. 24. Results on video Test3.mp4, using PRNet

IV. Analysis Of Outputs and Problems faced

1. The quality of the output varies depending upon various factors such as the camera view, illumination and also occlusion because Dlib fails to predict the facial landmarks correctly. For example,the following figure 25 shows a profile view of the faces in the frame. In such types of views the face swapping is bad.

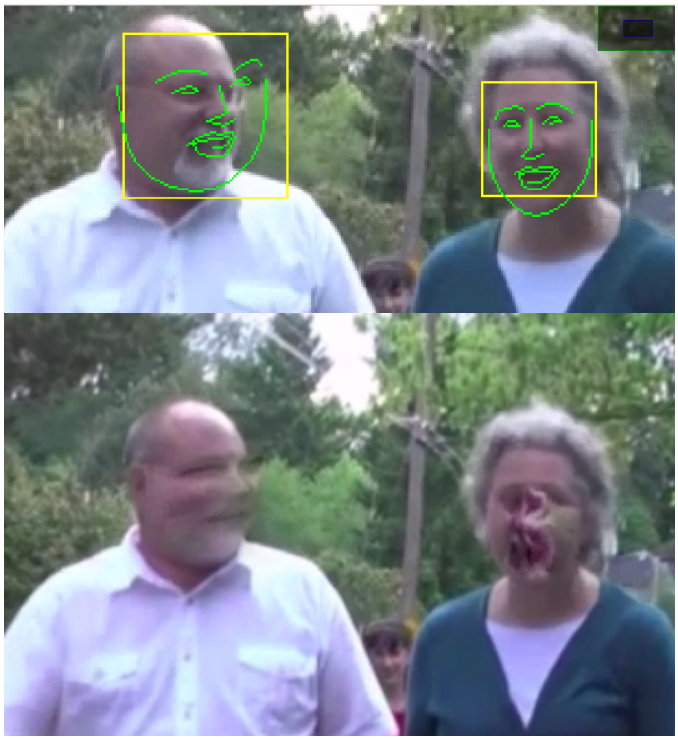

Fig. 25. Top : DLib keypoints for faces. Bottom :Incorrect swapping of faces

For images with occlusion, as seen in Figure26, even though the swapping occurs correctly, the face in the right has a finger occluding the face, which also shows up after swapping to the face on the left side in the frame.

Fig. 26. Occlusion and the output

2. The type of blending also has an impact on the quality of outputs. Faces with a lot of difference in skin tone, give good output when we use Mixed Cloning, whereas with faces in a single video with same illumination and same skin color, gave us good output with Normal Cloning. The outputs with different cloning methods can be seen in the figure 27.

Fig. 27. Different Cloning Methods. Top: Mixed Clone , Bottom: Normal Clone

3. TPS has an additional parameter for regularization. When using PRNet to predict keypoints, and using TPS to swap, it produces artifacts. These might be the result of not applying enough regularization and tweaking the lambda parameter could give us more good results. This can be observed in Figure 9 and 13. In figure 13, the second face has some discrepancies.

4. We drop a frame whenever dlib's predictor does not return sufficient keypoints in the face. This problem of dropping frames due to the inability to predict keypoints/facial landmarks in sparse illumination conditions is somewhat solved by PRNet's predictor. For e.g when we generated outputs for Test3 using dlib's predictor we got 21 frames but in case of PRNet we got 65. This shows PRNet's superiority.

V. Conclusion and Final Thoughts

We have successfully performed face swapping using the traditional and deep learning approaches discussed in the previous sections. Our final thoughts on the project and these method are -

Triangulation is the least effective of the three methods since it approximates the 3D information of the face using 2D shapes, in our case triangles. This approximation is sometimes not able to capture all the information of the complex human face. Although, it's benefit compared to TPS is it's running time since it's complexity is $O(n\log(n))$ compared to $O(n^3)$ of TPS respectively.

TPS gives slightly better outputs since it captures more of the complex structure of the face. But, it is slow and sometimes it requires the regularization parameter(λ) to be tweaked in order to get good output.

PRNet works really well since it accounts for the 3D information of the face(depth information) and can generate good results even when the view of the face is not parallel to the camera.

References

[1] Manual Registration with Thin Plates,Herve Lombaert, http://profs.etsmtl.ca/hlombaert/thinplates/

[2] FaceSwap, https://cmsc733.github.io/2019/proj/p2

[3] Thin Plate Spline editor, Jarno Elonen, http://elonen.iki.fi/code/tpsdemo

[4] F. Bookstein, "Principal warps: thin-plate splines and the decomposition of deformations," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 6, pp. 567–585, 1989.

[5] Thin-Plate Splines,David Eberly, Geometric Tools

[6] G. Donato and S. Belongie, "Approximate Thin Plate Spline Mappings," Computer Vision — ECCV 2002 Lecture Notes in Computer Science, pp. 21–31, 2002.

[7] https://mathworld.wolfram.com/ThinPlateSpline.html

[8] https://github.com/YadiraF/PRNet

[9] https://www.learnopencv.com/delaunay-triangulation-and-voronoi-diagram-using-opencv-c-python/

[10] https://www.cs.unc.edu/ lazebnik/research/fall08/lec08faces.pdf