CMSC 733 Project 2 Report
Face Swap

Darshan Shah
M. Eng Robotics
University of Maryland
College Park, Maryland 20740
Email: dshah003@umd.edu

Mayank Pathak
M. Eng Robotics
University of Maryland
College Park, Maryland 20740
Email: pathak10@umd.edu

I. Introduction

Given an image and a video stream, this project aims to implement end to end pipeline to swap faces. We achieve this goal with both traditional and deep learning approaches and then analyse the outcomes from both the methods.

II. Phase I: Traditional Approach

In this traditional pipeline, we implement image processing techniques using OpenCV and dlib. The overall pipeline is as follows: a. we read the image and detect the number of faces and then detect facial fiducials from the face. This is done for both, the source image and the target image. b. Triangulation and hence meshing is done based on the facial fiducial points c.a. Warping each of these meshes from the source to their corresponding mesh in the target image. d. Once the warping is done, the cropped image is blended with the target image to get a smooth image. The following sections cover these steps in detail.

1) Input image: Figure 1 and 2 shows the input image used for developing and testing the Phase 1 of this project

The image is loaded and converted to Gray-scale. We made use of the `get_frontal_face_detector()` and `shape_predictor()` functions from the dlib library. These further detect faces in the given image and also returns a list of 68 points each corresponding to a very specific landmark feature on the face. Due to which we get point correspondences for both the faces. This correspondence proves to be very crucial when applying triangulation and creating meshes. The output of the dlib feature detector is as shown in figure 3. Not all the points in the image are non-colinear to form triangles hence, we decided to only consider the points lying on the outside od the face. We do this by computing the Convex hull from all the points using the `convexHull()` function from openCV.

A. Warping using Triangulation

1) Triangulation: The facial landmarks obtained in the previous section act as the bases for the triangulation to be done in this step. We have made use of the Subdiv2D package provided by openCV. The `getTriangleList()` funciton returns the coordinates of all the vertices of each of the triangles formed during triangulation. This process of triangulation is performed on both, the source and Target image. Figure 4 shows the output of the triangles formed on both the Source and Target images.

2) Transformation and Warping: After the process of triangulation, we have obtained corresponding triangles in both the source and target images. We take advantage of Barycentric coordinate system to find the transformation of each pixel from triangle in source image to the triangle in target image. The Barycentric points are represented by α, β, γ

We estimate the bounding box of each triangle and iteratively compute The Barycentric coordinates of each pixel and

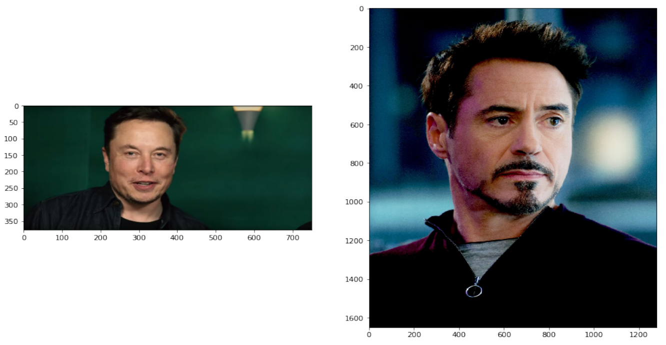


Fig. 1. The input image set 1

Fig. 2. The input image set 2

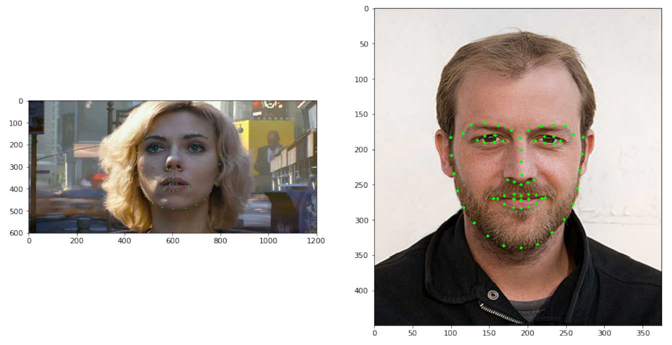


Fig. 3. 62 features detected by dlib's frontal face detector

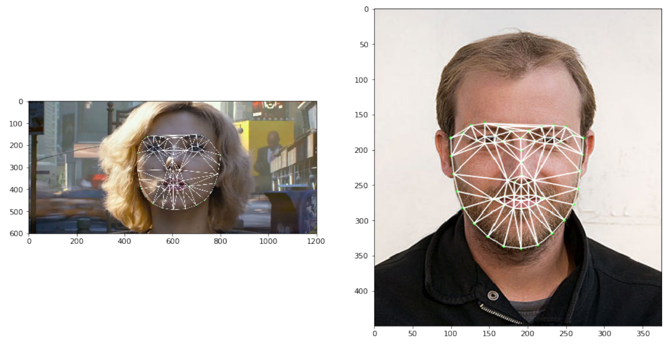


Fig. 4. Output of Delaunay Triangulation on the input image.

check if they lie inside the triangle or not. This is done by performing the following check:

$$\alpha >= 0, \beta > 0, \gamma > 0, \alpha + \beta + \gamma <= 1$$

If the pixel does lie within the triangle, with the help of barycentric coordinates, it's corresponding pixel location is found in the target triangle. To prevent holes in the image while transforming, we use inverse transformation as opposed to forward transformation. However the results are still not satisfactory. Ideally, to get a nice image, we need to have as many triangles as possible. However, this is a workable approximation. Finally the target image is blended with the source image using `seamlessClone` function of openCV. The `seamlessClone` Since we are dealing with pixel level manipulation, this process is computationally intensive and takes a while to execute. Hence it isn't the best method for this application.

The output of Triangulation is shown in the figure 5

B. Thin Plate Splines

Thin Plate Splines(TPS) can model arbitrary complex shapes and hence provide better results as compared to traingulation method used earlier.

Parameters of TPS are obtained by performing the matrix manipulation as suggested by authors in [1]. Lambda is used as e$^-$5, to account for noise and make sure that the $(p+3) \times (p+3)$ matrix is non-singular.

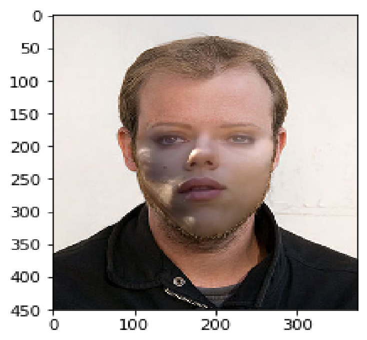
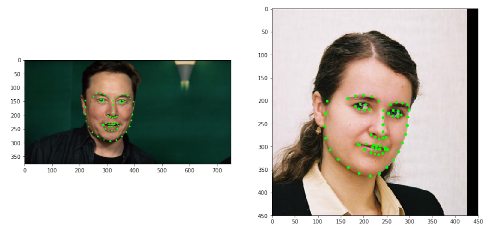


Fig. 5. Output of face swap by traingulation

(a) Sample input image for TPS



(b) face fiducials as detected by dlib

Fig. 6. Sample Destination and Source images for TPS

C.

III. Phase II: Deep Learning Approach

In this Phase, a pre-trained supervised encoder-decoder model is used to obtain the face fiducials [2]. The code from the paper is used as refernce for implementing this part of the project.

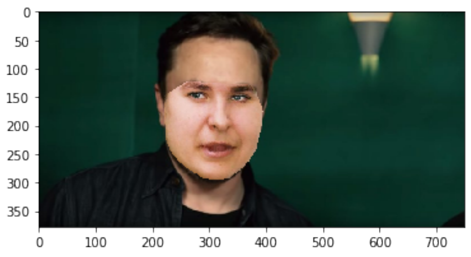
The encoder part of network begins with one convolution layer followed by 10 residual blocks, which reduce the 256 256 3 input image into 8 8 512 feature maps. The decoder part contains 17 transposed convolution layers to generate the predicted 256 256 3 position map. A kernel size of 4 is used for all convolution or transposed convolution layers, with ReLU layer for activation.

The trained model is saved in the project directory and is used for face swapping. Figure 8 shows some sample input images.

IV. Failure Cases

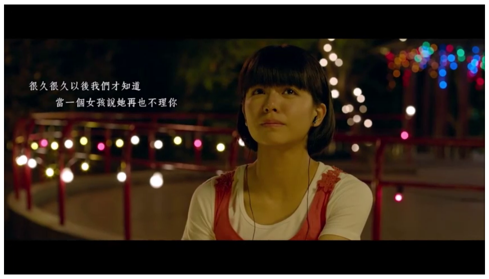
All the above implemented methods fail to detect a face if the face alignment is not proper as shown in figure 11a. Also, Triangulation method fails most out of the three methods in detecting faces provided the same data-set. For the figure

(a) Face masks for the images



(b) Output of TPS face swapping

Fig. 7. Face swap output for the sample images, using TPS.



(a) Sample destination image

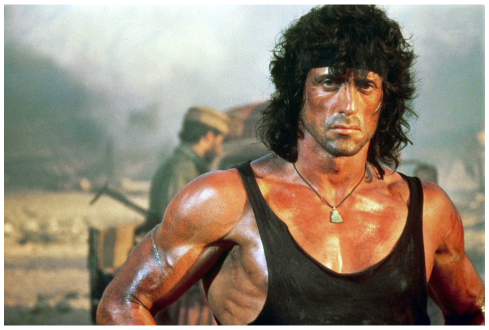

(b) Sample source image

Fig. 8. Sample input images used for testing network

11b TPS and PRNet manage to give an output whereas Triangulation fails to find a face.

V. Results Comparision

On observing the outputs, processing time and computational complexity of both the approaches, We can conclude that PRNet provides way better results, with less flickering and good color blending. The results of Triangulation are, however, not as satisfactory and the face warping is not up to par with other two methods. The process of Triangulation is piece-wise linear and assumes the transformation to be planar. whereas on the other hand, the neural net outputs a smooth face warp created from a 3D reconstructed face based on it's features.

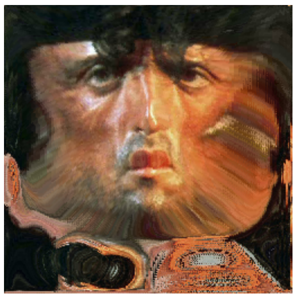
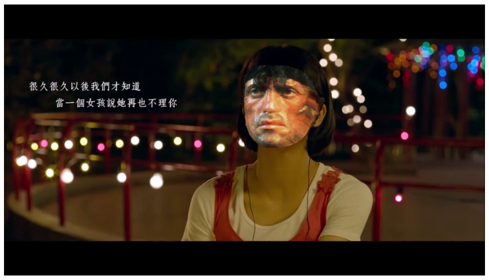
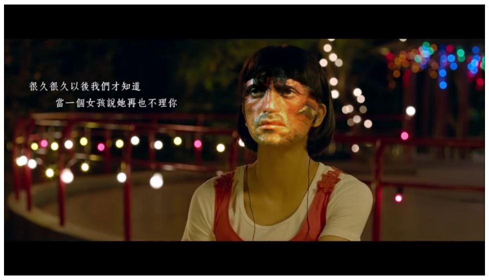


Fig. 9. Texture extracted from the source image

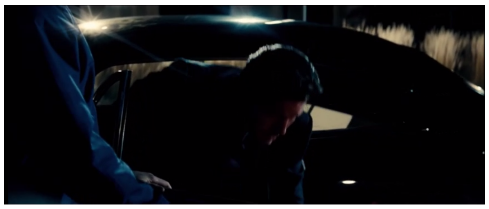


(a) Source face swapped on the destination image



(b) Output after seamless blending

Fig. 10. Face swap output for the sample images, using PRNet.


(a) Failure case 1 where face is not visible

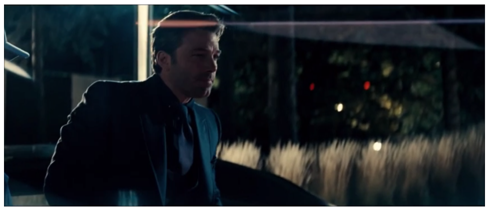
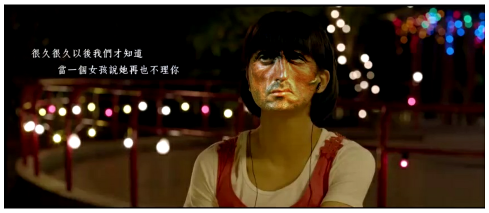
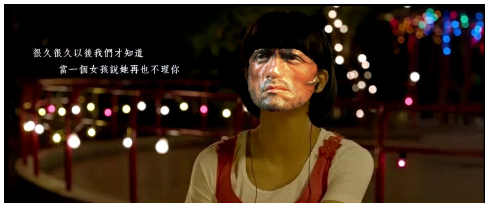

(b) Output by TPS method

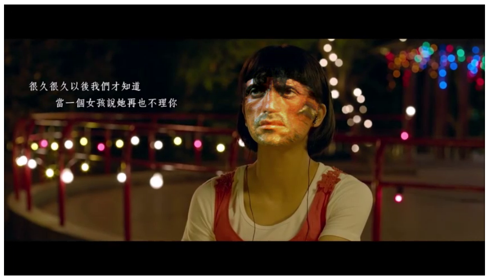
Fig. 11. Failure Cases



(a) Output by Triangulation method



(b) Output by TPS method



(c) Output by PRNet

Fig. 12. Comparision between all outputs

Between TPS and PRNet, PRNet gives better results accounting for different face alignments and illuminations. However, the Computation time required for PRNet is much more that what required for TPS. Also, close results can be obtained by applying edge dilution and seamless color morphing on the top of of TPS output.

References

[1] Gianluca Donato and Serge Belongie. Approximate thin plate spline mappings. In *Proceedings of the 7th European Conference on Computer Vision-Part III*, ECCV '02, pages 21–31, Berlin, Heidelberg, 2002. Springer-Verlag.

[2] Yao Feng, Fan Wu, Xiaohu Shao, Yanfeng Wang, and Xi Zhou. Joint 3d face reconstruction and dense alignment with position map regression network. *CoRR*, abs/1803.07835, 2018.