Project 2 - FaceSwap Submission

Abhishek Kathpal
CMSC733
UID:114852373
Email: akathpal@umd.edu
USING 2 LATE DAYS

***Abstract*—This project is focused on implementation of Face Swapping algorithm using both traditional and deep learning approach. The pipeline for the algorithm consists of detection of facial landmarks, inverse warping, blending and motion filtering. Facial landmarks are detected using dlib library. Delaunay Triangulation and Thin Plate spline are used for inverse warping. To detect the feature points more accurately, 3D face mesh is used in deep learning technique. All these techniques are implemented and discussed in detail in this project.**

I. PHASE1

A. Overview

The goal of the project is to implement Face detection pipeline to replace face in a video with celebrity as well as swapping two faces within the video. The pipeline for the traditional approach can be implemented using followng steps:
1. Detection of Facial Landmarks
2. Inverse Warping using Thin Plate Spline and Triangulation
3. Replacing the face
4. Blending the output to get an even texture and brightness.

These steps are described in detail in next sections.The pipeline is given by following figure:

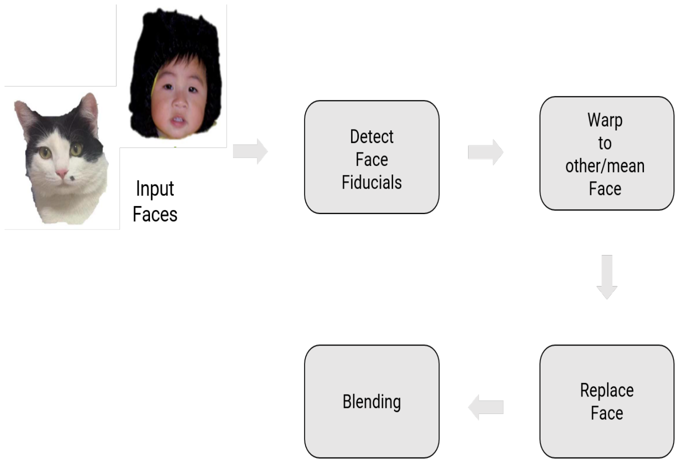

Fig. 1. Face Swapping pipeline

B. Facial Landmarks Detection

This is the most important step in the pipeline. This step help in finding corressponding points between two faces. There are many techniques for Face Features detection like using Hog Classifier, Haar Cascade filters etc. For this project, I have used inbuilt OpenCV library- dlib for detecting facial fiducials. Dlib library in OpenCV requires using a trained model file. For traditional approach, that trained file is based on hog filters and Linear SVM classifier. This gives out 68 facial landmarks.

The output for the facial landmarks on Scarlett Johanson and Robert Downey Jr.(Stark) is shown below:

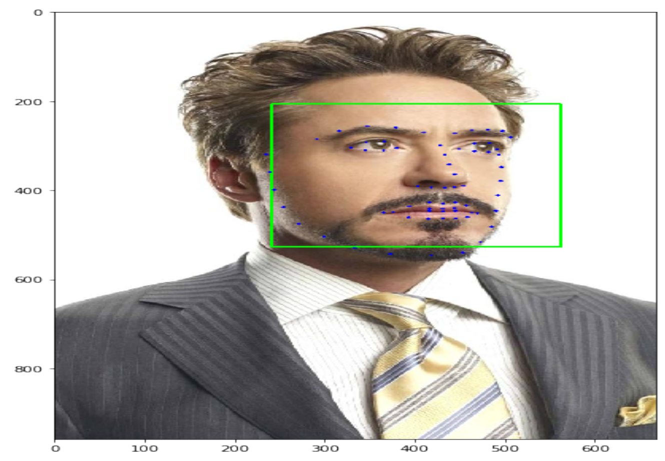

Fig. 2. Stark Facial Landmarks

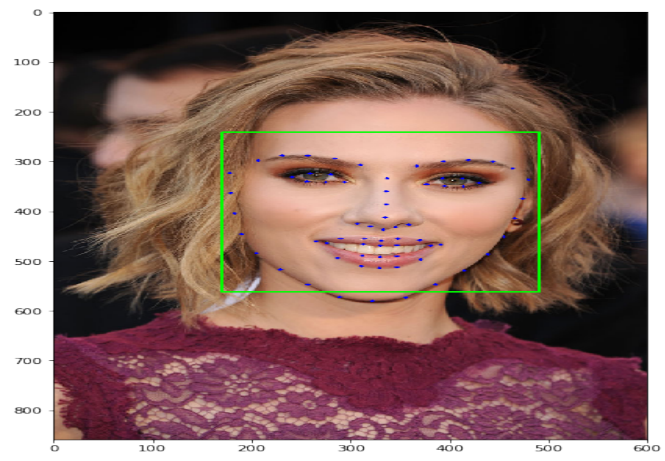

Fig. 3. Scarlett Facial Landmarks

C. Delaunay Triangulation

The next step after detection of facial landmarks is to warp the image of one face to another. Ideally, 3D information is required to properly warp it. But for this traditional approach we warped using 2D information. The image with facial points is subdivided into triangles and assuming that each triangle content is planar, we can warp using affine transformation.

Delaunay Triangulation is one of the fastest technique to obtain the triangulation in an image. It is equivalent to dual of Voronoi diagram. This techniques with each addition of point tries to maximize the smallest angle in each triangle.

The output after applying Delaunay Triangulation is shown in figure below:

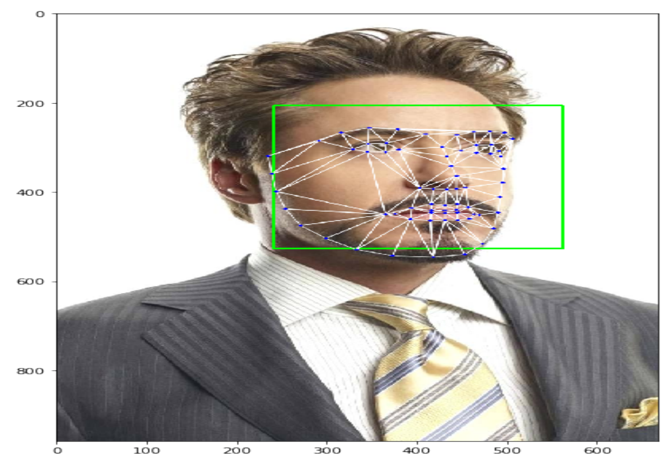

Fig. 4. Stark Triangulation

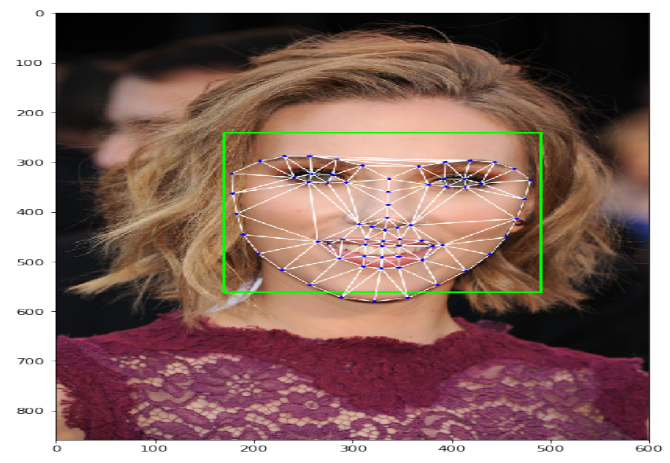

Fig. 5. Scarlett Triangulation

D. Inverse Warping using trangulation

After computing the triangulation I have observed that all the triangles of faces are not corresponding. To resolve this issue, I found two ways-

1. Using average of those two faces facial features and compute delaunay triangulation on those points and use that to find the corresponding triangles.
2. Using the convex hull points and doing triangulation using them. I found that triangles are coming out to be similar in both faces using this approach. This also reduces computation time but decreases fitting accuracy because some triangles near nose are bigger this way.

For implementation of inverse warping using triangulation, I have used 2nd approach. In later sections, better approaches for inverse warping are discussed.

For this part of project, I computed barycentric coordinates for each point within rectangle of destination image. These barycentric coordinates have property that their values will lie between (0,1] but in practical programming, that's not exactly true. As I was getting some holes in the boundary of triangles, so to avoid them I have set the range as (-epsilon,1+epsilon] of the coordinates to get the points within triangle.

Using the braycentric transformation matrix, I found the corresponding points in source image. As these points are coming to be as decimal, I have used scipy interpolate function to compute the accurate color at these positions from the neighborhood pixels.Using the pixel colors from those points I have replaced the color of pixels in corresponding points in the destination.

Another approach for trianulation is by using the warp affine inbuilt function. This gives the similar output but is much faster than barycentric approach.

The final output from this step is shown below:

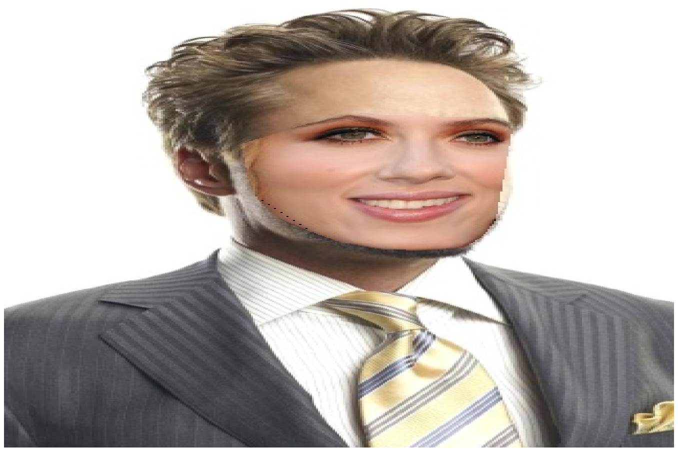

Fig. 6. Face Warping using Traingulation

E. Face Warping using Thin Plate Splines

As faces have compex structure, instead of using triangulation approach, It is better to used thin plate splines as they will be able to fit a smooth and differentiable function for such complex shape. TPS is like beating a thin sheet of metal at the feature points. We have 68 feature points, all these points are transformed by multiplying with weight of those control points.

These thin plates are shown in figure below:

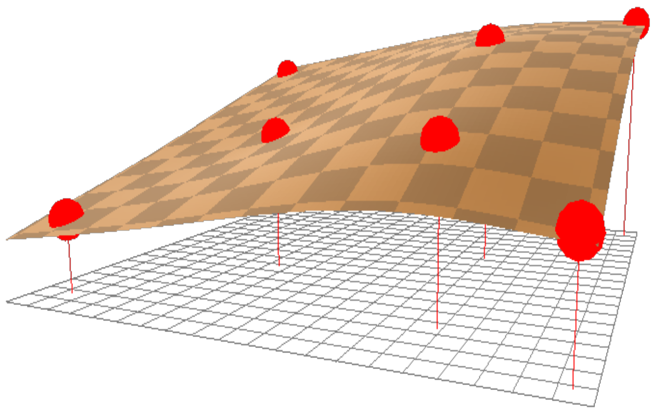

Fig. 7. Thin Plate Splines

These TPS are used for both x coordinates and y coordinates seperately and used to find the corresponding points in the images.

The output of warping using TPS is shown below:

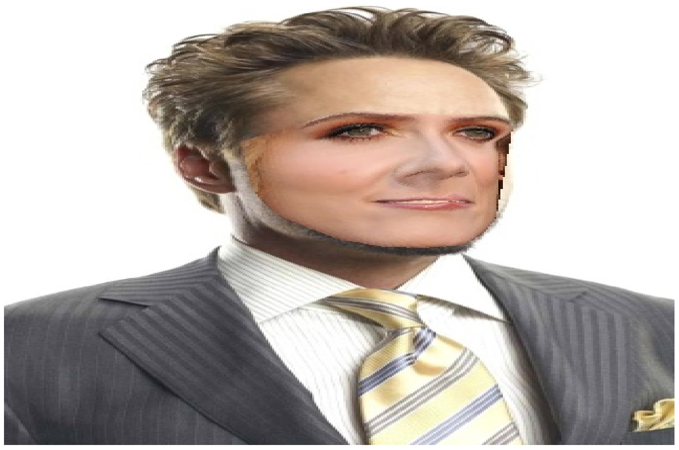

Fig. 8. Face Warping using TPS

It gives smooth output in comparison to previous technique used. The main difference however is because for triangulation convex hull points are considered but for this technique all the facial points are considered. The difference will be less visible if average point approach is used for triangulation for correspondences.

F. Blending

Now as we can see from previous section outputs, that there is a texture difference between the outputs of the images. To improve the output, poisson blending is used. OpenCV has inbuilt function seamless clone for doing this blending. I have used that function for blending these two images.

Two types of cloning can be performed using the inbuilt function- Normal and Mixed. In Normal Cloning the texture (gradient) of the source image is preserved in the cloned region. In Mixed Cloning, the texture (gradient) of the cloned region is determined by a combination of the source and the destination images. Mixed Cloning does not produce smooth regions because it picks the dominant texture (gradient) between the source and destination images.

For this project, I have used Normal cloning.

The output of this blending technique after warping is shown below-

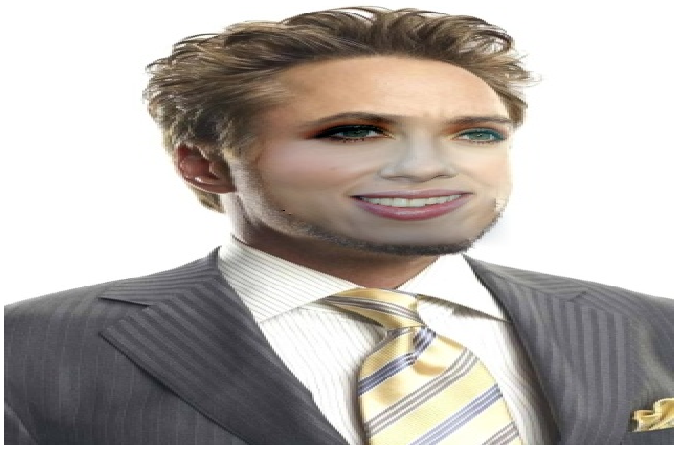

Fig. 9. Final Output using Triangulation

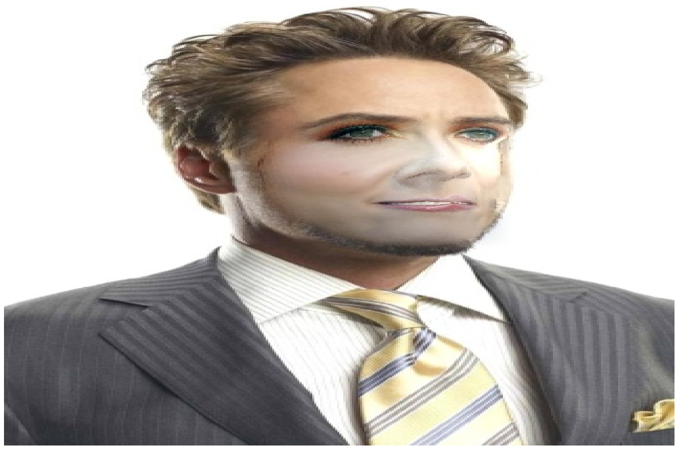

Fig. 10. Final Output using TPS

G. *Motion Filtering*

Our goal was to swap faces in the videos, after following the above pipeline I was able to get good results, but there were some flickering issues.
I thought of two ways to avoid this type of issue- 1. We can take average of faces from two consecutive frames after warping and then find a average face. Insert that between those frames. 2. The approach I tried was using cv2.fastNlMeansDenoisingColoredMulti The first argument is the list of noisy frames. Second argument imgToDenoiseIndex specifies which frame we need to denoise, for that we pass the index of frame in our input list. Third is the temporalWindowSize which specifies the number of nearby frames to be used for denoising.
I did not find much difference in output even after applying this filtering approach.

II. Phase2

A. *Overview*

PRNet generates a full 3D mesh of the face and its dense correspondence from a given single 2D image. A UV position map, which is a 2D image recording the 3D coordinates of a complete facial point cloud, and at the same time keeping the semantic meaning at each UV place. Position map Regression Network (PRN) is a convolutional neural network which jointly predicts dense alignment and reconstruct 3D face shape. The architecture of PRN is shown in figure below. Green rectangles are residual blocks and blue rectangles are transposed convolutional layers.

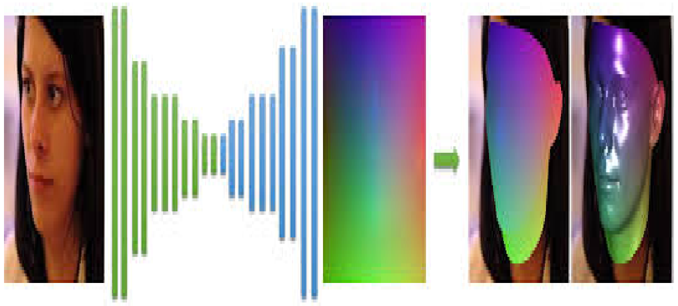
Fig. 11. PRN

The face swapping output generated from PR Network is shown in figure below:

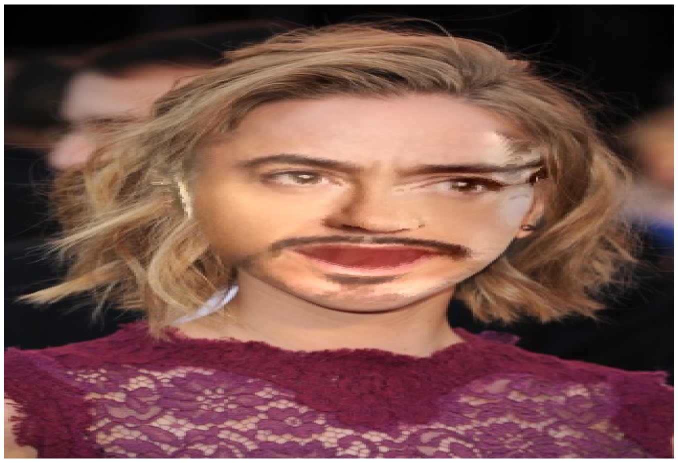
Fig. 12. Final Output using PRNet

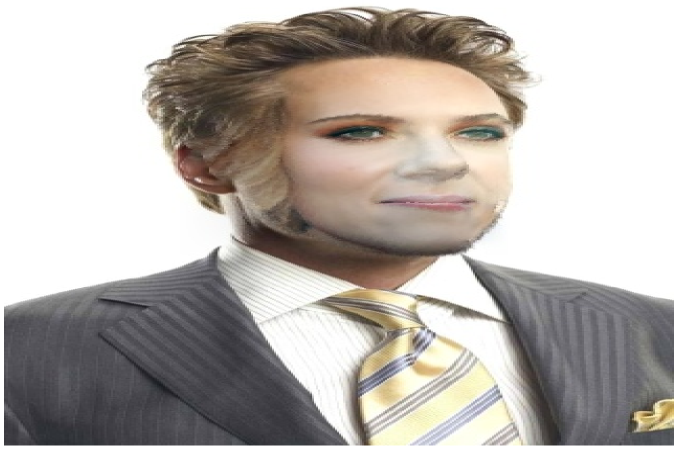
Fig. 13. Final Output using PRNet

As we can see from the outputs , the face swapping output picks features from forehead and sides as well.

This PRNet uses cnn model of dlib to find features. It gives output facial features as 68x3. It gives the values of facial features in three dimensional. Instead of previous traditional technique where we are getting facial features in 2-d.
To generate the final output, same pipeline is being followed as it was in traditional approach. The only difference is using the output from 3d mesh of face. The comparison of output is done in next section.

III. Output Discussion

I have saved all the output videos in TestSetOutput folder and Data folder. The output is generated using affine inbuilt, triangulation and barycentric, thin plate splines as well as prnet approach.

The fastest among the four approaches is using warp Affine function after traingulation. Barycentric approach is the slowest among these. I have to resize the inputs for some videos to speed up the output.

Using traditional approach, there are more frames in which dlib 68-point hog+svm detector failed to detect the faces especially in last Test3. But dlib used in prnet with cnn model is able to detect faces even sideways properly.

The best results are obtained from prnet in all the test videos.In some frames output from TPS and prnet are comparable. But prnet has some extra forehead features from the source where as traditional method does not have any forehead features.

I have attached the frame from Phase1 and Phase2 output. Phase2 is little blurry because of resizing.
Please follow the read me instructions to run the code.

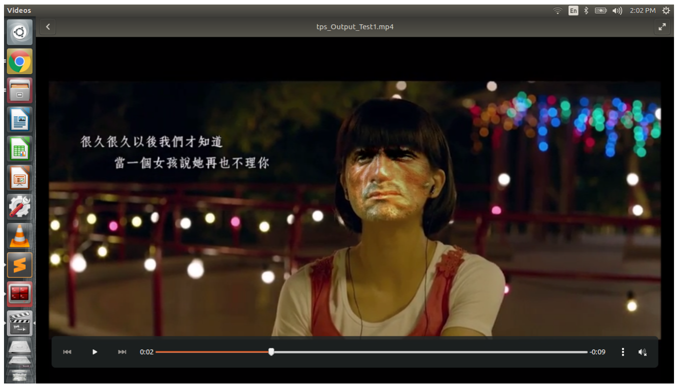

Fig. 14. Final Output using Phase1

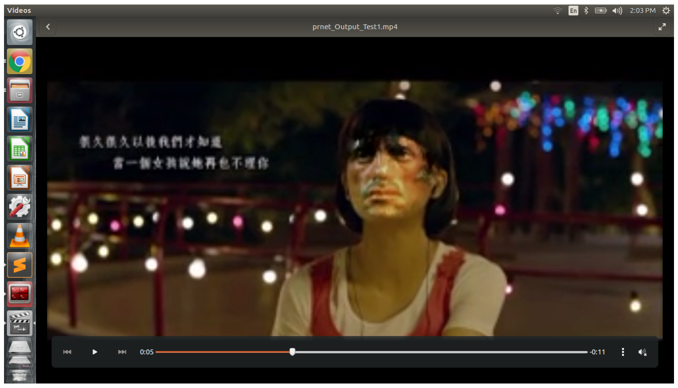

Fig. 15. Final Output Phase2

References

[1] Image Denoising- https://opencv-python-tutroals.readthedocs.io/en/latest/
[2] TPS- https://profs.etsmtl.ca/hlombaert/thinplates/
[3] Seamless Cloning - https://www.learnopencv.com/seamless-cloning-using-opencv-python-cpp/
[4] FaceSwap - https://www.learnopencv.com/face-swap-using-opencv-c-python/
[5] PRNET- https://github.com/YadiraF/PRNet
[6] Project Guidelines- https://cmsc733.github.io/2019/proj/p2/