CIS 581, Computer Vision, Fall 2018: Project
Face Swapping

Abhijeet Singh
abhsingh@seas.upenn.edu

Shikhar Brajesh
shikharb@seas.upenn.edu

I. Introduction

Face swapping is the technique of replacing the one person face with another person's face without making any change to the background or the body of the target. In this project, we perform face swapping on objects in one video to another. We take the face from a video ("source video") and transfer it to another video ("target video") while keeping the emotions of the source video. In such a situation, the result will have the background and body of the target video and the face and emotions of the source video. The opposite, i.e. preserving the emotions of the target face, is also performed and the results are shown in section IV*.

II. Methodology

A. Pipeline

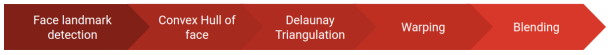

Fig. 1. Pipeline

We perform face swapping using a series of steps shown in the figure 1. For each pair of source video frame and target video frame, we perform these steps in the order below.

1) **Face landmark detection** - Facial features are detected using the *face_recognition* python library [1]. This library uses *Dlib* [2] that in turn uses deep learning algorithms to accurately detect faces from images. It gives 72 landmarks for a forward facing face that correspond to various regions such as *nose bridge*, *left eye*, *lips*, *right eye*, *left eyebrow*, *chin* etc. These landmarks are simply represented as the (x, y) coordinates of these features. With this detailed and specific information of the face features for both the source frame and the target frame, we move on to the next step.

*Due to some ambiguity in the definition of "source" and "target" in the project description, we decided to perform face swapping while preserving both source emotion (section III) and target emotion (section IV).

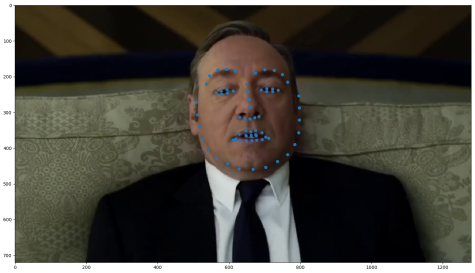

Fig. 2. Face landmarks

2) **Convex hull of face** - Here we find the enclosing region of the face that will ultimately be replaced. From the facial landmarks in the previous step, we get this by finding a convex hull of the points. The result is shown in the figure 3. We use the OpenCV [4] function *cv2.convexHull()* for this step.

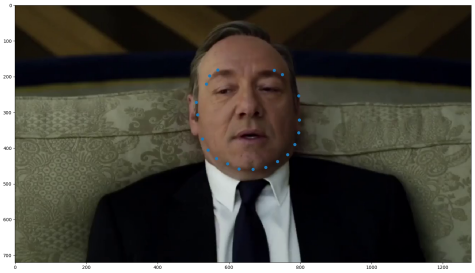

Fig. 3. Convex Hull

3) **Delaunay triangulation** - Next, we want to warp the enclosing region (the convex hull) of the face to the target image's hull. One way to do that is to morph the images from source to target. An easy way to accomplish this is to find the Delaunay triangulation of the target points (figure 4) and then warp each corresponding triangle of the source frame into the target frame.

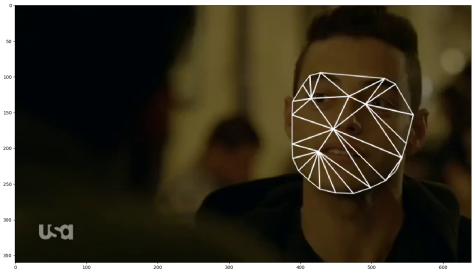

Fig. 4. Delaunay triangulation

We found that the warping step improves considerably if we take a few points from interior of the face for the triangulation step, instead of finding the Delaunay triangles for the convex hull points only. We select points as shown in figure 5. We cannot select all interior points as that will leak the emotion of the target in the resulting image, so we select only one point from each of the landmarks (eye, lips, nose etc). This way, we maintain the emotion of the source image and get good results after warping.

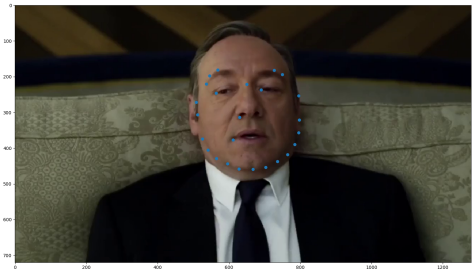

Fig. 5. Convex Hull with internal points

4) **Warping** - Warping is done using affine transform. From the previous step, we get corresponding Delaunay triangles of the source image and target image. That's all we need as an affine transformation matrix can be defined by selecting three corresponding 2D points. For each triangle, we find this matrix and warp all the points within the triangle to the target video frame. We get a result as shown in figure 6.

Fig. 6. Warped image

5) **Blending** - The final step is to seamlessly blend the warped source image into the target frame. This is achieved using the OpenCV function *cv2.seamlessClone()* which takes care of the illumination of the target and applies it to the warped source image to produce a naturally looking blended image.

Fig. 7. Blended source image

B. Optical Flow

The steps described in the previous section lead to good results a lot of jitter is observed. Including interior points while calculating the Delaunay triangles (as mentioned in the previous section) helped mitigate a lot of jitter. We try to improve the resulting video further by using optical flow. Face swapping is performed for every 5^{th} frame in the target video. For the others, the warped source image is transformed into the next frame by finding the optical flow (of the target face) from the current target frame to the next target frame. This leads to improvement in resulting video quality (less jitter) and also reduces the Face Swapping processing time (as facial landmark detection takes a considerable amount of time on a CPU).

C. Multiple frames

Till now, we were using one frame of the source and one frame of the target (at the same time interval) to perform the face swapping. We can do better by finding the closest source frame to a given target frame and use that to warp into the target frame.

First we tried to find the "closest" source frame using the *face_recognition.api.face_distance* from [1]. But it did not

give good results as this distance basically tries to find similar faces from a bunch of faces of different people. Our problem is to find the best orientation of the source frame that fits in the target face's frame. So we tried some heuristics for that and came up with two features to represent the "orientation". First feature was the ratio of the distance of left eye and nose tip to the distance of the right eye and nose tip. This gives whether the face is oriented towards left or right (depending on the ratio is less than 1 or greater). The second feature was the ratio of the distance of the top lip from the bottom lip to the distance of the top lip from the nose tip. This should tell us whether the person in the frame is speaking or not. We use ratio so as to make it scale invariant of face size.

We use these features and for each target frame, find the closest source frame using the Euclidean distance of the features. Then, we proceed on to face swapping using this closest source frame.

D. CLAHE

We use CLAHE [3] to improve contrast in images. This step is useful for good facial landmarks detection in cases of illumination changes as can be seen in figure 8 and 9.

Fig. 8. Original image

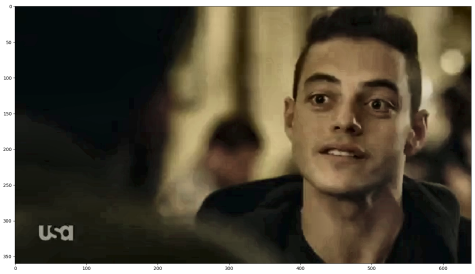

Fig. 9. Image after CLAHE

E. Swapping faces within a video

We also tried to swap people's face within a video (as opposed to a separate source and target video). The results can be seen in the next section.

F. Real-time Face Swapping

Using optical flow, we can also try to perform real-time face swapping on easy level videos. For each 20^{th} frame we perform face swapping and for other frames we use optical flow to move the source face on the target frame. The results are mentioned in the next section, though they are not particularly pleasing.

III. Results

Click the links (in the list below) to go to the YouTube video.

A. Train data

1) On Easy videos:

- Frank Underwood to Mr Robot.
- Frank Underwood to Jon Snow.
- Mr Robot to Frank Underwood.

2) On Medium videos:

- Russo1 to Russo3.
- Russo1 to Russo2.
- Frank Underwood to Russo1.

3) On Hard videos:

- Joker to Frank Underwood.

4) Two faces in the same video:

- Original. Face Swapped

5) Using multiple frames:

- Real-time speed for Frank Underwood to Mr Robot.

6) Using multiple frames:

- Joker to Frank Underwood.
- Joker to Leonardo.
- Russo1 to Russo2.
- Russo1 to Russo3.

We can see (specifically on the Joker to Frank Underwood video) that selecting from multiple frames improves the resulting video for harder cases.

B. Test data

IV. Preserving target face emotion

We also tried experimenting with face swapping while preserving the emotions of the target video. The only difference in technique was in the step of calculation of Delaunay triangles. To calculate the Delaunay triangles, earlier we were taking the points in the convex hull of the target and one interior point for each of the features like nose, eyes etc. Now, to preserve the target face emotion, we take all the facial landmarks and calculate the Delaunay triangles (shown in figure 10).

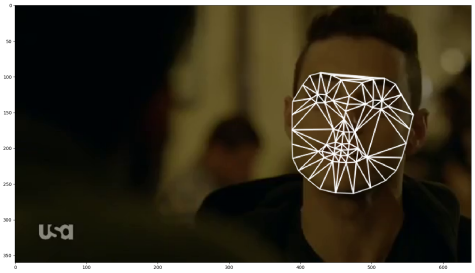

Fig. 10. Delaunay triangulation with all facial landmarks

Now, when we warp each triangle from the above triangulation, the pixels will be from the source face but the facial emotions - denoted by the shape of lips, eyes etc - will be of the target face. So if the target face (like Mr Robot in the above figure) has big eyes and wide open mouth, the resulting swapped face will also have big eyes and wide open mouth. No other change was made in the algorithm to achieve target face emotion.

The resulting videos are linked below -

A. Train data

1) On Easy videos:

- Frank Underwood to Mr Robot.
- Frank Underwood to Jon Snow.
- Mr Robot to Frank Underwood.

2) On Medium videos:

- Russo1 to Russo3.
- Russo1 to Russo2.
- Frank Underwood to Russo1.

3) On Hard videos:

- Joker to Frank Underwood.

4) Two faces in the same video:

- Original. Face Swapped

5) Using multiple frames:

- Real-time speed for Frank Underwood to Mr Robot.

6) Using multiple frames:

- Joker to Frank Underwood.
- Joker to Leonardo.
- Russo1 to Russo2.
- Russo1 to Russo3.

B. Test data

V. Future Work

The process described above works quite well on easier videos - where the subject does not move abruptly and the it remains mostly forward facing. These limitations can surely be removed by using better face detection algorithms that can detect side-view faces for example.

Another area of improvement can be to increase the number of features while using the Multiple Frames technique as described previously. We use only two features (namely to detect left-right orientation and to detect whether the subject is speaking or not) and we can surely add more to detect the best matching face.

We can also look into the optical flow technique and use it in a better way to track faces effectively.

References

[1] Face Recognition, `https://face-recognition.readthedocs.io/en/latest/readme.html`

[2] Dlib, `http://dlib.net/`

[3] Reza, A. M. (2004). Realization of the contrast limited adaptive histogram equalization (CLAHE) for real-time image enhancement. Journal of VLSI signal processing systems for signal, image and video technology, 38(1), 35-44.

[4] Bradski, Gary, and Adrian Kaehler. "OpenCV." Dr. Dobbs journal of software tools 3 (2000).

[5] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2018-12-09].

[6] Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).